Supporting Quality Assurance with Automated Process-Centric Quality Constraints Checking

Christoph Mayr-Dorn
Johannes Kepler University
Linz, Austria
christoph.mayr-dorn@jku.at

Michael Vierhauser
Johannes Kepler University
Linz, Austria
michael.vierhauser@jku.at

Stefan Bichler
Johannes Kepler University
Linz, Austria

Felix Keplinger
Johannes Kepler University
Linz, Austria

Jane Cleland-Huang
University of Notre Dame
Notre Dame, USA
JaneHuang@nd.edu

Alexander Egyed
Johannes Kepler University
Linz, Austria
alexander.egyed@jku.at

Thomas Mehofer
Frequentis AG
Vienna, Austria
thomas.mehofer@frequentis.com

***Abstract*—Regulations, standards, and guidelines for safety-critical systems stipulate stringent traceability but do not prescribe the corresponding, detailed software engineering process. Given the industrial practice of using only semi-formal notations to describe engineering processes, processes are rarely "executable" and developers have to spend significant manual effort in ensuring that they follow the steps mandated by quality assurance. The size and complexity of systems and regulations makes manual, timely feedback from Quality Assurance (QA) engineers infeasible. In this paper we propose a novel framework for tracking processes in the background, automatically checking QA constraints depending on process progress, and informing the developer of unfulfilled QA constraints. We evaluate our approach by applying it to two different case studies; one open source community system and a safety-critical system in the air-traffic control domain. Results from the analysis show that trace links are often corrected or completed after the fact and thus timely and automated constraint checking support has significant potential on reducing rework.**

***Index Terms*—software engineering process, traceability, developer support**

I. Introduction

Software quality assurance (QA) focuses on ensuring and attesting that the engineering processes result in appropriate quality of the software. To this end, various regulations, standards, and guidelines stipulate stringent traceability paths [1], [2], but don't prescribe the corresponding, detailed software engineering process. Examples in safety-critical systems include the FDA principles in the medical domain, DO-178C/ED-12C for airborne systems, ED-109A for air traffic management systems, and Automotive SPICE in the automotive industry. Here, QA engineers need to inspect fine-grained constraints over properties of engineering artifacts (i.e., requirements, models, code, test cases, etc.) as well as trace links at specific points in time (i.e., in different process steps such as requirement elicitation, specification refinement, coding, or test case specification). The current practice in industry, however, is using semi-formal descriptions to specify processes [3]. As a result, software engineering processes are rarely "executable", meaning that there is little to no automated support for checking whether these processes are followed, and to what extent deviations occur during development.

In this paper we specifically address the problem of *Developers* and *Quality Assurance Engineers* being overwhelmed by the complexity and extent of adhering to, and evaluating, QA constraints. Typically, developers work on multiple projects, sometimes simultaneously, with each project having different quality standards or guidelines.

In an informal study with our industry partners, developers reported being stressed about potentially missing important steps mandated by quality assurance. QA Engineers, on the other hand, need to conduct countless, tedious, often mind numbing checks that involve (manually) navigating across diverse artifacts and tools to ensure that the required constraints are fulfilled at the right process step. These checks are error prone and rarely conducted in time to provide immediate feedback to developers. Remediating problems later on, however, interrupts developers who may have already moved on to other steps or projects, causing even further delay.

In this paper we present a novel approach that aids developers and quality assurance engineers to reduce the effort required in ensuring that development activities adhere to the intended process—ultimately leading to less rework. Our approach relies on passive process execution, achieved by tracking process steps via monitoring engineering artifacts such as requirements, design documents, issues, change requests, and tests. This is complemented by constant evaluation of quality constraints. The key novelty is treating (quality) constraints neither as an implicit part of the engineering process model nor as completely disjunct from it. Instead, we propose treating (quality) constraint evaluations as first class citizens: i.e., as explicit development artifacts that determine process progress. This contrasts with existing work on traceability [2], [4] where links required by regulations and standards are typically verified by an auditor at the end of a process stage or prior to shipping the final product [5]. Similarly, work on constraint checking [6], [7] primarily focuses on consistency among diverse artifacts without addressing consistency issues

between these artifacts and the underlying process.

The key contributions of this paper are as follows:

- A process model that decouples process control and data flow from QA constraints.
- A passive process engine which explicitly tolerates inconsistencies [8] and allows engineers to temporarily deviate from the process while providing them with timely feedback on QA constraint evaluation results.
- A prototype that helps developers clearly understand when they have completed a step or what they still need to provide (e.g., specific content or trace links).
- An evaluation against an open source system for unmanned aerial vehicles (UAVs) and an industrial air traffic control system (ATC), that measures the extent to which the prescribed process was followed.

The remainder of this paper is structured as follows. Section II motivates our work by describing constraint checks for the development of safety-critical systems. Sections III and IV provide an overview of our approach and introduce details on modeling the process and constraints. We then evaluate our approach using two distinct case studies in Section V, and conclude with a discussion of results (Section VI), threats to validity (Section VII), and related work (Section VIII).

II. Motivating Example

Our industry partner Frequentis is a world leading voice communication provider for air-traffic control and command-control centers. In this domain the DO-278/ED-109 standard [9] specifies traceability requirements according to different design assurance levels, ranging from "Catastrophic" to "Minor" or "no Effect".

However, while DO-278/ED-109 defines *which* trace links need to be available, it does not specify *when* these trace links need to be established or *who* should/must perform this task.

Figure 1 (left) depicts a partial traceability information model (TIM) and excerpts from a simplified development process representation (right) for one Frequentis product. The process example indicates that once high-level requirements (HLReq) and high-level design specifications (HLSpec) are reviewed, HLReq may be refined into low-level requirements (LLReq) and cause updates to the low-level design specification (LLSpec). Thus, developers should not start refining LLReq without the outcome of the review even though HLReqs and teams are already assigned to work packages during the development iteration planning phase. Additionally, before the implementation of LLReq can start, trace links from HLReq to LLReq and trace links from LLReq to LLSpec need to be reviewed. Constraints that are set up for the development process are not only used to check for the existence of an artifact or the existence of a trace link between the right type of artifacts. Constraints build on the properties of an artifact at a specific process progress. At the end of step S3 (cf. Figure 1) the respective LLReqs need to be in state "Complete", each having a trace link to a HLReq. Upon completion of step S7 when a LLReq's verification method is "Demonstration", then there must exist at least one trace link to a respective Test Case which is not of type "Software Test Case" (but, for example, a simulation, demonstration, or acceptance test). These constraints are complementary to human-in-the-loop QA measures such as the trace reviews in S5 and S6. The automated checks reduce the effort for these reviews by ensuring the traces under reviews are syntactically correct and thus the review can be done more efficiently.

Engineers who are updating the LLSpec (in S4), for example, thus need to be aware when they may start with their step to avoid rework if additional refinements of an LLReq occur. Similarly, engineers in S7 need timely feedback when they can claim to have finished implementation and thus trigger the review in S8.

Knowing the state of the process helps assess the risk of deviation. Starting early on HLReq refinement (S3) may be too risky if non of the requirements have gone through review in S1 and S2 but perhaps necessary and ok due to time pressure when S2 has only a few requirements left to review. The following sections use this TIM, process, and constraints as a running example to better explain process and constraint modeling, as well as execution.

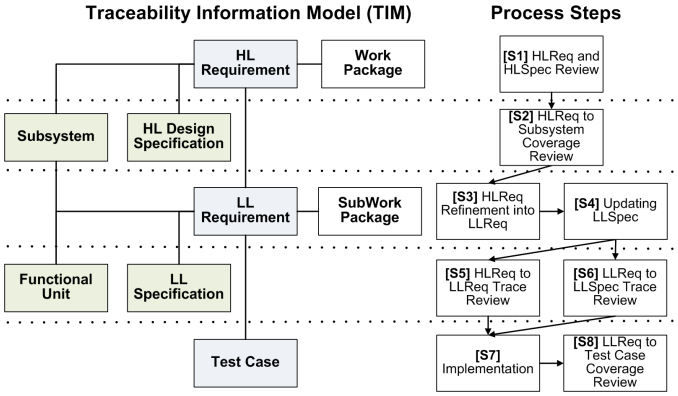

Fig. 1: Frequentis' simplified traceability information model (TIM) excerpt (left) and process model excerpt (right).

