2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 978-1-6654-1219-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSE-Companion52605.2021.00033

Guiding engineers with the Passive Process Engine Environment

Christoph Mayr-Dorn, Stefan Bichler, Felix Keplinger, Alexander Egyed
Johannes Kepler University
Linz, Austria
firstname.lastname@jku.at

***Abstract*—Research as early as the 90s identified rigid, active process enactment as detrimental to engineers' flexibility. While software engineering processes thus are rarely "executable", engineers would benefit from guidance in safety critical domains where standards, regulations, and processes are often complicated. In this paper, we present the Passive Process Engine Environment (*P2E2*) that tracks process progress in the background and automatically evaluates quality assurance constraints even in the presence of process deviations. Our approach is engineering artifact agnostic and comes with two exemplary tool connectors to Jira and Jama. Video at: https://youtu.be/kXwU_baVWoQ**

***Index Terms*—software engineering process, deviation, developer guidance, constraints, monitoring**

I. Introduction

Software engineering processes together with software quality assurance (QA) aim at providing the necessary level of software quality. Research as early as the 90s identified rigid, active process enactment as detrimental to engineers' flexibility. We observed this need for flexibility during informal studies at our industry partners, which revealed that engineers deviate temporarily from the intended process. Indeed, the current practice in industry is using semi-formal descriptions to specify processes [1], rather than rigidly enforced processes. As a result, software engineering processes are rarely "executable", meaning that engineers have little to no automated support for checking whether their work is in line with the process and whether their work is affected by deviations of others. Given the often complicated standards, regulations, and processes in safety critical domains, engineers quickly feel overwhelmed and stressed about potentially deviating too much or correcting too late – resulting in costly rework and/or delays. Deviations come in different forms: forgetting to set a trace between requirement document and design document as mandated by QA, starting the implementation of non-reviewed requirements, skipping a mandatory step, etc.

In this paper, we present our **P**assive **P**rocess **E**ngine **E**nvironment (*P2E2*) that allows to track the engineering process' progress in the background purely based on artifact changes (where artifacts include requirements, models, code, test cases, issues, work packages, etc.). Engineers need not interact with the *P2E2* except for obtaining feedback whether they should start with their work, respectively whether their work fulfills given quality constraints. The *P2E2* combines two key novelties: first, it treats (quality) constraints neither as an implicit part of the engineering process model nor as completely disjunct from it. Instead, (quality) constraint evaluations are first class citizens: i.e., as explicit development artifacts that may also determine process progress. Second, the *P2E2* continues to track progress even as engineers deviate from the process.

We evaluated *P2E2* with Dronology [2], an open source system for unmanned aerial vehicles (UAVs) and an industrial air traffic control system (ATC) to investigate to what extent QA violations occurred [3].

II. Usage Scenarios

The *P2E2*'s deviation tolerant process tracking capabilities are useful for various, complementary purposes. In the following, we describe three scenarios where *P2E2* provides guidance to engineers.

1 - Process Guidance: *P2E2*'s main purpose is to raise process progress awareness amongst engineers by providing them with information on step progress and quality constraint fulfillment on demand. Engineers typically use a multitude of tools and diverse artifacts for their work, spreading implicit process information across these. Obtaining an accurate picture of which artifacts are available at what maturity requires significant coordination effort among engineers and/or querying of different tools. On top of that, an engineer needs to be aware of the precise process definition and applicable QA constraints. Providing an integrated view on the process in a tool that the engineer is free to use, but not forced to conduct work in, has large potential to reduce coordination overhead and rework.

2 - Automating QA: Introducing a process guidance system for a complete process at a fine granular level is a non-trivial and work-intensive endeavour. The *P2E2* enables an incremental roll-out by focusing on the quickly achievable benefits, such as automating tedious QA tasks, first. Rather than modeling each step, QA engineers formulate which quality constraints should be fulfilled at which stages in a process. This progress-aware checking allows to ensure trace links mandated by regulations and standards are checked as early as possible rather than at the process end. For example, consistency of trace links between low-level and high-level requirements should be checked early, while trace links from low-level requirements to test cases are not available until much later in the process. Engineers thus receive timely feedback whether they are truly done with their work by consulting the *P2E2* Process Dashboard to ensure they haven't missed

978-1-6654-1219-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSE-Companion52605.2021.00033

anything (rather than being interrupted by QA engineers at a later time when they have moved on to another task).

