ProCon: an Automated Process-Centric Quality Constraints Checking Framework

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*—When dealing with safety-critical systems, various regulations, standards, and guidelines stipulate stringent requirements for certification and traceability of artifacts, but typically lack details with regards to the corresponding software engineering process. Given the industrial practice of only using semi-formal notations for describing engineering processes – with the lack of proper tool mapping – engineers and developers need to invest a significant amount of time and effort to ensure that all steps mandated by quality assurance are followed. The sheer size and complexity of systems and regulations make manual, timely feedback from Quality Assurance (QA) engineers infeasible. In order to address these issues, in this paper, we propose a novel framework for tracking, and "passively" executing 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 three case studies: a safety-critical open-source community system, a safety-critical system in the air-traffic control domain, and a non-safety-critical, web-based system. Results from our analysis confirm that trace links are often corrected or completed after the work step has been considered finished, and the engineer has already moved on to another step. Thus, support for timely and automated constraint checking has significant potential to reduce rework as the engineer receives continuous feedback already during their work step.**

***Index Terms*—software engineering process, traceability, developer support, quality assurance, process deviation, constraint checking**

I. Introduction

Software quality assurance (QA) focuses on ensuring and attesting that the implemented engineering processes result in appropriate quality of the software. This not only includes code quality, but also pertains to the quality of the procedures, documentation, and available artifacts [1]. To this end, various regulations, standards, and guidelines stipulate stringent traceability paths [2], [3] without prescribing a corresponding, detailed software engineering process. Examples in safety-critical systems include the FDA principles in the medical domain [4], [5], DO-178C/ED-12C for airborne systems [6], ED-109A [7] for air traffic management systems, and Automotive SPICE [8] in the automotive industry. To achieve compliance, QA engineers need to inspect fine-grained constraints related to properties of engineering artifacts, such as requirements, models, code, and test cases, 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 to employ semi-formal descriptions to define processes [9]. As a result, there exists a crucial gap between the process model and the tool environment in which engineers (implicitly) enact the process. As a result, currently little to no automation support is available for engineers to understand whether they correctly follow a process, or to what extent they (temporarily) deviate from it.

In this work, we focus on the problems that *Developers* and *Quality Assurance Engineers* face when dealing with these processes, as adhering to, and evaluating QA constraints is complex and can quickly become overwhelming. Typically, developers work on multiple projects, sometimes simultaneously, with each project potentially adhering to different quality standards or guidelines.

We conducted an informal study with our industry partner Frequentis that applies the V-Model to develop (among other products) safety-critical air-traffic control software. Developers reported being stressed about potentially missing important steps mandated by quality assurance. A sub-problem, the challenge to correctly provide traces between engineering artifacts, is also commonly found in the automotive industry [10]. 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. We observed that when quality checks are performed in batch for efficiency reasons towards the end of the development cycle, developers may only receive feedback as late as 6-12 weeks after completing their work [11]. Remediating problems late in the process interrupts developers who may have already moved on to other steps or projects, causing disruptions and extra effort as they need to re-understand their past work context.

In this paper, we extend our prior work that was published at the International Conference on Software Engineering (ICSE'21) aimed at reducing the effort required in ensuring that development activities adhere to the intended process. Our approach provides automated support to developers and quality assurance engineers [12], and relies on passive process execution, i.e., by tracking process steps via monitoring engineering artifacts such as requirements, design documents, issues, change requests, and tests rather than engineers having to explicitly interact with a process engine. This is complemented by continuous evaluation of the specified 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 are used to monitor and determine process progress. This contrasts with existing work on traceability [3], [13] 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 [14]. Similarly, work on constraint checking [15], [16] primarily focuses on consistency among diverse artifacts without addressing consistency issues between these artifacts and the underlying process. Compared to past work on process-centric environments (PCEs) such as PRIME [17], our approach remains lightweight and requires minimal integration efforts - such human effort is one of the reasons past approaches were not readily adopted.

The key contributions of our work are as follows:

- A process model that decouples process control and data flow from QA constraints.
- A passive process engine that explicitly tolerates inconsistencies [18] 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), an industrial air-traffic control system (ATC), and a web-based recreational activity system that measures the extent to which the prescribed process was followed.

The prototype, process models, constraints, and historical development data used in this paper are available on Figshare[1]

The additional content, extending our original work, includes the following three main extensions: (1) an extended discussion on the background, as well as related work in the area, (2) an enhanced process model, and (3) significantly expanded evaluation including another industry evaluation case study and a preliminary usability study.

The remainder of this paper is structured as follows. Section II discusses the background of our work and motivates it by describing constraint checks for the development of safety-critical systems. Sections IV and V provide an overview of our approach and introduce details on modeling the process and constraints, and Section VI further focuses on constraint execution. We describe our evaluation method in VII, supported by three distinct case studies, and introduce our prototype implementation in Section VIII and evaluation setup in Section IX. We report details of the evaluation in Section X, and conclude with a discussion of results (Section XI), threats to validity (Section XII), and related work (Section III).

II. Background & Motivation

Our work combines the areas of process management, traceability, and safety-related regulations. In this section, we provide a brief introduction to each of these areas and describe the context of our process engine using a motivating example from our industry collaborator.

Safety-critical software typically is subject to stringent regulations, where certain artifacts as well as trace links between the specification, individual requirements, test cases, and source code need to be thoroughly documented and provided during the certification process. For example, medical devices are subject to diverse international regulations, and software developed for the aerospace industry must comply with ISO/IEC12207 and DO-178C guidelines.

In the domain of Non-Airborne Systems, the DO-278/ED-109 standard [19] defines requirements for traceability of artifacts, according to different design assurance levels. Creating, maintaining, and validating these links before certification is a costly and labor-intensive task that is typically performed manually, with little to no tool support. However, while DO-278/ED-109 describes the different types of assurance levels, ranging from "Catastrophic" to "No Effect", and the types of trace links that need to be provided, it does not specify *when* these trace links need to be established or which role (i.e., *who*) should perform this task.

A. Motivating Scenario

In this domain, of Non-Airborne Systems, our long-term industry collaborator Frequentis is a world-leading provider of voice communication solutions for air-traffic control and command-control centers. Their product portfolio in the air traffic domain ranges from aeronautical information management solutions, over remote digital towers, to traditional towers, with reliable voice communication playing a major role. Monitoring radio channels to obtain situational awareness, communicating with air traffic participants, and coordinating with other stakeholders in emergency situations introduce diverse use cases that all require real-time audio processing software and hardware, as well as the user interface, to work exactly as expected.

As part of their requirements engineering and development process, Frequentis refines high-level requirements and assigns them to different work packages for engineers to work on. In

[1] https://doi.org/10.6084/m9.figshare.12840053

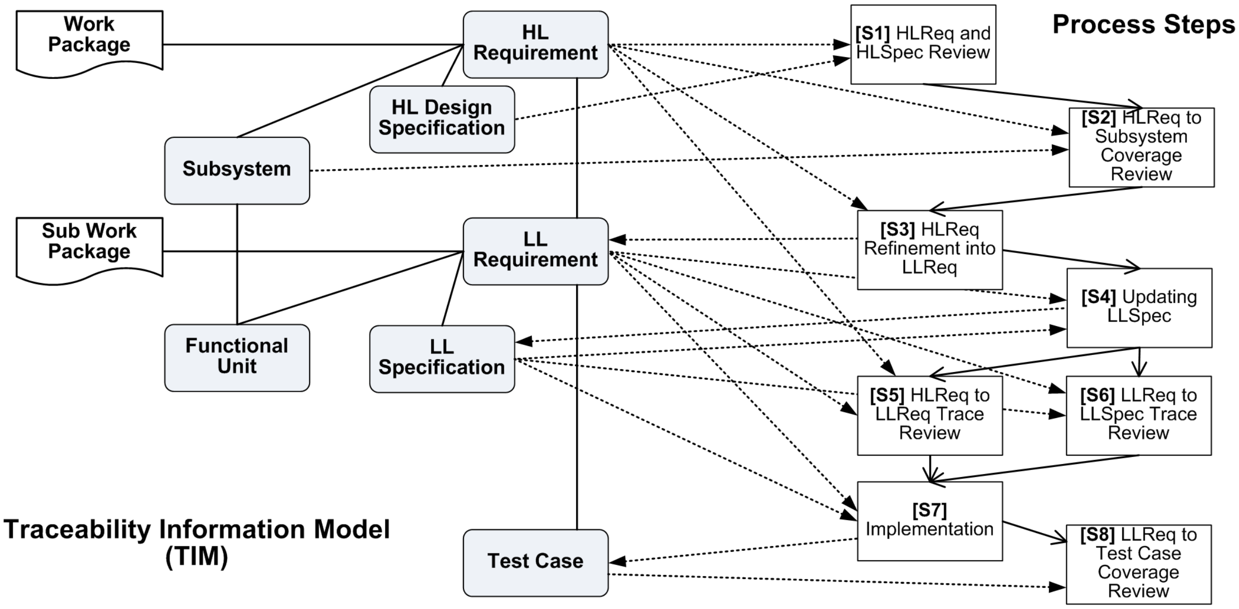

Work Package
HL Requirement
HL Design Specification
Subsystem
Sub Work Package
LL Requirement
Functional Unit
LL Specification
Test Case
Traceability Information Model (TIM)
Process Steps
[S1] HLReq and HLSpec Review
[S2] HLReq to Subsystem Coverage Review
[S3] HLReq Refinement into LLReq
[S4] Updating LLSpec
[S5] HLReq to LLReq Trace Review
[S6] LLReq to LLSpec Trace Review
[S7] Implementation
[S8] LLReq to Test Case Coverage Review

Fig. 1: Frequentis' simplified traceability information model (TIM) excerpt (left) and process model excerpt (right) labeled with Steps S1 to S8. Full lines depict explicit, bi-directional traces between artifacts, and dotted lines depict exemplary artifact read and write usage in the various process steps. Colors map to elements in the DO-278/ED-109 standard.

order to meet the prescribed regulations, trace links need to be established between requirements, work packages, and the respective test cases. The left-hand side of Figure 1 provides an overview of the partial traceability information model (TIM) used. The right-hand side provides excerpts from a simplified development process representation consisting of a sample set of process steps (S1 to S8) for one of their products. We use the described TIM, process, and constraints as a running example to describe our passive process engine framework.

In the example, high-level requirements (HLReq) and the corresponding high-level design specifications (HLSpec) are reviewed and subsequently refined into low-level requirements (LLReq). This, in turn, may require low-level design specifications (LLSpec) to be updated. Therefore, it is advisable for developers to wait for the outcome of the review before they start refining LLReq, even though HLReqs and teams are already assigned to work packages during the development iteration planning phase. Furthermore, before the implementation of LLReq can start, the corresponding trace links from HLReq to LLReq and trace links from LLReq to LLSpec need to be reviewed. In order to ensure adherence to the prescribed standards, constraints for the engineering process are introduced which rely on the properties of an artifact at a specific process progress. These not only check for the existence of an artifact or respective trace links, but further ensure that the correct links are set up between the right type of artifacts. For example, in the case of Frequentis' development process, at the end of step S3 (cf. Figure 1) the respective LLReqs need to be in state "Complete", and contain a trace link to a HLReq. Furthermore, upon completion of step S7, when a LLReq's verification method is "Demonstration", at least one trace link to a respective Test Case must be established. This is further constrained by the fact that this test case must be a simulation, demonstration, acceptance test, or any other form of test, except the type "Software Test Case".

Engineers who are updating the LLSpec (in S4), for example, thus need to be aware of when they can proceed with their step in order 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 prematurely on HLReq refinement (S3) may be too risky if the requirements have not gone through review in S1 and S2 but perhaps necessary and expedient when S2 has only a few requirements left to review.

These constraints are not meant to replace human-in-the-loop QA measures, such as the trace reviews in S5 and S6, but to complement manual activities. Such (continuous) automated checks reduce the effort for reviews by ensuring the traces under review are syntactically correct and thus the review can be performed more efficiently.

B. Process definition vs. Process execution

As the scenario above exemplifies, in most software engineering environments, one cannot expect engineers to precisely follow the prescribed process definition. Various factors such as time pressure, unclear or missing information, or changing customer expectations, cause rework and make iterations necessary. Modeling all such possible "deviations" from an ideal process is often impractical and hence not done. The process, however, has an important guiding purpose, while additional QA constraints define properties of the engineering artifacts and their relations. The key aspect here is that strictly following the process is as detrimental to software quality as largely ignoring it. Hence, we make the case for having less detailed, but nevertheless, informative processes that are mapped to tools (in which they are executed), while tolerating deviations. This, however, should be no motivation to prescribe a waterfall model.[2] Tracking process progress is crucial for engineers to assess whether an action leads to a deviation, or to assess who might be affected by a deviation. Tracking progress in the presence of deviations is crucial to understanding whether a deviation is critical, acceptable, and repairable. The next section discusses related work and its shortcomings with regard to supporting deviations.

