Herausgeber et al. (Hrsg.): Name-der-Konferenz,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 11

Feature Trace Recording – Summary

Paul Maximilian Bittner[1], Alexander Schultheiß[2], Thomas Thüm[3], Timo Kehrer[4], Jeffrey M. Young[5], Lukas Linsbauer[6]

Abstract: In this work, we report about recent research on Feature Trace Recording, originally published at the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) 2021 [Bi21]. Tracing requirements to their implementation is crucial to all stakeholders of a software development process. When managing software variability, requirements are typically expressed in terms of features, a feature being a user-visible characteristic of the software. While feature traces are fully documented in software product lines, ad-hoc branching and forking, known as clone-and-own, is still the dominant way for developing multi-variant software systems in practice. Retroactive migration to product lines suffers from uncertainties and high effort because knowledge of feature traces must be recovered but is scattered across teams or even lost. We propose a semi-automated methodology for recording feature traces proactively, *during* software development when the necessary knowledge is present. To support the ongoing development of previously unmanaged clone-and-own projects, we explicitly deal with the absence of domain knowledge for both existing and new source code. We evaluate feature trace recording by replaying code edit patterns from the history of two real-world product lines. Our results show that feature trace recording reduces the manual effort to specify traces.

Keywords: feature traceability, feature location, disciplined annotations, clone-and-own, software product lines

1 Summary

For comprehending, maintaining, and extending existing software, it is crucial to find locations of interest in a software system quickly, reliably, and exhaustively. To that end, tracing requirements to their implementation is one of the most common activities of developers and crucial to all stakeholders of a software development process. When managing software variability, requirements are typically expressed in terms of *features*. A *feature trace* identifies those artefacts of the software system that implement a certain feature, thus indicating where, how, and which features are implemented.

While features, their dependencies, and their locations are fully documented in software product-line engineering, it is rarely adopted in practice for multiple resons. In practice,

[1] University of Ulm, Germany, paul.bittner@uni-ulm.de
[2] Humboldt-University of Berlin, Germany, alexander.schultheiss@informatik.hu-berlin.de
[3] University of Ulm, Germany, thomas.thuem@uni-ulm.de
[4] Humboldt-University of Berlin, Germany, timo.kehrer@hu-berlin.de
[5] Oregon State University, Corvallis, Oregon, USA, youngjef@oregonstate.edu
[6] TU Braunschweig, Germany, l.linsbauer@tu-braunschweig.de

development often begins with only a single variant of the software system to reduce complexity and costs, or because the need for future variants is unknown. To derive a new variant of the system, developers clone the whole software system to alter specific parts independently from the previous variant (e. g., using branches or forks), also known as clone-and-own. Unfortunately, propagating changes such as bug fixes to other cloned variants is increasingly difficult and ambiguous with a growing number of variants because knowledge of feature traces is scattered across the team or even lost.

We propose *feature trace recording*, a semi-automated methodology to infer feature traces semi-automatically upon source code changes. The main idea is that by recording feature traces during development, they neither have to be recovered retroactively, suffering from uncertainties, nor specified explicitly in a separate step, which causes overheads to the actual source code editing and becomes increasingly difficult as time passes since the last edit. Our core design philosophy is that contributing domain knowledge in the form of feature traces must place minimal burden on the user in order to be accepted. As developers might not always know to which feature an edited artefact belongs, and because feature traces are initially absent in clone-and-own development, we specifically deal with absent domain knowledge on both new and existing source code.

In our envisioned methodology, software development is performed according to a session-oriented editing model where developers *may* specify the feature or feature interaction they are currently implementing. From the edits developers make under such a *feature context*, our recording algorithm infers feature traces for changed source code automatically. Thereby, feature trace recording is not tied to any specific task in the workflow of developers, such as commits to a version control system. Moreover, by employing *disciplined annotations* with Abstract Syntax Trees, we release developers from manual, laborious, and error-prone tasks, such as assigning opening and closing brackets to specify feature traces which are scattered among several code fragments.

2 Data Availability

The original publication is accessible under the DOI 10.1145/3468264.3468531. A preprint is also available online at `https://github.com/SoftVarE-Group/Papers/raw/master/2021/2021-ESECFSE-Bittner.pdf`. Our artifact is available on Github (`https://pmbittner.github.io/FeatureTraceRecording`) and Zenodo (DOI: 10.5281/zenodo.4900682).

Bibliography

[Bi21] Bittner, Paul Maximilian; Schultheiß, Alexander; Thüm, Thomas; Kehrer, Timo; Young, Jeffrey M.; Linsbauer, Lukas: Feature Trace Recording. In: Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE). ACM, New York, NY, USA, pp. 1007–1020, August 2021.