The Journal of Systems and Software 83 (2010) 2496–2512

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Decision support for moving from a single product to a product portfolio in evolving software systems

Muhammad Irfan Ullah [a,*], Günther Ruhe [a,b], Vahid Garousi [b]

[a] Department of Computer Science, University of Calgary, Calgary, Alberta T2N 1N4, Canada
[b] Department of Electrical and Computer Engineering, University of Calgary, Calgary, Alberta T2N 1N4, Canada

ARTICLE INFO

Article history:
Received 14 February 2010
Received in revised form 25 May 2010
Accepted 21 July 2010
Available online 5 August 2010

Keywords:
Software product lines
Software product evolution
Software product management
Decision support
Behavioral analysis
Open-source systems

ABSTRACT

Successful software systems continuously evolve to accommodate ever-changing needs of customers. Accommodating the feature requests of all the customers in a single product increases the risks and costs of software maintenance. A possible approach to mitigate these risks is to transition the evolving software system (ESS) from a single system to a portfolio of related product variants, each addressing a specific customers' segment. This evolution should be conducted such that the extent of modifications required in ESS's structure is reduced. The proposed method COPE+ uses preferences of customers on product features to generate multiple product portfolios each containing one product variant per segment of customers. Recommendations are given to the decision maker to update the product portfolios based on structural analysis of ESS. Product portfolios are compared with the ESS using statechart representations to identify the level of similarity in their behaviors. A proof of concept is presented by application to an open-source text editing system. Structural and behavioral analysis of candidate portfolios helped the decision maker to select one portfolio out of three candidates.

1. Introduction

Successful software systems continuously evolve to accommodate new features. The variation of customers' needs and increased system complexity triggers the creation of product lines. A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission (Clements and Northrop, 2001). SPL is a viable approach if a development organization plans to target a wide and diverse customer-base. Typically SPLs evolve from existing products or systems that are successful and therefore attract customers from a wide variety of domains. A number of real-world software systems show the presence of this phenomenon, e.g., Microsoft Windows operating system: Windows Vista offers separate product variants for the *home* (Starter, Basic, Premium), *business* (Business, Enterprise) and *power users* (Ultimate) market segments.

According to Helferich et al. (2006) most of the existing SPL development methodologies either focus on technical details (Schmid, 2002; Kishi et al., 2002) without involving customers in product line definition process or they identify marketing techniques for this purpose without prescribing how to translate the results of these techniques into tangible products (Rommes, 2003). The second issue with existing methods is that they assume the product line is being developed from scratch, hence they do not consider ESS's structural information (code and design) while defining product variants (Moraes et al., 2009). As a note on terminology, in this paper we use the term product variant to refer to an individual product in a SPL.

The proposed method COPE+ (Customer Oriented Product Evolution) attempts to address these shortcomings for the specific evolution scenario when a single ESS is evolved into a product line. It generates solutions by considering both the business and the technical perspectives. Note that there are a huge number of organizational, financial, technical and customer related criteria that can influence product evolution decisions. Given the uncertainty in the data and dynamically changing environment, it is neither our intention nor feasible to propose the best possible solution to the decision problem addressed in this paper. The results of COPE+ should be considered as a first-level base plan to initiate the decision-making process for product evolution. The main contributions of this paper are:

(1) The decision support method COPE+.
(2) Application of an evolutionary algorithm to analyze the structure of an ESS.

* Corresponding author. Tel.: +1 403 210 9708; fax: +1 403 284 4707.
E-mail addresses: miullah@ucalgary.ca (M.I. Ullah), ruhe@ucalgary.ca (G. Ruhe), vgarousi@ucalgary.ca (V. Garousi).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.07.049

(3) A proof of concept of the method COPE+ using an open-source software system.

The remainder of this paper is organized as follows: Section 2 presents the research questions. Section 3 presents some of the existing works related to proposed solution approach. Section 4 presents the method COPE+. Section 5 illustrates the method by application on an open-source software system (jEdit). Section 6 discusses the applicability and value of COPE+ and Section 7 presents future work.

2. Research questions

The fundamental research question addressed in this paper is: "Should an evolving software system (ESS) facing feature requests from a diverse customer-base be transitioned into a product line consisting of a portfolio of product variants, and if the answer is *yes*, how this should be done?" The question is part of a release-planning scenario with a decision to be made for the forthcoming release of an ESS.

It is assumed that the list of new features to be added in the ESS, the voting of customers on these features (using one or more criteria on a pre-defined scale) and the impact of each one of these features on the ESS's structure are given as input to the solution method. The system structure representation is based on the *Module View* of the architecture as defined by Bass et al. (2003). This is a non-runtime representation of architectural elements (modules).

The goal is to use these inputs to find customer segments (if any) and propose product variants to meet customers' expectations by modifying a minimum number of modules[1] of the existing system structure. The decision maker is also presented with the level of impact of each feature on ESS's modules.

The specific research questions addressed in this paper are:

RQ1: From a given set of candidate product portfolios for enhancing an evolving software system, which one minimizes the structural impact on the set of modules representing the evolving software system?

RQ2: For the purpose of product evolution, how to determine the level of behavioral conformance of the candidate product variants (in a given product portfolio) with the evolving software system?

