International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:5, 2014

User-Driven Product Line Engineering for Assembling Large Families of Software

Zhaopeng Xuan, Yuan Bian, C. Cailleaux, Jing Qin, S. Traore

***Abstract*—**Traditional software engineering allows engineers to propose to their clients multiple specialized software distributions assembled from a shared set of software assets. The management of these assets however requires a trade-off between client satisfaction and software engineering process. Clients have more and more difficult to find a distribution or components based on their needs from all of distributed repositories.

This paper proposes a software engineering for a user-driven software product line in which engineers define a Feature Model but users drive the actual software distribution on demand. This approach makes the user become final actor as a release manager in software engineering process, increasing user product satisfaction and simplifying user operations to find required components. In addition, it provides a way for engineers to manage and assembly large software families.

As a proof of concept, a user-driven software product line is implemented for Eclipse, an integrated development environment. An Eclipse feature model is defined, which is exposed to users on a cloud-based built platform from which clients can download individualized Eclipse distributions.

***Keywords*—**Software Product Line, Model-driven Development, Reverse Engineering and Refactoring, Agile Method

I. Introduction

WITH the increasing number of components in software production, strong coupling is likely to occur more often, thus increasing software complexity. Users are proposed a set of predefined products, the list of available variants being updated at the only editor initiative. Variability becomes harder to maintain and to evolve, and users end up installing products containing a majority of features irrelevant for their actual needs.

To mitigate this issue and to help software developers strengthen their ability to increase product variability for satisfying users real needs, users will set their own product configurations as release managers that can be shared, then trigger the build-up process and download the automatically packaged software distribution closely meeting their needs. This approach will help adopting organizations enhance their software engineering discipline, get a valuable and deeper insight into their users' actual needs, thus complementing efficiently the software usage logs, and convert more users. By having customized software building process triggered from users configurations, smaller, and less resource-hungry installable packages may occur more often, which can then be run on commodity hardware they could not work on otherwise.

In this paper, the proposed approach uses User-Driven Software Product Line Engineering (UDSPLE) as a means to include the user into software development life cycle. The first impact is that Feature-Oriented Domain Analysis (FODA) [1] must be extensively performed in order to create the Feature Model of software domain. Features of the domain model are created from user perspective and exposed to him, so the model may not display the same level of details and granularity as developers would have if they were the only users of the feature model. Assets are then associated to each feature, but this is handled by engineers. The user then chooses the set of features he wants to use in the software and submit the configuration that triggers the build process. To make such a process successful, dependency must be automatically solved and model should be designed from user perspective. This puts a constraint on having a well defined and up-to-date build up infrastructure and dependency resolution framework. Finally the user will be proposed the customized built up software for download.

In the remainder of this paper, Section II details concept and background on which the proposed solution is built upon. Section III illustrates the current problems. Section IV details the conceptual solution and its architecture and workflow. Section V reports the case study performed on Eclipse distribution platform to assess the proposed solution, and Section VI makes the conclusions based on UDPSLE and summary of UBSPLE-Based Eclipse Distribution Platform.

II. Conceptual Background

A. Software Product Line Engineering (SPLE)

SPLE is a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization [2]. The combination of mass customization and a common platform allows the reuse of a technology common base and, at the same time, to bring out products in close accordance with customer requirements [2]. There are four different parts in SPLE:

- Software asset inputs: a collection of software assets [7];
- Decision model: it could be FM or domain-specific languages (DSLs) [8], [9];
- Production mechanism and process: the means for composing and configuring products from the software assets ;
- Software output: The final industrial software distributions

Zhaopeng Xuan is student with the ECE Paris Engineering School (phone: 0033-770527208; e-mail: xuan@ece.fr).

Yuan Bian, C. Cailleaux, Jing Qin and S. Traore are students with the ECE Paris Engineering School (e-mail:bian@ece.fr).