III. Related Work

A. Process-centric software development environments

Process-centric software development environments (PCSDE) have received significant attention in the '90s. We discuss an exemplary selection below, for a detailed review see [21] or [22]. Step-centric modeling and active execution frameworks such as Process Weaver [23], SPADE [24], Serendipity [25], EvE [26], or PRIME [17] 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 [27], PROSYT [28], or Merlin [29] 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 [30], OIKOS [31], or EPOS [32] utilize Event Condition Action (ECA) rules or pre- and post-conditions, thereby providing significant freedom of action to the engineer but offering 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. Provence [33] 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 of the engineering life-cycle rather than general-purpose processes. DevOpsML [34] aims at reducing the effort to describe continuous integration and deployment processes. Amalfitano et al. [35] aim to fully automate the execution of the testing process and to automatically generate appropriate traceability links. Similarly, Hebig et al. [36] 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 [37]. At a micro-level, Zhao et al. propose Little-JIL for describing fine-grained steps involved in refactoring [38] and to help developers track artifact dependencies during rework [39].

A few approaches on general-purpose process modeling and execution (e.g., [40]–[43]) focus on step-centric languages such as SPEM and BPMN, which both imply active execution where engineers cannot deviate from the prescribed process.

B. Business Process Modeling

In the business process modeling domain, significant related work focuses on formally verifying processes [44] rather than attempting to fix them. Fixing is limited to achieving sound process models but is not applicable to instances as we aim for. The few, recent approaches that address inconsistencies and their repair exhibit limited expressiveness for specifying constraints: LTL for expressing task and event constraints [45], [46] or Mixed-Integer Programming for determining runtime compliance of task and resource allocation [47]. Business process compliance checking approaches determine whether complex sequences of events and/or their timing violate particular constraints. Ly et al. analyze frameworks for compliance monitoring [48] 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, recent work such as [49] or [50], lacks this crucial support for defining constraints on artifact details.

Notably, the processes studied and used for evaluation in the business process management or information systems domain exhibit complex decision-making about which task to do next, or which task must not be done, and how much time between tasks may pass. Evaluation domains thus often include administrative processes, medical processes, or legal processes but virtually never software engineering processes. In the software engineering domain, processes are simpler but instead require a focus on keeping artifacts consistent with each other. Hence structural (i.e., data-centric) constraints are required which our approach checks proactively, subsequently highlighting that they are not yet fulfilled. Often the necessary guidance is not so much about which task to do next, but when to do it.

In comparison to dedicated software engineering process environments introduced above, general (business) process support, as provided by enterprise tools like SAP, is not applicable in software engineering environments as such support rigidly controls what steps may be worked on without any possibility of deviation.

[2] Even W. Royce, the original author of the waterfall model [20], considers the execution of a strict waterfall process as risky and likely to fail.

C. Traceability

Several researchers have proposed techniques for continuously assessing and maintaining software traceability [13]. Event-Based Traceability (EBT) uses a publish-subscribe model to notify developers when trace links need to be updated [51] while Rempel et al. proposed an automated traceability assessment approach for continuously assessing the compliance of traceability to regulations in certified products [52], [53]. These approaches are orthogonal to our work as they are process-unaware, and hence provide little to no guidance for which step in a process a trace link must be available. Furthermore, we assume engineers have chosen a suitable traceability strategy [54] and assessed that the resulting TIM (supported by flexible traceability management tools such as Capra [55]) also conforms to the relevant guidelines [3].

D. Model Consistency

QA constraint checking exhibits some similarities to cross-artifact consistency checking. Examples include work on model-to-model [16], [56], [57] or model-to-code checking [15]. 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.

E. Research Gap

The main research gap that we address in this paper is the lack of guidance in the presence of deviation from the intended process. Existing approaches either need to explicitly model the possibility to deviate or are too flexible (hence not providing sufficient guidance). Additionally, most approaches require engineers to interact with the process environment in order to track the process progress rather than a process environment passively observing the engineers' activities in their tools and inferring process progress in the background from these activities.

IV. Approach – The ProCon Framework

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

As part of ProCon, a passively executable *Process Specification* describes the sequence and alternatives of engineering activities (i.e., the "control flow") and the corresponding software engineering artifacts serving 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 to the 90s [23], [24], [58], [59] (c.f. previous Section). ProCon is different insofar as (i) the process is tracked in the background, based on the software engineers' activities performed in the tools they are using in their daily work, and as such, it does not require 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.

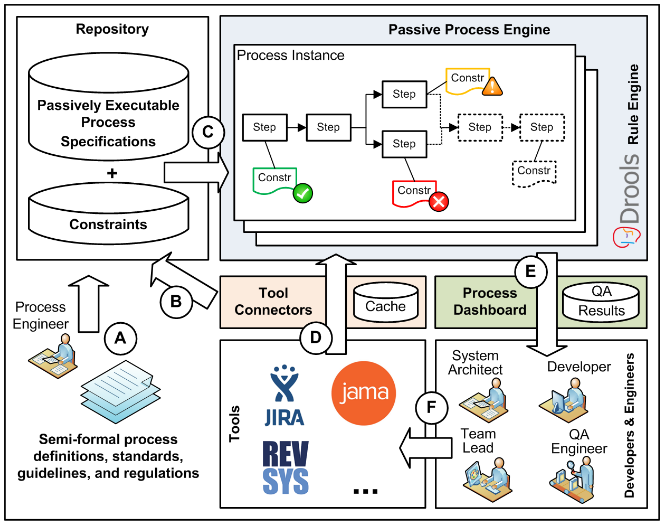


Fig. 2: The ProCon framework.

We refer to these abilities as "passive execution". Instead of simply monitoring the process, our framework determines available future steps, detects premature steps, and makes this information immediately available to engineers. ProCon, is capable of detecting and tracking deviations from a predefined process via a series of process and quality constraints. These constraints, and their respective evaluation results, are treated as first-class citizens in the software engineering process (model), and hence represent explicit software engineering artifacts in their own right. This in turn allows constraints to be explicitly evaluated as soon as actions are performed so that the evaluation results can provide valuable insights about the status of the process beyond whether a step has been completed or not.

A. ProCon High-level Architecture

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

Existing **Semi-informal process definitions, standards, guidelines, and regulations** (A) 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 include (parts of) the Software Process Engineering Metamodel (SPEM) [60], PDFs representing flowcharts, or simple text documents outlining the steps and responsibilities of the different roles.

The process definition documents 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 access to artifacts often in machine-readable formats such as JSON data. These connectors enable sophisticated tasks such as obtaining artifact updates via polling or subscriptions, and managing which artifacts are relevant for a process. It is, therefore, necessary to keep track of these artifacts, provide caching mechanisms for quick repeated access, and keep this cache up to date (cf. Section VIII).

Passively Executable Process Specifications and **Constraints** (C) are manually derived (cf. Section V-A and V-D, 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 the state of an artifact and respective events from tool connectors (E), and 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, in turn, utilizes to update the process progress and quality constraint evaluation results (cf. Sections VI and VI-A, respectively).

Finally, a web-based **Process Dashboard** makes process progress and QA constraint evaluation results (F) continuously available to software engineers. It enables control of the passive process instance: triggering new instances, observing their progress, and eventually archiving them (cf. Section VIII).

B. ProCon Usage

ProCon users are primarily QA engineers and developers, but may include other stakeholders, such as product managers and team leads. While the former can use the framework with a focus on quality assurance, which is the focus of this paper, the latter can leverage the framework to track and manage 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). Stakeholders comprise dedicated process engineers, QA engineers, project managers, and others, depending upon how the organization assigns responsibilities for (i) defining how the development team needs to work to adhere to regulations and (ii) monitoring and improving these practices. Responsibility for providing a tool such as ProCon would fall under the responsibility of an IT department similar to maintaining other infrastructure such as source code repositories and issue tracking systems.

The definition 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, or what trace links are available to navigate from a Jira [61] issue to a requirement managed in Jama [62]. The outcome of the modeling use case is a passively executable process specification and its quality constraints.

It is, however, important to note that our approach does not require modeling the complete process if tools do not facilitate 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 these ambiguous situations. In line with Lee Osterweil's key observation that "software processes are software, too" [63] we advocate the use of contemporary software engineering practices such as developing in iterations, testing, versioning, and issue tracking. A key element here is replaying the artifacts' history to test the constraints' ability to detect deviations [64].

Upon process instantiation, the passive process engine obtains artifacts and their changes as they occur in various tools (D). The engine then tracks the progress of each step and its 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, conduct the necessary artifact changes via the tools, trigger reevaluation, and confirm 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).

V. Process and Constraint Modeling

ProCon relies on a dedicated process meta-model that structures the constraint and artifact space. Furthermore, a number of process steps guide the (passive) execution of the process during the engineering life-cycle.

A. Process Meta-Model

One major challenge, when passively executing processes, is to determine the steps that are currently available for an engineer 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. In comparison to existing approaches, the major differences are not the basic building blocks (i.e., the process steps), but rather the way transitions between these steps are defined and subsequently triggered. Figure 3 (left) provides a simplified UML class diagram of the main elements in the process specification and instance

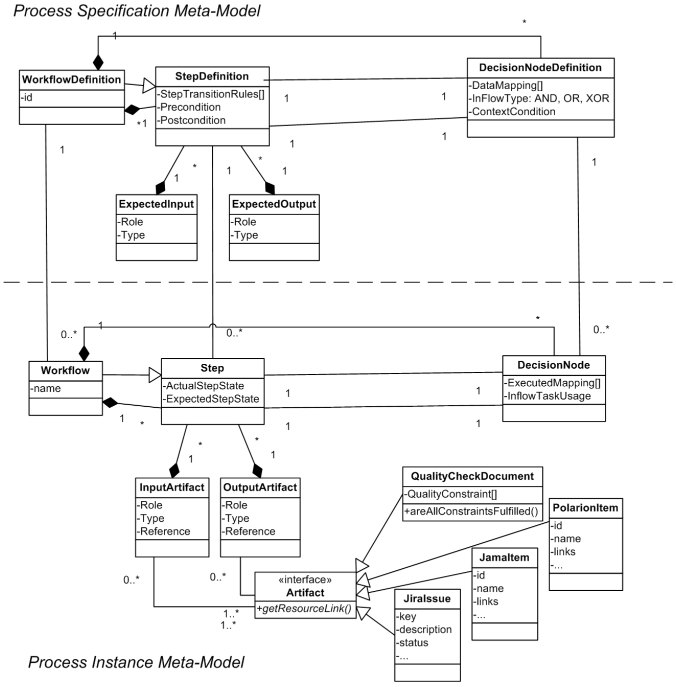


Fig. 3: Meta-model of the process specification (top) and process instance (bottom) - simplified view.

meta-model. The two main elements of the process model are *Steps* and the *Decision Nodes* attached to them.

A **StepDefinition** describes what an engineer *"should"* do (in contrast to "must" do – as prescribed by a more restrictive process). Examples include: "refine a requirement", "implement a feature", or "define a test case". A defined step, in turn, has zero or more *Input* artifacts that represent data required for making a decision, creating a new artifact, or artifacts that need to be modified. Furthermore, zero or more *Output* artifacts are defined, describing the result of executing the step (e.g., having modified an input artifact or created a new artifact). Input and output artifacts can represent arbitrary kinds of information such as requirements, tests, issues, or trace links. While the **StepDefinition** only defines what type of Artifact is expected, identified via its **Role**, the **InputArtifact** and **OutputArtifact** in the process instance will contain the references to the actual artifacts of a concrete instantiated step. In addition to the textual description of an engineer's activity, a step consists of a set of event-condition-action (ECA) rules that define which event(s) stemming 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. Therefore, the set of available decision nodes defines the control flow of the overall process. A decision node's *DataMapping* 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". As the **WorkflowInstance** is also a **Step**, the DataMappings also enable the mapping of workflow input into step input, and step output to workflow output. The **ExecutedMapping**s then track which artifacts have already been mapped.

