

Visualizing Change in Agile, Safety-Critical Systems

Jane Cleland-Huang, *Member, IEEE*, Ankit Agrawal,
Michael Vierhauser and Christoph Mayr-Dorn

Abstract—Safety-critical software systems must be developed using rigorous safety assurance practices. This has led to the phenomenon referred to as the “big freeze” in which the cost, effort, and difficulty of introducing new functionality to an already certified product is prohibitively expensive. However, present day agile processes have greatly matured to the extent that organizations who have traditionally used waterfall approaches are actively experimenting with agile practices even across relatively high-criticality domains. At the same time, organizations experienced in agile development are increasingly building Cyber-Physical Systems (CPS), often without sufficient knowledge or instrumentation to adopt appropriate hazard analysis and safety assurance practices. The challenge in both scenarios is to manage, and more importantly understand change, and to further leverage automated software traceability to support the incremental development and maintenance of a safety case. This article explores solutions for visualizing and understanding change in highly-incremental, safety-critical development contexts.

Index Terms—Agility, Safety-Critical Systems, Visualization, Software Traceability

I. INTRODUCTION

SAFETY-CRITICAL software must deliver functionality while ensuring that the system is safe for its intended use (1). In many domains the software must be approved or certified prior to use. For example, software developed for the aerospace industry has to comply with ISO/IEC12207 and/or DO-178C guidelines, while medical devices must meet diverse international regulatory guidelines (2). Building such systems requires extensive hazard analysis and safety assurance processes that have led to the phenomenon referred to as the “big freeze” in which the cost, effort, and difficulty of introducing new functionality to a certified product is prohibitively expensive. As a result, there is significant industry interest in moving towards a more agile approach.

Many organizations working in safety-critical domains are experimenting with agile methods (3), while organizations experienced in agile development are increasingly building Cyber-Physical Systems (CPS), such as factory floor robots, Unmanned Aerial Systems (UAS), and medical devices, often without sufficient knowledge or instrumentation to adopt appropriate hazard analysis and safety assurance practices (4).

J. Cleland-Huang and A. Agrawal are with the Department of Computer Science and Engineering at the University of Notre Dame. E-mail: Jane-Huang@nd.edu

M. Vierhauser is with the Department of Business Informatics – Software Engineering at the Johannes Kepler University Linz, Austria

C. Mayr-Dorn is with the Institute for Software Systems Engineering at the Johannes Kepler University Linz, Austria

These two trends – emerging from opposite ends of the process spectrum – are pointing to a new way of developing safety-critical software which embraces the rigor of safety-critical development environments while experiencing the benefits of more incremental, faster delivery cycles made possible by agile solutions. The importance of using traceability to manage change in safety-critical software development has been broadly recognized (3; 5) but has not yet been adequately addressed within agile projects. The highly iterative environment of an agile project, makes it difficult to maintain accurate traceability, evaluate the impact of change on system safety and to accurately maintain a safety case (6).

We introduce innovative solutions for evaluating traceability of individual versions and for visualizing the impact of change across subsequent software releases. We provide examples from our own *DroneResponse* project, which falls into the lower-to-mid end of the safety-spectrum, but introduces clear safety concerns associated with the deployment of semi-autonomous UAVs in an urban setting. We present four traceability views which we deem particularly helpful for maintaining traceability and for supporting safety-related change impact analysis. All of these views are automatically generated by our *SAFA* (Software Artifact Forest Analysis) tool. *SAFA* is designed for use across a range of projects including traditional safety-critical projects which are adopting agile techniques, as well as agile projects which need to improve their safety assurance practices. *SAFA* does not prescribe any specific traceability structure or notation, but instead focuses purely on the visualization of evolving artifacts and the roles they play in mitigating hazards and demonstrating system safety. Furthermore, safety-critical software exists on a criticality spectrum, and our experiences with *SAFA* have been in domains of medical devices, Positive Train Control, and Defense, as well as lower criticality projects such as deployment of Unmanned Aerial Vehicles in urban areas.

