Feature-Oriented Traceability

Thorsten Berger
Chalmers | University of Gothenburg

Context and Motivation. Features are commonly used to describe the functional and non-functional aspects of a system. Features are abstractions over implementation assets and understood by many different roles, including domain experts, architects, and developers. As such, features are often used for communication, planning, and keeping an overview understanding of a system. Some software-engineering methods advocate the explicit use of features, such as feature-driven development (FDD) and software product line engineering with feature modeling. Especially the latter requires abstractions (*features*) to cope with complex product lines—portfolios of system variants tailored towards specific requirements, such as different market segments, hardware, or non-functional properties (e.g., performance or energy consumption).

Variants are typically developed using clone&own [1]—that is, copying and adapting existing variants to new requirements. This strategy is simple and allows experimenting with new ideas and rapidly prototyping variants. However, it does not scale well, and maintaining variants quickly becomes costly. Then, variants often need to be migrated to an integrated product-line platform. Such a platform is often configurable and allows deriving variants by selecting dedicated features in a configurator tool. Unfortunately, the product-line migration is costly and risky, requiring architectural and organizational changes, as well as recovering features and their locations.

We believe that recording features and their locations early during clone&own facilitates feature maintenance and evolution, including a later platform migration. Established feature traceability would avoid the expensive recovery of feature traces (e.g., when modifying, removing or reusing features) or allow analyzes and predictions on the level of features.

Questions. Our past and current work targets three questions.

Q1: *What are effective feature-traceability methods?* Most established methods retroactively recover feature traces and are heavyweight, imposing significant setup (e.g., creating a traceability database) and ongoing costs (e.g., updating traces in the database). We believe that effective methods should encourage developers to continuously record features during development and should be lightweight, facilitating easy integration into engineering processes with low overhead.

Q2: *What are the benefits and costs of feature traceability?* The cost of creating and maintaining traceability links should be lower than the expected benefit. We conjecture that feature traceability does not only enhance feature-oriented engineering activities and analyzes, but also support traceability tasks not related to features, such as change-impact analysis of requirements. We believe that some of the low-level and fine-grained traceability links (e.g., between requirements and test cases) can be replaced by fewer, but higher-level traceability links based on features as pivotal (and intuitive) elements.

Q3: *How to leverage expert knowledge?* Most feature-traceability techniques focus on the fully automated recovery of feature locations, typically using information-retrieval techniques. Yet, these techniques are inaccurate and have not found widespread adoption. Since features are highly domain-specific [2], expert (developer) knowledge should be taken into account. We believe that effective feature-traceability methods are hybrid, supporting developers with partial automation.

Principal Ideas. Towards Q1, we conceived a lightweight feature-traceability approach relying on embedded annotations. During implementation, developers record features and the feature locations as annotations in code [3]. Our feature dashboard tool allows exploiting annotations by extracting and visualizing them [4]. Towards Q2, we studied costs and benefits of embedded annotations [3], which showed that their cost is low compared to their benefit for feature maintenance and evolution. We currently investigate the benefit of feature traceability for other traceability tasks, such as change-impact analysis for requirements. We strive to conduct user studies on feature traceability compared to traditional traceability, investigating how the notion of features is perceived and under what conditions feature traceability is beneficial. Towards Q3, we currently conceive a hybrid approach that learns from past annotations and recommends new ones. Developers should proactively record annotations, but should be reminded about potentially forgotten traces (e.g., upon commit). We systematically experiment with machine-learning techniques by replaying the history of feature annotations in a case study.

Contributions. We contribute to the grand challenges of traceability by raising the abstraction level of traceability to the (intuitive) notion of features, by arguing for feature traceability as a way to provide more short-term benefits of traceability in order to improve its general acceptance, and by providing empirical data on the costs and benefits of feature traceability.

Future Directions. While our work is motivated by the needs of variant engineering in clone-based development, we believe that making features explicit and establishing feature traceability also enhances single-system engineering. Investigating traceability strategies around the notion of features in single systems is a valuable future research direction.

References

[1] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wąsowski, A survey of variability modeling in industrial practice, in: VaMoS, 2013.

[2] T. Berger, D. Lettner, J. Rubin, P. Grünbacher, A. Silva, M. Becker, M. Chechik, K. Czarnecki, What is a feature? A qualitative study of features in industrial software product lines, in: SPLC, 2015.

[3] W. Ji, T. Berger, M. Antkiewicz, K. Czarnecki, Maintaining feature traceability with embedded annotations, in: SPLC, 2015.

[4] B. Andam, A. Burger, T. Berger, M. Chaudron, Florida: Feature location dashboard for extracting and visualizing feature traces, in: VaMoS, 2017.