3. Related work

There are four main areas that relate to the method proposed in this paper. These areas are scoping, decision support for software product evolution, structural analysis for system evolution and behavioral modeling. In this section, we present a brief overview of works in these areas and their relationship with the proposed method.

3.1. Scoping

Almost all systems development methods perform scoping at initial product definition stage to identify what (functionality) will be part of the system and what (functionality) will not be part of the system. This activity becomes even more important in SPL development since a product line contains multiple product variants. Most frameworks and methodologies for SPL development include scoping as a distinct activity. It is used to identify products within the product line (product portfolio scoping), features of these products as well as the features that will be developed for reuse (asset scoping) (Schmid, 2002). Since, the proposed method COPE+ determines portfolio of products, therefore, we will only consider the scoping techniques that generate such results. The scoping techniques that only determine asset scoping are not being analyzed.

There is a broad range of scoping techniques available for SPL development. On one side of the spectrum, there exist relatively informal techniques which rely on surveys, workshops and other *Market Analysis* techniques to elicit scope of the product line. An example of such techniques is the scoping activity in the Software Product Line Practice Framework (PLP) (Clements and Northrop, 2001). On the other side of the spectrum, there are very technical and formalized scoping techniques. PULSE-Eco v2.0 is one such approach (Schmid, 2002). It relies on technical details such as economic benefit analysis to define products in the SPL. Clements and Northrop (2001) combine five *practice areas* (understanding relevant domains, market analysis, scoping, technology forecasting, building a business case) to formulate the *What-to-Build Pattern* that helps an organization determine what products ought to be in its software product line. Northrop and Clements (2007) present PLP Framework v5.0, the most recent version of this framework. They report that one of the three major risks in scoping is "essential stakeholders do not participate". PLP Framework v5.0 proposes *Customer Interface Management* practice area to engage customers in the product line scoping. Among other suggestions, Northrop and Clements (2007) recommend establishing user groups to help product line development organizations identify and prioritize customers' needs. They also note that customers with their own agenda can dominate such user groups and workshops to influence scope of the products. Hence, the risk of all the customers not involved in the scoping process is still not mitigated. COPE+ explicitly involves all the customers to vote on the proposed new features. Since customers can do this independently from each other, therefore, the risk of some of them forcing their agenda on others is not possible. Note that customers' involvement means the results generated by COPE+ are based on inputs (votes) of all the customers on product features. During the execution of the method, there is no customer involvement.

QFD-PPP proposed by Helferich et al. (2005) targets a research question similar to the one addressed by COPE+, however, there is a fundamental difference between the two approaches. An implicit assumption in QFD-PPP seems to be "greenfield" development, since they do not analyze impact on system's structure. Greenfield development assumes building entirely new systems from scratch. Kishi et al. (2002) present a decision-making framework for product line scoping. Requirements for individual product variants and the product line are evaluated using architectural alternatives. Final scope is defined based on whether the decision maker favors individual product's optimization or the product line's optimization for a given architectural candidate. There is no stakeholder involvement in the scoping process proposed by Kishi et al. (2002). Rommes (2003) proposes a scenario-based approach that enables stakeholders to participate in the scoping process.

A major shortcoming in the existing scoping techniques is that some of them focus on the customers' side (Rommes, 2003; Helferich et al., 2005) while others (Schmid, 2002; Kishi et al., 2002) focus on the technical side resulting in a gap between what the customers are asking and what is being actually developed. The second issue with the above-mentioned scoping methods is that they assume the product portfolio is being developed from scratch (Moraes et al., 2009). According to Moraes et al. (2009), from the existing SPL scoping methods, only Riebisch et al. (2001) base their scoping decisions on the ESS. They use decision tables for this purpose. The method is presented at a conceptual level without any operational details. To the best of our knowledge, no

[1] Modules refer to elements of implementation as defined by Bass et al. (2003). They represent a code-based way of considering the system.

existing method addresses the question of how to define product variants from an ESS by considering both the explicit preferences of customers on product features and the impact of features on ESS's structure. Our proposed method COPE+ addresses this problem by combining the voice of the customer and the ESS's structural analysis to suggest product variants.

We also acknowledge a large body of knowledge on product line design in the fields of marketing and management information systems. Researchers in these fields have investigated the optimal product line design problem from two viewpoints: buyer's welfare and seller's welfare. Buyer's welfare problem seeks to identify a product line that maximizes the total utility for all customers. Seller's welfare problem, subject to the customers' objective of choosing product features that maximize their utility, tries to find a best subset of products from a specified finite set of alternatives that will maximize the total value of the product line to the seller (McBride and Zufryden, 1988). Our formulation of the product line design problem is closer to the seller's welfare case. Proposed method COPE+ attempts to define product portfolios by incorporating both the customers' prioritization of features and ESS's structural constraints.

The product line design problem has been formulated under various conditions for both the buyers' welfare and sellers' welfare cases. For example, the monopoly based segmentation assumes the developing organization decides what product is offered to a customer and the only choice left to the customer is either to accept or reject that product (Frank et al., 1972). This is not the way most real-world market segmentations work. To create this condition, the developing organization must isolate customers from each other, deliver customized marketing programs to individual segments without leakage to other segments and it must have the legal authority to deny a segment access to what is being offered to another segment. Moorthy (1984) developed a theory of self-selection making the market segmentation more realistic by allowing each customer to have access to the complete array of products put out by the developing organization. Our work follows this approach by allowing the customers to prioritize all the features planned for the upcoming release.