III. Approach – The ProCon Framework

In this section we provide a comprehensive overview of ProCon, our passive **Pro**cess execution and quality **Con**straint support framework. Two key aspects that characterize our framework are: first, the integrated handling of explicitly distinct processes and constraints, and second, the tracking of engineering progress achieved through explicitly linking process descriptions to software engineering artifacts.

In ProCon, a passively executable *Process Specification* describes the sequence and alternatives of carrying out engineering activities (i.e., the "control flow") and the software engineering artifacts that serve as inputs and outputs of those activities (i.e., the "data flow"). Explicitly modeling a software engineering process for controlling the software engineering life-cycle is not new, with a plethora of research dating back until the 90s [10]–[13]. ProCon is different insofar as that (i) the process is tracked in the background, based on the

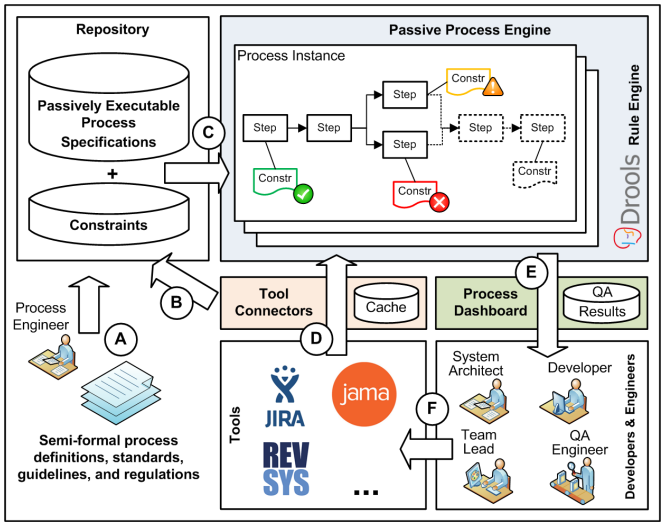

Fig. 2: The ProCon framework.

software engineers' activities performed in the tools they are using in their daily work rather than requiring engineers to interact with a process engine, (ii) engineers are free to deviate from the process, (iii) engineers may receive guidance even in the presence of deviation, and (iv) ProCon supports control- and dataflow conditions as well as constraints across diverse artifact types and tools.

We refer to these abilities as "passive execution" as our framework goes beyond simple process monitoring by determining available future steps, detecting premature steps, and making this information immediately available to engineers.

With ProCon, deviations from the process are tracked via process and quality constraints. These constraints, and their respective evaluation results, are treated as first-class citizens in the software engineering process (model), hence are explicit software engineering artifacts of their own. This in turn allows for explicitly evaluating constraints as soon as actions are performed and constraint evaluation results can provide valuable insights about the status of the process beyond whether a step has been completed or not. Additionally, treating constraints and their evaluation results as first-class citizens makes reuse and maintenance less cumbersome. Constraints may be modified over time to accommodate changes in the organization's process, or may apply in diverse process contexts, making them amenable to product line engineering approaches. Imagine applying the concepts of software product line engineering for better managing the variations in process and QA concerns found in a larger organization [14].

A. ProCon High-level Architecture

ProCon consists of four major elements (depicted in Figure 2) that together allow defining, checking, and maintaining the process and development artifacts of an organization:

Existing **Semi-informal process definitions, standards, guidelines, and regulations** serve as the initial input for ProCon. Typically, these already exist within an organization and describe (and motivate) the prescribed processes and quality assurance measures. Process definitions may be SPEM [15] documents, PDFs with flowcharts, or simple text documents outlining the steps and responsibilities of the different roles.

The process definitions are complemented by – and used in conjunction with – a variety of diverse **Tools** within an organization to create, update, and maintain the artifacts that represent the input and output of the different process steps. Artifacts such as issues often serve as a (partial) informal representation of process instances. *Tool Connectors* (B) for the respective tools provide (machine readable – e.g., as JSON data via a REST API) access to artifacts. These connectors take on sophisticated tasks such as obtaining artifact updates (via polling or subscriptions), managing which artifacts are relevant for a process and thus need to be monitored, caching the artifacts for quick repeated access, and keeping this cache up to date (cf. Section V-A).

Passively Executable Process Specifications and **Constraints** (C) are manually derived (cf. Section IV-A and IV-B, respectively) based on the above two main inputs. Process Specifications formalize the (semi-)informal process definitions and guidelines and allow a fine-grained mapping to the respective artifacts, properties, and their changes observable via the tool connectors.

The **Passive Process Engine** (D) manages instances of passively executable process specifications (or *Process Instance*s for sake of brevity). The passive process engine obtains artifact state and events from tool connectors (E), feeds these changes into a rule engine in which the process progress conditions and QA constraints are evaluated. The rule engine eventually fires events that the process engine utilizes to update the process progress and quality constraint evaluation results (cf. Sections IV-C and IV-D, respectively).

A web-based **Process Dashboard** makes Process progress and QA constraint evaluation results (F) continuously available to software engineers. It allows controlling the passive process instance: triggering new instances, observing their progress, and eventually archiving them (cf. Section V-A).

B. ProCon Usage

ProCon users are primarily QA engineers and developers, but also include stakeholders such as product managers and team leads. The former can use the framework with a focus on quality assurance (the focus of this paper), while the latter can use the framework with a focus on process progress.

Based on this, ProCon supports two distinct use cases. In the *process and constraint modeling use case*, various stakeholders map the informal process definitions, standards, etc. (A) to passively executable process specifications (C). This phase involves analyzing how engineers currently produce evidence of process execution in the various tools (B). The tool connectors describe what artifact properties are available and thus what change events may serve as process progress triggers during process execution: for example, what (custom) properties are used for various Jira issues, what trace links are available to navigate from a Jira [16] issue to a requirement

managed in Jama [17]. The outcome of the modeling use case is a passively executable process specification and its quality constraints.

Note, that our approach does not require modeling the complete process if tools don't allow access to certain information. One would typically start with those parts that are most important, or most error prone, etc and focus on these. Ambiguity may arise, for example, if rules expect one trace to navigate to an artifact but find multiple ones, then a random one will be selected. Mitigation includes correcting and/or extending QA-constraints to identify the ambiguous situation.

In line with Lee Osterweil's key observation that "software processes are software, too" [18] we advocate the use of contemporary software engineering practises such as developing in iterations, testing, versioning, and issue tracking. A key element here is replaying of the artifacts' history to test the constraints' ability to detect deviations [19].

The second use case focuses on the passive process execution. Upon process instantiation, the passive process engine obtains artifacts and their changes as they occur in tools (D). The engine then tracks the progress of each step and attached constraints. It determines completed and in progress steps (Figure 2 center, with solid border), which steps an engineer is free to start next, and which ones should not yet start (dashed border). Artifact and process updates trigger reevaluation of constraints (document symbols with icons). ProCon users then access the process progress and constraint evaluation results (E). An engineer may notice an unfulfilled constraint, conducts the necessary artifact changes via the tools, triggers reevaluation, and confirms quality constraint fulfillment. Note that users exclusively affect process progress and constraint evaluation results via tool interactions (F) and never via direct interaction with the passive process engine itself (except for explicit triggering of quality constraint evaluation). In the next section we describe the process and constraint modeling in more detail as well as process execution.

IV. Process, Constraint Modeling, and Execution

The following section describes how Process Specifications are structured and executed, and discusses the elements and transitions of the ProCon Process Specification metamodel.

A. Process Specification Model

The challenge when passively executing processes is determining the steps that are currently available for engineers to work on, steps that represent work in progress (but perhaps shouldn't be worked on yet), and finally, steps that have been (successfully) completed. The prevalent differences to existing approaches hereby are not the basic building blocks (i.e., the process steps) but rather in how transitions between these steps are defined and subsequently triggered. Figure 3 (left) provides a simplified UML class diagram of the process model's main elements. The two main elements of the process model are *Steps* and *Decision Nodes* attached to them.

A **Step** describes what an engineer *"should"* do (in contrast to "must" do – as prescribed by a a more restrictive traditional process). For example, refine a requirement, implement a feature, define a test case. A step has zero or more *Input* artifacts attached that represent required data to make a decision or artifacts that need to be modified. It further has zero or more *Output* artifacts that describe the effect of having executed the step (e.g., having modified an input artifact or created a new artifact). Input and output artifacts can represent any kind of information such as requirements, tests, issues, or trace links. In addition to the textual description of an engineer's activity, a step consists of a set of event-condition-action rules that define which event(s) from the engineering environment (e.g., an artifact update), given additional constraints (i.e., the condition) trigger the inclusion of an artifact to a step's output artifact set (i.e., the action part of the rule). For example in S5, "When an engineer posts a review URL as a Jira issue comment, then add that link as the step's output artifact".