II. MANAGING CHANGE IN AGILE PROJECTS

Safety-critical system design requires rigorous and systematic hazard analysis and safety assurance using techniques such as Software Fault Tree Analysis (FTA) or Software Failure Mode, Effects, and Criticality Analysis (FMECA) (1) to identify hazards and prepare countermeasures in the form of mitigating, safety-related requirements. Safety-critical systems have traditionally been developed using carefully controlled processes, such as waterfall, V, or W models or relatively heavyweight incremental processes (e.g., the Rational Unified

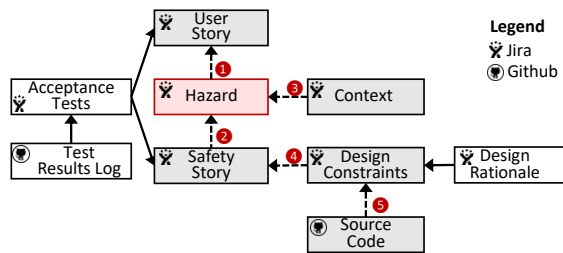


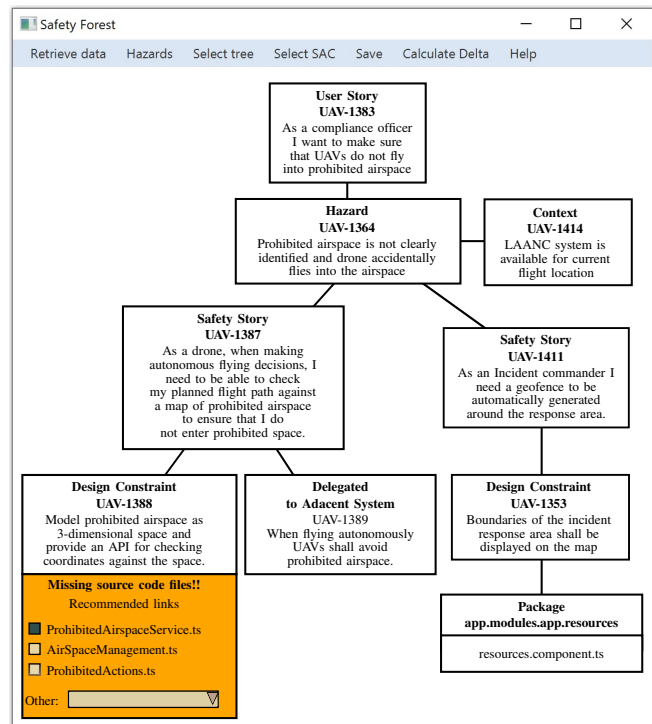
Fig. 1: A project’s TIM defines artifacts and traceability paths and can be customized for each development environment. The paths needed to generate the trace slice depicted in Fig. 2a are labeled in red.

Process) which emphasize detailed planning, upfront design, and phase-based quality assurance gateways. As the cost, effort, and difficulty of introducing new functionality to a certified product can be prohibitively expensive, many organizations are moving towards a more agile approach.

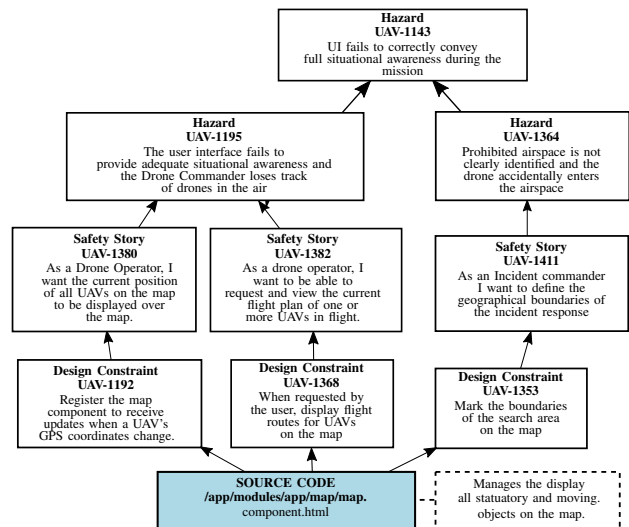
Several authors have proposed techniques for integrating agility with safety-critical software development in order to achieve compliance to diverse regulations (7). Kuhrmann et al. found that agile practices and methods, such as water-scrum-fall, are common in safety-critical contexts (8). *R-Scrum* emphasizes traceability of artifacts and continuous compliance to certification standards throughout the development process (3) and introduces the concept of “Living Traceability”. *Safe-Scrum* (5) separates safety-related activities by splitting the project backlog into two distinct parts containing *safety* and *functional* stories. It defines the traceability task as a separate activity within each sprint, but does not address the challenges of creating, evolving, and using links in the highly iterative, rapidly changing environment of an agile project.

