Editor in Chief: **Ipek Ozkaya**
Carnegie Mellon Software Engineering Institute
ipek.ozkaya@computer.org

Can We Really Achieve Software Quality?

Ipek Ozkaya

"SOFTWARE QUALITY" IS defined as the field of study and practice that describes the desirable attributes of software products. There are two dimensions of quality in consideration here. The first is the fact that the software is free of errors, that is, free of bugs and vulnerabilities, including missed or misunderstood requirements and errors in design, functional logic, data relationships, process timing, validity checking, and coding errors. The second, which is more subtle, is the fact that the system has a handle on its technical debt and satisfies different quality and quality attribute characteristics. The ISO/International Electrotechnical Commission (IEC) 25010:2011 standard groups these characteristics under eight categories: functional suitability, reliability, operability, performance efficiency, security, compatibility, maintainability, and transferability.[1] However, in reality, quality and quality attributes vary in multiple other dimensions as well. In addition, business and contextual goals drive tradeoffs that directly influence the assessment and guarantee of quality. For example, safety-critical domains incorporate finer granular notions of safety and fault tolerance as well as the ability to certify that these properties are adequately met.

Assuring software quality along both of these dimensions revolves around establishing and ensuring that the processes and practices of developing software result in a system that has the proper quality to meet its business and user goals. Consequently, assuring software quality advocates for a systematic approach to evaluating the quality of and adherence to software product standards, processes, and procedures. This activity includes making sure that appropriate standards and procedures are established and followed throughout the entire software development lifecycle, including requirements engineering, software design, software architecture, implementation, code reviews, design conformance, software configuration management, testing, release management, software integration, and deployment.

The collection of all these practices constitutes software quality assurance; however, more often than not, software quality assurance is mostly associated with testing practices. Software quality assurance cannot be achieved when confined only to executing tests and conformance activities. Software engineering practices need to prioritize the application of techniques in which enforcing software quality is already incorporated. Moreover, without embracing an assured-by-design and implementation approach along with tools that assist developers to accomplish it, achieving software quality will continue to mostly rely on our ability to test extensively.

More Formalism Versus More Agility

Assuring software quality requires that the implemented software achieves its desired behavior without any unintended consequences. Techniques available for such assured confidence gravitate toward increased formalism, for example, formal methods for model-based approaches to capture traceability from requirements to design to implementation. While more formalism assists to check for nonconformances, it comes with the cost of the time it takes to

Digital Object Identifier 10.1109/MS.2021.3060552
Date of current version: 16 April 2021

IEEE Software Mission Statement

To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

CONTACT US

AUTHORS

For detailed information on submitting articles, visit the "Write for Us" section at www.computer.org/software

LETTERS TO THE EDITOR

Send letters to software@computer.org

ON THE WEB

www.computer.org/software

SUBSCRIBE

www.computer.org/subscribe

SUBSCRIPTION CHANGE OF ADDRESS

address.change@ieee.org
(please specify *IEEE Software*.)

MEMBERSHIP CHANGE OF ADDRESS

member.services@ieee.org

MISSING OR DAMAGED COPIES

contactcenter@ieee.org

REPRINT PERMISSION

IEEE utilizes Rightslink for permissions requests. For more information, visit www.ieee.org/publications/rights/rights-link.html

use these techniques (the time it takes to specify the software to the level of detail that is required as inputs for the effective use of methods with higher degrees of formalism and rigor). Any technique that introduces increased rigor, when not supported by appropriate automation, may imply shifting resources from delivering new functionality. Therefore, there is often debate about how increased formalisms help in designing-in and assuring software quality, but risk compromising from agility.