A **Decision Node** describes how the completion of one or more *Steps* – and additional conditions – leads to the execution of subsequent steps. The set of decision nodes thus defines the process' control flow. A decision node's *DataTransfer* declaration describes how the output of one step becomes the input of a subsequent step, thereby defining the process' data flow. For example, "The LLReq output artifacts of S3 and LLSpec output artifacts of step S4 become the input artifacts to step S6". Note that a step may only have one preceding and one subsequent decision node to avoid conflicting control or data flow. Only a decision node may link to multiple steps. Ultimately, a process consists of a set of steps and decision nodes that create a single connected, directed graph. Note that currently loops are not yet supported as the use cases at our industry partner did not require this feature for two reasons: first, artifacts can be updated over and over again until QA-constraints are fulfilled. Second, longer, explicit loops such as sprints are typically represented as separate sub/processes and thus are "spawned" separately.

The specific activation conditions are placed on the *InFlow*s (from preceding steps to decision node), on the node itself, and on the *OutFlow*s (from decision node to subsequent steps). Figure 3 (right) outlines how these conditions reflect in a decision node's state.

- *AVAILABLE*: upon instantiation, a decision node is in the *AVAILABLE* state. Depending on a decision node's *InFlowType*: AND, OR, XOR, it will check upon each update of preceding steps whether all, at least one, or exactly one InFlow are fulfilled to transition the node into *PASSED_IN_CONDITIONS*.
- *PASSED_IN_CONDITIONS*: in this state, conditions independent on any preceding step, such as process or date-time centric conditions need to be fulfilled before transitioning into *PASSED_CONTEXT_CONDITIONS*.
- *PASSED_CONTEXT_CONDITIONS*: in this state, the decision node evaluates OutFlows. OutFlow conditions serve as a filter to dynamically select the relevant subsequent steps to enable. Depending on a decision node's *OutFlowType* (AND, OR, XOR), it will check whether all or at least one OutFlow is fulfilled, then transitioning

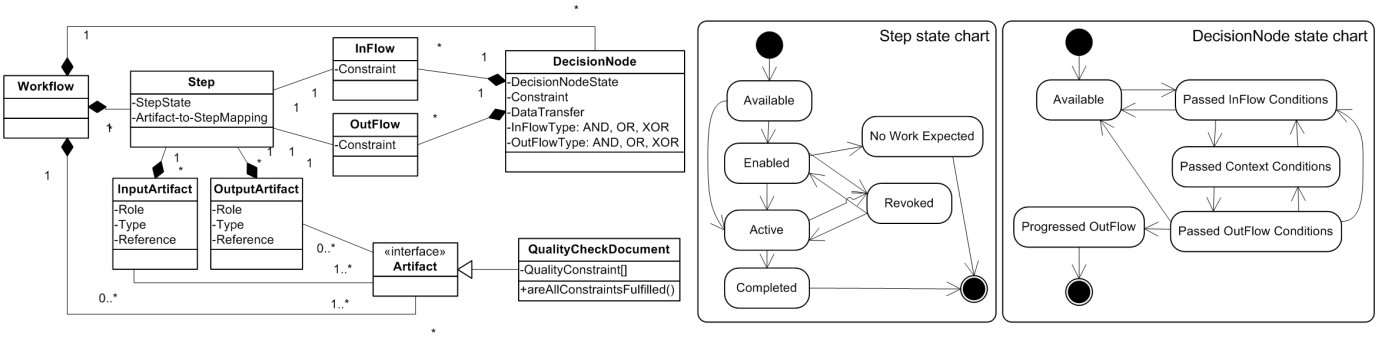

Fig. 3: Process specification meta model: UML class diagram and state charts.

into *PASSED_OUT_CONDITIONS*. In case of XOR, we still need to enable all steps of fulfilled OutFlows as this indicates a situation where the engineer decides which one is the relevant one.

- *PASSED_OUT_CONDITIONS*: this is a temporary state to separate between having fulfilled all OutFlow conditions and having also instantiated the subsequent steps including data transfer.
- *PROGRESSED*: all subsequent steps identified by OutFlow conditions are instantiated and have the required input artifacts available according to the specified data transfer.

As long as a decision node hasn't reach *PROGRESSED*, any change to the InFlow, context, and OutFlow conditions will transition the state back to an earlier state.

Note that the conditions on InFlow, context, and OutFlow are optional. For example, the decision node between step S1 and step S2 (not shown in Figure 1) does not have any of these three conditions. Upon completion of S1 it will immediately transition through all these states into state Progressed, thus serving as a simple trigger to activate step S2.

In addition to the decision node state, tracking each step's state is vital to provide developers with feedback on which steps are ready for starting, which ones have started too early, and which ones should not be done. Figure 3 (middle) depicts a step's life-cycle modeled as a finite state machine.

- *AVAILABLE*: when a step is instantiated it resides in the *AVAILABLE* state, indicating that it's input is not sufficient yet (i.e., it has not yet obtained the necessary data from the preceding decision node).
- *ENABLED*: once all specified input conditions are met (for example, required input artifacts are available), a decision node causes the step to transition to *ENABLED*, indicating that an engineer is free to start working on it.
- *ACTIVE*: when *Artifact-to-Step Mapping*s signal that artifacts attached to the state are updated or modified, the step becomes *ACTIVE*, indicating that an engineer is actively working on it.
- *NO_WORK_EXPECTED*: when multiple mutual exclusive steps are *ENABLED* it can initially not be determined which of these an engineer has chosen. Once, one of the alternative steps transitions into *ACTIVE*, the remaining steps transition into *NO_WORK_EXPECTED*.
- *REVOKED*: when a step's input conditions are no longer fulfilled, it transitions into *REVOKED* to indicate that an engineer should not/no longer work on this step, or if starting, respectively continuing, may need rework later on. Once the input conditions are re-established, the step transitions back into *ENABLED* or *ACTIVE* (depending on its previous state). The key difference between a REVOKED step and *NO_WORK_EXPECTED* step is that the former should eventually be carried out, while the latter should not be carried out at all.
- *COMPLETED*: when all output conditions are met (e.g., all required output artifacts are available), the step transitions into *COMPLETED* and triggers the evaluation of the step's subsequent decision node.

B. Quality Constraint Integration

For each step, *Quality Constraints* can be defined describing conditions the created or updated output artifacts must adhere to. A constraint can refer to a step's input and output artifacts, its metadata, or process metadata.

Each constraint has an identifier that allows triggering constraints not only upon artifact changes but also manually, on demand. This is important to provide control to the user and to allow, for example, an engineer to trigger a constraint check to reassure him/her that a step is indeed complete and nothing has been overlooked. The language in which constraints are implemented is flexible and can vary depending on the domain or application scenario. The only requirement is that it has to provide respective evaluation results (we provide further examples on constraints and the constraint language as part of the use cases in Section V) .

Every time a quality constraint is evaluated, a corresponding new instance of *Quality Constraint Evaluation Result* is created, reporting the result (i.e., fulfilled or unfulfilled) and lists the artifacts subject to that constraint. For example, a constraint for step S6 checks whether each low-level requirement (LLReq) output artifact has a trace link to a high-level requirement (HLReq) which must be in state "released". Therefore, the respective constraint evaluation result will contain the list of LLReqs that fulfill this constraint, and a list of LLReqs

that violate this constraint. For every constraint, we further store timestamps indicating when the constraint evaluation result was (last) evaluated, and when the evaluation result last changed from violated to fulfilled and vice versa. This, for example, allows to infer how recent results are when shown on the Process Dashboard. The evaluation results of all quality constraints associated with the same step are collected and bundled in a *Quality Check Document* which is added to the output artifacts of that step.

The process model itself remains largely independent from constraints and their evaluation results. While a process step's completion constraint must check whether a *Quality Check Document* output artifact exists which contains only positive *Quality Constraint Evaluation Result*s, it doesn't need to understand the particular constraints that resulted in the evaluation success. This loose coupling allows to execute the same process with different levels of quality assurance by switching in and out different quality constraints.