The objective functions in McBride and Moorthy's formulations of the product line design problem are composed of measures such as expected frequency of purchases, dollar consumption or profitability of the product, etc. Although our method is targeting a similar problem, the focus is to provide decision support by analyzing the structural and behavioral knowledge of the system under analysis.

3.2. Decision support for software product evolution

The need for decision support arises when decisions have to be made in complex, uncertain and/or dynamic environments (Ruhe, 2003). There are several works that address provision of decision support for various aspects of software systems development. Turban et al. (2004) suggest that decision support systems are most suitable for semi-structured and unstructured problems. This is one of the key characteristics of wicked problems, i.e., they are difficult to formulate (Rittel and Webber, 1984). Planning releases for a single software product has been classified as a wicked problem (Carlshamre, 2002). Planning for a portfolio of products is even more challenging. This calls for greater efforts to develop methods and tools for supporting decision-making process in product line engineering. SPL development leads to additional decision problems on top of the typical ones in traditional single product development. For example, which product(s) should be developed as part of the product line and which feature(s) should be shared across them (scoping related decisions)?; will the product line approach lead to higher revenues and profit (investment related decisions)?; should the architecture be optimized for individual product variants or the product line (architecture related decisions)?; should the business units be the owners of shared features or a centralized product line engineering division (organization related decisions)?

These are some of the decision problems confronting product line development organizations. We will analyze the decision support methods that address a similar decision problem as addressed by COPE+. Product Line Potential Analysis (PLPA) is a decision support approach (partly) addressing the fundamental research question presented in Section 2 (Fritsch and Hahn, 2004). It is applied in a one-day workshop for personnel of a business unit. Authors have developed a set of criteria which they suggest is important for answering this decision problem. These criteria are classified in four categories: *main criteria* (essential for product line development and have to be fulfilled), *inclusion criteria* (indicates product line already exists), *supporting criteria* (applied if a business unit has problems that product line approach addresses) and *exclusion criteria* (to rule out factors that reduce economic advantage of product line approach). Each one of these four categories has a set of fine-grained criteria. Authors have prepared a questionnaire to get information on these criteria. The participants of the business unit are asked to fill out the questionnaire in the workshop. Their answers are then mapped to the criteria to address the decision problem. The results of this method are: *yes* (the product line approach is suitable for these products and markets), *no*, or *investigation required*. It does not, however, provide any details on how the existing products be evolved into a product line (Fritsch and Hahn, 2004).

Product Line Technical Probe (PLTP) assesses an organization's readiness to adopt product line development approach (Clements and Northrop, 2001). It requires an organization-wide effort in which assessors gather information through structured interviews of key stakeholders. The results are a set of findings identifying the potential benefits, risks, as well as an assessment of the organization's expertise regarding product line development approach (Clements and Northrop, 2001). Product Line Benefit and Risk Assessment (Schmid and John, 2002) method analyzes the benefits and risks for each of the technical domains associated with the product line. Hence, instead of just saying *yes* or *no* to product line engineering, this method prioritizes technical domains for reuse potential. This method has been modeled on the pattern of existing process maturity assessment methods.

The above-mentioned methods require participation of the product managers, technical team leads and architects. Although these methods assess the existing products, they evaluate the entire organization's capability to transition to product line development approach. These methods only address the first part of our decision problem, i.e., whether the organization under assessment should transition to product line development approach to develop its products or not. The details on how the existing products be evolved into a product line are not addressed by these methods. Finally, customers' input is not explicitly used to answer the decision problem. Some of the approaches, such as PLTP, do mention that assessors can interview the customers as part of assessment process.

Application of COPE+ does not require an organization-wide assessment effort as performed by most of the methods mentioned above. More importantly, it does not require participation of the senior management for generation of results. Focus of COPE+ is on the product evolution and it does not address organizational issues. It should be noted that the goal (of the method COPE+) is not to propose the best possible product portfolio, which is unrealistic because of the uncertainties in the input data, conflicting customers' needs and dynamically changing environment. Rather the focus is to put a first-level base plan on the table which will ini-

tiate the decision-making process through collaboration amongst stakeholders.

3.3. Structural analysis for system evolution

Locating software features in source code of an ESS is an important problem in software maintenance. As the software systems evolve, most often, the corresponding traceability documentation (identifying design and source code artifacts implementing a given feature) is not updated due to time and effort constraints. Hence, the information regarding design and source code artifacts of the ESS that are implementing a given software feature is not always readily available. Numerous techniques have been proposed to address this problem. According to Rohatgi et al. (2009) a large group of these techniques uses execution traces for software features to identify their impact on the system. The modules specific to the feature under analysis are the ones that are only invoked in its corresponding trace. These methods require significant human intervention to generate results. A few other feature location techniques are based on concept analysis and clustering. Our method uses the results of an existing feature impact analysis technique which is based on clustering (Kuhn et al., 2007).

