Social Software Product Lines

Raian Ali[1], Carlos Solis[1], Fabiano Dalpiaz[2], Walid Maalej[3], Paolo Giorgini[2] and Bashar Nuseibeh[1, 4]

[1] *Lero – The Irish Software Engineering Research Centre, Ireland.* [2] *University of Trento, Italy.* [3] *Technische Universtität München, Germany.* [4] *The Open University, UK*

Abstract*. Software product lines are an engineering paradigm meant to systematically configure software products of reusable assets so that development effort and time are minimized. Configuring a high-quality product is a challenging design activity, mainly because quality is a dynamic property and hardly predictable by designers at design time. In this position paper, we propose Social Software Product Lines (SSPL) as a new development paradigm which involves users as collaborators in judging software products quality and guiding configuration in a lifelong style. SSPL paradigm advocates two principles. The first is that quality has to be evaluated iteratively during the product operation so that quality evaluation is maintained up-to-date. The second is that users are the main evaluator of quality and their feedback is a primitive driver of reconfiguration. At runtime, SSPL keeps obtaining users' quality feedback and planning upon reconfiguration to deliver the product shown most adequate by the users' community. We discuss motivations and foundations of SSPL and outline a set of research directions.*

Keywords- Software Product Lines; Social Software Engineering; Models at Runtime, Users feedback.

I. INTRODUCTION

Software product lines engineering (SPLE) is a software engineering paradigm, which aims to construct products by configuration of reusable software assets [1]. It is based on capturing the commonality and the variability between the possible products belonging to a certain domain. A product family consists of a space of product configurations and the assets needed to implement each configuration. Products are generated by a systematic derivation of a configuration from the product family. SPLE helps to minimize costs by not starting development from scratch and to accommodate users' diversity by offering selectable multi-configurations.

Traditionally, products configuration is a design time activity driven by specific requirements elicited from prospective users and guided by certain common practice rules [2]. Design time configuration is appropriate when the software to produce is not subject to frequent changes. For example, university library and hotel booking systems would not change often, as the domain is well-known and it exhibits stable rules. However, other application areas, such as mobile applications, would rather be subject to multiple changes during operation as users' demands and trends besides competitive technology would be rapidly varying. Dynamic Software Product Lines (DSPL) paradigm aims to cope with such changes via autonomous product reconfiguration at runtime so reconfiguration costs and time are minimized [3].

Several factors influence the configuration of software products such as organizational rules, law, user preferences, required resources, usage cost, and the context wherein software operates. Some of these factors are static, which makes configuration decisions possible at design time. Some other factors are volatile, which necessitates a lifelong reconfiguration to ensure the up-to-dateness of the derived product. For example, context is a configuration driver, which influences the applicability of each software product [4,5]. Context changes at runtime might activate certain requirements and can also limit the space of configurations which are applicable and able to reach the set of activated requirements.

In line with the view presented in [6], social feedback could also make users collaborators in the product configuration process. It mainly concerns the users' judgment of the quality of a product as a means to reach their requirements. Users of each software product provide social feedback to express their satisfaction degree concerning the quality of that product. Social feedback is unpredictable by designers, varies over time, and is often un-monitorable by relying on solely automated means. These properties make it a primitive driver for products configuration, which is irreplaceable by other means. Social feedback is the main ingredient for the collective judgment of users' community about a software product. One of the ultimate goals of configuration process is to choose a software product, out of a space of available products, which maximizes the social collective satisfaction about quality. Such configuration is desirably achievable autonomously by the system at runtime to minimize the costs of the manual iterative configuration.

In this position paper, we propose Social Software Product Lines (SSPL) as a new software development paradigm which treats users as collaborators in the configuration of software products. SSPL enables obtaining users' quality feedback and analyzes it to reconfigure products in a lifelong style. Users provide their feedback about each configuration so that the most appropriate configurations will be applied and vice versa. Thus, configuration is guided by the collective judgment of the users' community at runtime. We present a motivating example in Section 2, discuss the main principles of SSPL in Section 3, enumerate preconditions for applying SSPL as a development paradigm in Section 4, discuss a set of research challenges in Section 5, and conclude the paper in Section 6.

II. MOTIVATING SCENARIO

We consider the development of assistance software to help overseas students about the typical procedures they need to go through when starting their study (registration, accommodation, immigration office, etc.). The software can be configured to deliver the assistance in different ways. One configuration is based on the use of *automated assistance*, which includes demos and intelligent agents, etc. Another configuration is based on *personal assistance*, which establishes a remote connection with one of the volunteer students who knows about the requested procedure. Each of these two configurations is a high level description as it incorporates different other sub-configurations in turn.

