Received: 3 July 2019 | Accepted: 5 July 2019
DOI: 10.1002/smr.2221

WILEY Journal of Software: Evolution and Process

EDITORIAL

The new era of software reuse

pilar

For more than 40 years, software reuse has been considered the cornerstone of many software development processes in order to save development effort and costs and to increase the quality of software. The evolution of different kinds of reusable assets has been wide. For instance, on one hand, diverse and modern programming languages have brought different ways to achieve reusable code more efficiently and adapted to modern software platforms. On the other hand, diverse and modern family-oriented engineering methods have enabled the reuse of not only code but also any other engineering software-related development artefact. However, although software reuse is not mainstream in current software conferences, researchers and developers still build software with reuse in mind. Nowadays, the challenge for building smart software systems that exploit contextual information to provide a smart reaction or adaptation to the environment, new Internet application and services that rely on the notion of micro-services as a form of reusable assets, or open data approaches used to achieve interoperability and reuse of data, is current trends where developers can find new opportunities for software reuse. This special issue on The New Era of Software Reuse, associated with the 17th International Conference on Software Reuse (ICSR 2018), aims to increase the impact of related research in software reuse. We received seven submissions and after a rigorous review process, we accepted four articles for publication based on quality and quantity of the new technical contribution. Each accepted article explores a different perspective of software reuse and presents different techniques to achieve it. The list of accepted articles is outlined below.

In "A Formal Framework for Measuring Technical Lag in Component Repositories—and its Application to npm," Zerouali and his colleagues state that reusable open source software components are typically distributed through online package repositories like npm for JavaScript. Such component repositories form complex dependency networks that are subject to frequent evolution. As a consequence, many reusable components have outdated dependencies, increasing their risk of suffering from bugs and security issues in those dependencies. While upgrading one's dependencies reduces this risk, it incurs additional effort and may be subject to backward incompatibilities. The technical lag measure captures this delicate balance in a generic way, by quantifying the difference between the currently deployed component release and the ideal (eg, most recent, most stable, or most secure) release. Based on the technical lag, component maintainers can make more informed decisions about whether and when to upgrade. In this manuscript, a generic formal framework for measuring this technical lag is proposed. The framework makes it possible to measure lag in different ways, such as based on the time difference or the version difference. The framework was empirically validated by instantiating it in multiple ways to the npm package dependency network, in order to empirically analyze the presence of technical lag in npm packages and assess its evolution over time. The authors observed that the presence of technical lag relates to the use of too restrictive dependency constraints, and that technical lag is increasing over time. External GitHub projects relying on pm packages suffer from a higher technical lag than npm packages themselves. Deep transitive dependencies tend to induce a high technical lag. The findings suggest the need for more awareness of, and integrated tool support for, controlling technical lag in reusable software libraries.

In "A Delta-oriented Approach to Support the Safe Reuse of Black-box Code Rewriters" by Benni and his colleagues, the authors argue that large-scale corrective and perfective maintenance is often automated thanks to rewriting rules using tools such as Python2to3, Spoon or Coccinelle. Such tools consider these rules as black-boxes and compose multiple rules by chaining them: that is, giving the output of a given rewriting rule as input to the next one. Therefore, it is up to the developer to identify the right order (if it exists) among all the different rules to yield the right program. In this paper, a formal model compatible with the black-box assumption is defined that reifies the modifications (Δs) made by each rule. Leveraging these (Δs), the authors propose a way to safely compose multiple rules when applied to the same program by (i) ensuring the isolated application of the different rules and (ii) identifying unexpected behaviors that were silently ignored before. This approach supports the reusability of code rewriting rules at large scale. We assess our results on two large-scale case studies: (i) identifying conflicts in the Linux source-code automated maintenance (by analyzing 35 semantic patches applied to 19 versions of Linux) and (ii) fixing energy anti-patterns existing in Android applications available on GitHub (by analyzing 22 anti-patterns rewriting rules on 19 Android applications).