3 - Research foundation: the current *P2E2* capabilities don't yet include sophisticated deviation analysis and repair aspects [4]. Hence, engineers need to coordinate with their coworkers and based on the process state to what extent a deviation affects their work and how (respectively when) to fix a deviation. The *P2E2* , however, provides the technical research foundation for building such capabilities. In addition, the events emitted by the *P2E2* provide all the vital details required for process mining as artifact changes are readily associated with process instances and process steps. Process mining algorithms can then provide insights into typical task duration, deviations, and their repairs, which in turn may be applied to improve the engineering process and guidance.

III. Related Work

Research in the 90s resulted in a number of approaches in support of Process-centric software development environments (PCSDE). Step-centric modeling and active execution frameworks [5]–[8] 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 are not well supported. In contrast, systems utilizing event-condition-action (ECA) rules or pre- and post-conditions [9], [10] provide freedom of action to the engineer but offer limited guidance.

More recent work focuses on specific aspects in the engineering life-cycle rather than general purpose processes. DevOpsML [11] aims at reducing the effort to describe continuous integration and deployment processes. Amalfitano et al. [12] aim to fully automate the execution of the testing process and to automatically generate appropriate traceability links. Similarly, Hebig et al. [13] 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 [14]. At a micro-level, Zhao et al. propose Little-JIL to help developers track artifact dependencies during rework [15].

A few approaches on general purpose process modeling and execution (e.g., [16]–[18]) focus on step-centric languages such as SPEM and BPMN, which both imply active execution where engineers cannot deviate from the prescribed process.

IV. Process Model

The two main elements of the process model are *Steps* and *Decision Nodes* attached to them (Figure 1 provides a simplified UML class diagram of the process model's main elements). A **Step** describes what an engineer *"should"* do. For example, refine a requirement, implement a feature, define a test case. The challenge when passively executing processes is determining the steps that are currently available for engineers to work on, steps that represent work in progress (but perhaps shouldn't be worked on yet), and finally, steps that have been (successfully) completed. A passive process specification, therefore, consists primarily of the corresponding transitions conditions in the form of ECA rules. These define which event(s) from the engineering environment (e.g., an artifact update), given additional constraints (i.e., the condition) trigger the inclusion of an artifact to a step's output artifact set (i.e., the action part of the rule) as well as whether the step is now ready to be worked on, in progress, aborted, or completed. The artifacts used in the ECA rules are defined by the step.

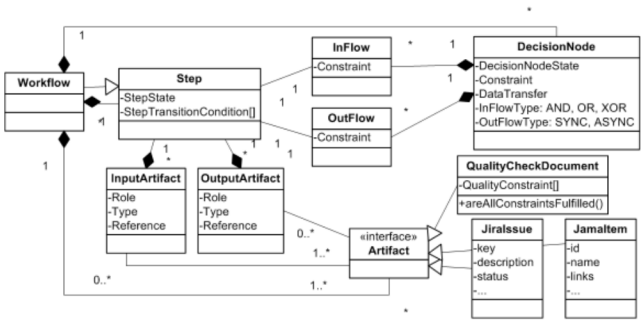

Fig. 1: Process specification meta model: UML class diagram. JiraIssue and JamaItem are exemplary integrated artifact types.

A step has zero or more *Input* artifacts attached that represent required data to make a decision or artifacts that need to be modified. It further has zero or more *Output* artifacts that describe the effect of having executed the step (e.g., having modified an input artifact or created a new artifact). Input and output artifacts can represent any kind of information such as requirements, tests, issues, or trace links.

A **Decision Node** describes how the completion of one or more *Steps* – and additional conditions, again defined as ECA rules – leads to the execution of subsequent steps. The set of decision nodes thus defines the process' control flow. A decision node's *DataTransfer* declaration describes how the output of one step becomes the input of a subsequent step, thereby defining the process' data flow. The specific activation conditions are placed on the *InFlow*s (from preceding steps to a decision node), on the node itself, and on the *OutFlow*s (from decision node to subsequent steps).