C. Passive Process Execution

Every process model comes with an activation condition, typically an explicitly added artifact such as a change request issue or work package issue that already serves as some form of process representation. The process engine instantiates the process and checks which steps and decision nodes can be instantiated based on the provided activation artifact. Instantiation occurs incrementally, i.e., only when a step reaches the state ENABLED, will the process engine instantiate the step's subsequent decision node. Similar, only when a decision node is in state PASSED_OUTFLOW_CONDITIONS, will the engine instantiate the steps identified by the outflow links. With step instantiation, the engine also instantiates the quality check constraints. Thus, as long as a step doesn't exist, none of its constraints will be checked.

In the passive process engine, incremental step and decision node instantiation is insufficient as engineers may decide to work on steps not yet ready. When artifact changes trigger *Artifact-to-Step Mapping*s, the process engine will instantiate the corresponding "premature" step and attempts to find an existing preceding decision node that this step can be linked to. If none exists yet, the step remains dangling in the process until the process progress catches up: i.e., upon instantiating a decision node, the engine checks if a dangling step exists that should be linked via one of the decision node's out flow links. From the engine's point of view, there is no difference between missing a step and starting the next, or starting too early on the next step. It will continue either way. In such as case, however, then the engine will not be able to fully execute a DataTransfer that requires the output of the skipped/incomplete step. The consequence is then highlighted via the step's status as having insufficient input artifacts.

Note that assessing the impact of prematurely starting a step and how to mitigate any potential change propagation is outside the scope of this paper. Premature steps stand out from regular steps by having transitioned directly from AVAILABLE to ACTIVE.

D. Constraint Checking

We made the deliberate decision to execute constraint checks upon explicit request and not automatically triggered by every single artifact change. Often a single change is not indicative of step completion. The reasons behind this are manifold. First, as quality constraints often span across multiple artifacts, a single change to an artifact is insufficient, multiple changes need to occur. For example in S6: not just one low level requirement (LLReq) needs to be set to "released" but all linked ones. Second, an artifact may be involved in multiple constraints, thus a change would trigger execution of multiple constraints. Third, reevaluation on every change puts unnecessary burden on the process engine, respectively its constraint evaluation subsystem. Forth, when quality constraints involve diverse artifact types managed in different tools, change events from these tools may not be readily available as the engineer carries out the change in these tools but need to be fetched periodically. Jama, for example, doesn't offer automatic event notifications but requires polling with subsequent explicit fetching of changed artifacts. This would result in executing checks on stale data.

V. Evaluation

We evaluated ProCon against two distinct use cases for which we created process specifications and constraints, and implemented connectors for different issue tracking and requirements management tools. The first use case, Dronology [20] – an open source project – represents an more agile, lightweight process, whereas the second one – a safety-critical system in the air traffic management domain – describes a rigid, standardized process with stringent quality assurance criteria. We report on the application of ProCon to the two use cases and our findings and lessons learnt. Supplemental online material [21] provides artifact data and trace links, process definitions including constraints, and the experiment results. The prototype source code is publicly available [22].

A. Prototype Implementation

We implemented a prototype to evaluate the two use cases and to obtain feedback from our industry partner Frequentis.

Tool Connectors: To cover a reasonably large set of artifacts from our industry partner Frequentis and from Dronology, we implemented connectors for Jira, a web-based tool for planning, issue tracking, and reporting, and Jama, a tool for requirements management, traceability, and test management. The Jira Connector uses the Atlassian Java REST API to retrieve artifacts and their attributes and is used to periodically poll for changes in these artifacts. Similarly, the Jama connector uses the Jama REST API. To reduce load on network and tools, the tool connectors cache Jira and Jama artefacts in a CouchDB (a schemaless JSON database).

Process Engine: The Process Engine is implemented in Java, containing an implementation of the process specification metamodel and a rule engine for checking constraints. We opted for the Drools rule engine [23], a Business Rules Management System that can be easily integrated into a Java

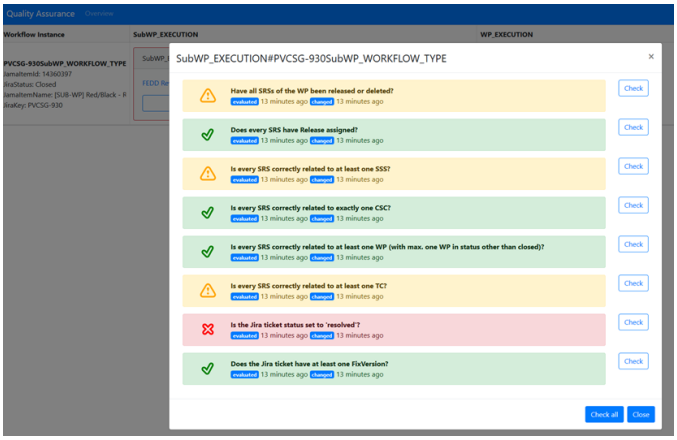

Fig. 4: ProCon Process Dashboard.

application and allows easy access to Java objects (representations of Jira and Jama artifacts) within rules written in a Java dialect. Additionally, we persist the processes and Quality Check Documents (including Quality Constraint Evaluation Results) attached to the different process steps in a Neo4J graph database. The graph structure simplifies fetching all related process steps, decision nodes, input and output artifact (references), and quality check documents in a single query to be shown on the process dashboard.

Passive Process Specifications and QA Constraints: The Drools rule engine evaluates process progress conditions and quality constraints. We therefore defined quality constraints as well as the decision nodes' control and dataflow conditions in respective Drools rules files.

Process Dashboard: Figure 4 shows the user interface for inspecting quality constraint evaluation results. The results contain browseable links to the original artifacts thus enable the engineers to quickly switch into their usual tools (here Jama and Jira) to fix any unfulfilled constraints.

B. Open Source Use Case: Dronology

Dronology is a UAV management and control system providing a full project environment for managing, monitoring, and coordinating the flights of multiple UAVs. It can interact with real hardware as well as a high-fidelity Software-in-the-Loop simulator that enables experimentation with virtual UAVs. Dronology was developed, with both students and professional developers, over several years as a research incubator with various development artifacts publicly available [20], [24]. For the purpose of this case study we obtained permission to use data from multiple sprints maintained in Jira from 2017 to 2019. This includes the following artifacts: Bugs, Hazards, Requirements, Design Definitions, Tasks, and Sub-Tasks.

Process and Constraint Creation: We treat each of these issues as "small sub processes" where the issue's state represents a process step, and quality constraints for each step describe the conditions that need to be fulfilled to transition from one step to the next, respectively complete the process (i.e., close the issue). Given the lean nature of typical agile open source development processes the states were limited to the default process steps in Jira, "Open", "InProgress", and "Closed". Based on the information available, we identified the following eight quality constraints and allocated them to the steps where they are most useful (note that some constraints are reusable for multiple issue types). Process and constraint creation took around 3 hours. We then contacted one of the lead developers to confirm the validity of the constraints and the process. An overview of the constraints can be found in Table I. At the end of step "Open", we require constraints D-C1 to D-C5 to be fulfilled, and at the end of step "In Progress" we require constraints D-C6 and D-C7 to be fulfilled.

C. Industry Use Case: Frequentis

For the Frequentis use case we selected a safety-critical product for voice communication in air traffic control centers. The product consists of several subsystems for interfacing with radio transceivers, managing near-real time voice streams, and providing operator user interfaces. Frequentis follows a V-model like engineering process. Specifically, for this case study, we focused on sub work packages (SubWP). This process for each team (each responsible for one subsystem) starts with high-level requirements resulting in the actual implementation and the successful execution of test cases. This covers steps S3 to S8 of the motivating scenario. Trace links between SubWP's and low-level requirements are therefore the main focus of QA constraints defined as the completion conditions of steps S3 (represented by ATC-C1 to ATC-C4), S4 (ATC-C5), S7 (ATC-C6 and ATC-C7), and S8 (ATC-C8) (cf. Table I). Frequentis uses Jama to manage all artifacts and trace links depicted in Figure 1 and uses Jira to manage the engineering process.

Process and Constraint Creation: We defined the constraints together with a QA engineer within two hours, and took another two hours to specify the process. Frequentis' informal process definition precisely defines how engineers need to set properties of Jira and Jama artifacts for completing the various steps. Changes to these properties serve as step completion signals in our process engine.

D. Data Gathering

We were granted access to the Dronology project's Jira server REST API to obtain artifacts and their change history. The data set consists of 802 process instances (i.e., Jira issues): 199 Tasks, 211 Sub-tasks, 109 Bugs, 247 Design Definitions, and 36 Hazards. From Frequentis we obtained Jira issues related to the aforementioned SubWPs. Each SubWP managed in Jira has a corresponding Jama artifact with respective trace links to LLReqs and subsequent artifacts. We used the Jama REST API to navigate across these trace links to collect all Jama artifacts (including their history) that are relevant for constraint evaluation. This resulted in a set of 109 SubWPs and ∼14,000 linked Jama items (out of which 1,121 are LLReqs).

