Software product lines evolution for valuable reusability

Eddy Ghabach, Mireille Blay-Fornarino, Franjieh El Khoury

Software product lines evolution for valuable reusability

GHABACH Eddy[1,2], BLAY-FORNARINO Mireille[2],EL KHOURY Franjieh[1,3], BAZ Badih[1]

Université Saint-Esprit de Kaslik (Liban) [1], Université Nice Sophia Antipolis – i3s (France) [2] Université Lyon 1 (France) [3]

ghabach@i3s.unice.fr,blay@i3s.unice.fr,franjiehkhoury@hotmail.com, franjiehelkhoury@usek.edu.lb, badih.baz@usek.edu.lb

Nowadays, adopting software product line (SPL) development approach becomes a successful strategic decision in software development since the rapid time to market necessity is guaranteed by SPLs due to assets reusability [1,2]. However, the expansion of the market segment implies a boost of user's requirements that should be satisfied by quickly developing new products [1]. Thus, an agile evolution of SPLs becomes a necessity.

The general purpose of a SPL is the automated construction of a new product based on the reusability of existing features [2]. A feature is a characteristic defined by the domain experts [3] that abstracts a set of software-related resources called assets. Thus, a feature model (FM) represents all the products of the SPL and permits capturing products commonalities and variability [3]. To generate a new product, a user selects a set of features via a process called configuration by respecting the constraints defined in the FM [2].

Despite that SPLs permit reusability, generating a new product that uses features related to different products is not supported by a classic FM [3]. In other words, if two products P_1 and P_2 leaded respectively to the injection of the features F_1 and F_2 in the FM, the latter does not support the generation of a new product P_3 that uses both F_1 and F_2 since they refer to different products. However, in our desired approach we want to design evolved FMs that make the previous operation feasible. In addition, to benefit from a valuable reusability, we are interested in focusing on the fact that a product can be constructed from a subset of the SPLs' features – regardless of referring to one or many products, or from a set of features that some of them are not part of the SPL. Thus, we should identify the features that are not part of the SPL once requested, connect them – if required – to the existing features that we still need from the SPL and integrate them in the SPL to be able to use them later. On the other hand, it is necessary to guarantee an agile evolution of the SPL, thus our approach should adopt mechanisms that automate as much as possible the software development process, minimize its overhead and simplify its complexity.

Many research papers concentrated on SPLs engineering [6], variability management [5], FMs configuration [3] and organizational SPL development challenges [1], but few of them focused on the evolution of SPLs [4]. However, after the successful adoption of SPL as a software development approach, evolving an SPL becomes a necessity for a sustainable development.

In the SPL engineering field, we define an information system as a set of interconnected assets corresponding to some well-defined features. Each information system is identified by a unique version. Thus, an information system product line can be defined as a set of features belonging to a specified domain, where this set consists of the union of the features defined or used by all the information system versions. In other words, our global FM is the union of the features of

the produced versions. Thus, the set of features or a subset of it consists of a configuration. We classified the configurations into two categories: predefined configurations where an existing version fulfills the required configuration and non-predefined configurations where a configuration refers to a set or a subset of features that are not strictly fulfilled by an existing version – thus a new version should be produced.

Based on best practices of the domain experts, developing and evolving SPLs requires to involve the client in the different phases of this process [1,6]. Thus we defined our perspective as follows: once a client arrives with new requirements, we navigate through the line feature model and respectively through the configuration knowledge engine, then we select all existing features that fulfill the client requirements. In case all the requirements are covered, we generate the product and deliver it to the client, either using an existing version (that was produced using the same features) or by creating a new version (by assembling a set of existing features). In other case, where a subset of the requirements is not covered, we determine the features that should be added (if any) or modified and respectively we determine the assets to be added, modified or deleted, before creating a partial version of the desired product. We mean by partial version, a version that contains the assets related to the features selected existing already in our line, in addition (if possible) to a skeleton of the assets related to the new/updated feature-related assets. Afterwards, we validate the extra requirement with the client based on the partial version, before updating the partial version by adding the new required assets. At this level, we should on one side, create the new version requested by the client (v_i), and on other side, integrate the new artefacts in our product line (PL) in order to reuse them in future configurations.

To develop our approach, we created multiple products concentrated on a set of domain-related features, and for each new production process we applied a different configuration category and we created our new product by selecting, adding, modifying and removing artefacts from the existing versions. Therefore, we tracked the different steps from the choice of the versions that we were based on to create the new version, the operations that we used on the features, activities and assets, and the trace of the steps that we did to execute each step. Thus, we defined some rules, constraints and operators to be used during the creation of a new product.

Finally, to adopt our approach it is necessary to provide an asset per version model to identify the assets related to each version, an asset per feature model to identify the assets related to each feature, a version to feature model to identify the features fulfilled by each version, an algorithm that automatically selects the minimum number of versions providing the maximum of the requested features for a given configuration and an algorithm that automatically performs the selection of assets per feature and provides suggestions to the product developer.

[1] Bosch, J., Bosch-Sijtsema, P.M.: Introducing agile customer-centered development in a legacy software product line. Softw. Pract. Exp. 41, 8, 871–882 (2011).

[2] Trigaux, J.C., Heymans, P.: Software Product Lines: State of the art. (2003).

[3] Czarnecki K, Helsen S., Eisenecker3 U., Staged Configuration through Specialization and Multilevel Configuration of Feature Models, Software process improvement and practice, 143-169, (2005).

[4] Borba P., Alves V, Sena D, Investigating the Safe Evolution of Software Product Lines, GPCE' 11, ACM (2011)

[5] Brummermann, H. et al.: Variability issues in the evolution of information system ecosystems. Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems. pp. 159–164 ACM, New York, NY, USA (2011).

[6] Bosch, J.: Toward Compositional Software Product Lines. IEEE Softw. 27, 3,(2010).