In addition, SPLE process consists of two main lines of activities:

- Domain Engineering, focusing on development of the core assets for reuse
- Application Engineering, focusing on final products development using the core assets according to customer requirements

Variability management is the distinctive feature of SPLE. This concept holds two dimensions [2]:

- Variability in time, in which a software artifact evolves through different versions.
- Variability in space, in which the artifact takes different shapes at the same time.

SPLE is mostly concerned with variability in space. To model variability, FODA must be performed in order to capture commonalities and variability during the requirements analysis phase. It defines a feature as *"a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems"* [14]. In this respect, a Feature Model captures the commonalities and manages the variable features of systems, in a family of systems or in a product line [10].

According to [5], SPLE is not widely spread in open source community. Although this has been a long-time preoccupation [7], [8], reports of successful cases of product line engineering practice in open source community do not abound.

B. Component-Based Software Engineering (CBSE)

The primary role of CBSE is to address the development of systems as an assembly of parts (components), the development of parts as reusable entities, and the maintenance and upgrading of systems by customizing and replacing such parts. This requires established methodologies and tool support covering the entire component and system lifecycle including technological, organizational, marketing, legal, and other aspects [3]. Known software implementing the CBSE paradigm is Eclipse IDE, NetBeans and modern web-browsers (Chrome, Firefox, and Safari).

As mentioned above, Eclipse IDE is a platform based on CBSE with a large family of components, which support 12 distributions with fixed version of components. In this paper, Eclipse will be used as a case study to illustrate present proposed concept.

III. Problem Statement

From CBSE perspective, core conception is to identify components and dependencies. CBSE describes the product from component level and developer perspective. As a consequence, a user cannot customize its own production. Such as the color of their car, users could choose red or black color, but the manufacturers just provide some chemical materials which are composition of these two colors for users to choose, and nearly no one could get the expected final product. This analogy describes the main problem of CBSE. With rising number of components for large families of software, not only the clients do not understand the way to build, but also for manufacturers, it will be more difficult to manage them and the dependencies. Thus there are four obvious problems for CBSE-based product:

- Engineers have difficulty to manage all components in distributed repositories, especially for third-party components.
- Some functions in CBSE-based product are rarely used by users.
- Users can hardly find a required component from scratch on demand.
- The product with all components requires high performance of disk and memory of computer.

Thus for CBSE-based Eclipse, with rapidly rising number of components dispatched in a fixed and limited set of distributions, users with specific needs often do not find a suitable Eclipse distribution from the official website. They are even at a loss trying to figure out which distribution best suits their needs. Building a customized Eclipse distribution from scratch is not an option for most users, so they end up downloading the distribution that matches the most their profile, to the best of their judgment. More than often this is followed by a long, confusing and sometimes frustrating upgrading procedure by installing missing features from Eclipse or third-party repositories. So this paper is aimed at explaining the conception of UDSPLE proposed by the Eclipse case.

IV. Effective Solution

Solution due to FM was proposed as a part of the FODA method. Meanwhile, based on the conception of SPLE, FM is suggested as a method to describe the problem space [8], [11]. Since then, SPLE has been suggested, and as FM could adapt most part of industrial software, the proposed solution could be applied in most domains, as explained below.

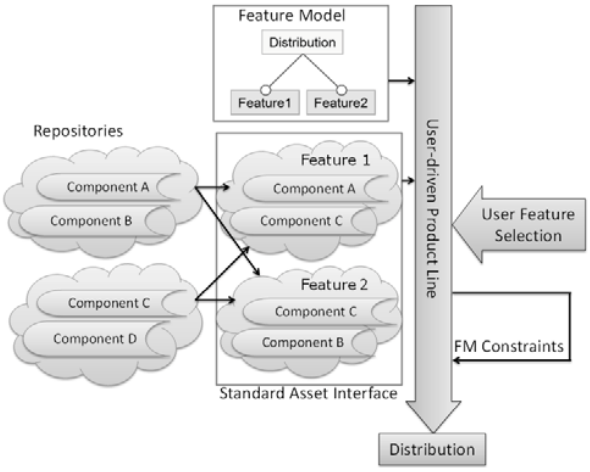

Fig. 1 Software product line process

The UDSPLE process is displayed in Fig. 1, where are described the four parts of Software Product Line (SPL) [6], defined in Section II *A*. It is proposed that at early stages of software development, FM provides the basis for scoping the system family by recording and assessing information such as