We used our trace link replay tool [19] to reset the entire state of the dataset of Jira issues and Jama items and their trace links to the earliest change event and then replayed

Constr.	Description	Issue Type
D-C1a	The issue traces to one or more Design Definitions	Tasks
D-C1b	The issue traces to one or more Design Definitions directly, or via its parent	Task, Sub-Task
D-C2	The issue does NOT trace to a Requirement	Task, Sub-Task
D-C3	The issue has an assignee	Bug, Task, Sub-Task
D-C4	The issue traces to a Requirement	Design Def.
D-C5	The issue is mitigated by a Requirement (i.e., trace type: isMitigated) or refined by a Hazard (i.e., trace type: isRefined)	Hazards
D-C6	The issue has all related bugs (if any) closed	Task, Sub-Task
D-C7	The issue has all sub-tasks (if any) closed	Bug, Task
ATC-C1	All traced LLReq have status "released"	SubWP
ATC-C2	All traced LLReq have a release assigned	SubWP
ATC-C3	All traced LLReq have a trace link to at least one HLReq	SubWP
ATC-C4	No traced LLReq has a trace to another SubWP with a status other than "closed"	SubWP
ATC-C5	All traced LLReq have a trace link to exactly one Functional Unit	SubWP
ATC-C6	All traced LLReq have a link to at least one test case matching the requirement's verification method	SubWP
ATC-C7	The SubWP's Jira issue has at least one "Fix version".	SubWP
ATC-C8	The SubWP's Jira issue is set to "resolved"	SubWP

TABLE I: Constraints for the Dronology system (prefix "D") and the Frequentis system (prefix "ATC").

every single change in the correct temporal order. The changes occurred between April 2017 and December 2019 for Dronology and between May 2018 and June 2020 for Frequentis, respectively. Using the replay tool allowed us to start from the beginning of the development process and, step-by-step, simulate (i.e., "replay") changes made by engineers (e.g., modify the state of artifacts in Jira, add trace links, etc.) allowing us to automatically trigger constraint checks and track the process state the same way as in a "live" environment. For each constraint evaluation we evaluated (i) whether a step's Quality Check Document was fulfilled; (ii) which constraints were (not) fulfilled; and (iii) whether a step became active without the constraints of the previous one(s) being fulfilled.

Additionally, for a process instance (i.e., a Jira artifact), we collected the following metrics: number of Quality Check Documents un/fulfilled; number of un/fulfilled constraints; number of constraint checks performed; and the maximum number of past steps with unfulfilled constraints (i.e., how many steps an engineer advanced ahead without having the completion condition of the previous steps fulfilled).

E. Process Replay Results

Table II reports the QA constraint evaluation results across multiple process instances, grouped per process type. Row *AlwaysOk* shows the numbers of process instances where engineers only progressed to subsequent steps when all quality constraint in previous steps were fulfilled. Row *EventualOk* shows the processes for which all constraints were eventually fulfilled. The row *CompleteNotOk* shows processes for which some constraint were never fulfilled. Row *IncompleteOk* counts those process instances that were not finished by the end date of the timeframe but had all mandated constraints up to their current state fulfilled. Row *IncompleteNotYetOk* counts the partially completed process instances with unfulfilled constraints but no progress beyond those not fulfilled steps, in contrast to those with progress beyond in row *IncompleteProgressedNotOk*. Percentage values are reported relative to the sum of completed process instances, respectively sum of incomplete process instances.

Dronogy: we noticed that for "Task" processes no completed process instance (i.e., finished "Task") ever fulfilled every constraint before moving from one step to the next, yet around 30% fulfill all their constraints at the end, with $\sim$70% remaining unfulfilled at the end. "Sub-task" processes see $\sim$30% of instances "correctly" carried out, with only $\sim$50% not fulfilling their constraints. "Bug" processes are almost always correctly executed. "Design Definition" and "Hazard" processes are either correctly carried out from the beginning (the vast majority), or remain with unfulfilled constraints. Observing the incomplete process instances we encounter an expected large number of processes with unfulfilled constraints (i.e., hinting at steps with associated QA constraints that are not complete yet). However, we notice that only in a low percentage ($< 20\%$ for *IncompleteProgressedNotOk*) of instances have engineers started too early on subsequent steps without having fulfilled the previous steps' constraints.

Frequentis: we noticed two aspects. First, the amount of SubWPs ultimately Ok reaches almost 90%, with 10% SubWPs remaining with unfulfilled constraints. We manually investigated the violating artifacts (exclusively LLReqs) and the comments attached to the SubWP Jira issue. Given that Jira is used as the primary communication and coordination mechanism amongst the distributed teams and QA department, the comments provide an accurate and sufficiently complete track of the SubWPs history. For the 10 "CompleteNotOK" SubWPs we found that in two cases SubWPs were used for documentation purposes rather than development (thus no trace links to Functional Units were present). In one case test cases were not applicable, and in three cases multiple Functional Units were linked rather than one. This was due to the fact that the configuration subsystem affects multiple Functional Units. Three times a test case was referenced in the Jira comments (but no corresponding trace link in Jama was created). Once an additional SubWP was traced without closing the older one, and three times LLReq were marked for proposed future changes (and thus being no longer in state "released"). Note that some SubWPs experienced multiple, diverse violations. Second, we found that 11 SubWPs are "IncompleteOk" even we know that all the work was done. Manual investigation revealed that the Jira custom fields which are used by the passive process engine as a signal to advance

the process were not used by the engineers, hence the process remained in the first step. We further discuss implications of these findings in Section VI.

Constraint Fulfillment - Dronology: Table III reports the differences in how often a constraint was fulfilled (limited to completed process instances). A majority of constraints was fulfilled most of the time ($\sim$90% and higher). The lower fulfillment rates for constraints D-C1a and D-C1b ($<$55%) are the main reasons "Task" and "Sub-task" processes exhibit low *AlwaysOk* and *EventualOk* values in Table II. Yet, constraints applied across multiple process types (i.e., D-C1a/b, D-C2, D-C3, D-C6, D-C7) exhibit similar fulfillment rates.

Frequentis: For this case we generally observed high fulfillment rates for all constraints. The 12 unfulfilled constraint instances are distributed across the 10 "CompleteNotOk" SubWPs described above. Recall, a Dronology constraint typically requires the existence of a trace link to one artifact (e.g., D-C1a: a Task traces to a least one Design Definition). In contrast, for Frequentis a constraint requires that all linked artifacts (i.e., LLReqs in ATC-C1 to ATC-C6) fulfill specific conditions. For ATC-C5, for example, a single LLReq out of 10 that doesn't have a trace link to a Functional Unit will cause the entire constraint to fail (regardless of whether all other LLReqs are correct). We, therefore, also looked at the number of times an artifact (primarily an LLReq) was part of a constraint violation. With 1,121 LLReqs and six constraints involving an LLReq, there are potentially 6,726 opportunities that cause an overall constraint to fail. We observed 128, which is less than 2%. Out of 128 LLReqs that were part of a violation (due to missing, wrong, or too many trace links) only 3 of these cases were part of two different constraint violations. 98 LLReqs belonged to a single SubWP that was used for documentation (and needed no Functional Unit trace links), additional 12 LLReq belonged to single SubWP where Test cases were not applicable. The remaining 18 LLReqs violations were spread across the other eight "CompleteNotOk" SubWPs.

F. Performance

An engineer typically checks constraints at the level of Quality Check Documents to ensure all QA demands for their step are fulfilled (and not necessarily for each individual constraint). Overall the replay of 26,926 change events over 109 simultaneously active process instances resulted in 18,241 Quality Check Document evaluations. The resulting replay of events from the Tool connector's cache including constraint evaluation took $\sim$6.5 minutes (averaged over 10 evaluation runs). This corresponds to $\sim$0.02 seconds necessary for evaluating all quality constraints within a single Quality Check Document: a duration that allows frequent and timely feedback to developers.

