Additional Proposal Information

DISCLAIMER: This document accompanies the corresponding proposal to provide details on several of its aspects. **We should remark that the submitted proposal is a stand alone document. Therefore reading this document is NOT required for the assessment of the proposal.** Instead, this document is only complimentary for the reviewers that would optionally like to have more details on the items marked with ★ in the proposal document.

1.1.4 State of the Art Research

Software Product Lines and their evolution. A second challenge is the evolution of SPLs. There has been some incipient work to address this challenge. For instance, the work by Pleuss et al. [17] captures evolution but at the feature level, that is, their main focus is on the evolution of *feature models*, the de facto standard for modelling variability [14], and they do not consider the evolution of the implementing artifacts and their mappings to features and feature interactions.

Traceability and feature location. A *trace* is a link between a source and a target artifact [19]. *Traceability* is defined as the potential for traces to be established and used, and as an attribute of an artifact or collection of artifacts [19]. There has been substantial work of traceability for SPLs. For instance, the work by Jirapanthong et al. [20] presents a tool called XTraQue that defines rules to generate traces between feature models and UML diagrams. In contrast with our work, their approach is tailored for these types of diagrams and does not address traces of feature interactions. Another example is the work by Tsuchiya et al. [21]. They compute traces between requirements documents and source artifacts by relying on configuration management logs and clone detection [5, 22]. The main contrast with our proposed work is that they do not retrieve links from/to features and feature interactions. Heider et al. propose an event-based approach for SPL traceability that records the changes performed to the artifacts through an IDE across the development life-cycle [23]. Their focus is on *decision models* rather than features and feature interactions.

The first thread in feature location is the extensive work on *software diffing* where the goal is to identify the differences between two or more software artifacts. Examples on this thread are UMLDiff and VTracker, the former computes differences for UML designs and the latter is domain independent [26].

1.2 Methods – Proposed Research Approach

RC1: A Generic Evolution Knowledge Database. The first novelty of this research contribution lies in that, to the best of our knowledge, there are no other models that describe the evolution of features and feature interactions as we described them in Section 1.2. The starting point for the model we proposed is the work by Batory et al. [55], who provide an algebra for expressing features, variability, and the notion of variation points. The second novelty is that, to the best of our knowledge, there is no general infrastructure or framework to support the representation of evolution of features and feature interactions that consist of multiple types of artifacts. Rather, the state of the art for evolution in SPLs focuses mostly either on source code or on some domain-specific models, but not on more than one artifact type simultaneously.

RC2: Portfolio Mining Engine and Product Generation Engine. Our published work uses basic structural similarity at the AST level to detect differences among the variants' artifacts. Changes such as variable renaming or some forms of statement reordering are currently not detected. The first novelty of this research contribution is using advanced diffing and clone detection techniques for detecting changes between features and feature interactions with the goal of establishing traces with their implementing artifacts. The state-of-the-art in diffing and clone detection does not consider granularity level of features or feature interactions coming from multiple variants, and such techniques have only been employed to detect similarities among features once they have been modularized (e.g. with feature oriented approaches). The proposed leverage of advanced diffing and clone detection techniques will permit detecting more complex forms of changes and differences among variants' artifacts and thus obtain more accurate traces.

The second novelty is that for the implementation of the Product Generation Engine we will use the feature oriented composition paradigm for the generation of the composed products from the generic representation of their artifacts and corresponding traces. This work will advance the state of the art because current feature oriented approaches only deal with code artifacts and their extensibility capabilities beyond this type of artifacts are rather limited. In addition, we will explore how combining of feature oriented techniques and annotation-based techniques could produce a more flexible composition. For the implementation of the Product Generation Engine, we again expect to leverage existing tool support for EMF-based artifacts.

RC3: Automated Variant/Product Repair Infrastructure. The current state of the art on automated software repair has focused only on the single systems. The novelty of this research contribution then lies at the application of automated software repair techniques for software product portfolios, that is, considering multiple variants and artifacts for finding the potential repairs. Achieving this research contribution will require advances in search-based techniques (e.g. novel problem representations and evolutionary operators), and significant generalizations of existing variability-aware analysis techniques (see [48]) for supporting the generic artifact representation of our framework.

RC4: Empirical Framework Evaluation. This research contribution is spread among all the work packages as explained in the next subsection on this appendix. We should point out that in the area of SPLs, there is no such thing as a benchmark that could be used for the comparison of different related research approaches. One driving goal of this contribution is then to gather a corpus of case studies as extensive and varied as possible, and to make this corpus available for the benefit of other researchers within the community.

1.4 Work Plan

In this subsection we aim to provide more details on the rationale of each work package and how we plan to address their corresponding tasks.

WP1. Development of Evolution Knowledge Database – RC1, RC4

Task 1.1. aims to establish a feature algebra model to formally represent features and feature interactions and their traces to the artifacts that realize them. This is the foundation on which the Evolution Knowledge Database is built, that is, this knowledge repository will capture the formalism expressed in our algebra model.

