A Vision for Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants

Stefan Fischer[1] Lukas Linsbauer[2] Roberto E. Lopez-Herrejon[3] Alexander Egyed[4]

Many companies build a portfolio of similar product variants, each tailored to different customer needs. The number of such product variants varies widely. We have observed anything between a handful and 1000+ variants and correspondingly large code sizes. The key characteristic of these product variants is that they share a high degree of common functionality (i.e. features) but still differ. Currently the state of the art investigates this problem in form of Software Product Lines (SPLs), which are single, configurable systems from which all desired product variants can be derived. The drawback of SPLs is that they require considerable upfront investments because all possible product variants need to be pre-engineered into SPLs. If it is not possible to predict all these product variants SPL approaches are problematic. And even if all product variants were known a-priori, not all companies could afford building them due to the associated high cost.

In practice we rarely observed full-blown SPLs. Instead, we found that companies often resorted to an ad-hoc practice of clone-and-own, where a new product variant is created by modifying existing variants that closely match the new variant's needs while copying and pasting from other variants as needed. Clone-and-own has three informal steps:

1. *extraction:* locating reusable artifacts (e.g. code) in the existing variants and,
2. *composition:* copying/merging those artifacts that closest match the desired needs into a new product variant, and
3. *completion:* adapting the new product variant to account for needs that did not exist thus far in any existing variant (i.e. new requirements) or could not be extracted.

This paper provides a vision for an approach called ECCO for supporting software engineers in applying clone-and-own [Fi14, Li15, Fi15]. ECCO stands for Extraction and Composition for Clone-and-Own and it allows software engineers to incrementally develop software portfolios, one product at a time, while supporting the reuse of already available product variants. To accomplish this we automated parts of the clone-and-own process, the *extraction* and the *composition*. The *completion* step is still manual, but our approach guides the software engineer with hints.

[1] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, stefan.fischer@jku.at
[2] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, lukas.linsbauer@jku.at
[3] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, roberto.lopez@jku.at
[4] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, alexander.egyed@jku.at

The *extraction* step locates artifacts that implement a feature based on commonalities and differences in product variants. The main assumption is, that if two product variants have features in common, then the artifacts they have in common trace to these features. Moreover, the *extraction* is able to deal with feature interactions, which refer to the fact that the implementation of a feature may change depending on the presence or absence of other features.

The *composition* step is the reverse operation of the *extraction*. It merges extracted fragments based on selected features by software engineers who intend to build a new product variant. The fragments are selected not only based on the selected features but the *composition* also takes feature interactions into account. Moreover, references between artifacts and the order of artifacts are considered. The result of the *composition* is a product consisting of the selected features, as far as the *extraction* was able to distinguish them. For features that were not adequately extracted, ECCO provides a set of hints to guide the software engineer during the *completion* step.

In the *completion* step the software engineers finalize a new product variant by adding features/interactions that did not exist thus far. The engineers use the hints provided by ECCO to find missing or surplus features/interactions in the product, i.e. features/interactions that never occurred in any previous product or that could not be separated from others. When two artifacts are merged that never existed together before then ECCO provides suggestions of the orderings of these artifacts which the engineers can choose from. After a product is completed, it can be fed back into the *extraction*, which will refine the knowledge.

ECCO is thus an incrementally evolving SPL that does not require major upfront investments but still facilitates the reuse of already existing product variants. Software engineers do not have to change their development practices. They can continue to develop single product variants the way they are used to but get automated support in doing so. Our approach assumes that product variants exhibit similar code structures (i.e. a common architecture) which is a valid assumption based on our experiences thus far (SPLs also make this assumption). Please find a detailed empirical evaluation in [Fi14].

Acknowledgment The research reported has been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH, and the Austrian Science Fund (FWF) project P25289 and P25513.

Literaturverzeichnis

[Fi14] Fischer, Stefan; Linsbauer, Lukas; Lopez-Herrejon, Roberto Erick; Egyed, Alexander: Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In: ICSME. S. 391–400, 2014.

[Fi15] Fischer, Stefan; Linsbauer, Lukas; Lopez-Herrejon, Roberto E.; Egyed, Alexander: The ECCO Tool: Extraction and Composition for Clone-and-Own. In: ICSE. S. 665–668, 2015.

[Li15] Linsbauer, Lukas; Fischer, Stefan; Lopez-Herrejon, Roberto E.; Egyed, Alexander: Using Traceability for Incremental Construction and Evolution of Software Product Portfolios. In: SST. S. 57–60, 2015.