The perception that achieving quality by construction with rigor and agility is not possible is a weakness of how we execute the techniques, not the weakness of the techniques. For example, gated process checks and conformance steps are inserted, such as triggering recertification every time a change happens in safety-critical software, even when model-based software engineering techniques are embraced. The emphasis should be on isolating the changes and automating their traceability to provide evidence that the changes do not risk quality, rather than relying on manual and qualitative conformance steps. At the other extreme, when software delivery is pressed for time and faces resource challenges, we shortcut the application of practices that enforce and check for quality even when automated, such as analyzing the codebase for design faults beyond obvious bugs or completing all of the needed tests.

The Quality Triangle Revisited

The project management triangle, also referred to as the *iron triangle*, suggests that the expected quality of any work is constrained by the project's budget, schedule, and scope (features implemented). If we believe the quality triangle to be correct, achieving software quality is always incomplete. We accept that we always deliver below par as we always have to trade off one aspect of the cost, schedule, and scope triad. There are legitimate challenges that make it quite difficult to break the tight coupling among these elements and their influence on software quality.

In 2018, Vector Consulting Services conducted a survey among 2,000 decision makers about trends and challenges in software engineering in recognition of the 50th anniversary of software engineering. The study revealed that organizations continue to struggle to achieve quality along with cost and efficiency.[2] The reasons participants cited included the ever-increasing pressure to reduce costs while increasing development speed. This pressure will no doubt continue to challenge software quality. Automating test, analysis, integration, and deployment—in particular, through embracing DevOps practices and tools—is one response that software engineering organizations are giving in an effort to not give up on quality. Empirical evidence suggests that organizations incorporating automated security analysis and DevOps practices (also referred to as *DevSecOps*) observe improved security through the improved discovery of vulnerabilities by using integrating analysis as well as monitoring tools in their development and deployment environments.[3] Despite their demonstrated effectiveness; however, not all DevOps and automated analysis tools are adopted by developers. The reasons why developers fail to adopt DevOps tools include challenges in configuring them and mismatches in how the tools adapt to the developers' workflows.[4] When developing such tools, we need to prioritize understanding developers' workflows. Also, if new workflows are beneficial, we need to prioritize how to overcome barriers of adoption. Improved tool support

THE MANY FACETS OF SOFTWARE QUALITY

In this issue of *IEEE Software*, we feature articles that discuss software quality from different engineering practices' perspectives, spanning defect management, developing safety-critical systems in agile environments, a comparison of tools that assist in detecting technical debt, and a focus on two different aspects of testing, including black- and white-box testing and fuzzing. This collection of articles is well representative of the wide range of practices that need to work in concert to achieve software quality.

Lopez et al.'s article, "Bumps in the Code: Error Handling During Software Development," looks at software errors from the perspective of their potential contributions to the experience of software developers and their growth.

In "Automatic Recovery of Missing Issue Type Labels," El Zanaty and colleagues describe an approach where they apply machine learning to classify issues by defect type to improve data analytics.

Cleland-Huang et al. suggest the use of traceability links to visualize and analyze changes to support safety assurance in agile development environments in their article, "Visualizing Change in Agile Safety-Critical Systems."

In "Software Safety Analysis to Support ISO 26262-6 Compliance in Agile Development," Sandgren and Antinyan present a software safety-analysis method to support compliance to ISO 26262.6, the road vehicle function safety standard, in agile projects.

Avgeriou and colleagues summarize the current state of the practice when it comes to tools that offer features in an attempt to quantify technical debt with different metrics in their article, "An Overview and Comparison of Technical Debt Measurement Tools."

Arcuri presents experiences with the automated testing of RESTful application programming interfaces (APIs), and Böhme et al. present outcomes of the 2019 Shonan Village Center workshop on fuzzing and symbolic execution in their articles, "Automated Black- and White-Box Testing of RESTful APIs With EvoMaster" and "Fuzzing: Challenges and Reflections," respectively.

will improve ability to enforce quality conformance with less time and effort, consequently will reduce cost and schedule barriers to achieving quality.

Can We Really Achieve Software Quality?