Change impact analysis techniques and tools are used in industry during maintenance and evolution of software systems. Several tools are available in this domain such as the IBM Rational Suite (RequisitePro, Application Developer, Software Architect and Data Architect) (Sundman, 2007) and NDepend (NDepend, 2010) which works with Microsoft's Visual Studio integrated development environment.

Each one of the new features impacts a subset of modules in the ESS's structure. These subsets (for individual features) may overlap with each other partially or completely. By generating multiple sequences for implementation of given features, our method attempts to identify subsets of features that impact a cohesive set of modules in ESS's structure. Selection of such feature subsets (for product variants) will avoid a large spread of modules in ESS's structure from being impacted. The assumption is that if the number of impacted by the features in a product variant is reduced, the effort to develop that product variant will also be reduced.

To achieve the above goal, in Ullah et al. (2009), we used three different code-based heuristics, e.g., selecting the module with the least number of lines of code to provide this structural impact information to the decision maker. These heuristics were based on greedy approaches. We realized that it is impossible to generate good quality solutions with greedy heuristics, since no global search was done. The other option is exhaustive search, i.e., evaluating architectural impact for all possible permutations of features. The numbers of permutations increase exponentially with the increase in the number of features, thus making exhaustive search prohibitively expensive. Additionally, the search space is non-linear, i.e., it is impossible to linearly predict the number of modules that will be impacted when the features are implemented in a different sequence or different subsets of features are implemented. Thus, linear optimization techniques such as linear programming cannot be used and non-linear techniques (e.g., branch and bound, and meta-heuristics) are needed.

In recent years, there is an increasing trend to apply search-based optimization algorithms for solving software engineering problems. One of the most popular optimization approaches in Search-Based Software Engineering (SBSE) is genetic algorithms (GAs) (Harman et al., 2009). Given the limitations of greedy heuristics, cost of exhaustive search and non-linear search space, we have used a search-based optimization technique (GA) for evaluating the impact of features on existing system's structure. GAs are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. They are able to direct search into a region of better performance within the search space (Mitchell, 1996).

3.4. Behavioral modeling

The future systems (product variants) are non-existing, i.e., they do not have any source code. Therefore, we had to select a suitable modeling notation to represent the future and existing software systems for comparison. In the system and software modeling literature, different models are used to represent different properties of a software system. The UML (latest version 2.2) (OMG, 2009), consisting of 11 diagram types, is a widely used standard in modeling software systems.

The focus of this work is to propose customized products for each cluster of customers. Customers express their opinion by voting on product features. One of the commonly used definitions of a software feature is "a behavioral aspect of the system that represents a particular functionality, triggered by an external user" (Eisenbarth et al., 2003). Hence, we decided to select a modeling notation that can represent how a system will behave when a user initiates a specific functionality. These system-level usage models for the product variants are compared with the same for the ESS. To represent the behavior of a software system by UML, the following four types of models are used (OMG, 2009).

(1) Activity diagram
(2) Statechart diagram
(3) Use case diagram
(4) Interaction diagram

Since we want to evaluate the behavioral similarity of the ESS with the product variants in a given product portfolio, we need to select a type of behavioral model for which conducting systematic and automatable similarity matching is possible. In this paper, we have chosen statecharts to model ESS and product variants' system-level usage since statechart diagrams are one of the suitable choices for this type of behavior modeling (details in Section 4.4). The technique we have used for comparing the statecharts is based on directed graphs and is applicable to activity diagrams as well. Comparison of use case system-level usage is not handled by this technique since those do not support control flow. Interaction diagrams were not considered since they emphasize the object interactions in lower level of granularity (i.e., one interaction diagram for each use case) than system-level usage modeling.

It is assumed that the statechart model for the ESS is available as input to COPE+. If not already available, manually developing a statechart for an ESS will require significant effort. There exist methods and tools that can automatically generate (reverse engineer) statecharts from a code base for fairly large systems with minimal human interaction (van Zeeland, 2009). Once a statechart model is available for the ESS, it can be reused to generate the same for product variants. This will require manual effort since the design models and code for the future systems do not exist. Empirical studies have identified the cost of developing UML models in software maintenance and evolution projects (Dzidek et al., 2008).

There are two broad categories of techniques for comparing statecharts (or directed graphs). The first category contains similarity measures that are either *cost-based*: requiring the number of modifications needed to transform one statechart into the other or *feature-based*: relying on structural features such as vertex degrees for comparing graphs (Sanfeliu and Fu, 1983).

In our method, where states and transitions contain semantic information, these structure-based (*cost and feature-based*) similarity measures are not very useful. Hence, we favored the second category which contains similarity measures based on behavioral comparison. One such technique is known as *Extremal Quantitative*