what features are important to enter a new market, or to remain in an existing market, what features incur a technological risk, what is the projected development cost of each feature, and so forth [8], [11]. Besides, FM proposes a standard interface shared with developers and users, and provides better knowledge for dependencies, not only for official development group, but also for third part development ones. The following sub-sections explain how SPL works in Eclipse Distribution system.

A. Automatic Dependency Derivation

Very simple Eclipse FM has been built up, in Fig. 2, it is described by Eclipse Feature IDE and presents Eclipse FM as a Feature Diagram (FD), a family of popular modeling languages used for engineering requirements in SPL [12].

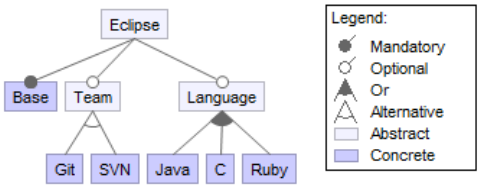

Fig. 2 Eclipse features diagram

Each block in Fig. 2 is a feature in FM, and the FM process will automatically make the following derivation group to create FM constraints [4], [13].

$Root \Leftrightarrow Eclipse$
$Base \Leftrightarrow Eclipse$
$Team \Rightarrow Eclipse$
$Language \Rightarrow Eclipse$
$(GIT \vee SVN \Rightarrow Team) \wedge\neg (GIT \wedge SVN)$
$Java \vee C \vee Ruby$

From now, a benchmark has been defined without assets for Eclipse that provides a basis for scoping system family. Meanwhile, FM has proposed dependencies and constraints among all user-viable aspects.

B. Integration and Configuration of Software Assets

In Fig. 3, Eclipse bundles or components are pro-vided from multiple repository clouds. In Eclipse embedded provisioning system (P2), it is only needed to select one remote repository each time for installing or updating some bundles into Eclipse. Also users need to find the URL of repository by themselves, as Eclipse organization does not know about existence of other third part group repositories. To simplify users operation and to improve components management, an approach is here proposed to integrate all software input assets from multiple official or non-official repositories at first, which are presented as Standard Asset Interface in Fig. 3. Eclipse repository is constructed by P2 layout which is a specified format. So a particular tool called Repository Analysis System (RAS) is created to analyze and collect resources from multiple repositories at the same time with a given repository URL list. It is proposed for each development team to register its repository URL before publishing new components.

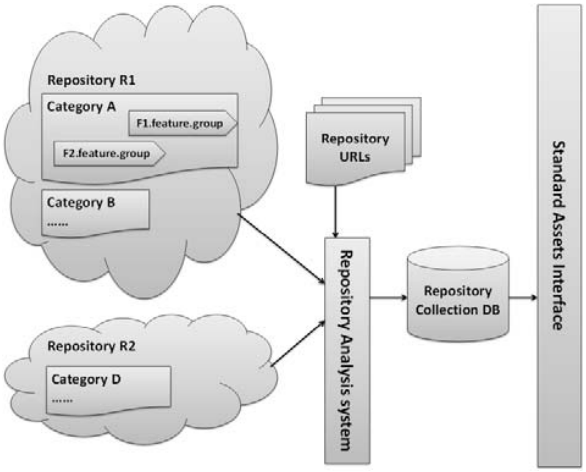

Fig. 3 Architecture for integration of software input assets

After integration of software input assets in Eclipse distribution system, a Web interface is proposed for administrators to configure the assets for each Feature presented in Fig. 3. Until now, this paper has explained the UDSPLE for Eclipse Distribution Platform. It could be noticed that UDSPLEA indirectly expose the build-up process to users to lead a building process. Meanwhile, the development plan and scope are described by FM, the development artifacts are in fact still located in remote clouds, created and maintained by multiple distributed teams. On another hand, through a web interface, the users become final actors as release managers to assembly the production based on their individual needs.