Let us revisit the question I started with: Can we really achieve software quality? If assuring software quality spans every technique and engineering activity in the software development and deployment lifecycle, and if we have to trade off among cost, schedule, and scope to achieve any quality, the implication by definition is that quality can only be relative and not assessed in absolute terms. The implication of this observation is that we always need to give up on quality in our software systems. There are concrete actions that we can take to avoid comprimising from quality, such as the following:

EDITORIAL STAFF

IEEE SOFTWARE STAFF

Managing Editor: Jessica Welsh, j.welsh@ieee.org
Cover Design: Andrew Baker
Peer Review Administrator: software@computer.org
Publications Staff Editor: Cathy Martin
Publications Operations Project Specialist: Christine Anthony
Compliance Manager: Jennifer Carruth
Publications Portfolio Manager: Carrie Clark
Publisher: Robin Baldwin
Senior Advertising Coordinator: Debbie Sims
IEEE Computer Society Executive Director: Melissa Russell

CS PUBLICATIONS BOARD

M. Brian Blake (VP of Publications), Hui Lei, Antonio Rubio, Diomidis Spinellis

CS MAGAZINE OPERATIONS COMMITTEE

Diomidis Spinellis (MOC Chair), Lorena Barba, Irena Bojanova, Shu-Ching Chen, Gerardo Con Diaz, Lizy K. John, Marc Langheinrich, Torsten Möller, Ipek Ozkaya, George Pallis, Sean Peisert, VS Subrahmanian, Jeffrey Voas

IEEE PUBLICATIONS OPERATIONS

Senior Director, Publishing Operations: Dawn M. Melley
Director, Editorial Services: Kevin Lisankie
Director, Production Services: Peter M. Tuohy
Associate Director, Information Conversion and Editorial Support: Neelam Khinvasara
Senior Managing Editor: Geraldine Krolin-Taylor
Senior Art Director: Janet Dudar

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Software* does not necessarily constitute endorsement by IEEE or the IEEE Computer Society.

To Submit: Access the IEEE Computer Society's Web-based system, ScholarOne, at http://mc.manuscriptcentral.com/sw-cs. Be sure to select the right manuscript type when submitting. Articles must be original and not exceed 4,700 words including figures and tables, which count for 200 words each.

IEEE prohibits discrimination, harassment and bullying: For more information, visit www.ieee.org/web/aboutus/whatis/policies/p9-26.html.