In order to avoid conflicting control or data flows, ProCon limits the preceding and subsequent decision nodes of a step to one single node. 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 (further well-formedness properties are discussed in Section V-C). 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. We do, however, support recursion by instantiating the same process as a sub-task.

Similar to more traditional process models, a DecisionNode allows the specification of *InFlowTypes* that determine whether all prior steps (AND), at least one step (OR), or exactly one step (XOR) must be completed before any subsequent step is activated. The **InflowStepUsage** tracks which steps have been considered for triggering further activation. The decision node's optional *ContextCondition* determines whether that activation occurs immediately, or only once that condition is fulfilled. This allows refinement of the InFlowType (e.g., at least two steps that need to be completed before the process continues) or awaiting deadlines (e.g., wait for the next day) and similar aspects.

In contrast to the relatively simple decision nodes, a step needs to track its state in much more detail in order to provide developers with feedback on which steps are ready to be started, which ones have started too early, and which ones should not be done at all. Figure 4 and Figure 5 depict a step's expected and actual life-cycle, each modeled as a finite state machine (FSM). Note that the actual step life-cycle FSM extends the expected life-cycle FSM with additional transitions and triggers (depicted in red), while both share the same set of states shown in Table I.

Beyond these standard building blocks, the process meta-model includes additional modeling elements to support our advanced use cases:

- **Constraining Step Input and Output**

The main purpose of pre- and post-conditions of a step is to allow further refinement of input and output artifacts. The engine itself merely checks if there is at least one artifact available for each required input and required output. The step transition ECA rules may have the side effect of adding multiple artifacts per expected output instead of just one, and this is considered normal behavior. Step S3 HLReq

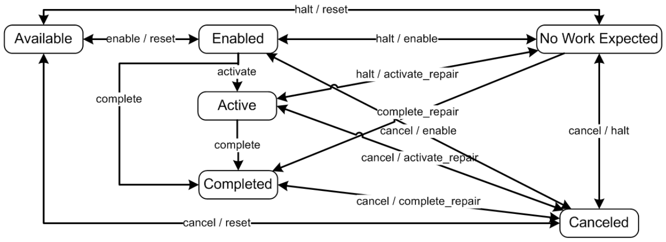


Fig. 4: Expected step life-cycle FSM: transitions list the respective triggers. For bidirectional transitions, the text before the "/" describes the trigger to the right or bottom, the remaining text defines the trigger for transitioning to the left or top.

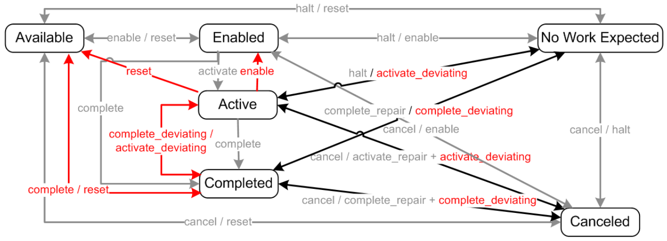


Fig. 5: Actual step life-cycle FSM: red triggers and transitions are in addition to the expected live-cycle FSM, unchanged transitions provided in grey.

State	Description
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) and/or its optional precondition is not yet fulfilled.
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 *StepTransitionRules*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.
CANCELED	When "business logic" dictates that a step can/should no longer be executed (and that condition cannot be encoded in the step's precondition) then the step is canceled. The difference between CANCELED and NO_WORK_EXPECTED is that the former is determined by process constraints, while the latter is determined by the engineer selecting one step out of many XOR alternative steps.
COMPLETED	When all expected output artifacts are available and optional quality constraints, as well as optional post-condition are met, then the step transitions into COMPLETED and triggers the evaluation of the step's subsequent decision node.

TABLE I: Overview of process step states.

Refinement into LLReq, for example, may specify that all refined LLReq are collected within one required output. In this case, a post-condition can ensure, for example, that there are at least as many refined LLReqs as there are provided Input HLReqs.

- **Reactivation of XOR steps:** The states COMPLETED, CANCELED, and NO_WORK_EXPECTED are not necessary final states. In the case of two or more steps from a set of XOR alternatives, the first step that becomes ACTIVE will cause the other step to transition to NO_WORK_EXPECTED. When the active step, however, becomes canceled, then the other steps now become relevant again and hence can transition from NO_WORK_EXPECTED back into AVAILABLE, ENABLED, etc. depending on the fulfillment of pre-conditions and firing of StepTransitionRules.

The same occurs if upstream changes (i.e., changes of artifacts in prior steps, thus process deviations) affect input artifacts in such a manner that a step becomes relevant again. Suppose we have two testing step types: unit test and manual test (that exclude each other) that become enabled depending on the validation type of their input requirement. If a delayed assessment causes the change of the requirement's validation type, then the testing steps' precondition evaluations will be inverted.

- **Deviations from expected state:** Aside from the expected task life-cycle FSM, we use the actual life-cycle FSM to track deviations between the defined and the actual process. For example, an engineer has performed a change but input artifact changes cause the pre-conditions to fail. Then the actual life-cycle FSM would transition into AVAILABLE (which is not possible in the expected FSM). This results in an explicit deviation whenever actual and expected life-cycle FSMs are not synchronous. Continuing with this example, eventually, this step might be not the selected one in an XOR, thus both FSMs transition into NO_WORK_EXPECTED, thereby resolving the inconsistency. Similar deviations are possible when a step is COMPLETED, CANCELED, or NO_WORK_EXPECTED but further work is observed once that state is reached. Here the actual FSM would transition into ACTIVE.

To distinguish when a transition to the active and completed state is deviating in contrast to repairing, we append "_deviating" and "_repairing" to the triggers in Figure 4 and Figure 5.

B. Process Notation

The data model described above determines all information required by ProCon to instantiate a process. Our intention was not to mandate a particular visual notation, and we opted against using a preexisting language, such as BPMN [65], for two main reasons. First, we aimed to reduce the data model to only its core elements that are absolutely necessary. For example, BPMN defines far more elements than what is supported by our framework, hence making process design much harder for a process engineer, as they would need to recall which elements to use and which ones have no effect. Second, our constraints are defined as Drools rules and subsequently best written using a Drools editor, hence requiring the process engineer to switch between several tools.[3] However, as we define a canonical process format in JSON one could produce a transformation from a BPMN process model to our data model. This would be rather straightforward as noted above in Section IV: the key modeling elements themselves such as steps and AND, OR, and XOR decision nodes used for defining the process structure are not new. The novelty is in the way they are interpreted and allow for flexibility by ProCon.

C. Well-formed Process Specification

The process definition meta-model prescribes that a step must have exactly one predecessor decision node and exactly one successor decision node, but no further rules on what the emerging graph, containing these two element types, should look like. A well-formed process specification consists of a single kickoff decision node leading to steps that ultimately converge in a single-end decision node. Hence, only one decision node has no preceding steps, and only one decision node has no subsequent steps. Furthermore, whenever a decision node has k subsequent steps (with $k > 1$), a decision node that "collects" these k branches (i.e., having k preceding steps) needs to exist at some point in the process. In the prototype implementation section, we show how our web-based process editor assumes responsibility for automatically ensuring such well-formed process specifications without burdening the process designer with this aspect.

D. Quality Constraint Integration

For each step, additional *Quality Constraints* can be defined, describing conditions for the newly created or updated output artifacts. Such a constraint can refer to all the information that is available for a step, such as its input and output artifacts, its local metadata, and its global process metadata.

Each constraint has an identifier that enables the triggering of constraints – not only upon artifact changes – but also manually, upon demand. It is important to provide control to the user, for example, by allowing an engineer to trigger a constraint check to reassure him/her that a step is indeed complete and nothing has been overlooked. ProCon does not prescribe a specific constraint engine or language in which constraints are defined, as this can vary depending on the domain and application scenario. The only requirement is that the adopted solution provides respective evaluation results that can be further processed. Further examples of constraints and the constraint language are discussed as part of the case studies described in Section IX).

Every time a quality constraint is evaluated, a corresponding new instance of *Quality Constraint Evaluation Result* is created. This evaluation result not only reports the result (i.e., fulfilled or unfulfilled) of the evaluation, but also lists the artifacts affected by the 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 each constraint, ProCon further collects timestamps indicating when the constraint evaluation result was last evaluated, and when the evaluation result last changed from being violated to fulfilled and vice versa. This, for example, allows a user to determine how recent the results are when displayed in the Process Dashboard. The evaluation results of all quality constraints associated with the same step are collected and bundled into 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 does not need to understand the particular constraints that resulted in the evaluation success.

This rather loose coupling allows the same process to be executed with different levels of quality assurance by simply replacing quality constraints. Inversely, a quality constraint may be used in different process contexts when the respective process step provides the artifacts on which the constraint is evaluated.

E. Supporting Semi-structured Artifact Properties

Development tools quite frequently support artifacts that can not be sufficiently customized to match the underlying development process. In such cases, users typically revert to providing information in text fields or rich edit fields where data is entered using HTML tags or markdown. To this end, the process engine utilizes the full potential of Java that comes with defining the rules and constraints in the Drools rule engine. ProCon supports the extension of individual artifacts with arbitrary key-value pairs that are populated when an artifact changes based on custom drools rules. These rules transform semi-structured information from the artifact to structured key-value pairs that can then be used in preconditions, post-conditions, StepTransitionRules, etc., hence separating the concern of how to extract information from artifacts from the concern of processing that information.

VI. Passive Process Execution

Every process specification model comes with a set of required input artifacts, typically representing a change request issue or work package issue that already serves as some form of informal process representation. There are three ways in which a process instance can be created. First, manually when a ProCon user provides the identifier(s) of the input artifact(s) via the framework's user interface. Second, when a subprocess

[3] We are currently working on unifying the process structure and constraints to remove the need for tool switching.

step definition is instantiated, the step instance then creates the corresponding process passing its input artifacts along. Third, when a rule in an existing process instance spawns a new separate process. The difference between the second and third option is that in the former case, the framework ensures that output from the subprocess is mapped back into the subprocess step and any relevant state changes of the subprocess step such as canceling are propagated to the subprocess. In the latter case, the spawned process remains completely independent from the spawning process. The last option is particularly useful for creating multiple fine-grained processes, for example, one for each story within an epic.

The process engine instantiates the process and triggers the initial decision node instance, which in turn instantiates the first step(s) and executes the data mappings from process input(s) to step input(s). Further instantiation happens incrementally, only when a step reaches the state `ENABLED`, and the process engine instantiates the step's subsequent decision node. Similarly, only when a decision node's context condition is fulfilled, the engine instantiates the subsequent steps. 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.

However, as passive process execution does not mandate and enforce a fixed, predefined step, incremental step and decision node instantiation are not sufficient in our case. For example, when an engineer decides to work on a step that is not yet available, as soon as the artifact changes trigger *StepTransitionRules*, the process engine must instantiate the corresponding "premature" step and retrieve an existing preceding decision node that this step should be linked to. If no preceding decision can be found, 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. From the engine's perspective, 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 a case, however, the engine will not be able to fully execute a DataMapping that requires the output of the skipped/incomplete step. The consequence is then highlighted via the step's status as having insufficient input artifacts. In the case of a premature step, the missing input artifacts will eventually be mapped when the prior steps are complete. As a decision node instance keeps track of executed DataMappings, it notices which artifacts have not yet been mapped. In the latter case, when a step is skipped, the user may manually provide the missing input artifacts via ProCon's web interface.

Missing input has multiple effects: when StepTransitionRules rely on missing input, they will not trigger any state transitions in the step. The same rules might use the input to navigate to or to identify the step's output, which in turn also will prevent the quality constraint checks from being evaluated. As long as input is missing, the step will not reach a `COMPLETED` state, even if the user considers the step done. In short, missing input will typically lead to failure of the step's state to accurately reflect the step's true progress in the real world. However, ProCon will continue to support the user in other parts of the process, or even in other steps in the downstream process which will then most likely also be treated as "premature".

As a side effect, in iterative development environments, with often missing or incomplete artifacts, the step states become less informative as steps will reach and remain less often in the `COMPLETED` state. For engineers assigned to specific tasks, ProCon nevertheless informs about which artifacts are not ready yet or currently changed again. For stakeholders, such as team leads and QA Engineers interested in an overview of one or more processes, percentage data might be then more insightful.[4] To this end, the process engineer needs to consider the extent and parts of a process subject to significant iterative activities when determining the process scope, e.g., foreseeing one process instance per development iteration.[5]

A. Constraint Checking

ProCon provides two possible ways of how constraint checks can be performed. By default, QA constraints and StepTransitionRules are evaluated upon every single artifact change. As ProCon tracks which process instance accesses which artifacts and forward the respective change events, only constraints relevant for the process instances need to be checked.