C. Assembling Process for Users

As noticed in Section III *A* and Fig. 3, in SPL, one import part is Production mechanism and process, which is presented as FM constraints and User Feature Selections. The former is defined by FM in Section III *A*, and the User Feature Selections could be defined as a file which includes all selected and dependent Features. The selection of Feature means that UDSPL will automatically expose the functions of selected features and corresponding dependent ones. For instance, if user selects Feature GIT in Fig. 2, based on actual FM constraints, User Feature Selection actually contains Feature Eclipse, Feature Base, Feature Team and Feature GIT. Since then, UDSPL composer will compose the assets of selected Features to produce a customized Eclipse distribution. This means the user will drive the build-up process and build components by selecting Features from FD, and then FM will valid users selection based on the constraints to add or to remove dependent Features, or show up a warning when user selection violates FM constraints.

Thus, upon Features selection by the user, Eclipse distribution system has already a record of requirements before using the software. It provides a much earlier step for analyzing the value of each Feature in the market and for optimizing FM to provide more suitable architecture.

At last the composer of SPL is changed by the type of input assets. So in this paper, attention will be given the technical aspect of composer for Eclipse artifacts which will be explained in next part.

V. Case Study and Discussion

Based on UDSPLE and relevant technologies, an Eclipse distribution platform has been successfully implemented where the user can find features and get customized Eclipse distributions easily. This implementation expresses author's idea, and also reflects the value of UDSPLE.

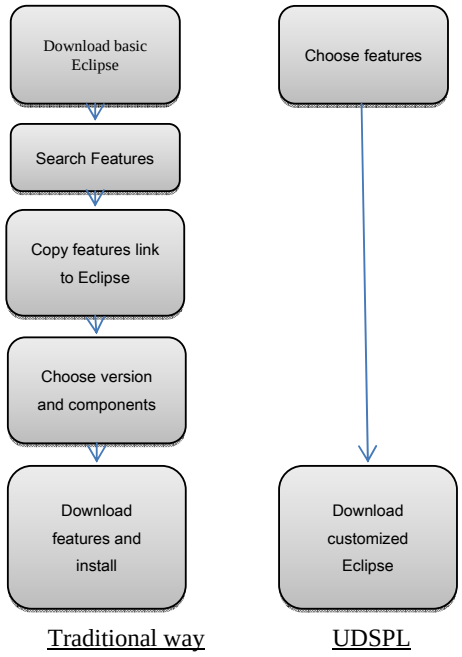

Fig. 4 Comparison of methods to get customized eclipse

After implementation and tests, the two ways of having customized Eclipse distribution by traditional Eclipse components and by UDSPL are compared, see Fig. 4. There are some advantages and progress from user perspective. Before, a user might spend hours searching online and fix compatibility problem to get some specified Eclipse features. With UDSPL platform, the user can select features and download his Eclipse directly and save considerable time and efforts while ensuring software high quality, because the platform solves all compatibility problems for the user. Moreover, UDSPL is of great potential because by using the latest architecture and data science technologies such as HDFS, performance could be much better quality.

VI. Conclusion

In order to both catch up with the high speed of software development and satisfy user requirement, easy-to-run user-driven software product line engineering has been presented in which end users can define and trigger actual software distribution build-up on demand. The main idea is to build up this distribution with a small set of actually needed features running on commodity hardware, instead of directly distributing software with fixed bundles. By enabling the user to choose the needed features according to his profile or preferences before actual distribution building, one can significantly decrease the time wasted on slowly running update software and simplify the choice amongst various features. Also in this way, software resources can be efficiently managed and highly reused.