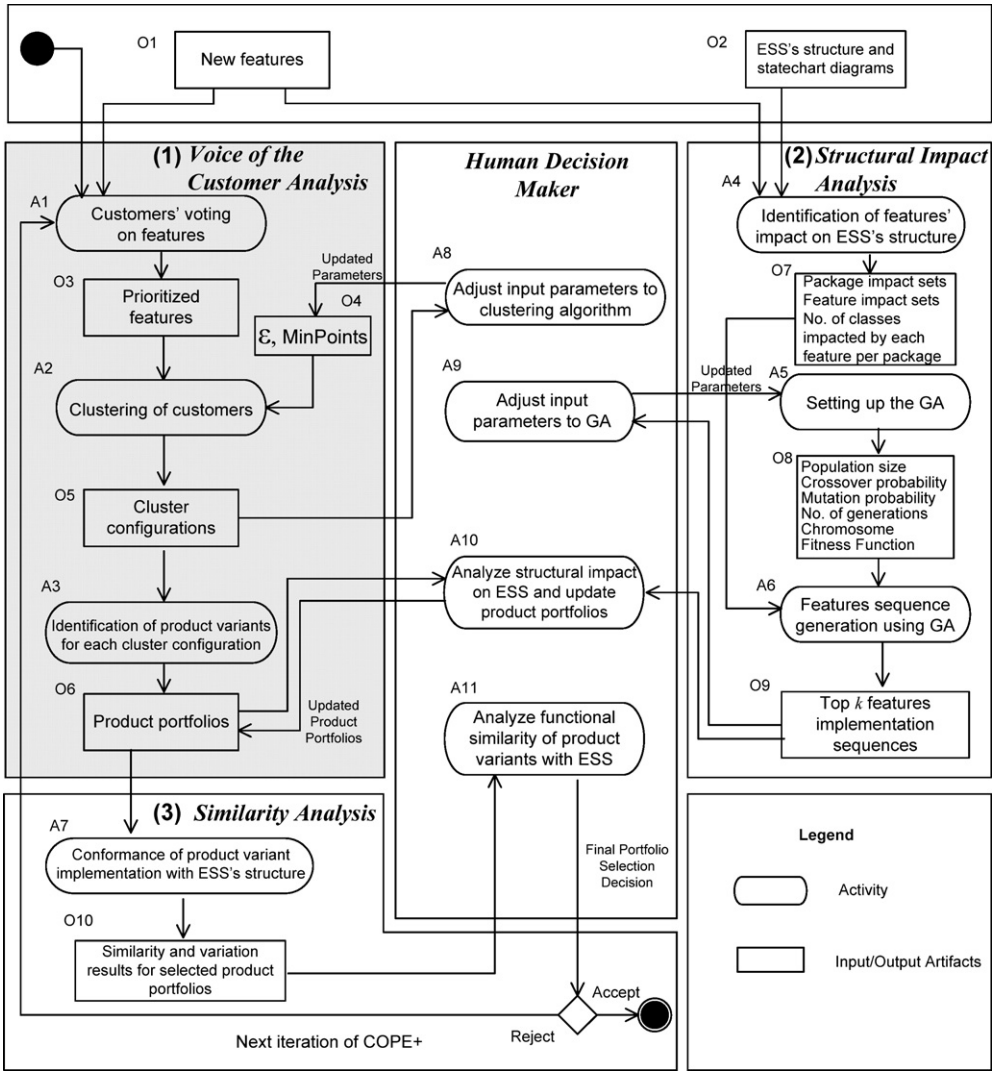

Fig. 1. Workflow of COPE+ illustrated as a UML activity diagram.

Simulation (also referred to as *mq-simulation*) (Sokolsky et al., 2006). Interested reader is referred to Sokolsky et al. (2006) for a more detailed comparison of the two categories of similarity measures and their applicability. Nejati et al. (2007) have proposed a hybrid *Match* operator for comparing statecharts which combines static matching and behavioral matching. The behavioral matching part of their operator is analogous to the mq-simulation technique. We have also applied this technique in COPE+; the details are presented in Section 4.4.

4. Decision support method COPE+

4.1. Overview

COPE+ is a decision support method to address the product evolution problem described in Section 2. The idea of offering decision support arises when decisions have to be made in complex, uncertain and/or dynamic environments. In software development and evolution, many decisions have to be made concerning processes, products, tools, methods and techniques. From a decision-making perspective, all these questions are confronted by different objectives, constraints, and a huge number of variables under dynamically changing requirements. Very often, this is combined with incomplete, fuzzy or inconsistent information about all the involved artifacts, as well as with difficulties regarding the decision space and environment (Ruhe, 2003).

For a focused problem domain as studied in this paper, emphasis is on providing methodology for generation, evaluation, prioritization and selection of solution alternatives. Typically, modeling, measurement, empirical and simulation-type investigations are combined with intelligent methods of analysis and reasoning to predict the impact of decisions on future life-cycle performance. From a high-level standpoint, the decision support method COPE+ has three modules. Fig. 1 shows the workflow of COPE+ illustrated as a UML activity diagram. Elements with identifiers *Ai* are the activities which take one or more input artifacts (data) identified as *Oi*. An activity manipulates input artifacts to produce at least one output artifact. These activities are arranged in three modules. The human decision maker is involved at various stages of the decision support to analyze and update the data and results. A brief overview of the modules is presented below.

The first step in decision support for evolution of ESS is to identify segments of customers having a similar structure of preferences on product features. This is the goal of module 1 *Voice of the Customer Analysis*. Module 1 applies cluster analysis (A2) to the votes of customers on product features (O3). This results in a cluster configuration classifying customers into clusters (O5). Each cluster of customers is offered one product variant. Collectively all product

variants for a given cluster configuration form a product portfolio (O6). Multiple cluster configurations are generated by varying the input parameters to the clustering algorithm (O4).