V. Tool Architecture

The *P2E2* consists of following main components (depicted in Figure 2): Process Dashboard, Process Definition Registry, Tool Connectors, Passive Process Engine, and a Rule Engine.

The process dashboard provides the primary interaction means for stakeholders to register **Passively Executable Process Specifications** (A), instantiate them (B), and inspect the current status (i.e., inspecting process progress and QA constraint evaluation results) (H).

A Passively Executable Process Specification consists of a two parts: (i) a description of the high-level process structure and data-flow and (ii) a set of rules that specify step progress triggers, process control-flow conditions, artifact fetching, and QA constraints.

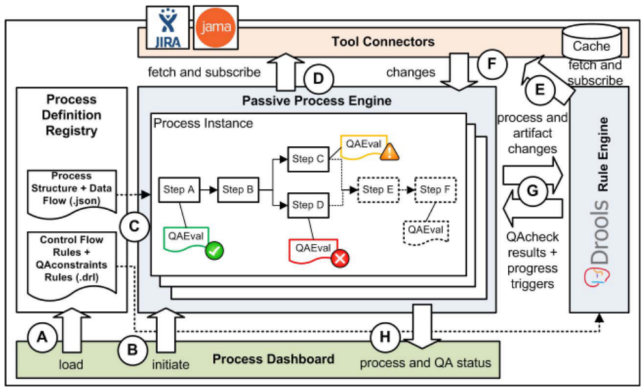

Fig. 2: *P2E2* main architecture components.

A **Process Instance** is created by selecting a process specification and providing the ids of the required input artifacts (C). The **Passive process engine** then fetches the artifacts via the **Tool Connectors**. At the same time it registers for any artifact updates (D). Similarly, the rule engine fetches artifacts (E) when traversing trace links or relations among artifacts.

Tool Connectors obtain artifacts from their origin services, e.g., a Jira issue from a Jira server, a requirement from IBM Doors, or commit information from a Git repository, typically via the services' REST API. A Tool Connector takes on sophisticated tasks beyond obtaining artifact updates (via polling or subscriptions). It manages which artifacts are relevant for a process and thus need to be monitored, and keeps a cache of these artifacts up to date.

When the Passive Process Engine receives an artifact update event (F), it looks up which rules in the **Rule Engine** component rely on that artifact and triggers them (G). Depending on condition evaluation outcome, the Rule Engine signals process progress events back to the Passive Process Engine. The Passive Process Engine subsequently uses these events to update the process state and determine which steps to activate, which ones to deactivate, and which artifact (references) to copy from the output of one step to the input of a subsequent step. These changes in turn are fed back into the rule engine (E) potentially triggering further progress.

A. Implementation Details and Usage

The *P2E2* prototype is implemented in Java on top of the Axon framework and is available on Github [1] with a demonstration video available on youtube.[2] The Axon framework implements the event-sourcing pattern, thus any change in the process and any artifact change is captured as an event. These events establish an audit trail how the process came about and allow replaying the process event by event for inspection at which time a quality constraint was not (yet) fulfilled.

Process structure and data flow are specified in a json format (directly loaded into the passive process engine) while control flow conditions and QA constraint are written as Drools rules (.drl files) – see dashed arrows in Figure 2. The Drools rule engine [19], a Business Rules Management System, can be easily integrated into a Java application and allows easy access to Java objects (representations of e.g., Jira artifacts) within rules written in a Java dialect. Currently, the control flow and QAconstraint rules come with some boilerplate overhead. Writing new rules requires copying and slightly adjusting the boilerplate code. A blockly-based web editor (briefly shown in the video) is work in progress from which currently the json process structure, data flow mapping, and later rule boilerplate code is automatically generated.

We provide tool connectors for Jama and Jira (via their REST interface) as an example how to access artifacts, cache them in a CouchDB (json based NoSQL database), and periodically poll for updates. The passive process engine and the rule engine provide the process id to the tool connectors when fetching an artifact. Hence the connectors send artifact updates only to the relevant process instances. Other event sources such as the developer's IDE are currently not supported.