The development team is uncertain about the right configuration of software with regards to each of the possible procedures. That is to say, the decision about the correct software product to generate is uncertain. For example, some procedures are complex and might require *personal assistance* rather than *automated assistance* and vice versa. Moreover, the configuration seeming to be correct currently may not remain infinitely correct. For example, if currently the use of *personal assistance* has some social implications affecting negatively its quality, this might change over time as students and volunteers may become more comfortable with it due to some cultural changes, or vice versa.

As a solution, the development team would leave the decision between configurations to the users (*students* and *volunteers*) themselves. The decision is taken by users collectively and iteratively during the software operation. The way to do that is to allow users to express their judgment of the quality of each configuration (*personal assistance*, *automated assistance*) for each procedure (*registration, accommodation, etc.*). The configuration shown to be more adequate for a certain procedure will be the one to produce when assistance about that procedure is asked for. This process has to be iterative so configuration is able to cope with changes that may happen in users' judgment over time.

III. SSPL: FOUNDATIONS & ARTIFACTS

SSPL is a software development paradigm in which users are given a voice by treating their quality feedback as a main driver for configuration. In SSPL, the product is not delivered statically and the configuration activity is not done only once at design time. The product is delivered dynamically and the configuration is iteratively done at runtime. The configuration is a lifelong process guided by the feedback the users provide about the use of each product configuration. Thus, the product to deliver at a certain time is derived in light of the collective quality judgment provided by the users' community so far. Figure 1 outlines SSPL configuration loop. SSPL analyzes users' feedback about each configuration and elects upon the configuration shown to have the best quality according to the social feedback, operates it and obtains the feedback from the users of that operation. These activities are done in a lifelong style so the best configurations will be socially selected similarly to natural selection in biology where species (configurations) fitting their environment (users' community) survive.

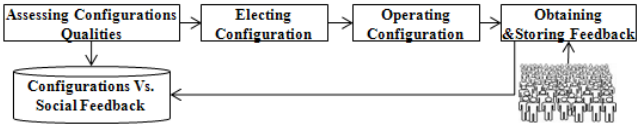

Figure 1 Social Software Product Line Loop

The crux of SSPL is the treatment of social feedback as a primitive driver for configuration. Social feedback is essential for a feasible and correct product configuration in software product lines for the following reasons:

1. *Uncertainty*. The design decision about the configuration to enact aiming for a good-quality product is often taken under uncertainty. Designers are not the main quality judges. Quality is rather judged by users. Users' quality feedback is essential to validate each configuration so that uncertainty is coped with via involving users in taking the design decisions. For example, the development team may not be certain if *automated assistance* is better configuration in comparison to *personal assistance* for students who ask for help regarding complex procedures. Thus, students themselves will be the quality assessors and the decision will be taken by them collectively.

2. *Un-monitorability*. Users' quality feedback concerns beliefs they have in mind, which is in most cases undecidable via other means than the explicit disclosure of users. Thus, such feedback is often non-inferable by relying on only automated means. For example, whether students and volunteers find *personal assistance* configuration efficient and comfortable is a personal judgment, which requires those users to provide it. Automated monitoring and analysis of the attitude of users could partially infer their quality feedback. However, this helps to minimize the amount of feedback and the intervention of users but does not replace it entirely in the general case.

3. *High-variability*. The number of configurations incorporated in a product family could be high in large-scale systems. This means that validating all these configurations at design time is a hard and time consuming activity which influences the delivery of product on time and the development costs as well. SSPL allows for crowd-sourcing the validation of the varying configurations so that development time and costs are minimized. For example, each of the configurations in the assistance software (*personal assistance* and *automated assistance*) is a high level description of multiple other configurations (imagine a Feature Model [7] of this system). The *personal assistance* might have variations for the way of establishing communication (voice, video, via public or designated messengers) and explanation (supported by demos, presenta-

tions, remote screen sharing etc.) and other functionalities. The large number of possible configurations could be rapidly validated when the users' community (students and volunteers) is part of the validation team while using the system.