The product variants (within each product portfolio) generated in module 1 reflect customers' opinion on features. To implement these product variants, the ESS's structure will have to be changed. Therefore, it is necessary to analyze the impact of product portfolios on ESS's structure. The goal is to decrease these structural changes. Module 2 *Structural Impact Analysis* uses a search-based algorithm to arrange features in sequences such that the features impacting the same modules are placed adjacent to each other (A6). Top k sequences are used to evaluate the product variants (O9). Decision maker is given specific recommendations to introduce changes in the product variants by adding or deleting features to reduce their impact on ESS's structure (A10). Module 2 supports the decision maker to address the first research question (RQ1) presented in Section 2.

The product portfolios which exhibit good fit with the ESS's structure are selected for behavioral comparison with the ESS in module 3 *Similarity Analysis*. The customers are interested in the functionality provided by the product variants. Behavioral models of product variants and ESS highlight this functionality. Using such models, COPE+ compares the product variants with the ESS (A7). The goal is to measure the conformance of each product variant's implementation (in the selected product portfolios) with the ESS's structure (O10). Comparison is performed on statechart representations of the ESS and product variants. The differences of each product variant with the ESS are also identified (O10). Module 3 provides this information to the decision maker for final selection of product portfolios (A11). Therefore, this module helps to address the second research question (RQ2) presented in Section 2.

COPE+ can be customized by the users according to their organization's business and technical parameters. They can select suitable analysis techniques in the three modules according to the data availability related to customers and the ESS. The specific techniques for structural analysis of ESS and similarity analysis, presented in this paper should be considered as some example techniques to generate required results. Detailed description of the activities is given below:

4.2. Module 1 – voice of the customer analysis

The focus of this module is to elicit customers' preferences on new features and use them to offer customized products to all segments of customers. The details of the activities of module 1 have been presented in our previous works (Ullah and Ruhe, 2007; Ullah et al., 2009) and hence will not be discussed in this paper. This work assumes that the results of the module 1 are available for input to subsequent activities of COPE+. A brief introduction to the notation and terminology that will be referred to in other modules is given below:

We consider a set of customers represented by $C = \{c(1), c(2), \ldots, c(l)\}$. A given set of features $F = \{f(1), f(2), \ldots, f(m)\}$ is studied for their (complete) implementation in an evolving system. The features might represent the decision of a product manager related to the feature set added for an upcoming product release. The granularity of a feature is flexible. It is assumed that the granularity is defined in a way that the number of features under consideration is in the order of up to 100, as otherwise it becomes more and more effort consuming for customers to conduct prioritization. Customers vote on the features using a 9-point Likert scale (1: least desired to 9: extremely desired). A feature vote represents the perceived *value* of that feature to a customer. Additional criteria can also be defined for voting such as frequency of use or urgency.

Definition 1 (cluster configuration): We consider a set of customers C. A partition of the set C is called a cluster configuration. A cluster configuration is represented as $CC(i) = \{cc(i,j), \ldots, cc(i,k)\}$, where as $cc(i,j)$ is the jth cluster of customers in the ith cluster configuration.

Definition 2 (product variant): We assume a given cluster configuration $CC(i)$. A product variant $p(i,j)$ is defined as the subset of the set of features F offered to the cluster of customers $cc(i,j)$.

Note that each product variant contains all the features of the ESS in addition to the new features corresponding to its customer segment. One product variant is proposed per cluster of customers for all cluster configurations.

Definition 3 (product portfolio): The set of product variants corresponding to a given cluster configuration $CC(i)$ constitute a product portfolio $PP(i) = \{p(i,j), \ldots, p(i,k)\}$. One product portfolio is proposed per cluster configuration.

Multiple cluster configurations are generated by varying the input parameters to the clustering algorithm. The corresponding product portfolios are the decision alternatives for transitioning the ESS. The activities in the next two modules evaluate these candidate solutions and recommend changes to improve them.

4.3. Module 2 – structural impact analysis

The second module pro-actively analyzes the expected impact of the given features on ESS's structure. A search-based algorithm determines possible implementation sequences of features such that the impact on ESS's structure is minimized. Note that we aim to minimize the impact of implementing new features on ESS's structure; however, we do not claim that it is the absolute minimal in formal mathematical terms. This notion of minimization has been used in several SBSE works. For example, Doval et al. (1999) and Mancoridis et al. (1998) applied GA to arrange modules of software systems in subsystems such that couplings amongst these subsystems are minimized. Interested reader is referred to Harman et al. (2009) for a comprehensive review of SBSE literature. Module 2 consists of the following three activities.

4.3.1. Identification of features' impact on ESS's structure

For this work, ESS's structure is evaluated at the abstraction level of packages[2] (in the object-oriented paradigm, a package is a composition of classes). The system's structure is represented as $K = \{k(1), k(2), \ldots, k(n)\}$ where each $k(i)$ represents a package.

Definition 4 (package impact set): Each feature $f(i)$ impacts a set of packages $\Phi(i) \subseteq K$ in the ESS's structure. We refer to $\Phi(i)$ as *Package Impact Set* for feature $f(i)$.

Definition 5 (feature impact set): In the opposite direction, each package $k(j)$ (partially) implements a set of features $\Gamma(j) \subseteq F$ referred to as *Feature Impact Set* for package $k(j)$.