In "Variability management in safety-critical systems design & dependability analysis," de Oliveira and his colleagues highlight the importance of safety-critical systems for many application domains and where safety properties are a key driver to engineer critical aspects and avoid system failures. For the benefits of large-scale reuse, software product lines (SPLs) have been adopted in the critical systems industry. However, the integration of safety analysis in the SPL development process is nontrivial. Also, the different usage contexts of safety-critical systems complicates component fault modeling tasks and the identification of potential hazards. In this light, better methods become necessary to estimate the impact of dependability properties during Hazard Analysis and Risk Assessment. Existing methods incorporating the analysis of safety properties in SPLs are limited as they do not include hazard analysis and component fault modeling. In this paper, the authors present a novel DEPendable Software Product Line Engineering (DEPendable-SPLE) approach, which extends traditional SPL processes to support the reuse of safety assets.

The work describes a detailed analysis of the impact of product and context features on the SPL design, safety analysis, and safety requirements. The approach is validated using a real case study from the aerospace domain to illustrate how to model and reuse safety properties, but also to achieve a reduction of the the effort and costs for certifying system variants.

In "REI: An integrated measure for software reusability," Zozas and his colleagues state the need that an efficient selection among candidate reusable assets should be performed in terms of functional fitness and adaptability. In this article, the authors propose a reusability index (REI) as a synthesis of various software metrics and evaluate its ability to quantify reuse, based on the IEEE Standard on Software Metrics Validity. REI overcomes the limitations of the literature by synthesizing various metrics that influence reusability and considering multiple aspects of reusability, such as structural quality, external quality, documentation, availability, etc. The novelty of the proposed index lies in the fact that a holistic (ie, not only dependent on structural aspects of software) and quantifiable measure of reusability is proposed. As validation, the proposed index is compared to existing ones through a case study on 80 reusable open-source assets. The results of the study suggest that the proposed index presents the highest predictive and differential power; it is the most consistent in ranking reusable assets and the most strongly correlated to their levels of reuse. To illustrate the applicability of the proposed index, they run a pilot study, where real-world reuse decisions have been compared to decisions imposed by the use of metrics (including REI).

In addition, the guest coeditors of this special issue together with John Favaro have documented a significant piece of the history of software reuse in the form of an overview article titled "Opportunities for Software Reuse in an Uncertain World: From Past to Emerging Trends." In this invited paper, the authors describe part of the history from old forms of software reuse to current trends such as open data reuse, services and micro-services, reusable features, and the reuse of safety-critical assets. In addition, the authors provide an updated view of current trends of reuse practices in different application domains. Also, in order to understand what reuse practitioners think, the authors conducted an online survey to interview practitioners and developers about 14 questions in a similar vein to the survey that was conducted in 1993 by Bill Frakes. The results comparing both surveys for similar questions bring an interesting outcome about the evolution of reuse practices and how these are integrated with other software engineering processes and open source approaches. To conclude, the authors provide a research road map for the future as a set of four main trends that may guide researchers into new forms of software reuse.

Finally, we would like to thank the authors of all submissions and the reviewers for their effort and commitment in providing a rigorous review process and a valuable contribution to the discipline of software reuse. We are also very grateful to Gerardo Canfora, Editor-in-Chief, for his continuous support in this special issue.

ORCID

Rafael Capilla https://orcid.org/0000-0002-6943-1285
Barbara Gallina https://orcid.org/0000-0002-6952-1053
Carlos Cetina Englada https://orcid.org/0000-0001-8542-5515

Rafael Capilla[1]
Barbara Gallina[2]
Carlos Cetina Englada[3]

[1]*Department of Informatics, Rey Juan Carlos University, Madrid, Spain*
[2]*Mälardalen University, Västerås, Sweden*
[3]*Escuela de Arquitectura y Tecnología, Universidad San Jorge, Zaragoza, Spain*

Correspondence
Rafael Capilla, Department of Informatics, Rey Juan Carlos University, Madrid, Spain.
Email: rafael.capilla@urjc.es