Variant Analysis in Changing System Landscapes

Vasil L. Tenev Martin Becker
Fraunhofer IESE, Kaiserslautern, Germany
{vasil.tenev, martin.becker}@iese.fraunhofer.de

***Abstract*—In order to raise understanding of similarity in a set of related systems, the Variant Analysis approach has been developed and applied in various industrial settings over the last decade. The paper discusses typical analysis goals and the respective approaches. Additionally, it motivates the need for an incremental and iterative analysis approach to support settings, where the scope of the analysis changes over time. Examples to this end are adding, updating, or removing variants, versions, or system modules.**

I. Introduction

Software-intensive systems are intended to change. Changes to the systems may be the result of an organization evolving, of end-users discovering that the software system does not fully meet their needs, several customers demanding customized system variants or indeed as a result of those needs (requirements) themselves are changing.

In order to raise understanding of similarity in a set of related systems, the Variant Analysis approach has been developed in [2] and has been successfully applied in several industrial settings over the last decade. However, if one applies the approach in larger and evolving product lines the following *challenges* arise:

(i) Variant Analysis can be applied in various settings answering different analysis questions. However, its unclear where and how to apply the Variant Analysis.

(ii) If the product line and its system variants are evolving and one likes to track the evolution of similarity over time, the respective Variant Analysis has to be conducted again and again, also just minor parts in the system variants have been changed.

(iii) If there is a large number of variants that need to be analysed the understandability of the respective visualizations declines. One approach to overcome this limitation is to conduct the Variant Analysis in several iterations.

In order to cope with challenge (i), the paper discusses different Variant Analysis settings, in which we have applied the analysis so far, and experiences we have made therein. In order to approach challenge (ii) & (iii), we discuss the need and possibilities for an incremental Variant Analysis in changing system landscapes, e. g. for adding or removing variants, versions, or system modules.

The paper is structured as follows: the following section briefly introduces the Variant Analysis approach and its key visualisations. Afterwards, it discusses typical industrial analysis settings, their challenges and our approach to overcome them. The paper closes with a discussion of possibilities and limitations of incremental Variant Analysis approaches and our planned next steps.

II. Variant Analysis

The Variant Analysis approach [2] and the corresponding tool address the problem of analysing and visualizing similarity of many (possibly large) variants of a software system. This analysis problem is especially relevant when several software system variants were created in the process of cloning or forking, in order to accommodate increasing customization. While system cloning is a frequent phenomenon in the software industry [1], code duplication causes maintenance problems which, after some time, create a need to merge these systems and introduce a systematic reuse approach. Consequently, Variant Analysis aims at recovering and visualizing the similarity information using reverse engineering [3].

At the heart of the Variant Analysis, system variants are represented as intersecting sets of content elements, and the elements are placed similar between any n systems into the intersection of the respective n sets. Using the resulting set model and the system structure hierarchy, different similarity visualizations can be used to explore the similarity measures from different perspectives. The approach is scaling for tens of compared software systems and millions lines of code. The current Variant Analysis tool analyzes similarity of text files such as source code. However, the underlying models and visualizations can also be used for other types of data, even beyond the software domain [5].

III. Variant Analysis Settings

Besides the original application scenario described above, the provided technique of multi-comparison and similarity visualization can be used also in other scenarios where the similarity information is useful. So far we have used Variant Analysis in the following analysis settings:

- *Planning of incremental refactoring*: An organization wants to understand, where they should start the incremental merging of variants into reusable core assets. To this end, one searches for clusters with high similarity in the phylogenetic tree and then analyses the variation in the selected cluster in more detail.
- *Grow and prune*: An organization follows the "grow-and-prune" approach to product line development [4], where the phases of unconstrained code growth, possibly resulting in cloned features, are interleaved with consolidation phases in which the new code is restructured to a reusable form. In this case, four variants typically need to be analysed: (i) the initial version of the core asset CA_i from which (ii) instance assets IA_i have been derived and which have been grown into (iii) the new version of the instance assets IA_{i+1}, and (iv) the latest version of

CA_j. Variant Analysis was especially helpful, if several branches had to be pruned in one merge step.