The package impact sets can be generated through source code-based or model-based feature impact analysis techniques (Arnold, 1996). The level of impact of a given feature on ESS's packages is determined as the number of classes impacted by this feature per package. This information will help the decision maker to identify features that can be added or removed from product variants in candidate product portfolios to adjust the level of impact on ESS's structure for feature implementation. It is assumed that the information about the impact of each feature on the packages of ESS is available as input to the method.

The above concepts are illustrated in Fig. 2. The arrows originating from feature $f(i)$ indicate the packages impacted for implementing this feature. The small squares represent classes within each package of a given ESS. The classes that are impacted by the feature $f(i)$ are shaded. The arrows originating from package $k(j)$

[2] It is possible to extend COPE+ to conduct impact analysis at other levels of packaging such as classes, files, etc.

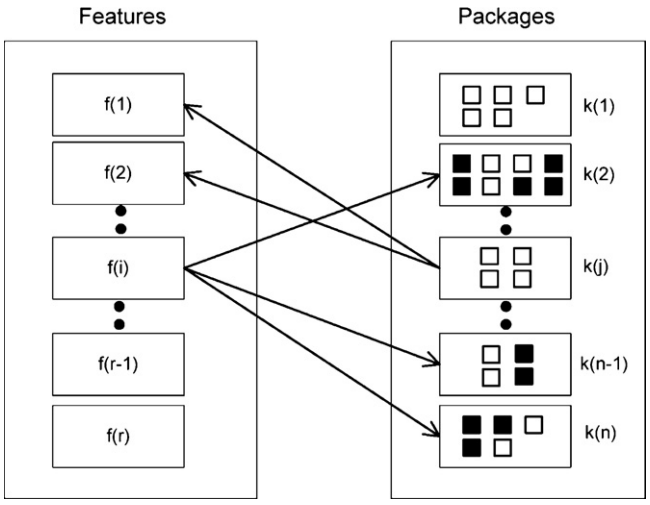

Fig. 2. Feature f(i) impacts packages k(2), k($n-1$) and k(n) in ESS's structure. Package k(j) (partially) implements features f(1) and f(2).

identify the features that are (partially) implemented by this package. A real example of feature impact analysis will be discussed in Section 5.2. The reason for analyzing the impact of features on ESS's packages is to reduce the number of packages that will require modification for each product variant. This is achieved by application of a genetic algorithm as discussed in the next subsection.

4.3.2. Setting up the genetic algorithm

In our previous work we have applied heuristics to determine the impact of features on ESS's structure (Ullah et al., 2009). These heuristics were based on greedy approaches, e.g., selecting features that impact the least number of classes in ESS's modules. Other measures such as lines of code, number of methods, number of connections to other modules, etc., were also used to construct heuristics for feature impact analysis. It is extremely difficult to predict which heuristic will produce good quality results for the system under analysis. In order to improve the quality of results, we have applied a genetic algorithm (GA) to determine the sequence of implementation of features into the ESS. A variety of successful applications of GAs in the context of software engineering problem solving have been discussed by Harman et al. (2009).

In a GA, a possible solution to the problem is represented as a chromosome (also called a genome). For our problem, we structured the chromosome as a 2D array (featureImpact[x][y]) representing x features each impacting up to a maximum of y packages in the ESS. For the implementation purposes, we have used GAlib, an open-source GA framework (GAlib, 2009). It provides four classes of genetic algorithms and a variety of representations for genomes. It also provides pre-defined operators such as initialization, mutation and crossover. A fitness function has to be defined by the user according to the specific problem to which the GA is being applied. For more details on GAlib, readers can refer to GAlib (2009) and Wall (1996). The design of our GA is briefly explained next.

The crossover operator is a procedure to mate individuals from the population to generate new off-springs. We have used one-point crossover procedure for the 2D array as it is commonly used in other SBSE works (Chang et al., 2001). After crossover, a small number (1 percent) of chromosomes are mutated. Swap mutation (Wall, 1996) is applied in our algorithm in which elements of the chromosome are swapped. We selected values for population size, crossover probability, mutation probability and number of generations, based on a few experimentations and also referred to the literature for guidance (Harman et al., 2009).

The fitness function is used to evaluate the quality of a chromosome (or a genome) representing a possible solution of the problem. Our fitness function is based on the concept of *ordered feature commonality* (OFC) described below. Given any feature in the sequence, the feature resulting in higher OFC value than others is selected for the next place in the sequence. Instead of local optimizations (a greedy approach), the genetic algorithm is able to select a solution which tries to maximize the overall OFC value for the sequence of features.

Definition 6 (ordered feature commonality): To quantify the idea of reducing structural impact, we define *ordered feature commonality* (OFC) for any two features f(i) and f(j) such that f(i) comes in the implementation sequence before f(j)[3] (represented as f(i) → f(j)) as:

$$\mathrm{OFC}_{\mathrm{f}(i)\rightarrow\mathrm{f}(j)} = \frac{\left|\Phi(i)\cap\Phi(j)\right|}{\left|\Phi(j)\right|} \tag{1}$$

Example: To illustrate the computation of *ordered feature commonality*, we refer to a popular open-source text editor called jEdit. It is used in Section 5 as an example software system to illustrate the whole COPE+ method. To calculate the *ordered feature commonality* using Eq. (1), we use three features (out of the set of nine features from the jEdit example):

f(1) = DC (Domain Concepts),
f(2) = UI (User Interface) and
f(4) = TB (Text Buffers).