Task 1.2. aims to provide the concrete generic representation to actually implement the Evolution Knowledge Database. Our starting point here will be the Eclipse Modelling Framework (EMF)[6] because it is an standard for Model-Driven Engineering, which is supported by an extensive set of tools and plugins within the Eclipse IDE. The main challenges of this task would be the scalability of this representation format, specially when dealing with complex feature interactions and multi-artifacts.

Task 1.3. will aim to implement generic transformations to-from the generic representation to the actual artifacts. For achieving this goal, we plan to rely on the extensive infrastructure available for EMF-based metamodels. For example, the transformation tools ATL and Xpand. Achieving this task will allow us to add the implementation of new variants to the

[6] http://eclipse.org/modeling/emf/

Evolution Knowledge Database with multiple artifact types, and also generate products in their corresponding artifact types.

For Task 1.4, in our published work we have already collected a basic set of representative case studies. With our industrial collaborations we expect access to other larger and more complex case studies. In addition, we have reached out to the Mining Software Repositories research community to identify open source case studies that could be suitable for our work.

The empirical evaluation of Task 1.5 will focus primarily on the flexibility, scalability and performance of the generic representation and related artifact transformations. We will assess this part with the selected case studies and it will not involve experiments with users, but rather profiling measurements on the built prototypes.

WP2. Development of Portfolio Mining Engine and Product Generation Engine – RC2, RC4

Task 2.1 and Task 2.2 will respectively survey and analyze in further detail existing approaches for diffing and clone management, with the goal of identifying the most suitable techniques and tools to adapt and extend for our framework. The main challenges of these two tasks are enabling these types of techniques to handle our generic artifact representation.

Task 2.3 refers to the implementation of the Product Generation Engine. The main challenge of this task is the realization of feature oriented composition based on model and program transformations applied to the generic artifact representation.

Task 2.4 refers to the empirical evaluation of the work package. For this purpose we will employ the identified case studies, on which we will profile the performance of the product generation (i.e. composition via model/program transformations), and perform a usability study of the supporting prototype. For the latter, we are interested in finding any potential issues with the definition and realization of the composition for the different artifact types. Some of the questions we plan to answer are: How easy or difficult is it to integrate a new artifact type into the framework? How to provide support for the user to define the expected composition for the new artifact?, How to establish and exploit dependencies between artifacts of different types?.

WP3. Development of Automated Variant and Product Repair Infrastructure – RC3, RC4

Task 3.1 and Task 3.2 aim to develop the structural consistency checking that will be required by the automated repair infrastructure. The challenges of these tasks are creating the extensions needed on both types of techniques to be able to handle the generic artifact representation used by our framework. They are challenges because the state-of-the-art on both types of techniques has focused only on single artifact types, mostly for analysis of source code.

The main novelty of this work package is the creation of an automated repair infrastructure for product portfolios. This is the purpose of Task 3.3 and Task 3.4. We propose using SBSE techniques, in particular those based on Genetic Programming, that have been proven successful for coping with certain types of repairs for single systems. Completing these tasks entails: *i)* advancing the state-of-the-art in search-state representations by considering all the aspects involved in the repair infrastructure such as artifact fragments expressed with the generic artifact representation of our knowledge database, *ii)* devising and adapting SBSE algorithms with adequate fitness functions that can efficiently exploit the knowledge that can be derived from the static analysis and safe composition techniques of the artifacts to guide the search for repairs.

Task 3.5 deals with the empirical evaluation of the repair infrastructure. We plan to perform this evaluation with the case studies collected from previous tasks, including those we already have identified, the ones we expect from our industrial collaborations, and any others that we could obtain from the open source community. This evaluation has two main objectives: *i)* to profile the performance of the repair mechanism, considering aspects such as standard metrics like recall and precision, *ii)* to devise and apply experiments that compare, for a set of specific development and evolution tasks, groups of developers that use our framework with groups that do not use it. For the latter objective, we plan to perform our experiments in a classroom setting and within an industrial application site. Our driving motivation for this type of experiments is to identify common and best practices, as well as to point out potential limitations of our framework or issues with its IDE implementation. We will follow the guidelines by Wohlin et al. (see [50]) for carrying out our empirical experiments.

4.1 Information on Research Institution

According to Microsoft Scholar, the JKU ranks among the top 1% of software engineering institutions in the world over the last five years, see URL `http://academic.research.microsoft.com/RankList?entitytype=7&topDomainID=2&subDomainID=4&last=5`

4.2 Support requested

Non-Personnel Costs Requested. The travel budged of EUR 19,000 corresponds to roughly two to three travels per person per year and includes trips to companies to gather existing software portfolios to evaluate as case studies. The visit to collaborators will be to facilitate joint publications and foster collaborations, while the field trips to companies will be to gather existing software portfolios to evaluate as case studies.