All of these processes are characterized by incremental change and frequent safety evaluations. “Living Traceability” aims at establishing (semi-)automated tool support for creating, maintaining, and utilizing trace links. Existing traceability approaches, such as Capra¹ and OSLC² standards, enable modeling and integration of trace information across heterogeneous tools and artifacts; however, they do not specifically address the challenges of evolving trace links in a fast-paced agile environment. In practice, organizations implementing traceability solutions, often create a Traceability Information Model (TIM) to specify the key artifacts and their desired traceability paths, and then focus traceability efforts on demonstrating that each *hazard* has been mitigated (9; 10) and that all regulatory requirements have been met. Integrating traceability checks into the software process, by tracking the evolution (11) of trace links, aids project stakeholders to systematically establish links at appropriate times.

Automated solutions, based on techniques such as information retrieval, deep-learning, and software repository mining, can generate candidate trace links across requirements, design, tests, and other artifacts (12). While they are not sufficiently accurate to produce high-quality links in a fully automated way, they can be used to recommend links to users in order to help them to maintain relatively complete and accurate links



(a) An artifact tree is anchored around a hazard and depicts a trace slice from the hazard down to its mitigating artifacts.



(b) An inverted Source Code graph starts at a source code file and traverses up the artifact tree to all potentially impacted hazards.

Fig. 2: Artifact trees are generated automatically using our SAFA tool based on information in the project’s TIM

as the system evolves (13). The trace links that are created and evolved throughout the development process are essential for iteratively assessing change and its impact on system safety (14; 15). These links provide the foundation for our SAFA approach.

III. THE DRONERESPONSE PROJECT

We illustrate SAFA using examples from our *DroneResponse* project³. *DroneResponse* deploys small groups of semi-