VI. Discussion

The analysis of the Dronology data set indicated that the actual process (in some cases significantly) deviated from the planned process. Upon requesting feedback, a project lead at Dronology explained that while guidelines and a development process were in place, it was not always feasibly to follow them by the letter. Student teams were involved in the development of some of the components, and while they have been trained on the process, they still lacked experience in following all prescribed rules and guidelines. Furthermore, besides the software development aspect, the focus was also on obtaining a data set of trace links and that the process had to be adapted to the availability of open source developers. Rather than forcing a change of process which might be infeasible, the insights gained here could be used to decide where to better place the QA checks, e.g., making constraint check results available upon reviewing a pull request. Here ProCon would then highlight where traces are missing or are incorrectly set. A trace recommendation technique such as [25], [26] could further assist in establishing the trace itself.

In contrast, the analysis for Frequentis' SubWPs confirmed that engineers followed the high and stringent quality standards expected in the (highly safety-critical) ATC domain. The finding that 10% of process instances "EventualOK" confirms the QA engineer's experience that engineers need support for producing correct and complete trace links as significant additional work at a later stage was necessary. Our investigations into the "CompleteNotOk" instances highlighted that, on the one hand, corrections come with significant coordination effort and still may result in missing traces or incorrectly set artifact properties. On the other hand, the investigations highlighted the presence of edge cases where the QA constraints do not apply, reinforcing the need for sometimes *tolerating these inconsistencies*. ProCon offers two options in such a case: first to ignore the constraint evaluation results, and/or to adapt the process, respectively constraints. In either of these two cases, a rigid (thus inflexible), active process enforcement environment would have severely hampered the engineer's available actions, effectively forcing the engineer to work outside the defined process. Finally, the huge amount of $>$18.000 Quality Check Document evaluations explain why manually providing timely feedback is infeasible.

Using ProCon can have significant practical implications for QA engineers. Supported by automated checks for "standard" cases, they can shift attention and focus their time on edge cases and deviations from the process. Furthermore, they can allocate time for improving constraints checks, and investigating whether these checks and following the process actually result in better software quality [27]. Engineers can leverage the immediate feedback they receive on their work status and do not need to revisit their work at a later, inconvenient time. The various stakeholders no longer need to build (error prone) custom "helper tools" that are hard to maintain or to reuse.

We received very positive responses from engineers at Frequentis upon presenting ProCon with one team lead wishing to have it ready as a product by tomorrow, and a QA engineer joking to be out of work then. QA engineers at Frequentis used the prototype for writing new QA constraints during a company internal innovation event to showcase its potential and adaptability. While the prototype was applied only to one

	Task	%	Sub-task	%	Bug	%	Design Def.	%	Hazard	%	SubWP	%
AlwaysOk	0	0.0	55	31.2	94	98.9	134	82.7	26	72.2	78	79.6
EventualOk	42	31.1	31	17.6	0	0.0	2	1.2	0	0.0	10	10.2
CompleteNotOk	93	68.9	90	51.1	1	1.1	26	16.0	10	27.8	10	10.2
IncompleteNotYetOk	53	82.8	22	62.9	10	71.4	41	48.2	0	0.0	0	0.0
IncompleteProgressedNotOk	11	17.2	5	14.3	0	0.0	5	5.9	0	0.0	0	0.0
IncompleteOk	0	0.0	8	22.9	4	28.6	39	45.9	0	0.0	11	100.0
Total	199		211		109		247		36		109	

TABLE II: Dronology quality constraint evaluation results per process type

	Task (n=135)			Sub-task (n=176)			Bug (n=95)			Design Def. (n=162)			Hazard (n=36)				SubWP (n=98)		
	✓	χ	✓%	✓	χ	✓%	✓	χ	✓%	✓	χ	✓%	✓	χ	✓%		✓	χ	✓%
D-C1a	44	91	32.6													ATC-C1	95	3	96.9
D-C1b				94	82	53.4										ATC-C2	98	0	100.0
D-C2	135	0	100.0	168	8	95.5										ATC-C3	98	0	100.0
D-C3	132	3	97.8	164	12	93.2	94	1	98.9							ATC-C4	97	1	98.9
D-C4										136	26	84.0				ATC-C5	94	4	95.9
D-C5													26	10	72.2	ATC-C6	94	4	95.9
D-C6	133	2	98.5	176	0	100.0										ATC-C7	98	0	100.0
D-C7	121	14	89.6	175	1	99.4	95	0	100.0							ATC-C8	98	0	100.0

TABLE III: Quality constraint evaluation results per constraint type from completed process instances.

product group at Frequentis, we are currently rolling out the prototype to three more product groups, each having different rules (but use Jira/Jama), thus only the process and rules needs to be adapted. Given the excellent performance during replay (i.e., handling 27k artifact changes across 109 process instances within a few minutes) we are confident that adding more rules in the current rollout will not lead to performance problems. We subsequently expect to obtain more detailed insights into the prototype's practical use.

Aside from immediate practical implications, ProCon has huge potential as the foundation for additional support tools building on top. Passive process execution has the benefit of enabling inspection at any time to what degree the process is followed and where deviations have occurred (respectively are not mitigated yet). Deviations can thus be detected earlier, e.g., an engineer has started too early on a step. Alerts or mitigating actions may then be less invasive rather than significant rework later on. Other potential support mechanisms could guild the engineer in how to setup the correct output artifacts, or direct the engineer in how to fix a constraint violation or offers to automatically fix it [28].

VII. Threats to Validity

Internal Validity. We address researcher bias by modeling process and constraints from an open source system and a company rather than conducting controlled experiments. ProCon works on arbitrary artifacts, traces, and change events and was not specifically tailored to Jama or Jira.

External Validity. Based on the limited scope of our evaluation with two different systems, we can not claim generalizability of our findings. However, we argue in line with Briand et al. [29] that context-driven research will yield more realistic results. Our work evaluated the ability of ProCon to passively execute diverse engineering processes and QA constraints (simple ones from an open source system and more complex ones from industry) in a timely manner. We analyzed data from these two sources with one being "production data" from an industrial safety-critical system. Typically, being able to obtain such data, and furthermore being able to publicly report results is quite challenging as companies are reluctant to provide insights into their working processes at that level of detail, and open sources systems rarely come with such extensive explicit artifacts and trace information.

A. Limitations

The evaluation process is exemplary of the processes at Frequentis, but doesn't cover all of ED109. The model and engine however are not specific to ED109 and can be adopted to the specific process setting as shown with Dronology that followed a completely different process and TIM.

Adopting a different scenario then is mostly a matter of connecting different tools. Tools tend to come with a HTTP/REST interface, or client implementation (as did Jira and Jama with dedicate Java clients). Hence, it requires little effort in wrapping these clients for integration with the engine. New tools (and artifacts) are then accessible in the rules.

We also make the assumption that step completion can be detected from tools. The need for management, teamleads, and project leaders to obtain an accurate picture of progress, as well as having teams increasingly work distributed across multiple locations leads to a move away from informal signalling of completion toward explicit one, e.g., assigning a different member to an issue, setting a checkbox, setting the status of an issue, etc. Thus we believe that obtaining such indicators in almost all cases is reasonable.

VIII. Related Work

Several researchers have proposed techniques for continuously assessing and maintaining software traceability [4]. EBT (Event Based Traceability) uses a publish-subscribe model to notify developers when trace links need to be updated [30]

while Rempel et al., proposed an automated traceability assessment approach for continuously assessing the compliance of traceability to regulations in certified products [31], [32]. These approaches are orthogonal to our work as they are process-unaware, and hence provide little guidance for which step in a process a trace link must be available. Further, we assume engineers have chosen a suitable traceability strategy [33] and assessed that the resulting TIM (supported by flexible traceability management tools such as Capra [34]) also conforms to the relevant guidelines [2].

Process-centric software development environments (PCSDE) have received significant attention in the 90's. We discuss an exemplary selection below, for a detailed review see [35] or [36]. Step-centric modeling and active execution frameworks such as Process Weaver [10], SPADE [11], Serendipity [37], EvE [38], or PRIME [39] determine which steps may be done at any given moment, automatically executing them where possible. While such research supports detailed guidance, deviations from the prescribed process is not well supported. Approaches such as Shamus [40], PROSYT [41], or Merlin [42] specify for engineering artifacts which actions and conditions are available, and enforce their correct order – yet, without prescribing an overall step-based engineering process. Often the supported artifacts are limited to files and folders. Systems such as MARVEL [43], OIKOS [44], or EPOS [45] utilize ECA rules or pre- and post-conditions, thereby providing significant freedom of action to the engineer but offer limited guidance.