4. *Socialization vs. Personalization*. Personalization and socialization are two different mechanisms with the same goal: the fitness to users. SSPL reflects the collective judgment of users' community while personalization customizes software to the characteristics of individuals. While socialization does not replace personalization, it is essential where the system is highly variable and the individual users use the system for a relatively limited number of times. For example, a student would use the assistance software for one time to register to the university library, so it is infeasible to treat him individually and try all configurations to customize the software to him. Rather, the product line will use the feedback provided by all students who used the system in the past and benefit from that when new students ask for assistance and so on.

5. *Lifelong reconfiguration*. The quality of a product configuration is not a static property. The configuration which is proved to have a high quality at certain stage might turn to have a lower quality when time passes or vice versa. In SSPL, users will be allowed to give quality feedback during operation which allows for a continuous quality assessment of each configuration. Thus, the product line can cope with the unpredictable reasons which influence the quality (changes in trends, competitive technology, organizational settings, etc.) by responding to the feedback provided by users in a lifelong style. For example, the usage of voice recognition to help users in filling in the forms might be currently judged uncomfortable. When time passes, users may become more familiar with this technique and, thus, judge it differently. SSPL allows for coping with such lifelong dynamicity.

Figure 2 shows the main artifacts required to realize our proposed SSPL loop. The upper part contains design time artifacts (*Configuration*, *Quality Attribute*, *Context Attribute*) specified by designers and having static values. The lower part contains runtime artifacts (*Operation*, *Quality Feedback*, *Monitored Value*) specified by designers with values obtained at runtime.

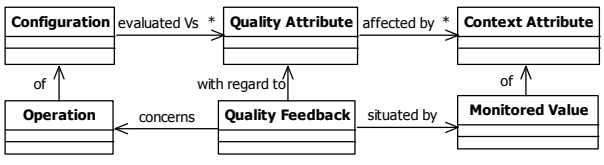

Figure 2 SSPL Main Artifacts

Configuration represents a synthesis of functionalities intended to reach the system main objectives. Feature model is a well-known technique for incorporating various configurations of distinguished characteristics of the systems, features, in one compact hierarchy [7]. *Quality Attribute* is a distinguished characteristic of the degree of excellence of a configuration. For example, *"comfortable"* and *"fast"* are quality attributes for each configuration of the student assistance software described earlier. *Context Attribute* is a distinguished characteristic of the environment within which the system operates which influences the quality of a configuration against certain quality attribute. For example, *"the student and volunteer speak the same native language"* would influence the quality attribute *"comfortable"* of the configuration *personal assistance*.

Operation is a single execution instance of a configuration. *Quality Feedback* is an assessment given by a user interacting with an operation about its quality against a certain quality attribute. *Monitored Value* is the value of a context attribute at the time of operation and feedback. For example, to assist a student about the immigration office procedure, the product line may operates (upon analyzing social feedback) a *personal assistance* configuration and establish connection with a volunteer. The monitored value of the context attribute *"the student and volunteer speak the same native language"* could be "*No"* and the feedback obtained from that student concerning the quality attribute *"comfortable"* could be *"medium"*.

IV. When to Apply SSPL

In this section, we discuss a number of preconditions for adopting SSPL as a development method. **First**, configurations should be different from the perspective of end users. End users usually do not understand the technical differences between configurations. They perceive visible features of software and thus the users' feedback is meaningful if the configurations differ at the feature level. **Second**, a significant portion of the users' community is willing to give feedback so that collective judgment is achievable. **Third**, feedback is not a subject to frequent radical changes, which lead to select inadequate configurations for a transitional period. For example, the crowd trend regarding mobile application is subject to rapid changes and quality feedback might change radically and rapidly. Thus, reconfiguration may potentially pass periods SSPL decisions are highly incorrect. **Fourth**, privacy and trust concerns are manageable. SSPL requires monitoring, amongst other things, the context attributes which include personal characteristics of users who are themselves a part of the system environment. When privacy compliance and trusting the system is unlikely to be achieved within reasonable time, users may refuse to provide feedback and may refuse to allow SSPL to monitor their context and, thus, SSPL is unlikely to work. **Fifth**, wrong decisions shall not lead to serious problems (e.g. critical domains are not supported). Indeed, SSPL is based on estimation of the collective judgment of users; thus, probability of wrong decisions is always there.