Additionally, we allow for manual user input, enabling an engineer to request explicit checks of artifact updates and subsequent evaluation. The reason for this is based on the fact that many tools do not provide an active notification mechanism when changes to artifacts occur. Jama, for example, doesn't offer automatic event notifications, but requires polling with subsequent explicit fetching of changed artifacts. In the case where polling intervals can not exceed a certain duration, to avoid introducing performance issues, engineers will not immediately notice effects of their work on QA constraints. This effect becomes even more severe when a single change is not indicative of step completion or QA constraint fulfillment. This is the case when quality constraints span across multiple artifacts and potentially even multiple tools: a single change to an artifact then 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. In such a situation, on-demand fetching of updates ensures the engineer that all QA constraint evaluations occur on the most recent artifact versions, and any violation will not be due to stale data.

B. Propagating Artifact Input/Output Changes

In the same manner as StepTransitionRules monitor artifacts for changes to add output artifacts (e.g., a requirement gets linked to a test case), they remove output artifacts when

[4] How to best aggregate and display meaningful percentage data is outside of the focus of this paper, and part of future work.

[5] We acknowledge that highly dynamic, highly iterative development environments perhaps might not benefit as much from our framework.

their conditions are no longer met (e.g., a link to a test case is removed again). The process engine ensures that such artifact usage changes are properly propagated to subsequent tasks. For example, in a case when S4 (Updating LLSpec) is completed and S5 (HLReq to LLReq Trace Review) has started. Now deviating from the process by having in the scope of S4 adding another trace between a HLReq and LLReq. Execution of S4's StepTransitionRules would result in another trace (also an artifact) to be added to S4's output which needs propagating as input to S5. Likewise, removing such a trace in S4 would result in removing the respective trace artifact from the input of S5. A change in input of a subsequent step may then in turn cause the revaluation of that step's transition rules and cause the post-conditions or quality constraints to change with impact on the step's state; with further change propagation potentially occurring.

C. Rule Execution Order

The majority of project/team-specific aspects such as task pre-conditions, post-conditions, StepTransitionRules including adding output, and quality constraints are defined as rules which in turn signal changes to the process. Together with the process engine internal change triggers such as task instantiation and DataMappings etc, these change evaluation sources need to be activated in a predetermined order to avoid "race" conditions: e.g., executing post-condition checks that signal task completion before adding new output that would cause these post-conditions to fail and hence not to signal task completion. Recall that the ultimate source of any activity in the process engine is an artifact change. Upon an artifact change, the change evaluation sources then are evaluated in the following order: [6]

- Adding input to tasks
- Pre-processing of artifact change events (e.g., when semi-structured artifact properties need to be parsed)
- StateTransitionRule which evaluate the pre-conditions
- StateTransitionRules which add output to tasks
- StateTransitionRules which activate or cancel the step
- Quality constraint evaluation
- StateTransitionRules which complete the step (i.e., the post-conditions)

VII. Evaluation Method

Evaluating our ProCon framework, we investigate five research questions regarding the *feasibility*, *flexibility*, *performance*, and *usability* of our approach. Specifically, we assess the general feasibility of our approach by evaluating it against real historical process data from three distinct case studies. For each case study, we created process specifications and constraints, and implemented connectors for issue tracking and requirements management tools.

A. Research Questions

While this research project was originally motivated by a pain point of our industrial collaborators (Frequentis), our first two research questions investigate the nature of process deviations and constraint violations (i.e., from a process view and a QA constraints view, respectively) to empirically evaluate the need for a tool such as ProCon.

RQ1: ***How frequently do process deviations occur, and to what extent are these temporary?***
Rationale: As part of this research question we investigate process deviation problems to uncover whether engineers continue their work even in the presence of QA violations, and ultimately the extent to which they require support to eventually fix these process deviations (i.e., investigating from a process perspective). To this end, we replay historical engineering process events and establish if and when engineers continued to a subsequent step in the presence of a constraint violation.

RQ2: ***Are quality constraint violations common in practice?***
Rationale: When only a small subset of QA constraints is violated, the effort for specifying detailed processes, transition rules, and constraints, as well as to maintaining ProCon might outweigh its benefits. We are, therefore, interested in obtaining insights into the potential benefits that ProCon can provide (i.e., investigating from a constraint perspective). Similarly to RQ1, during the replay of historical engineer process events, we determine which constraints are violated.

Our next two research questions focus on evaluating the ability of ProCon to track process progress under realistic conditions in a timely manner.

RQ3: ***Does ProCon sufficiently support flexible processes in which engineers frequently iterate and switch between alternative steps?***
Rationale: Software development is rarely a predefined sequence of activities but often requires rework, breaks, and adaptation. Hence, we investigate whether ProCon is able to support engineers with accurate feedback in the presence of frequent deviations from the prescribed process. For this purpose, we collect statistics on how often process steps are canceled, reactivated, revoked, and prematurely started.

RQ4: ***Is ProCon reacting to artifact changes sufficiently quick for practical application?***
Rationale: Engineers are least interrupted in their work when they can receive feedback on their actions before moving on to their next task. We collect performance data by measuring the time ProCon requires to evaluate constraints upon change events.

Our final research question focuses on ease of use.

RQ5: ***Is ProCon sufficiently simple to define processes and QA constraint to encourage practical applicability?***
Rationale: When specifying processes and their QA constraints require a steep learning curve, chances for uptake by practitioners, or the integration of the approach in an actual product, will be significantly diminished, and our research will fail to make an impact. We obtain insights into ease

[6] Note that typically only one or two such change evaluation sources become indeed active and trigger a consequent change such as a step state change.

of use by conducting a preliminary controlled experiment, where developers were asked to write process progress and QA constraints.

B. Case Studies

We address the research questions by applying ProCon to three distinct systems with unique processes and artifacts, from different domains and report our findings and lessons learned. The first case study, (CS-1) Dronology [66] – an open-source project – represents an agile, lightweight process. The second case study, (CS-2) is provided by Frequentis, a producer of safety-critical systems in the air traffic management domain, and describes a rigid, standardized process with stringent quality assurance criteria. The third case study, (CS-3) ACME-RA, is placed in a company developing non-safety critical web and mobile applications using constraints to measure process improvement. For the last research question, we mapped an existing traceability information model and process [67] (case study CS-4) onto Jira artifacts and their relations and asked a set of participants to specify process step transition rules and QA constraints.

CS-1: 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 [66], [68]. 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.

CS-2: Frequentis

For the Frequentis case study, we selected a safety-critical product for voice communication in air traffic control centers. The product consists of several different subsystems for interfacing with radio transceivers, managing near-real-time voice streams, and providing operator user interfaces. Frequentis follows a V-model [69] 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 II). Frequentis uses Jama to manage all artifacts and trace links depicted in Figure 1 and uses Jira to manage the engineering process.

CS-3: ACME-RA

ACME-RA[7] is in the business of hosting a recreational activities web platform. ACME-RA tracks development progress with Jira, heavily customizing available issue states and state transitions. Different issue types come with different states and transitions. Based on feedback from a developer at ACME-RE we selected issues of type "Task" as a representation for a non-trivial development sub-process. ACME-RA follows an agile development methodology, pulling issues in from a backlog for each sprint. Issues then undergo a set of possible transitions some of which require a particular engineering role to be involved. Once a new task is created, its initial state is `Open`. The state changes to `In Development` as soon as the work on the issue starts. After finishing the task the developer changes the state to `Ready for Review` when needed. A quality assurance engineer then picks up the task and assesses the issue before changing the state to `Reviewed`. Alternatively, no review is conducted and the task is regarded as finished, hence it changes to `Resolved` (one of the allowed end states). If required, testing can be performed after resolving the task (state `In Testing`). Both development and testing can be suspended (states `Suspended Development` and `Suspended Test`), for example, in case additional resources are required, and then resumed when these resources become available. Additional states track when a task needs to be in development again, is suspended again, or reviewed again.

CS-4: Siemens L+A

The case study from Siemens Logistics and Automation, described by Cleland-Huang et al. [67], involves various artifacts stakeholders use to capture the requirements and system components for automatically obtaining a shop floor layout. Business Goals (BG), Stakeholder Requests (SR), Minimal Marketable Features (MMF), and Business Use Cases (BUCs) are primarily used by business end users, while developers primarily interact via System Use Cases (SUCs), Concrete System Capabilities (CSCs), and Concrete System Components (CSComp). While a significant set of traces exist, the process of implementing a stakeholder request can be modeled rather straightforward as a sequence of "WriteOrReviseMMF", "RefineToSUC", and "CreateOrRefineCSC" steps.

Regulatory Certification

The case studies are subject to different regulatory certifications. The Dronology project (CS-1) aims to follow best practices and guidelines as, for example, specified in DO-178C [6] but does not require official certification, and hence is not certified. The ATC system by Frequentis (CS-2) is

[7] The company's identity and project names have to remain confidential due to the sensitive nature of the analyzed data.

developed to be compliant with ED-109A/DO-278A [7] and externally certified. In our evaluation, we focus mostly on the respective standards' traceability requirements, whereas the process description stems from engineers at the two case study organizations, describing how they manage software development and fulfillment of traceability constraints. ACME-RA (CS-3) and Siemens L+A (CS-4) are not subject to certification but were specifically chosen to demonstrate the applicability of our framework in non-regulated domains.

Differences in Case Study Results Reporting

Note that these case studies serve different evaluation purposes. CS-1 Dronology, CS-2 Frequentis, and CS-3 ACME-RA provide real historic process data to answer RQ1 and RQ2. While CS-2 has a simple underlying process but complicated constraints, CS-3 additionally was chosen to evaluate whether ProCon can also support a more complicated process where engineers repeat steps and may choose to skip steps (RQ3) but CS-3 yields more simple constraints than CS-2.

Performance evaluation in the scope of RQ4 only considers CS-2 and CS-3 as the former comes with the most complicated constraints among the four case studies and the latter comes with the most complicated process progress constraints among the four case studies. Both case studies come with a significant amount of related artifacts and change events. Hence they produce realistic loads on ProCon.

Finally, CS-4 Siemens L+A was selected for evaluating ease of use of specifying process and QA constraints as it comes with a reasonably complicated process structure, together with reasonably complicated QA constraints—thereby representing a middle ground between CS-2 and CS-3. As we have no access to historical process data for CS-4, we cannot report results for RQ1 to RQ4 for this case study.

VIII. Prototype Implementation

To support the evaluation, and to obtain feedback from our industry partner Frequentis about our approach and the provided tool support, we created a prototype implementation of ProCon.

Tool Connectors: To cover a diverse set of artifacts from our industry partner and from the Dronology case study, 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 the load on network and tools, the tool connectors cache Jira and Jama artifacts in a MySQL database in their native JSON format as obtained from the REST interface.

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 [70] rule engine, a Business Rules Management System that can be easily integrated into a Java application and allows easy access to Java objects (representations of Jira and Jama artifacts) within rules written in a Java dialect. The process engine is wrapped in a web application (see Figure 6) using the Command Query Responsibility Segregation (CQRS) pattern via the Axon framework[8]. Here any commands (A) (i.e., requests) to the engine are first checked for validity (B) against a lightweight version of the process instance, and then may result in events (C) that describe the expected change due to the command. These events are then propagated to the query side of the framework where the event is processed to realize the effect of the original command (D). From there, the user interface obtains the current process status and Quality Check Documents (E). Separating command and query side enables us to have different applications transparently connecting to events aside from the primary process dashboard (see below). As the CQRS pattern enables the replaying of events, we can utilize these to replay the progress of each process instance and inspect the fulfillment of QA constraints at any moment in time (F).