The approaches described so far have the implicit assumption that engineers primarily interact with the PCSDE for executing work. Our aim is to remain in the background, with engineers staying in their tools except for confirming QA constraint fulfillment. Provenence [46] has a similar goal, maintaining a process view from artifact change events. It's, however, limited to events from the file system, relying on moving files to dedicated folders to signal process-meaningful events. It also remains unaware of trace links between artifacts.

More recent work focuses on specific aspects in the engineering life-cycle rather than general purpose processes. DevOpsML [47] aims at reducing the effort to describe continuous integration and deployment processes. Amalfitano et al. [48] aim to fully automate the execution of the testing process and to automatically generate appropriate traceability links. Similarly, Hebig et al. [49] investigate how various software design and code artifacts dependencies emerge from MDE activities. When involving human steps, approaches often assume pre-defined process models and rigorous tool integration. Kedji et al. provide a collaboration-centric development process model and corresponding DSL [50]. At a micro-level, Zhao et al. propose Little-JIL for describing fine-grained steps involved in refactoring [51] and to help developers track artifact dependencies during rework [52].

A few approaches on general purpose process modeling and execution (e.g., [53]–[56]) focus on step-centric languages such as SPEM and BPMN, which both imply active execution where engineers cannot deviate from the prescribed process.

Business process compliance checking approaches determine whether complex sequences of events and/or their timing violate particular constraints. Ly et al. analyse frameworks for compliance monitoring [57] and highlight that the investigated frameworks have little or no inherent support for referencing data beyond the properties available in the respective events (hence no access to the actual artifact details and their traces/relations to other artifacts). They also show that hardly any approach supports proactive violation detection, the ability to continue monitoring after a violation, or root cause analysis in a manner useful for software engineering. Also very recent work, such as [58] or [59], lack this crucial support for defining constraints on artifact details. In general, processes in the software engineering domain are comparatively simpler but instead require a focus on keeping artifacts consistent with each other, hence data-centric constraints are required which our approach can check proactively and highlight that they are not yet fulfilled.

QA constraint checking exhibits some similarities to cross-artifact consistency checking. Examples include work on model-to-model [7], [60], [61] or model-to-code checking [6]. These approaches support the correct propagation of changes across artifacts once these artifacts are known to "belong" together. Our work, in contrast, supports the engineer in what state an artifact needs to be, and which trace links it needs to exhibit depending on the process progress.

IX. Conclusions and Outlook

We presented an approach for reducing the effort of ensuring that development activities adhere to quality constraints. The novel aspects are the decoupling of QA constraints from process control and dataflow. We thereby tolerate engineers to deviated from the process while informing them which constraints are yet unfulfilled and which steps are complete. Future work focuses on two main aspects. First we intend to study the effect of having our prototype in use by engineers at Frequentis. We aim to quantify the actual effort reduction and gather qualitative feedback for further improvements. Second, we will study QA engineers and process engineers during the creation, evolution, and maintenance of process models (including constraints) with ProCon to understand how their task can be supported even better.

Acknowledgment

This work was funded by the Austrian Science Fund (FWF) under the grant numbers P31989 and P29415-NBL, and by the state of Upper Austria LIT-2019-8-SEE-118. The Dronology case-study was supported by the United States National Science Foundation under grants SHF:1741781 and CPS:1931962.

References

[1] D. B. Kramer, Y. T. Tan, C. Sato, and A. S. Kesselheim, "Ensuring medical device effectiveness and safety: a cross–national comparison of approaches to regulation." *Food and drug law journal*, vol. 69 1, pp. 1–23, i, 2014.

[2] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang, "Mind the gap: assessing the conformance of software traceability to relevant guidelines," in *Proc. of the 36th Int'l Conf. on Software Engineering, ICSE '14*. ACM, 2014, pp. 943–954.

[3] P. Diebold and S. A. Scherr, "Software process models vs descriptions: What do practitioners use and need?" *Journal of Software: Evolution and Process*, vol. 29, no. 11, 2017.

[4] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Proc. of the on Future of Software Engineering, FOSE 2014*. ACM, 2014, pp. 55–69.

[5] R. Watkins and M. Neal, "Why and how of requirements tracing," *IEEE Software*, vol. 11, no. 4, pp. 104–106, 1994.

[6] A. Egyed, K. Zeman, P. Hehenberger, and A. Demuth, "Maintaining consistency across engineering artifacts," *IEEE Computer*, vol. 51, no. 2, pp. 28–35, 2018.

[7] H. Klare, "Multi-model consistency preservation," in *Proc. of the 21st ACM/IEEE Int'l Conf. on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '18. ACM, 2018, p. 156–161.

[8] R. Balzer, "Tolerating inconsistency," in *Proceedings of the 13th international conference on Software engineering*, 1991, pp. 158–165.

[9] EUROCAE, "ED 109 software integrity assurance considerations for communication, navigation, surveillance and air traffic management (cns/atm) systems," January 2012. [Online]. Available: https://standards.globalspec.com/std/1517993/eurocae-ed-109

[10] C. Fernstrom, "Process weaver: Adding process support to unix," in *Proc. of the 2nd Int'l Conf. on the Software Process-Continuous Software Process Improvement*. IEEE, 1993, pp. 12–26.

[11] S. Bandinelli, E. D. Nitto, and A. Fuggetta, "Supporting cooperation in the SPADE-1 environment," *IEEE Trans. Software Eng.*, vol. 22, no. 12, pp. 841–865, 1996.

[12] K. Pohl, K. Weidenhaupt, R. Dömges, P. Haumer, M. Jarke, and R. Klamma, "PRIME - toward process-integrated modeling environments: 1," *ACM Trans. Softw. Eng. Methodol.*, vol. 8, no. 4, pp. 343–410, 1999. [Online]. Available: https://doi.org/10.1145/322993.322995

[13] I. Alloui and F. Oquendo, "Managing consistency in cooperating software processes," in *Software Process Technology*, V. Gruhn, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 92–99.

[14] J. Simmonds, D. Perovich, M. C. Bastarrica, and L. Silvestre, "A megamodel for software process line modeling and evolution," in *Proc. of the 2015 ACM/IEEE 18th Int'l Conf. on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 406–415.

[15] "SPEM," 2008. [Online]. Available: https://www.omg.org/spec/SPEM/About-SPEM/

[16] "Jira," https://www.atlassian.com/software/jira, accessed: 2020-08-20.

[17] "Jama," https://www.jamasoftware.com/, accessed: 2020-08-20.

[18] L. Osterweil, "Software processes are software too," in *Engineering of Software*. Springer, 2011, pp. 323–344.

[19] C. Mayr-Dorn, M. Vierhauser, F. Keplinger, S. Bichler, and A. Egyed, "Timetracer: a tool for back in time traceability replaying," in *ICSE '20: 42nd International Conference on Software Engineering, Companion Volume, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 33–36. [Online]. Available: https://doi.org/10.1145/3377812.3382141

[20] J. Cleland-Huang, M. Vierhauser, and S. Bayley, "Dronology: An incubator for cyber-physical systems research," in *Proc. of the 40th Int'l Conf. on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '18. ACM, 2018, p. 109–112.

[21] C. Mayr-Dorn, M. Vierhauser, S. Bichler, F. Keplinger, J. Cleland-Huang, A. Egyed, and T. Mehofer, "Supporting Online Material," Aug 2020. [Online]. Available: https://figshare.com/s/1544630863a64d94d994

[22] C. Mayr-Dorn, S. Bichler, F. Keplinger, and A. Egyed, "Guiding engineers with the passive process engine environment," in *43rd International Conference on Software Engineering: Companion (ICSE-Companion)*. IEEE / ACM, 2021, p. to appear.

[23] "Drools," https://www.drools.org, accessed: 2020-08-20.

[24] J. Cleland-Huang and M. Vierhauser, "Dronology Public Datasets." [Online]. Available: https://dronology.info/research/datasets

[25] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *Proc. of the 40th Int'l Conf. on Software Engineering*, 2018, pp. 834–845.

[26] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[27] R. Conradi, C. Fernström, and A. Fuggetta, "Concepts for evolving software processes," *Software Process Modelling and Technology*, pp. 9–31, 1994.

[28] C. Mayr-Dorn, R. Kretschmer, A. Egyed, R. Heradio, and D. Fernández-Amorós, "Inconsistency-tolerating guidance for software engineering processes," in *43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE / ACM, 2021, p. to appear.

[29] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh, "The Case for Context-Driven Software Engineering Research: Generalizability Is Overrated," *IEEE Software*, vol. 34, no. 5, pp. 72–75, 2017.