¹<https://projects.eclipse.org/projects/modeling.capra>

²<https://open-services.net/>

³<http://DroneResponse.net>

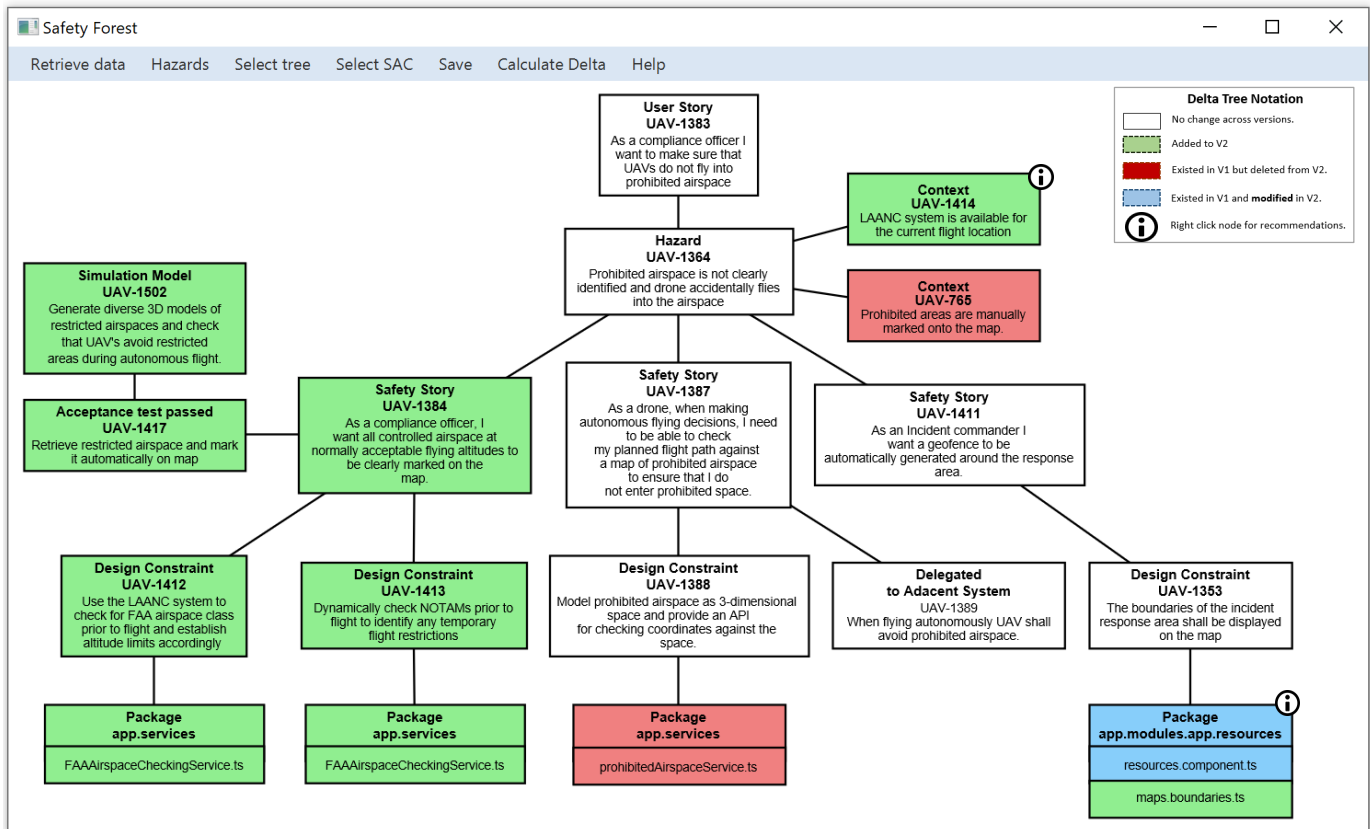


Fig. 3: A delta tree is fully generated using our SAFA tool to visually depict changes in the way a hazard is addressed across two versions. Artifacts are colored to show additions (green), deletions (red), and modifications (blue). Informational icons provide access to rationales and explanations of the underlying change.

autonomous UAVs to support emergency response scenarios such as search-and-rescue and fire reconnaissance. It is being developed in close collaboration with firefighters from the city of South Bend. DroneResponse is located at the medium to low end of the safety spectrum and therefore both the process, and the collection of artifact types and links, are appropriately lightweight. Later on we discuss how the same techniques can be used in projects with higher-levels of dependability requirements.

As we adopted a relatively agile approach, we specified DroneResponse’s requirements using a combination of user stories, safety stories, and design constraints. User stories were written from the perspective of human operators and UAVs in order to describe the basic functionality of the system. For example:

User Story (US1): “As a compliance officer I need to avoid UAVs flying into prohibited airspace.”

Next we performed a preliminary hazard analysis for each story to identify hazards that could prevent the realization of the story. An example hazard is:

Hazard (H1) Controlled airspace is not clearly identified and a UAV accidentally flies into prohibited airspace.

For each hazard, we then explored solutions and defined one or more *safety stories*. Each safety story describes a mitigation to the hazard from the perspective of an actor. Safety stories should not be confused with a safety-case but are likely

to contribute claims and evidence to a system-wide safety assurance case. Examples include:

Safety Story (SS1) “As a compliance officer, I want all controlled airspace to be clearly marked on the map.”

Safety Story (SS2) “As a UAV, I want to check my flight paths dynamically so that I can avoid flying into prohibited airspace.”

Finally, for each safety story, we identified and specified design constraints which provide a more formal specification of the system. An example is as follows:

Design Constraint (DC1) The FlightPlanner shall check the current FAA airspace class prior to flight and establish altitude limits according to local airspace restrictions.

Our intent was to create a light-weight agile environment whilst providing sufficient design details to clearly specify how hazards would be mitigated. We added the more formal layer of design-definitions between stories and code in order to track and validate that each safety story had been fully addressed.

IV. USING TRACEABILITY TO VISUALIZE CHANGE

SAFA leverages underlying trace links to visualize and explore the impact of change upon system safety. We introduce four visualization techniques that we deem helpful for evaluating trace links and understanding change. We show how carefully structured arrangements of artifacts, augmented by

the use of color and textual annotations, can provide critical safety insights.