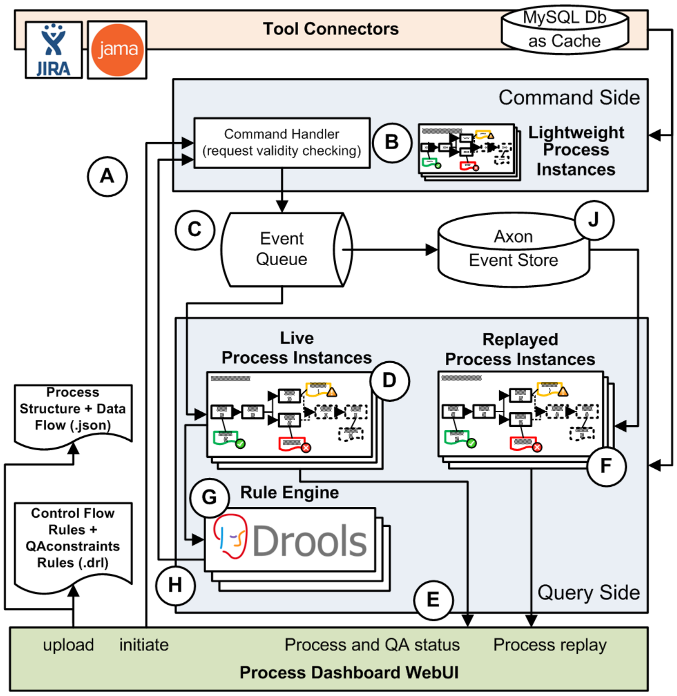


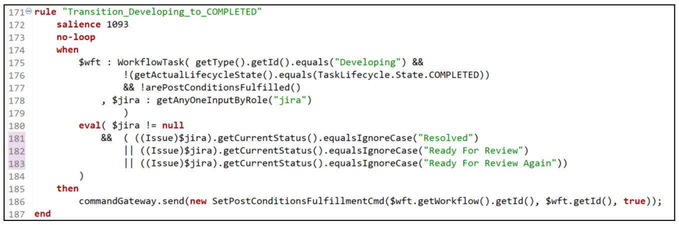
Fig. 6: ProCon Prototype Architecture.

Also, on the query side resides the rule engine, and any firing of step transitions and quality assurance constraints checks as the result of a change in the live process instance (G) potentially result in additional commands back to the process engine via the command side (H). This opens up to opportunity for manually overriding step transition conditions and QA check results, if so desired, by an authorized user role while ensuring that such commands (and their consequential events) are captured and persisted by the Axon Event Store (J) to form an audit trail.

[8] https://axoniq.io

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. Figure 7 provides an example rules excerpt that controls the completion of a process step by signaling that the step's postconditions are fulfilled. Most Drools code is boilerplate with the application-specific part focusing on the specific conditions of the StepTransitionRules. Similarly, quality constraints come with a significant portion of boilerplate code. To enable the process engineer to focus on the constraint formulation and avoid introducing errors in the boilerplate code, we provide a preliminary visual process editor (see, for example, a screenshot of the ACME-RA process in Figure 8) utilizing Blockly.[9] The editor ensures the well-formedness of the process steps, and generates the boilerplate code from the visual representation. In the rules excerpt of Figure 7 only lines 181 to 183 out of lines 271 to 187 had to be manually added, with the rest being generated automatically by the process editor. Overall, the process progress rules for the ACME-RA case study involved around 600 lines of code (not counting any java imports), out of which only around 70 had to be manually written. Similar for the quality constraints: for the four ACME-RA constraints, only 14 lines out of 150 (again without java imports) had to be manually defined.

The preliminary process editor enables a process engineer to define the steps, the types of artifacts used in input and output, their parallelism/alternatives, and which artifacts are utilized in StepTransitionRules and QA Constraints. Formulating the actual conditions of the rules and constraints is not yet supported as we are still investigating which type of visualization is best suited to specify complex conditions over artifact properties. Nevertheless, this structure is sufficient to generate all boilerplate code.



```
171 rule "Transition_Developing_to_COMPLETED"
172     salience 1093
173     no-loop
174     when
175         $wft : WorkflowTask( getType().getId().equals("Developing") &&
176                 !(getActualLifecycleState().equals(TaskLifecycle.State.COMPLETED))
177                 && !arePostConditionsFulfilled()
178             , $jira : getAnyOneInputByRole("jira")
179                 )
180         eval( $jira != null
181             &&  ( ((Issue)$jira).getCurrentStatus().equalsIgnoreCase("Resolved")
182                 || ((Issue)$jira).getCurrentStatus().equalsIgnoreCase("Ready For Review")
183                 || ((Issue)$jira).getCurrentStatus().equalsIgnoreCase("Ready For Review Again"))
184         )
185     then
186         commandGateway.send(new SetPostConditionsFulfillmentCmd($wft.getWorkflow().getId(), $wft.getId(), true));
187 end
```

Fig. 7: ProCon Drools StepTransitionRule example: signaling the "Developing" step's post conditions as fulfilled.

Process Dashboard: Figure 9 shows the user interface for inspecting quality constraint evaluation results. The results contain links to the original artifacts, enabling engineers to quickly switch to their commonly used tools (here Jama and Jira) to investigate and fix any unfulfilled constraints. The process dashboard is automatically updated whenever a step, decision node, or quality constraint evaluation changes without the user having to poll for updates in the browser. There the user also has the option to inspect available or missing artifacts and even manually add artifacts to step input and output in case the currently active rules fail to identify these from the process context.

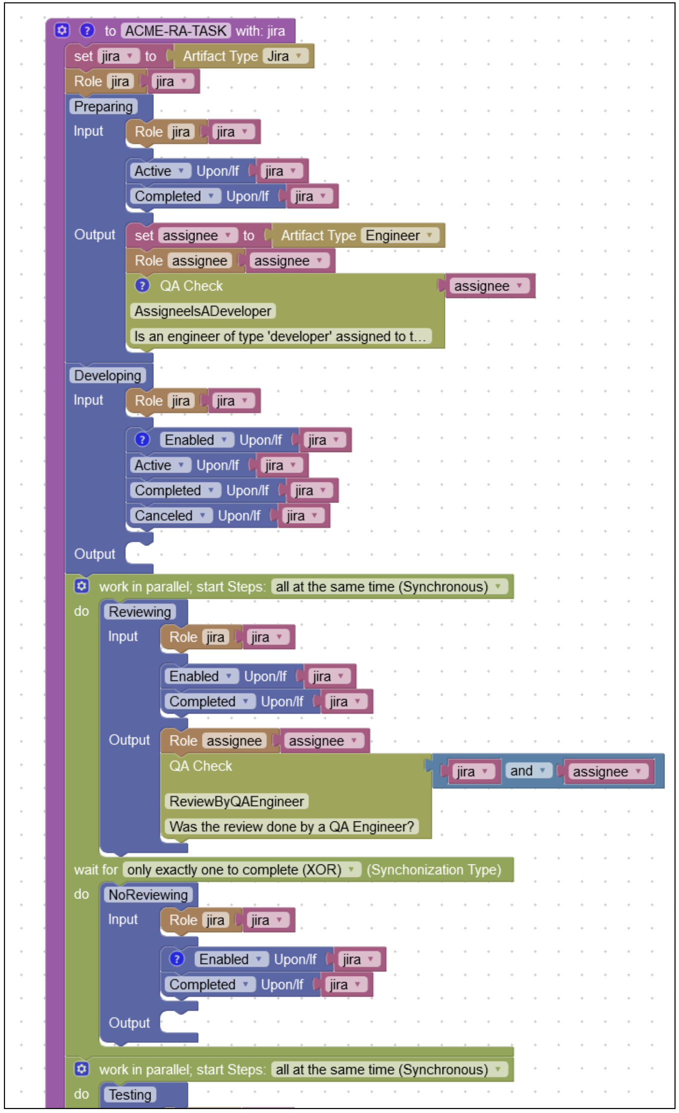


Fig. 8: ProCon Process Editor screenshot displaying an excerpt of the ACME-RA process.

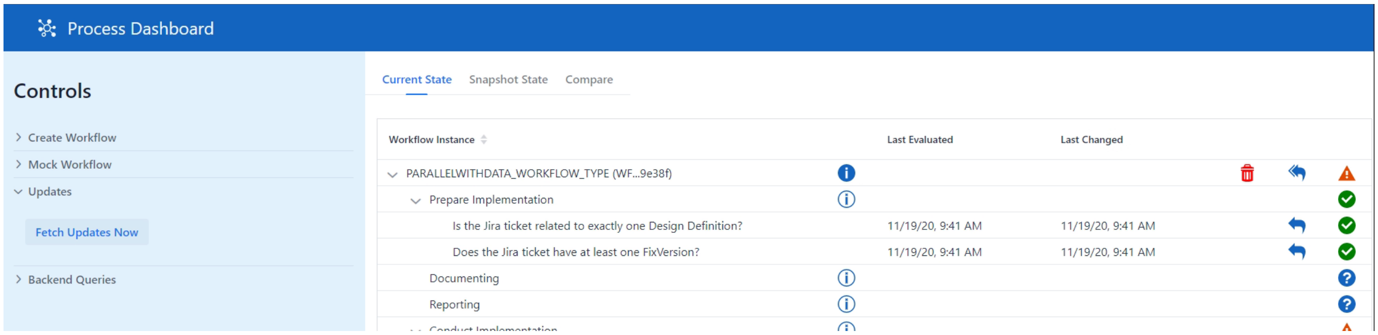
IX. Evaluation Preparation

Before ProCon can be used with a specific process and system, a number of preparatory steps are necessary, to identify, select, and integrate artifacts and tools, and create respective constraints. In the following, we briefly describe the relevant steps for our four case studies.

A. *Evaluation Setup*

CS-1: Process and Constraint Creation: For the purpose of this case study, we treat each of the collected Dronology issues as "small sub processes". The state of each issue represents a process step, and quality constraints for each step describe the conditions that need to be fulfilled to transition

[9] https://developers.google.com/blockly



Workflow Instance		Last Evaluated	Last Changed	
PARALLELWITHDATA_WORKFLOW_TYPE (WF...9e38f)	ⓘ			⚠
Prepare Implementation	ⓘ			✓
Is the Jira ticket related to exactly one Design Definition?		11/19/20, 9:41 AM	11/19/20, 9:41 AM	✓
Does the Jira ticket have at least one FixVersion?		11/19/20, 9:41 AM	11/19/20, 9:41 AM	✓
Documenting	ⓘ			?
Reporting	ⓘ			?
Conduct Implementation	ⓘ			⚠

Fig. 9: ProCon Process Dashboard.

from one step to the next and to complete the process (i.e., close the issue). Given the lean nature of typical agile open-source development processes, the states observed are 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). Defining the processes and constraints used in this evaluation took approximately three hours and did not include familiarizing with the project's artifact types and traceability strategy. We then confirmed the validity of the constraints and the process together with the lead developers of the project. An overview of the constraints can be found in Table II. 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.

CS-2: Process and Constraint Creation: For the second case study, we defined the eight constraints from the SubWP process together with a QA engineer from Frequentis. This was done in approximately two hours, and another two hours were spent extracting process information from documents and specifying the actual process. This duration did not consider familiarizing with the artifact types and their traces as this was done earlier in the collaboration with Frequentis and can be assumed to be common knowledge of a QA engineer. 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.

CS-3: Process and Constraint Creation: For the third case study, the web platform, we selected "Task" issues and transformed the state transitions into a process consisting of seven steps (see Figure 10). Any issue state that indicates a repetition of some activity was treated as a reactivation of the corresponding process step. For example, reopening an issue (i.e., `Reopened`) is interpreted as a reactivation of the "Prepare" step. Together with a developer from ACME-RA we discussed the conditions under which these transitions should occur and encoded three of them as quality assurance constraints (listed in Table II bottom). [10] Constraint RA-1 needs to be fulfilled at the end of step "Preparing", constraint RA-2 is required only in case a review is conducted (i.e., step "Reviewing"), constraint RA-3 is required only in case testing is carried out, and constraint RA-4 is checked for finalizing the "Completing" step. The four constraints were straightforward to encode within half an hour, while the step activation, completion, etc. rules required more time (around 4 hours), due to the extensive use of different issue states.

For the purpose of this paper, in the ACME-RA case study, we focus on demonstrating (aside from QA constraint checking) the ability of the passive process engine to handle revoked, reactivate, canceled, and prematurely started process steps.

CS-4: Process and Constraint Preparation: In a first step, we mapped the TIM to Jira by creating a dedicated Jira issue type for each artifact type, and mapped the process to a Jira story with the individual steps modeled as that story's subtasks. We then created example artifacts and traces (including an example process instance). Such a grounding of the TIM and process in actual tools is required in any case, independent of the application of ProCon.

Given the simplicity of the process compared to the more complex traceability information model, the preliminary usability evaluation focuses on the effort required for specifying process step transition rules and the QA constraints. To this end, we prepared the process specification with the ProCon process editor and generated the Drools templates for the process step transitions and quality constraints.