The case implementation of Eclipse platform based on presented fashionable user-driven software product line strongly supports the feasibility of proposed idea. With less difficulties and troubles, Eclipse users could get their expected distribution from Eclipse Distribution System instead of downloading the official version from Eclipse Website. In other words, the Eclipse Distribution System could replace the function of Eclipse official downloads, and Eclipse organization does not need to maintain the distribution of 14 fixed versions anymore. Finally, from business perspective, the Software Product Line extends the range of users by letting those select or buys their actually needed features, rather than force them to buy all features together no matter whether they are really necessary for their specific purpose.

As a final conclusion, by adopting this approach, software development organizations will further reduce maintenance cost of multiple pre-defined software distributions. So they will spend less time and effort in releasing trains, leaving more time for improving the quality of proposed features and for such a process more frequent releases of innovative features.

Acknowledgment

The authors are very much indebted to ECE Paris School of Engineering for having provided the environment where the work has been performed, to Drs L.M. Hillah and T.Ziadi, CNRS UMR 7606 - LIP6, Univ. Paris Ouest and Sorbonne Univ. Paris 06, for their constant help in the research, to DrJ. Templemore for guidance, and Pr. M. Cotsaftis for preparation of the manuscript.

References

[1] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson: *Feature-Oriented Domain Analysis* (FODA) *Feasibility Study*, Software Engineering Institute (SEI), Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.
[2] K. Pohl, G. Bockle, F. J. VdLinden: Software Product Line Engineering: Foundations, Principles and Techniques, 2000.
[3] I. Crnkovic, "Component-based software engineering - new challenges in software development," in Information Technology Interfaces, 2003. ITI 2003. Proceedings of the 25th International Conference on, June 2003, pp. 9–18.
[4] D. Batory, "Feature models, grammars, and propositional formulas.
[5] J. van Gurp, C. Prehofer, and J. Bosch, "Comparing Practices for Reuse in Integration-oriented Software Product Lines and Large Open Source Software Projects," Softw. Pract. Exper., vol. 40, no. 4, pp. 285–312, Apr. 2010.
[6] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.
[7] K. Czarnecki, Th.Bednasch, P. Unger, U.Eisenecker: Generative Programming for Embedded Software: An Industrial Experience Report, Proc.ACM SIGPLAN/SIG- SOFT Conf. on Generative Programming and Component Engineering (GPCE'02), pp. 156 –172, 2002; K.

Czarnecki, U. Eisenecker :*Generative Programming – Methods, Tools, and Applications.* Addison-Wesley, Boston, MA, 2000.

[8] M.Dalgamo, Danilo Beuche :*Software Product Line Engineering with FeatureModel*s, Proc. Intern. Software Product Line Conf. Tutorial SPLC 2013, Tokyo, August 26-30, 2013; A. Hein, M. Schlick, R. Vinga-Martins : Applying Feature Models in Industrial Settings,in *Software Product Lines: Experience and Research Directions*, Proc. 1st Software Product Line Conference - SPLC1, P.Donohoe, Kluwer Academic Publishers, pp.47-70, 2000;O.Spinczyk, DaniloBeuche: Modeling and Building Software Product Lines with Eclipse,OOPSLA Companion, pp.18-19, 2004.

[9] C.W. Krueger: New Methods in Software Product Line Development, *Proc. 10thIntern. Software Product Line Conf.*, pp.95-99, 2006.

[10] K.Czarnecki: *Staged Configuration Using Feature Models*, Springer, Berlin, 2004.

[11] S.K. DeBaud, J.M: A *Systematic Approach to Derive the Scope of Software Product Lines*, Proc. 21st Intern. Conf. on Software Engineering (ICSE), pp. 34 – 43, 1999.

[12] P.-Y.Schobbens, P. Heymans, J.-C.Trigaux : Feature Diagrams:A Survey and Formal Semantics, Requirements Engineering, *Proc. 14th IEEE Intern. Conf.* , pp.139-148, 2006.

[13] Batory: Feature Models, Grammars, and Propositional Formulas, *Proc. SPLC 9th Intern. Conf. on Software Product Lines*, pp.7-20, Springer-Verlag, Berlin, 2005.

[14] P. Clements, L. Northrop: Software Product Lines: Practices and Patterns, Addison-Wesley, 2001.