A. Visualizing Artifact Trees

Top-down trace slices allow human analysts to assess the hazard mitigation plan including associated design solutions (10), test cases, and supporting trace links. Our first visualization, depicted in Fig 2a, displays a trace slice for a subset of artifacts within a single version of the system. While visualizing trace slices is certainly not a new concept; it provides the foundation for the novel cross-version visualizations presented in this paper. The trace slice starts with an anchor node, such as a hazard. The anchor node is not necessarily the root node of the trace slice, but it represents the entity of interest in the analysis. In our case, we use hazards as anchor nodes, and then trace them backwards up the artifact tree to their associated user stories, and down the tree to see how they are mitigated in the system.

Trace link traversal is guided by the rules embedded into the project’s TIM. For example, the links used to generate our hazard slice are colored red in Fig. 1. By following these rules, SAFA automatically generates and visualizes the trace slice. In this example, the slice is anchored by the Hazard UAV-1364 stating that “Prohibited airspace is not clearly identified and the drone accidentally flies into the airspace”. If this hazard were to occur it would inhibit the user story “As a compliance officer, I want to make sure that UAVs do not fly into prohibited airspace” (UAV-1383). Additional nodes, directly and indirectly related to this trace slice contextualize the hazard (UAV-1438) and show how it is addressed through safety stories, design constraints, and ultimately implemented in the code. Test cases and other forms of validation such as formal models or simulations are not displayed here due to space constraints, but could equally well be included in the hazard slice.

Finally, we define a set of rules which SAFA uses to generate warnings. An example, stated informally is that “each Design Constraint must have one or more associated source code files”. The warning in Fig 2a highlights the fact that design constraint UAV-1388 is either not implemented or that related trace links have not yet been established. SAFA is supported by several underlying tools. In this example, a trace link evolution algorithm (13) recommends three source code files that could be linked to the design constraint, the first of which has been accepted by the user. Artifact trees can be generated repeatedly by developers throughout a sprint to visually track the mitigation of relevant hazards. They can be used to answer questions such as “Is the safety story sufficiently addressed in the design?”, “Have I fully implemented and tested the design constraints?”, “Are any known design constraints missing from the artifact tree?” This last question would imply missing trace links.

B. Bottom-up Analysis

Agile developers may find themselves modifying source code as they perform refactorings or as they add new features to the system. The underlying trace links support this activity

through visualizing a bottom-up graph as depicted in Fig. 2b. Starting at a source code file, all artifact trees which include a link to this file are identified and each traceability path is followed in an upwards direction through design constraints, safety-stories, and hazards as specified in the TIM. This view can be generated anytime a developer is working on code, and can be used to address questions such as “What is the potential impact of my current or planned code changes upon the overall safety of the system?”

C. Delta Analysis

Delta views represent one of the more novel aspects of SAFA by visualizing changes that have occurred between a *baseline* version and the *current* version of an artifact tree. In the example depicted in Fig. 3, a change has occurred in the operating context (i.e., see UAV-765 (red) vs. UAV-1414 (green)), a new safety story (UAV-1384), complete with design constraints, code, and an acceptance test has been added and is shown in green. Furthermore, source code has been added (green) and modified (blue). While SAFA’s artifact trees annotate nodes with warnings, the delta views introduce informational nodes which provide insights into the change. The goal is not to naïvely tell developers and analysts whether a change is safe or not, but rather to draw their attention to specific changes that warrant further analysis. We annotate nodes with an informational symbol to indicate that additional information is available. For example, clicking on the source code informational nodes provides links to commit messages and to results from a refactoring detection tool explaining specific changes in the code; whereas, clicking on the informational node attached to the green context node provides a rationale and explanation of the FAA’s new LAANC (Low Altitude Authorization and Notification Capability) system. Adding additional AI-driven insights associated with diverse patterns of change, represents an ongoing endeavor.

The delta tree therefore supports questions such as “What has changed in this version?”, “How do those changes impact safety?”, and “Are additional constraints or evidence needed for safety purposes?” Our goal is not to fully automate the safety analysis process, but to provide tools that empower developers to make informed, analytical, safety decisions.