Performance measurement We focus on measuring the time ProCon requires to check all constraints after an update event (i.e., a change to an artifact) occurred. We leave aside any time required to load the artifacts from their originating tools as this is heavily influenced by the tool's API[11] current load and availability of artifacts already in ProCon's cache. Hence, we measure the time from replaying the first of the change events to the end of processing the last change event for CS-2 Frequentis and CS-3 ACME-RA. We measured this

[10] Due to the limited amount of issue details, we were only able to encode a small subset of the actual constraints.

[11] Tools differ in the number of API calls required to obtain artifact details and artifact updates.

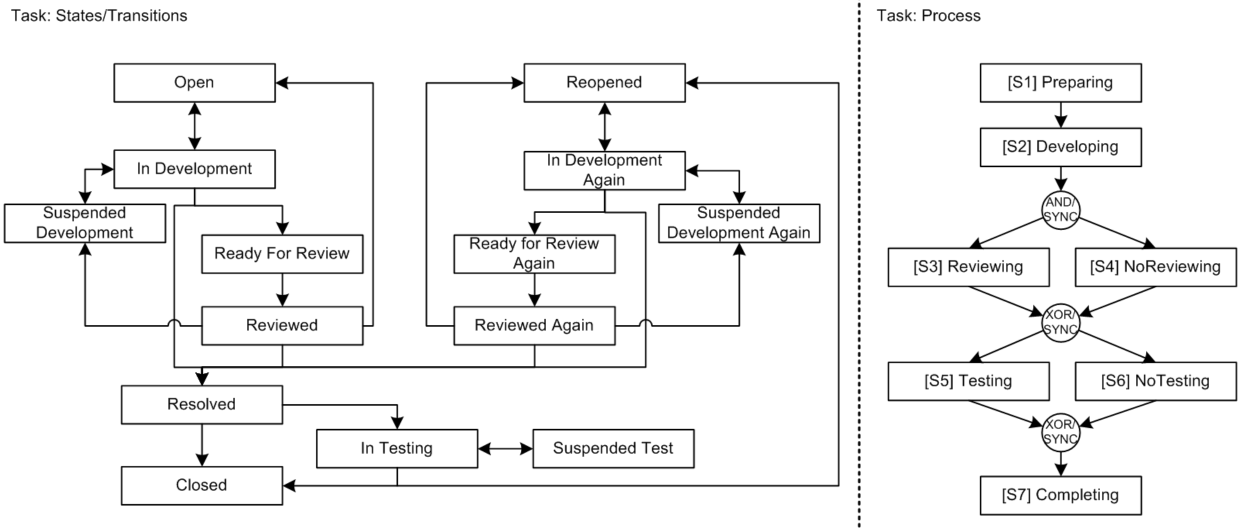


Fig. 10: ACME-RA transformation from issue states (left) to process specification (right) for issues of type "Task". Circles represent decision nodes.

interval for 10 replay runs on a standard Core i7 laptop, with 8 GB of RAM to obtain the average duration. Dividing the duration by the number of relevant events (i.e., those that potentially affect the constraint evaluation result) provides insights into the expected average processing time to evaluate one change event. We measure in the presence of multiple process instances as realistically a change event potentially affects multiple constraints (in the scope of multiple process instances) that all need separate re-evaluation.

User Experiment Preparation In the scope of the small, controlled experiment, we asked six software developers that had not used ProCon before to implement the StepTransitionsRules for activation and completion for each process step as well as the input to output DataMappings (experiment task 1) and to implement the six QA constraints textually given in the process specification (experiment task 2) by filling out the generated Drools templates. Overall, the participants had to write six StepTransitionRules, seven DataMapping rules, and six QAConstraints.

We recruited six participants, each of whom had some knowledge about software processes and basic knowledge about Drools, but had not used ProCon before to write quality constraints or process progress constraints. The participants had between 1 and 6 years of Java development experience in the scope of research project employment at our research institution.

The six participants had access to the ProCon development environment in Eclipse comprising the Drools Editor, ProCon Java API, with source code to access and inspect Jira issues, as well as code for executing their process and constraints rules. We additionally provided one example process step transition rule and one QA constraint from CS-3 as a reference.

During the 75 min experiment, the participants received a 15-minute introduction to the framework and development environment. They then had 30 minutes and 20 minutes available for the two experiment tasks, respectively, and 10 minutes to provide informal feedback at the end.

During the experiment, we documented how much of the task each participant was able to complete, and how many mistakes they made that remained in the final output. Additionally, participants received minor feedback on the Jira Artifact's API.

B. Data Gathering

For the Dronology project, we received access to the 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).

From ACME-RA we received a JSON dump of Jira issues of four multi-year projects: P1, a low-priority Android app development project; P2 and P3, two business-critical Android App development projects; and P4, a project integrating two types of recreational activities that involved experts beyond front-end, business logic, design, and database engineering (e.g., marketing and legal departments). In total the dataset contained 1,017, 2,676, 1,052, and 939 issues, respectively. The five most common issue types were "Task", "Bug", "Improvement", "Localization", and "Project Management" and make up between 80% and 90% of all issues. All issues had their change history reduced to changes to the properties assignee, state, fix version, and due date. Additionally, the

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
RA-C1	An engineer of role ’developer’ needs to be the issue assignee	Task
RA-C2	An engineer of role ’QA’ conducts the review	Task
RA-C3	An engineer of role ’QA’ conducts the test	Task
RA-C4	All sub-tasks are in state ’Closed’	Task
LA-C1	Each Process (story) must trace to at least one SR via a ’Realized’ Link	Story
LA-C2	Each linked MMF must trace to at least one SR via a ’Realizes’ link.	Subtask WriteOrReviseMMF
LA-C3	Each linked MMF must trace to at least one BUC via a ’Realizes’ link.	Subtask WriteOrReviseMMF
LA-C4	Each SR linked via an MMF must trace to at least one BUC via a ’Relates’ link.	Subtask WriteOrReviseMMF
LA-C5	Each SUC must trace to at least one (parent) SUC or a BUC via a ’Realizes’ link.	Subtask RefineToSUC
LA-C6	Each CSC must trace to at least one SUC via a ’Realizes’ link.	Subtask CreateOrRefineCSC

TABLE II: Constraints derived for the Dronology use case (“D”), the Frequentis (“ATC”), the ACME-RA (“RA”), and the Siemens L+A (“LA”).

anonymized user identifier were enhanced with role identifiers to distinguish between front-end developers, back-end developers, database developers, team leads, quality assurance engineers, graphics designers, marketing engineers, and bots. Project P1 was the first to start, its end interleaving with P2 and P3 which had a similar duration, their end again interleaving with the start of P4 which, at the time of data gathering, was not completely finished yet.

For this study, we retained only issues of type “Task”, that were successfully resolved (i.e., in state “Fixed”), with a non-empty set of child issues. This resulted in 46, 21, 119, and 81 process instances, respectively.

We used our trace link replay tool [64] to reset all datasets, particularly the Jira issues and Jama items and their trace links to the earliest change event and then replayed every single change in the correct temporal order. The changes occurred between April 2017 and December 2019 for Dronology, between May 2018 and June 2020 for Frequentis, and between December 2013 and January 2018 for ACME-RA, 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 separately for each change. In other words, after each change event, we evaluated all process and QA constraints against the updated process snapshot.

To answer RQ1 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 predecessor step(s) being fulfilled.

To answer RQ2, we collected the following metrics for each process instance (i.e., a Jira artifact): 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).

To answer RQ3, we additionally captured the number of step cancelations, reactivations, and revocations only when replaying CS-3 as the process descriptions for CS-1 and CS-2 didn’t make use of exclusive (XOR) branching.

X. Results

A. RQ1: Process Replay

Tables III and IV report details regarding the QA constraint evaluation results across multiple process instances, grouped per process type for the three case studies.

`AlwaysOk` represents the number of process instances where engineers only progressed to subsequent steps when all quality constraints in previous steps were fulfilled. `EventualOk` reports the processes for which all constraints were eventually fulfilled. `CompleteNotOk` shows processes for which at least one constraint was never fulfilled. `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. `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 that point as depicted 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 ∼70% remaining unfulfilled at the end. "Sub-task" processes see ∼30% of instances "correctly" carried out, with only ∼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. When examining the incomplete process instances we encountered 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 noticed that only a low percentage ($<$ 20% for IncompleteProgressedNotOk) of instances have engineers started too early on subsequent steps without having fulfilled the previous steps' constraints.

Frequentis: For the case study residing in the safety-critical domain, we observed two interesting aspects. First, the number of SubWPs ultimately Ok reaches almost 90%, with the remaining 10% SubWPs showing unfulfilled constraints. To further investigate these, we manually examined the violating artifacts (exclusively LLReqs) and the comments attached to the SubWP Jira issue. Given that Jira is used as the primary means for communication and as 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, and therefore no trace links to Functional Units were present. In one case test cases were not applicable, and in three cases more than one Functional Unit was linked. 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. The second observation we made was that 11 SubWPs are IncompleteOk, even though could confirm 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 XI.

ACME-RA: For all four projects we noticed that only a third or fewer of all process instances never incur a quality constraint violation. For P1 and P2 close to half or more of process instances remain with violations at their end. P3 and P4 fare much better in this respect with less than a third, or just 15% of completed process instance, respectively, having no constraint violations. The primary reason for unfulfilled constraints for completed process instances across projects P1, P2, and P3 was that the assignee for development was not a developer, but frequently assumed the role of team lead. For P4, the main reason was that not all child subtasks are closed or that a team lead or the assigned developer conducted the testing step. This violation temporarily also occurs in the other projects but is eventually resolved as all issues become closed at the end of the project. P4, however, was not completed at the time of data gathering and hence several process instances remained with an unfulfilled RA-C4 constraint.

Addressing the incomplete process instances: P3 has three incomplete processes. Upon manual inspection of the issue's change log, we noted that they missed the final transition from "resolved" to "completed". P4 has 11 incomplete processes which all come with a violation of RA-C1, i.e., not having a developer assigned for development but the team lead.

RQ1 Key Observation: Temporary deviation from the prescribed process in the form of prematurely starting process steps without having preceding quality constraints fulfilled is common in open-source system development and industrial settings. The open source CS-1 sees most deviations for processes with multiple constraints, the industrial safety-critical CS-2 has most process instances executed in the expected step sequence even in the presence of multiple complicated constraints, while the industrial non safety-critical CS-3 shows less deviation in processes that were part of more recent projects.

B. *RQ2: Constraint Fulfillment*

Table V (for CS-1 Dronology and CS-2 Frequentis) and Table VI (for CS-3 ACME-RA) displays for each constraint type and per each process type how often a constraint was fulfilled at the end of the process, and how often the constraint remained violated (i.e., an engineer did not fix it). In contrast to RQ1, where we observed the amount of fulfilled processes (and whether engineers deviated during the processes' lifetime), here we obtain insights into which constraints are more likely to be violated and hence are the root cause for a process to not fulfill all QA constraints.

Dronology: The left-hand side of Table V reports the differences in how often a constraint was fulfilled (limited to completed process instances). A majority of the constraints were fulfilled most of the time (∼90% and higher). The lower fulfillment rates for constraints D-C1a and D-C1b ($<$55%) are the main reason "Task" and "Sub-task" processes exhibit low AlwaysOk and EventualOk values in Table III. Yet, constraints applied across multiple process types (i.e., D-C1a/b, D-C2, D-C3, D-C6, D-C7) exhibit similar fulfillment rates, i.e., a constraint is typically equally well fulfilled, respectively violated, regardless in which process type it is used.

Frequentis: For the second case study we could observe significantly higher fulfillment rates for all constraints (Table V right). The 12 unfulfilled constraint instances are dis-

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

TABLE III: Quality constraint evaluation results per process type (Dronology case study)

Frequentis	SubWP	%	ACME-RA	P1	%	P2	%	P3	%	P4	%
AlwaysOk	78	79.6		15	32.6	3	14.3	32	27.6	21	30.0
EventualOk	10	10.2		8	17.4	5	23.8	49	42.2	38	54.3
CompleteNotOk	10	10.2		23	50.0	13	61.9	35	30.2	11	15.7
IncompleteNotYetOk	0	0.0		0	0.0	0	0.0	1	33.3	1	9.1
IncompleteProgressedNotOk	0	0.0		0	0.0	0	0.0	0	0.0	10	90.9
IncompleteOk	11	100.0		0	0.0	0	0.0	2	66.7	0	0.0
Total	109			46		21		119		81	

TABLE IV: Quality constraint evaluation results per process type (Frequentis and ACME-RA case studies)

tributed across the 10 `CompleteNotOk` SubWPs described above. Compared to the previous case study, a constraint for Dronology typically requires the existence of a trace link to one artifact (e.g., D-C1a: a Task traces to a least one Design Definition), whereas 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 links are correct). To account for this, we further 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 superfluous trace links) only 3 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 a single SubWP where Test Cases were not applicable. The remaining 18 LLReqs violations were spread across the other eight `CompleteNotOk` SubWPs.