The Process Dashboard makes use of the Vaadin framework for automated pushing of updates to the user's web browser. Figure 3 (middle) shows details of an example simple process and its tasks, providing details on which QA constraint where last evaluated (right), their evaluation result (completely/partially/not fulfilled or not evaluated yet), provides the ability to inspect input and output artifacts, and the option to request an explicit re-evaluation of QA constraints. The dashboard also allows uploading of new process definitions (left), and initiating new process instances.

VI. Prototype Evaluation

We used our TimeTracer tool [20] to revert artifacts and their trace links to their initial state, i.e., the start of the process. TimeTracer then provides artifact changes, step-by-step, thus simulating (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.

Execution of 802 open source processes instances (issues in Dronology involving eight QA constraints) and 109 industrial processes instances (also involving eight QA constraints) with *P2E2* allowed us to inspect whether QA constraints are fulfilled immediately upon a task's completion or rather at the end of the process. The results for Dronology showed that only 39% of process instances followed the defined process, compared to 80% in the industrial setting. These numbers hint at the potential of *P2E2* to make following the process easier for engineers.

VII. Conclusions

P2E2 guides engineers through the development process by highlighting which steps are completed, which ones are available to be worked on, and whether quality assurance constraints have been fulfilled. Engineers need not leave their engineering tools and are not limited by *P2E2* in their freedom

[1] https://github.com/jku-isse/c4s.p2ep.icse2021demo
[2] https://youtu.be/kXwU_baVWoQ

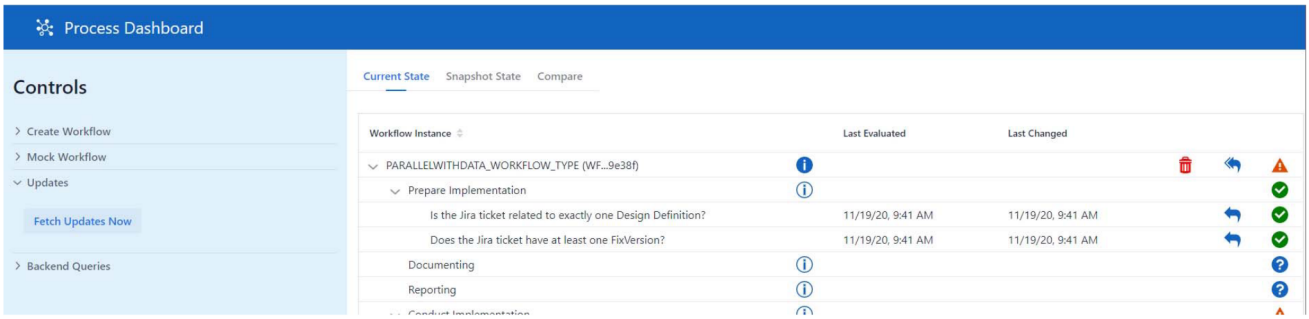

Fig. 3: *P2E2* - Process Dashboard screenshot excerpt.

to deviate when they see fit as the environment tracks progress in the background. The prototype is currently being rolled out at out industrial partner for evaluation in three separate friendly user groups. We thus expect to extensively report on developer acceptance, lessons learned, and best practices soon.

VIII. Data Availability

Data used in this paper are subject due to confidentiality agreements with out industry partner and therefore cannot be disclosed. The prototype, however, is archived on Figshare. [3]

Acknowledgment

The research reported in this paper has been funded by Austrian Science Fund (FWF) under the grant numbers P29415-NBL and P31989 as well as the Federal Ministry of Transport, Innovation and Technology, the Austrian Federal Ministry for Digital and Economic Affairs, and the Provinces of Upper Austria and Styria in the frame of the COMET Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG (K1-Centres Pro^{2}Future and SCCH).

References

[1] 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.

[2] 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.

[3] 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 International Conference on Software Engineering (ICSE 2021)*. IEEE / ACM, 2021, p. to appear.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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

[17] 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.

[18] 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.

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

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

[3] DOI: 10.6084/m9.figshare.13793513