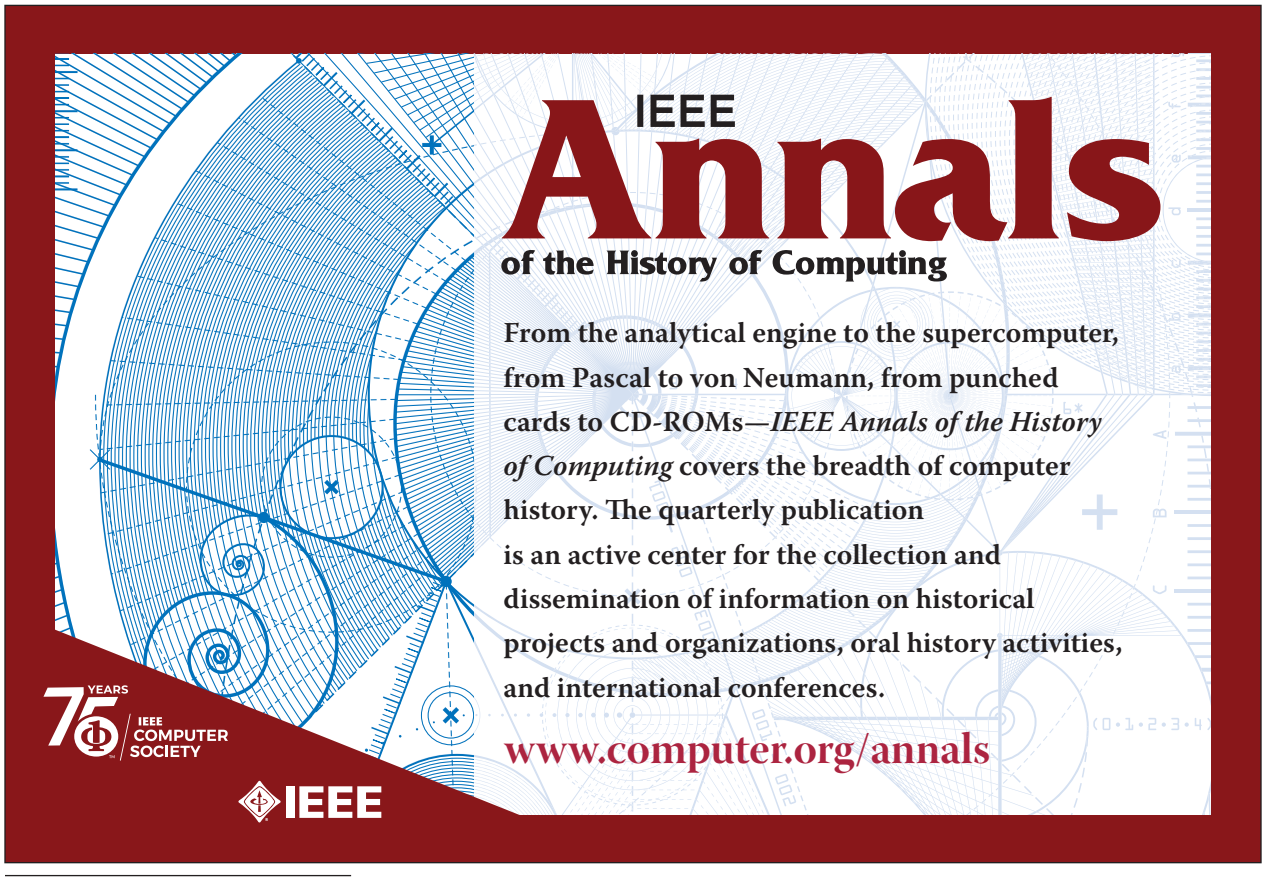
Digital Object Identifier 10.1109/MS.2021.3058030

- Continue to advance reliable, automated tool support, which will automate mundane analysis tasks by seamlessly integrating quality conformance into developer workflows and bring the cost and time of assuring quality down.
- Embrace the approach that implementing and assuring software quality is not a phase in the software development lifecycle, but it is a nonnegotiable aspect of all of the software engineering activities we conduct.
- Make sharing data and experiences in applying various techniques top priorities so that we can establish an improved empirical basis for choosing techniques and practices that get us closer to software that has assured quality by construction.

As software engineers, we need to reject a "good enough" quality mindset. Achieving software quality will be possible when better tools to enforce it will be widespread and software engineers will learn to make it a priority, no matter what.

References

1. *Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE)*—System *and* Software Quality Models, ISO/IEC 25010:2011.
2. C. Ebert, "50 years of software engineering: Progress and perils," *IEEE Softw*, vol. 35, no. 5, pp. 94–101, 2018. doi: 10.1109/MS.2018.3571228.
3. A. A. Ur Rahman and L. A. Williams, "Software security in DevOps: Synthesizing practitioners' perceptions and practices," in *Proc. Int. Workshop Continuous Software Evolution Delivery*, 2016, pp. 70–76. doi: 10.1145/2896941.2896946.
4. A. Rahman, A. Partho, D. Meder, and L. A. Williams, "Which factors influence practitioners' usage of build automation tools?" in *Proc. 2017 IEEE/ACM 3rd Int. Workshop on Rapid Continuous Softw. Eng. (RCoSE)*, pp. 20–26. doi: 10.1109/RCoSE.2017.8.



Digital Object Identifier 10.1109/MS.2021.3068501