ACME-RA: In addition to the overall fulfillment rate, we also tracked whenever a step was completed (for the first time) but a constraint was still violated (χc).

We could observe that Constraint RA-C1 was the constraint that was violated most frequently upon process completion for three out of the four projects (P1, P2, P3). We further noticed that there was hardly any improvement, in terms of violations, during the process' lifetime. Once a step was completed and exhibited a constraint violation, this violation was hardly ever repaired at a later stage. For these particular constraints, this is not necessarily surprising as the role of an issues' assignee is determined by the skills (i.e., front end, back end, etc.) and thus unlikely to change even when reopened. Given the high intermediary and final fulfillment rate of P4, we hypothesize that the engineer at ACME-RA better recalled the most recent project (i.e., P4) compared to the older ones and that the applicability of a "Task" issue has changed over time.

In contrast, for the remaining constraints RA-C2, RA-C3, and RA-C4, we observe generally high final fulfillment across all four projects. They, nevertheless, differ in their intermediary fulfillment, with RA-C3 being almost always fulfilled in P1 to P3, i.e., tests were done by a QA engineer, reviews not being immediately done by a QA engineer (RA-C2), and not much attention is given to check whether all sub issues are indeed closed before closing the issue (RA-C4). As all issues are gradually closed over the project's duration, so will this constraint be eventually fulfilled. The comparatively lower final fulfillment in P4 (compared to P1 to P3) was found to be due to the team lead or the assigned developer carrying out the tests instead of a QA engineer.

RQ2 Key Observation: Quality assurance constraint violations are common but to a lesser extent in industrial safety-critical environments, where QA constraints are mandated by regulation. In the open source CS-1 violations are mostly caused by forgotten or incorrect traces, while in the industrial safety-critical CS-2 violations are mostly caused by process edge cases. Constraint violations in CS-3 are primarily caused by the issue's assignee not having the expected engineering role.

C. *RQ3: Process Flexibility*

ACME-RA's process offers more flexibility to select a step compared to the processes from Dronology and Frequentis. Table VIII shows the number of steps that were started (i.e., became `ACTIVE`, are `COMPLETED`) and percentage of these steps being (temporarily) canceled, reactivated (i.e., having reached a completion state and then becoming active again), revoked (i.e., being active and then having the preconditions violated), and, finally, being prematurely started (i.e., having the predecessor step(s) incomplete). Note that we analyzed only completed process instances. Entries marked with a dash indicate that no process progress rules are in place, for example, to allow cancellation. Whether such rules can be in place depends on the underlying artifact details, here in particular the Jira issues states and their transitions. Hence, while only steps "Development" and "Testing" can be canceled, all steps can be reactivated, i.e., be marked completed but then experience additional engineering activities that violate the step's postconditions.

For ACME-RE we observed that engineers, for "Task" issues, made heavy use of the flexibility offered by the Jira state transitions. This flexibility is consequently also supported by the ProCon framework. We see every step that can be canceled was indeed temporarily canceled in every project (except for "Testing" in P1). Unsurprisingly, there is some correlation with the fulfillment of constraints. P1 and P2 see more cases of a non-developer assigned to the development step, while P3 and P4 fulfill this QA constraint more often, hence the latter two projects see fewer cases of premature starting the "Development" step. Premature execution of the "Closing" step is particularly high in P2, which is explained by the lack of either (No)Reviewing or (No)Testing steps being carried out (note the sum of (No)Reviewing and (No)Testing only amounting to 13 and 11, respectively, over a total of 21 process instances). Here, the engineers typically transitioned the "Task" issue into state "Closing" directly from "In Development". This is another case, where the valid transitions in Jira changed over time, but nevertheless, our framework was able to track this deviation. Inspecting the number of step revocations (i.e., implying repeated execution of the same step), we note that developing and testing are especially often repeated in P4. This could be an indicator that engineers aimed to provide more fine-granular feedback of their task progress by switching often between the respective Jira issues states. In contrast, reactivation primarily occurs for the "Preparation" step (across all projects), as this step doesn't come with preconditions that are invalidated (and hence no revocation is observable but rather only reactivation).

RQ3 Key Observation: ProCon is able to accurately describe the process progress even in the presence of frequent repetition, pausing, or skipping of engineering activities. This is especially relevant in highly iterative development processes (e.g., CS-3 P4) that frequently switch between development and testing phases.

D. RQ4: Performance of the ProCon Framework

Constraints related to certain Quality Check Documents are typically validated by engineers to ensure all QA demands for their step are fulfilled. Overall, the replay of 26,926 change events over 109 simultaneously active process instances from the Frequentis case study resulted in 18,241 Quality Check Document evaluations. The resulting replay of events from the Tool connector's cache including constraint evaluation took ∼6.5 minutes (averaged over 10 evaluation runs). This corresponds to ∼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.

Similarly, we evaluated the performance for the ACME-RA case study which exhibits simpler quality constraints, but more complex constraints for process progress tracking. Here we replayed almost 52,500 change events, out of which almost 10,500 were relevant for at least one process instance and thus resulted in a trigger of ProCon. On average, evaluating the 267 process instances took ∼2.1 minutes.

RQ4 Key Observation: With an average individual constraint evaluation duration of ∼0.02 seconds ProCon is able to quickly evaluate artifact changes and subsequently provide timely feedback on quality assurance constraints and process status even in the presence of many simultaneously active process instances. Rather, the main factor influencing timely feedback to developers is the rate at which artifact updates are made available to ProCon and how frequent developers visit the process dashboard.

E. RQ5: Ease of use

To evaluate how easily constraints can be created using ProCon, we conducted a preliminary user study to gain initial insights in the way ProCon is used and what challenges users face (c.f Section IX-A). Five out of six participants successfully encoded all six StepTransitionsRules. Results are more varied for the DataMapping rules (see Table VII). We found similar result diversity for QA constraint writing.

All participants stated that they found writing the constraints intuitive, with those not finishing them explicitly stating that they felt confident to complete them if given more time. Two of the six participants, however, found the DataMapping rules a bit confusing. One participant stating "*while the tasks themselves are easy, it is hard to enter the mindset.*" and another participant commenting "*having multiple rules for one step was confusing*". Specifically for the DataMapping rules, we hypothesize that they are more difficult to grasp as these constitute not true/false evaluations but require adding and removing of artifacts from the step's output.

During the experiment, the participants made frequent use of the test classes for inspecting the syntactic and semantic correctness of their rules. Overall, we believe these are very promising observations as the participants were able to specify a significant portion of a realistic process and its associated artifacts. Hence we conclude that ease of use is sufficiently high as to not impede framework uptake. We need to highlight, however, that in the field, process engineers overall would take more time as they would also write tests, generate test data, and replay artifact changes to obtain more confidence in the rules' correctness. At the same time, engineers would develop proficiency over time, while the participants had only 15 minutes of training and still did very well.

In addition to this initial study, we introduced ProCon to QA engineers at Frequentis during an internal company innovation

Dronology	Task (n=135)			Sub-task (n=176)			Bug (n=95)			Design Def. (n=162)			Hazard (n=36)			FRQ	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 V: Final quality constraint evaluation results (fulfilled ✓ and unfulfilled χ) per constraint type from completed process instances for CS-1 Dronology constraints (left) and CS-2 Frequentis constraints (right).

ACME-RA	P1 (n=46)				P2 (n=21)				P3 (n=119)				P4 (n=81)			
	✓	χc	χ	✓%	✓	χc	χ	✓%	✓	χc	χ	✓%	✓	χc	χ	✓%
RA-C1	24	22	22	52.2	8	13	13	38.1	82	36	34	70.7	69	2	1	98.6
RA-C2	1	1	0	100.0	1	2	1	50.0	18	18	0	100.0	5	6	1	83.3
RA-C3	37	1	1	97.4	7	0	0	100.0	49	1	1	98.0	5	4	4	55.6
RA-C4	46	22	0	100.0	21	13	0	100.0	115	47	1	99.1	65	42	5	92.9

TABLE VI: Quality constraint evaluation results per constraint type from all completed CS-3 ACME-RA process instances: differentiating between QA constraint violation upon first step completion (χc), unfulfilled at process end (χ), and fulfilled (✓) at process end.

	P1	P2	P3	P4	P5	P6
Transition Rules (6)	6	6	6	3	6	6
DataMapping Rules (7)	7	3	0.5	4	4	4
QA Rules (6)	6	2	2	5	0	6

TABLE VII: Process definition success for six participants for six Transition rules, seven DataMapping rules, and six QA Rules.

event, by providing them with an initial prototype for writing QA constraints. Our aim was to showcase ProCon's potential and adaptability and gather feedback from process engineers. After some initial training, they were able to successfully create additional constraints for their case study, based on the existing set of rules without the direct involvement of any of the authors. We received positive feedback from the three engineers (from the QA department, and one development team) encoding the rules and presenting the results at the end regarding the usability of ProCon itself and how constraints can be created and modified. This feedback provided the foundation for the prototype currently being rolled out for friendly user testing in three additional development teams.

RQ5 Key Observation: the preliminary results hint at an easy learning curve as beginners were able to write correct rules/constraints within an hour. Beginners made the least mistakes when writing step Transition rules, and found writing DataMapping rules the hardest. QA engineers at our industry partner found adapting existing constraints to new processes without any external support doable in a timely manner.

XI. Discussion

The analysis of the data collected from the first case study, Dronology, indicated that the actual process – in some cases significantly – deviated from the planned one. Upon requesting feedback, a project lead at Dronology explained that while guidelines and a development process were in place, it was not always feasible 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 introduce additional 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 [71], [72] could further assist in establishing the trace itself.

In contrast, the analysis of Frequentis' SubWPs confirmed that engineers do in fact follow the stringent quality standards one would expect 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 of 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

	P1 (n=46)					P2 (n=21)					P3 (n=116)					P4 (n=70)				
Step	Strt	Canc	Rea	Rev	Pre	Strt	Canc	Rea	Rev	Pre	Strt	Canc	Rea	Rev	Pre	Strt	Canc	Rea	Rev	Pre
Prep.	46	-	10.9	-	-	21	-	19.0	-	-	116	-	27.6	-	-	70	-	42.9	-	-
Dev.	46	19.6	4.3	13.0	47.8	21	4.8	0.0	33.3	61.9	116	16.4	2.6	30.2	31.0	69	31.9	2.9	44.9	2.9
Rev.	1	-	0.0	-	0.0	2	-	0.0	-	0.0	18	-	0.0	-	0.0	6	-	0.0	-	0.0
NoR.	45	-	0.0	4.4	-	11	-	0.0	9.1	-	98	-	0.0	10.2	-	64	-	0.0	12.5	-
Test.	38	0.0	0.0	5.3	-	7	42.9	0.0	0.0	-	50	36.0	0.0	6.0	-	9	33.3	0.0	22.2	-
NoT.	8	-	0.0	-	-	4	-	0.0	-	-	66	-	0.0	-	-	61	-	0.0	-	-
Clo.	46	-	2.2	-	2.2	21	-	0.0	-	47.6	116	-	0.9	-	0.9	70	-	7.1	-	7.1

TABLE VIII: Process flexibility for completed CS-3 ACME-RA process instances: **Start**ed step instances, thereof percentage **Canc**eled, **Rea**ctivated, **Rev**oked, and **Pre**maturely started. A dash entry indicates that no corresponding process progress tracking rules exists that could give rise to this phenomenon.

RQ	Summary
RQ1 CS-1	In this open source, safety-critical setting, the more restriction-rich Task and Sub-task processes exhibit a high amount of deviation that remain also when the process is considered finished.
RQ1 CS-2	In this industrial, safety-critical setting, there are only some cases of process deviation with most deviations repaired upon completing the process.
RQ1 CS-3	In this industrial, non-safety-critical setting, processes are deviated from regularly, but not to the same extent as the open source setting in CS1.
RQ2 CS-1	Out of 14 combinations of process types and QA constraints, only four exhibit no violation ever. Some QA constraints are violated in more than half of all process instances.
RQ2 CS-2	Half of all QA constraints are violated with the remaining QA constraints only violated in a few cases (less than 5%)
RQ2 CS-3	Projects appear to inconsistently apply QA constraints as QA violation extent fluctuates significantly among projects.
RQ3 CS-3	ProCon is capable of supporting premature and repeated step execution in the presence of exclusive step choices.
RQ4 CS-2	QA constraints were rapidly evaluated in the presence of complicated QA constraints and linear processes.
RQ4 CS-3	Process constraints were rapidly evaluated in the presence of simple QA constraints and complicated processes.
RQ5 CS-5	Beginners were able to write basic process and QA constraints in a short time.