[30] J. Cleland-Huang, C. K. Chang, and M. J. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Trans. Software Eng.*, vol. 29, no. 9, pp. 796–810, 2003. [Online]. Available: https://doi.org/10.1109/TSE.2003.1232285

[31] P. Rempel and P. Mäder, "Continuous assessment of software traceability," in *Proc. of the 38th Int'l Conf. on Software Engineering, ICSE 2016,*, L. K. Dillon, W. Visser, and L. Williams, Eds. ACM, 2016, pp. 747–748. [Online]. Available: https://doi.org/10.1145/2889160.2892657

[32] P. Rempel, "Continuous assessment of software traceability," Ph.D. dissertation, Technische Universität Ilmenau, Germany, 2016. [Online]. Available: https://www.db-thueringen.de/receive/dbt_mods_00029275

[33] P. Rempel, P. Mäder, and T. Kuschke, "An empirical study on project-specific traceability strategies," in *Proc. of the 21st IEEE Int'l Requirements Engineering Conf., RE 2013*. IEEE Computer Society, 2013, pp. 195–204. [Online]. Available: https://doi.org/10.1109/RE.2013.6636719

[34] S. Maro and J. Steghöfer, "Capra: A configurable and extendable traceability management tool," in *Proc. of the 24th IEEE Int'l Requirements Engineering Conf., RE 2016*. IEEE, 2016, pp. 407–408.

[35] P. Barthelmess, "Collaboration and coordination in process-centered software development environments: a review of the literature," *Inf. Softw. Technol.*, vol. 45, no. 13, pp. 911–928, 2003. [Online]. Available: https://doi.org/10.1016/S0950-5849(03)00091-0

[36] V. Gruhn, "Process-centered software engineering environments, a brief history and future challenges," *Annals of Software Engineering*, vol. 14, no. 1-4, pp. 363–382, 2002.

[37] J. C. Grundy and J. G. Hosking, "Serendipity: Integrated environment support for process modelling, enactment and work coordination," *Autom. Softw. Eng.*, vol. 5, no. 1, pp. 27–60, 1998. [Online]. Available: https://doi.org/10.1023/A:1008606308460

[38] A. Geppert, D. Tombros, and K. R. Dittrich, "Defining the semantics of reactive components in event-driven workflow execution with event histories," *Inf. Syst.*, vol. 23, no. 3–4, p. 235–252, May 1998. [Online]. Available: https://doi.org/10.1016/S0306-4379(98)00011-8

[39] K. Pohl, K. Weidenhaupt, R. Dömges, P. Haumer, M. Jarke, and R. Klamma, "Prime—toward process-integrated modeling environments: 1," *ACM Trans. Softw. Eng. Methodol.*, vol. 8, no. 4, p. 343–410, Oct. 1999.

[40] A. LaMarca, W. K. Edwards, P. Dourish, J. Lamping, I. Smith, and J. Thornton, "Taking the work out of workflow: mechanisms for document-centered collaboration," in *ECSCW'99*. Springer, 1999, pp. 1–20.

[41] G. Cugola and C. Ghezzi, "Design and implementation of prosyt: a distributed process support system," in *Proc. of the IEEE 8th Int'l Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99)*. IEEE, 1999, pp. 32–39.

[42] G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf, *MERLIN: Supporting Cooperation in Software Development through a Knowledge-Based Environment*. GBR: Research Studies Press Ltd., 1994, p. 103–129.

[43] N. S. Barghouti, "Supporting cooperation in the marvel process-centered sde," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 5, pp. 21–31, 1992.

[44] C. Montangero and V. Ambriola, "Oikos: constructing process-centred sdes," in *Software Process Modelling and Technology*, 1994, pp. 131–151.

[45] R. Conradi, M. Hagaseth, J.-O. Larsen, M. Nguyen, B. Munch, P. Westby, W. Zhu, M. Jaccheri, and C. Liu, "Object-oriented and cooperative process modelling in epos," *Software process modelling and technology*, pp. 9–32, 1994.

[46] B. Krishnamurthy and N. S. Barghouti, "Provence: A process visualization and enactment environment," in *Proc. of the European Software Engineering Con*. Springer, 1993, pp. 451–465.

[47] A. Colantoni, L. Berardinelli, and M. Wimmer, "DevopsML: Towards modeling devops processes and platforms," in *Proc. of the 23rd Int'l Conf. on Model Driven Engineering Languages and Systems*, ser. Models'20. ACM, 2020, pp. 1–11.

[48] D. Amalfitano, V. D. Simone, A. R. Fasolino, and S. Scala, "Improving traceability management through tool integration: an experience in the automotive domain," in *Proc. of the 2017 Int'l Conf. on Software and System Process, ICSSP 2017*, R. Bendraou, D. Raffo, L. Huang, and F. M. Maggi, Eds. ACM, 2017, pp. 5–14.

[49] R. Hebig, A. Seibel, and H. Giese, "Toward a comparable characterization for software development activities in context of MDE," in *Proc. of the Int'l Conf. on Software and Systems Process, ICSSP*, D. Raffo, D. Pfahl, and L. Zhang, Eds. ACM, 2011, pp. 33–42.

[50] K. A. Kedji, R. Lbath, B. Coulette, M. Nassar, L. Baresse, and F. Racaru, "Supporting collaborative development using process models: An integration-focused approach," in *Proc. of the 2012 Int'l Conf. on Software and System Process, ICSSP 2012*, D. R. Jeffery, D. Raffo, O. Armbrust, and L. Huang, Eds. IEEE, 2012, pp. 120–129.

[51] X. Zhao and L. J. Osterweil, "An approach to modeling and supporting the rework process in refactoring," in *2012 International Conference on Software and System Process, ICSSP 2012, Zurich, Switzerland, June 2-3, 2012*, D. R. Jeffery, D. Raffo, O. Armbrust, and L. Huang, Eds. IEEE Computer Society, 2012, pp. 110–119. [Online]. Available: https://doi.org/10.1109/ICSSP.2012.6225953

[52] X. Zhao, Y. Brun, and L. J. Osterweil, "Supporting process undo and redo in software engineering decision making," in *Proc. of the Int'l Conf. on Software and System Process, ICSSP '13*, J. Münch, J. A. Lan, and H. Zhang, Eds. ACM, 2013, pp. 56–60.

[53] M. Dumas and D. Pfahl, *Modeling Software Processes Using BPMN: When and When Not?* Cham: Springer International Publishing, 2016, pp. 165–183. [Online]. Available: https://doi.org/10.1007/978-3-319-31545-4_9

[54] R. Ellner, S. Al-Hilank, J. Drexler, M. Jung, D. Kips, and M. Philippsen, "espem – a spem extension for enactable behavior modeling," in *Modelling Foundations and Applications*, T. Kühne, B. Selic, M.-P. Gervais, and F. Terrier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 116–131.

[55] S. Alajrami, B. Gallina, I. Sljivo, A. Romanovsky, and P. Isberg, "Towards cloud-based enactment of safety-related processes," in *Computer Safety, Reliability, and Security*, A. Skavhaug, J. Guiochet, and F. Bitsch, Eds. Cham: Springer International Publishing, 2016, pp. 309–321.

[56] D. Winkler, L. Kathrein, K. Meixner, P. Staufer, M. Pauditz, and S. Biffl, "Towards a hybrid process model approach in production systems engineering," in *Systems, Software and Services Process Improvement*, A. Walker, R. V. O'Connor, and R. Messnarz, Eds. Cham: Springer International Publishing, 2019, pp. 339–354.

[57] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. van der Aalst, "Compliance monitoring in business processes: Functionalities, application, and tool-support," *Information Systems*, vol. 54, pp. 209 – 234, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437915000459

[58] C. Cabanillas, M. Resinas, and A. Ruiz-Cort's, "A mashup-based framework for business process compliance checking," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[59] D. Knuplesch, M. Reichert, and A. Kumar, "A framework for visually monitoring business process compliance," *Information Systems*, vol. 64, pp. 381 – 409, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437915301770

[60] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *IEEE Trans. Software Eng.*, vol. 37, no. 2, pp. 188–204, 2011. [Online]. Available: https://doi.org/10.1109/TSE.2010.38

[61] H. König and Z. Diskin, "Advanced local checking of global consistency in heterogeneous multimodeling," in *European Conference on Modelling Foundations and Applications*. Springer, 2016, pp. 19–35.