- *Monitor feature evolution in managed clones*: An organization developing cloned systems has decided not to merge them, but nevertheless still monitor the distribution of similarity in order to manage the evolution of cloned code and, for example, ensure that bug fixes are ported across the relevant variants.
- *Locate feature impact*: An organisation compares variants, which have a high degree of commonality but mainly differ in one variable feature, in order to locate the feature implementations in the variants.
- *Understand platform variability*: An organisation derives system variants from core assets, which support variability, e. g. via C-preprocessor annotations, and wants to understand how complex is variability realizations work. To this end, they derive specific variants from the core assets and compare them against the core asset to reason about the variability realization.

IV. The Need for Delta Variant Analysis

In larger system landscapes, e. g. high number of variants, high evolution rate, large system size, it's good practice to conduct the Variant Analysis in a iterative and incremental manner, typically with a slightly changing scope. In general, we consider system families in four dimensions: time (versions), space (variants), decomposition (system elements), and interplay (element relationships). Both time and space dimensions differ in their purpose and origin of existence, however there is no need to differentiate the analyses approach for them. The changes in the other two dimensions lead to new versions/variants. So, the changes in structure and relationships of system elements impact the overall variance.

This leads to the question, which items need to be (re-)analyzed and which results from previous analyses could be reused. Next, we discuss the necessary steps for incremental Variant Analysis in typical change settings.

- *Addition of variant/version*. Map the new variant items to existing information items, where possible. In other case, create new information items. Finally, analyse the variance and update figures for the considered items.
- *Removal of variant/version*. Update figures for information items that are included in the removed item.
- *Addition of system element*. Add new system elements and map them to new information items. Analyse variance and update figure accordingly.
- *Removal of system element*. Remove corresponding information items and update figures along the hierarchy.
- *Update of variant/version/system element*. Identify the information items that have changed and update their figures accordingly.
- *Addition/Removal/Update of relationships*. Any change of relationships is reflected either by influencing the mapping to information items, or by causing addition/removal/change of system elements, i. e. analyse and handle as a combination of these actions.

V. Conclusion

The complexities of software evolution can be significantly reduced by viewing evolving software as a software product line and using Variant Analysis to understand the similarity among the system variants.

In this paper, we have outlined different Variant Analysis settings in order to raise understanding where and how to apply the approach. The presented settings go beyond the original analysis setting, i. e. a commonality and variability analysis of clones variants. According to our experiences in large and changing systems settings, there is a need for an incremental and iterative analysis approach. To this end, we have discussed different change settings and the necessary steps to conduct Variant Analysis in a more delta-oriented fashion.

Our next steps include a deeper analysis of the change scenarios and a respective enhancement of the Variant Analysis approach and tool.

References

[1] Y. Dubinsky, Berger, S. Duszynski, K. Czarnecki. An exploratory study of cloning in industrial software product lines. In A. Cleve, F. Ricca, and M. Cerioli, editors, *17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013*, pages 25–34. IEEE Computer Society, 2013.

[2] S. Duszynski, J. Knodel, and M. Becker. Analyzing the source code of multiple software variants for reuse potential. In M. Pinzger, D. Poshyvanyk, and J. Buckley, editors, *18th Working Conference on Reverse Engineering, WCRE 2011, Limerick, Ireland, October 17-20, 2011*, pages 303–307. IEEE Computer Society, 2011.

[3] M. Lindvall, M. Becker, V. L. Tenev, S. Duszynski, and M. Hinchey. Good change and bad change: An analysis perspective on software evolution. *Trans. Found. Mastering Chang.*, 1:90–112, 2016.

[4] Th. Mende, F. Beckwermert, R. Koschke, and G. Meier. Supporting the grow-and-prune model in software product lines evolution using clone detection. In *12th European Conference on Software Maintenance and Reengineering, CSMR 2008, April 1-4, 2008, Athens, Greece*, pages 163–172. IEEE Computer Society, 2008.

[5] V. L. Tenev, S. Duszynski, and M. Becker. Variant analysis: Set-based similarity visualization for cloned software systems. In M. H. ter Beek, W. Cazzola, O. Díaz, M. La Rosa, R. E. Lopez-Herrejon, Th. Thüm, J. Troya, A. R. Cortés, and D. Benavides, editors, *Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017*, pages 22–27. ACM, 2017.