V. RESEARCH CHALLENGES

SSPL adopts openness-to-the-crowd principles so that configuration activity is crowd-sourced. On the one hand, this could reduce the responsibility of the development team, allow for a rapid and effortless and up-to-date configuration process, and also give the users the liberty to make their own choices. On the other hand, obtaining users' feedback and making use of it, is challenging for several reasons:

1. *Users Diversity*. Ideally, the majority of users provide similar quality feedback for a configuration under the same values of context attributes. However, when this is not the case and the quality judgment deviation between users is high, then the incorrectness probability of the collective judgment is also high. In other words, if the consensus of users' community is missing, SSPL cannot rely on their feedback. Thus, we need to devise analysis methods, measures, and rules that help to predict the significance of the collective judgment and devise strategies to cope with situations where significance is low. One strategy is to make configuration subject to runtime dialogue with the user, i.e., the configuration is done interactively.
2. *Specification*. The specification of SSPL artifacts is a hard task as it requires the designers to take some design decisions on behalf of the users' community. The designers might define a set of quality and context attributes which is incomplete, redundant, or irrelevant. For example, students and volunteers might find a quality attribute like "anonymity" relevant/irrelevant as opposite to the designers' specification. Users might find an attribute such as "user friendly" synonym to or part of another attribute such as "comfortable" differently from the designers' who specified these two attributes as unrelated. A valid SSPL specification might be achieved by strategies like allowing users themselves to define relevant quality and context attributes at runtime. Thus, the users play the role of designers besides the role of evaluators of the configurations quality.
3. *Judging the Unknown*. Users might judge a configuration without being aware of the other alternative configurations available in the product family. Hypothetically, the more the users know, the more significant their judgments are. For example, a student quality judgment of the *automated assistance* configuration may differ depending on whether he knows or not about the existence of *personal assistance* configuration. However, enforcing that user is aware of all available configurations is infeasible. This is because, practically, the user does have the ability to try or compare all configurations and this is also not one of his main concerns when using a system. Increasing the significance of feedback via maximizing the awareness about the space of available configurations is a research challenge.
4. *Transparency vs. Accuracy*. Users' feedback is given in and affected by certain environmental settings which we represented via context attributes. Similarly to the feedback, the values of these attributes are not always obtainable by relying on solely automated means and may require the intervention of users as well. For example, *"volunteer is busy"* is a context attribute which may influence students' judgment of the configuration *personal assistance* against a quality attribute like *"fast"*. Monitoring if the volunteer is busy could not be fully done by automated means and might require the volunteer to provide the value of this context attribute. Minimizing the size of input which the users need to provide for quality feedback and context values in order to maximize the computing transparency without losing accuracy is another research challenge.

VI. CONCLUSIONS

We have proposed SSPL as a new paradigm of software development, which build on the top of traditional software product line and dynamic software product lines principles and give the users a voice when configuring products. We argued that the role of users is primitive and cannot be replaced by other means. We have outlined the big picture, principles, motivations, preconditions and research challenges of SSPL. Giving users a voice in guiding adaptation, either at design time or at runtime, is the broader research area that we aimed at introducing in light of software product lines. Besides the potential benefits of enabling users to drive adaptation, a spectrum of research challenges is to be faced. Our future research consists on devising methods to engineer social intervention and weave it as a main component of the engineering of the whole adaptive system.

Acknowledgement This work has been partially funded by the EU Commission through the FastFix, Aniketos, and NESSOS projects, and by Science Foundation Ireland grant 10/CE/I1855.

REFERENCES

[1] Pohl, K.; Böckle, G. & Van Der Linden, F. Software product line engineering: foundations, principles, and techniques. Springer-Verlag New York Inc, 2005.

[2] Czarnecki, K.; Helsen, S. & Eisenecker, U. Staged configuration through specialization and multilevel configuration of feature models Software Process: Improvement and Practice, 2005, 10, 143-169.

[3] Hallsteinsen, S.; Hinchey, M.; Park, S. & Schmid, K. Dynamic software product lines Computer, IEEE, 2008, 41, 93-95.

[4] Ali, R.; Yu, Y.; Chitchyan, R.; Nhlabatsi, A. & Giorgini, P. Towards a Unified Framework for Contextual Variability in Requirements. In IWSPM'09.

[5] Ali, R.; Chitchyan, R.; & Giorgini, P. Context for Goal-level Product Line Derivation. In DSPL'09.

[6] W. Maalej; H-J Happel, A Rashid, When Users Become Collaborators: Towards Continuous and Context-Aware User Input, In OOPSLA'09.

[7] Kang, K.C. and Cohen, S.G. and Hess, J.A. and Novak, W.E. and Peterson, A.S., "Feature-oriented domain analysis (FODA) feasibility study", CMU/SEI-90-TR-021, Carnegie Mellon University, 1990.