We previously conducted a preliminary user study of SAFA’s delta views with ten software professionals, seven of whom had experience working in safety critical domains, and four of whom had over seven years of experience in industry (9). The ten participants were divided into two approximately equally skilled groups (A,B) and asked to evaluate six different hazards. Group A evaluated hazards 1-3 using the delta trees, and hazards 4-6 without them, while Group B did the opposite. Overall, participants identified all risk points using the delta views and only 80.6% without them. Nine participants (including everyone with more than 2 years of development experience) preferred the delta view over the use of paired artifact trees. Furthermore, one participant with 35 years of experience stated that he would “kill to have this (Delta View) in my workplace.”

- cross-national comparison of approaches to regulation.” *Food and drug law journal*, vol. 69 1, pp. 1–23, i, 2014.
- [3] B. Fitzgerald, K.-J. Stol, R. O’Sullivan, and D. O’Brien, “Scaling agile methods to regulated environments: An industry case study,” in *Proc. of the 35th Int’l Conf. on Software Engineering*. IEEE, 2013, pp. 863–872.
- [4] P. Diebold and S. Theobald, “How is agile development currently being used in regulated embedded domains?” *Journal of Software: Evolution and Process*, vol. 30, no. 8, 2018. [Online]. Available: <https://doi.org/10.1002/smr.1935>
- [5] T. Myklebust and T. Stålhane, *The Agile Safety Case*. Springer, 2018. [Online]. Available: <https://doi.org/10.1007/978-3-319-70265-0>
- [6] J. Cleland-Huang, O. C. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, “Software traceability: trends and future directions,” in *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 55–69.
- [7] B. Gallina, F. U. Muram, and J. P. C. Ardila, “Compliance of agilized (software) development processes with safety standards: A vision,” in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, ser. XP ’18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: <https://doi.org/10.1145/3234152.3234175>
- [8] M. Kuhrmann, P. Diebold, J. Münch, P. Tell, V. Garousi, M. Felderer, K. Trektene, F. McCaffery, O. Linssen, E. Hanser, and C. R. Prause, “Hybrid software and system development in practice: Waterfall, scrum, and beyond,” in *Proceedings of the 2017 International Conference on Software and System Process*, ser. ICSSP 2017. New York, NY, USA: ACM, 2017, pp. 30–39. [Online]. Available: <http://doi.acm.org/10.1145/3084100.3084104>
- [9] A. Agrawal, S. Khoshmanesh, M. Vierhauser, M. Rahimi, J. Cleland-Huang, and R. R. Lutz, “Leveraging artifact trees to evolve and reuse safety cases,” in *International Conference on Software Engineering, ICSE 2019, Montreal, Canada, 2019*, pp. 1222–1233. [Online]. Available: <https://doi.org/10.1109/ICSE.2019.00124>
- [10] L. C. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue, “Traceability and sysml design slices to support safety inspections: A controlled experiment,” *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, pp. 9:1–9:43, 2014. [Online]. Available: <https://doi.org/10.1145/2559978>
- [11] C. Mayr-Dorn, M. Vierhauser, F. Keplinger, S. Bichler, and A. Egyed, “Timetracer: A tool for back in time traceability replaying,” in *42nd International Conference on Software Engineering Companion (ICSE ’20 Companion)*, ser. ICSE 2020. New York, NY, USA: ACM, 2020.
- [12] A. D. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, “Information retrieval methods for automated traceability recovery,” in *Software and Systems Traceability*, 2012, pp. 71–98. [Online]. Available: https://doi.org/10.1007/978-1-4471-2239-5_4
- [13] M. Rahimi and J. Cleland-Huang, “Evolving software trace links between requirements and source code,” *Empirical Software Engineering*, vol. 23, no. 4, pp. 2198–2231, 2018. [Online]. Available: <https://doi.org/10.1007/s10664-017-9561-x>
- [14] T. Stålhane, G. K. Hanssen, T. Myklebust, and B. Haugset, “Agile change impact analysis of safety critical software,” in *Computer Safety, Reliability, and Security*, A. Bondavalli, A. Ceccarelli, and F. Ortmeier, Eds. Cham: Springer International Publishing, 2014, pp. 444–454.
- [15] J.-P. Steghöfer, E. Knauss, J. Horkoff, and R. Wohlrab, “Challenges of scaled agile for safety-critical systems,” in *20th International Conference on Product-Focused Software Process Improvement, PROFES, Barcelona, Spain, 2019*.