TABLE IX: Research Questions Summary per applicable case study.

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 explains why manually providing timely feedback is infeasible.

Finally, for the third case study, ACME-RA's software product has no stringent safety implications, and no external regulation that enforces the use of strict quality standards. The use of prescribed engineering processes is motivated mainly internally to obtain an accurate picture of the overall development progress within the individual projects. We observed simpler constraints, and lower degrees of following these constraints. Here ProCon is helpful in obtaining a true picture of the progress, e.g., by highlighting those "Task" processes that are officially closed, but still have child tasks that are not closed yet, hence indicating that there is still work to be done. Tracking processes and constraints violations also supports engineers in inspecting how often certain steps are indeed found to be applicable, and when executing these occasional steps, whether they are indeed executed as planned (i.e., are quality constraints indeed fulfilled). Over the course of time, process engineers may then learn whether the fulfillment rate of QA increases, and the rate of unfixed QA violations goes down. For the ACME-RA case study, our ProCon also demonstrates the ability to track repetition and deviation of process steps and the frequent encounter of such behavior, a strong signal that temporary deviations are pervasive in software engineering efforts.

A. *Implications for Practitioners and Researchers*

Based on the observations made from the case studies, we can conclude that ProCon can have significant practical implications for QA engineers. Supported by automated checks for "standard" cases, they can shift attention and focus 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 [73]. 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 their own (error-prone) custom "helper tools" that are almost infeasible to maintain or to reuse across multiple projects or teams.

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. While the prototype was applied only to one 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 need 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.

Beyond the immediate practical applicability, we would expect that our approach leads to cost savings by reusing constraints due to treating constraints and their evaluation results as first-class citizens (which also reduces maintenance costs). 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. Further, the concepts of software product line engineering could be used to manage the variations in process and QA concerns found in a larger organization [74]. We would also expect cost savings by reusing constraints across different systems subject to the same regulation. As a regulatory standard is applicable for a wider range of systems, constraints (and also processes) could be formalized at the level of the standard for reuse by affected companies. We are however aware, that initially reuse would occur primarily within an organization for two main reasons. First, constraints are rather tool-specific, especially how an organization makes use of their tools' extensibility through custom fields and custom link types. (We could imagine a mapping from a more generic/high-level standard-centric constraint to an organization's low-level tool-specific constraint; the feasibility of this is largely unclear). Second, from our discussions with various regulated companies, we understand that many see their precise processes and constraints for implementing a regulatory standard as confidential. We, hence, expect reuse at the level of a regulatory standard to find traction once processes and/or constraints are no longer considered a competitive advantage but rather an opportunity for jointly reducing development costs.

With respect to implication for researchers, ProCon has further potential to serve as a platform for additional research prototypes and support tools built on top of if. 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 set up the correct output artifacts, or direct the engineer in how to fix a constraint violation or offers to automatically fix it [11].

Our approach can serve as the basis for other research on supporting the software engineering process such as proactively driving the process through automated actions. Here, open points for investigation include how the process context can be used to better drive CI/CD pipelines, and automatically prepare engineering artifacts and engineering activities such as reviews. In general, the questions that emerge from automation also need to focus on the negative sides, such as engineers trusting too much in tool support or choosing guidance actions that are the most convenient for them but perhaps not optimal for the overall development process.

XII. Threats to Validity

Internal Validity. We address researcher bias by modeling process and constraints from an open-source system and two companies 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. [75] 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, as well as medium and more complex ones from industry) in a timely manner. We analyzed data from these three sources with two being "production data" from an industrial safety-critical system and an industrial non-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-source systems rarely come with such extensive explicit artifacts and trace information.

Construct Validity. For RQ1 and RQ2 we addressed the question of how frequently process deviations occur and which specific constraints are violated, by replaying real historical data. Hence, we stepped through the process as it occurred with exactly the same sequence of changes and evaluated the process and QA constraints. We thus measure the "official" state of the process as it would be used as evidence to demonstrate compliance with regulations. We cannot, in this way, measure the tacit process state implicit in the minds of engineers. Engineers might have used informal communication channels to convey status information while forgetting/delaying the update the indented process status signals (e.g., Jira issues status or checkboxes). We thus might generate a more pessimistic view of the process state. As the purpose of ProCon is to provide guidance to engineers and evidence for compliance, however, we believe it is important to minimize the gap between measured and tacit process states by highlighting deviations.

For RQ3, for assessing whether ProCon support flexible processes, we measured if for a process with alternative steps and engineers frequently repeating steps, we indeed find the reactivation, cancellation, and revocation of steps as indicated in the replayed artifacts' history.

For RQ4, we measured only the performance of the core process and QA constraints evaluation as this aspect is computationally expensive. Poll frequency for obtaining artifact updates might have an effect on the responsiveness as rare polling with the subsequent potentially large amount of updates could lead to longer constraint evaluation times. This aspect is also determined by the network load, tool load, and artifact update frequency which is different in each deployment scenario and hence needs to be assessed on a case-to-case basis.

For RQ5 we conducted a controlled experiment to evaluate ease of defining realistic constraints. Our aim was to obtain insights into whether the initial learning curve is sufficiently low to promote uptake by practitioners. Long-term use in a production environment needs to be separately investigated as actual constraint complexity might vary in practice. To mitigate any threats we provided information to participants about the concepts we were investigating, communicated the purpose of the study to our participants, carefully discussed the study setup and execution among multiple researchers, and conducted a pilot study.

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 and ACME-RA that followed a completely different process and TIM.

Adopting a different scenario then is mostly a matter of connecting different tools. Contemporary tools tend to come with a HTTP/REST interface, or client implementation (as did Jira and Jama with dedicated 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, team leads, 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 signaling 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.

Note, that overall, we cannot make any claims on the completeness of our approach as our research was primarily guided by the needs of our industry collaborators and the attempt to avoid investigating irrelevant aspects, the equivalent of the software development pitfall YAGNI: "you ain't going to need it". As we continue to apply our approach and framework to additional scenarios, we expect to identify missing aspects, especially along the lines outlined here [11]. However, we want to be able to validate the solutions to those aspects under realistic settings which, in the context of this work, typically requires an industry evaluation partner.

XIII. Conclusions and Outlook

In this paper, 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, which allows engineers to deviate from the process when necessary, whilst informing them which constraints are yet unfulfilled and which steps are already complete. Our framework achieves this flexibility by merely observing the engineers' actions in their tools rather than restricting them in their allowed activities. Constraints get constantly reevaluated upon changes even for steps that should not be worked on yet or which have been already marked as complete. Our evaluation using both, industry and OSS projects, revealed that engineers frequently deviated from the intended process for some time and that our approach can identify missing (traceability) information required by regulations.

Future work focuses on two main aspects. First, we intend to study the effect of having our prototype in use by engineers at Frequentis. This will include design, prototyping, and evaluation of advanced features, such as actionable guidance suggestions on how to return to the prescribed process upon deviation and change impact notification across steps. 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. This will include experimenting with constraint code completion techniques, automatically creating premature start conditions, and constraint deadlock checking techniques. Ultimately, we aim to quantify the cost savings by comparing the effort to specify and maintain processes to the benefits of easier and less error-prone collection of QA evidence.

XIV. Data Availability

The prototype and data used in this paper is available at Figshare https://doi.org/10.6084/m9.figshare.12840053.

Acknowledgment

This work was funded by the Austrian Science Fund (FWF) under the grant numbers P31989, P29415-NBL, and P 34805-N, and by the state of Upper Austria through LIT-2019-8-SEE-118 and LIT-2019-7-INC-316 as well as the LIT Secure and Correct Systems Lab. This work has been also supported by the FFG, Contract No. 881844: "$Pro^2Future$". The Dronology case study was supported by the United States National Science Foundation under grants SHF:1741781 and CPS:1931962.

References

[1] D. Galin, *Software quality assurance: from theory to implementation*. Pearson education, 2004.

[2] 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.

[3] 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.

[4] "Code of Federal Regulations - Title 21 - Part 820 Quality System Regulation," https://www.ecfr.gov/current/title-21/chapter-I/subchapter-H/part-820?toc=1, accessed: 2021-12-01.

[5] M. McHugh, F. McCaffery, and V. Casey, "Adopting agile practices when developing software for use in the medical domain," *Journal of Software: Evolution and Process*, vol. 26, no. 5, pp. 504–512, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1608

[6] B. Brosgol and C. Comar, "Do-178c: A new standard for software safety certification," ADA CORE TECHNOLOGIES NEW YORK NY, Tech. Rep., 2010.

[7] J. A. Jiménez, J. A. M. Merodio, and L. F. Sanz, "Checklists for compliance to do-178c and do-278a standards," *Computer Standards & Interfaces*, vol. 52, pp. 41–50, 2017.

[8] G. Macher, A. Much, A. Riel, R. Messnarz, and C. Kreiner, "Automotive spice, safety and cybersecurity integration," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2017, pp. 273–285.

[9] 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.

[10] S. Maro, J.-P. Steghöfer, and M. Staron, "Software traceability in the automotive domain: Challenges and solutions," *Journal of Systems and Software*, vol. 141, pp. 85–110, 2018.

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

[12] C. Mayr-Dorn, M. Vierhauser, S. Bichler, F. Keplinger, J. Cleland-Huang, A. Egyed, and T. Mehofer, "Supporting quality assurance with automated process-centric quality constraints checking," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 1298–1310. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00118

[13] 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.

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

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

[16] 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.

[17] 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.

[18] R. Balzer, "Tolerating inconsistency," in *Proc. of the 13th Int'l Conf. on Software Engineering*, 1991, pp. 158–165.

[19] 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

[20] W. W. Royce, "Managing the development of large software systems: concepts and techniques," in *Proceedings of the 9th international conference on Software Engineering*, 1987, pp. 328–338.

[21] 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

[22] 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.

[23] 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.

[24] 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.

[25] 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

[26] 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

[27] 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.

[28] 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.

[29] 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.

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

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] X. Zhao and L. J. Osterweil, "An approach to modeling and supporting the rework process in refactoring," in *Proc. of the 2012 Int'l Conf. on Software and System Process*, 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

[39] 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.

[40] 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

[41] 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.

[42] 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.

[43] 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.

[44] S. Morimoto, "A survey of formal verification for business process modeling," in *International Conference on Computational Science (ICCS)*. Kraków, Poland: Springer, 2008, pp. 514–522.

[45] F. M. Maggi, M. Montali, M. Westergaard, and W. M. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *International Conference on Business Process Management (BPM)*. Clermont-Ferrand, France: Springer, 2011, pp. 132–147.

[46] F. M. Maggi, C. Di Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *International Conference on Advanced Information Systems Engineering (CAiSE)*. Thessaloniki, Greece: Springer, 2014, pp. 457–472.

[47] A. Kumar, W. Yao, and C.-H. Chu, "Flexible process compliance with semantic constraints using mixed-integer programming," *INFORMS Journal on Computing*, vol. 25, no. 3, pp. 543–559, 2013.

[48] 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

[49] 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.

[50] 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

[51] 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

[52] 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

[53] 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

[54] 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

[55] 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.

[56] 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

[57] 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.

[58] 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

[59] 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.

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

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

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

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

[64] C. Mayr-Dorn, M. Vierhauser, F. Keplinger, S. Bichler, and A. Egyed, "Timetracer: a tool for back in time traceability replaying," in *Proc. of the 42nd Int'l Conf. on Software Engineering, Companion Volume*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 33–36. [Online]. Available: https://doi.org/10.1145/3377812.3382141

[65] S. A. White, "Introduction to bpmn," *Ibm Cooperation*, vol. 2, no. 0, p. 0, 2004.

[66] 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.

[67] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *Computer*, vol. 40, no. 6, pp. 27–35, 2007.

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

[69] Verein zur Weiterentwicklung des V-Modell XT, "Das deutsche referenzmodell für systementwicklungsprojekte," http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.3/Dokumentation/V-Modell-XT-HTML/index.html, 2021, [Last accessed 01-12-2021].

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

[71] 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.

[72] 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.

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

[74] 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.

[75] 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.