For this illustration we will use five packages (out of thirty packages from the jEdit example) that partially implement these features:

k(6) = menu_a,
k(13) = search,
k(16) = options,
k(17) = gui and
k(27) = print.

The (partial) package impact sets of these features for the jEdit system are:

Φ(1) = {menu_a, search, options, gui},
Φ(2) = {search, options, gui} and
Φ(4) = {options, gui, print}

Suppose the features are implemented in the sequence DC → UI → TB. Using Eq. (1), the OFC values for this sequence are as follows:

$$\begin{aligned}\mathrm{OFC}_{\mathrm{f}(1)\rightarrow\mathrm{f}(2)} &= \frac{3}{3} = 1\\ \mathrm{OFC}_{\mathrm{f}(2)\rightarrow\mathrm{f}(4)} &= \frac{2}{3} = 0.67\end{aligned} \tag{2}$$

The first result in Eq. (2) means that when f(2) is implemented after f(1), no additional packages are impacted. In other words, f(2) has 100% commonality with f(1) with respect to impact on ESS's structure. The second result in (2) means that out of the total packages impacted by f(4) in ESS's structure, 67% are the same as impacted by f(2). Therefore, f(4) impacts only 33% new packages in ESS's structure when it is implemented after f(2).

Selection of features impacting the same set of packages in ESS's structure will reduce the effort for implementing the features as

[3] Note that this is a start–start dependency relationship, meaning implementation on successor feature cannot start before the start of implementation of the predecessor feature.

argued by Saliu and Ruhe (2007) in the context of release planning. Ordered feature commonality is not defined symmetrically since we are assuming that features are being implemented in a sequence. Thus, the number of packages that the successor feature impacts depends on predecessor feature's impact on the system structure. If both features are implemented together then it will be symmetrical.

Definition 7 (cumulative ordered feature commonality): Given a sequence x for implementing m features, the *cumulative ordered feature commonality* (COFC) is defined as the sum of the OFC values for the first m-1 features.

$$\mathrm{COFC}(x) = \sum_{i=1}^{i=m-1} \sum_{j=2}^{j=m} \left(\mathrm{OFC}_{\mathrm{f}(i)\rightarrow \mathrm{f}(j)}\right) \quad (3)$$

The goal is to identify sequences for implementing the given set of features resulting in high COFC values. For this purpose, the search space $\prod(x)$ consists of the set of all permutations $\prod$ of m features. For each permutation $\pi = (\pi(1), \ldots, \pi(M))$ of feature indices $\{1, \ldots, M\}$ a unique sequence for implementing the features is determined. The fitness function $F(x)$ is composed of the COFC value for the sequence x represented as the permutation π. The objective is to maximize $F(x)$.

$$\text{Maximize } F(x) = \mathrm{COFC}(x) \quad (4)$$

In the above jEdit example, there are six possible permutations to arrange the three features f(1) = DC, f(2) = UI and f(4) = TB. Using Eq. (4), the highest value of $F(x) = 1.67$ is achieved when the features are implemented in the sequence DC → UI → TB. A higher value of *ordered feature commonality* means the later feature in the sequence is reusing a higher ratio of the packages from the earlier feature, hence, smaller impact on ESS's structure.

4.3.3. Features sequence generation using GA

The genetic algorithm (presented in Section 4.3.2) returns sequences $S(i)$ of implementation of features with high COFC values. The human decision maker can modify the input parameters to the GA and generate another set of sequences of features. Top k sequences are selected for analysis of product portfolios generated by module 1. Using these k sequences, two types of recommendations are given to the human decision maker for each product variant. The first type of recommendations is related to the existing features in the product variants, i.e., whether an existing feature should be kept in the product variant or removed. The second type of recommendations is related to the new features that can be included in the product variant with minimal additional impact on ESS's structure. This is illustrated in Section 5.2.

4.4. Module 3 – similarity analysis

After the product portfolios have been updated (by the decision maker) based on the results of module 2, the decision maker can select one or more product portfolios for comparison with the ESS. This comparison identifies the level of conformance of each product variant's implementation (in the selected product portfolios) with the ESS's structure as well as specific differences between them. This information helps the decision maker in the final selection (or rejection) of product portfolios.

4.4.1. Conformance of product variant's implementation with ESS's structure

The system and software modeling literature is rich with techniques to compare various types of models as presented in Section 3.4. Here we demonstrate how system-level usage modes of ESS and product variants represented as statecharts can be compared using an existing technique.

A *state* represents a recognizable situation that exists over an interval of time (Gomaa, 2004). An *event* occurs at a point in time and usually causes transition of the system to another state. Note that in a statechart, a *state* is represented as a *node* and an *event* is represented as a *transition* between two states. In our method, the nodes of the statechart represent the states of the system, e.g., Edit Mode or Help Mode of a text editing system. The transitions of the statechart represent the events that cause the system to move from one state to another, e.g., pressing the Help button will cause the system to transition to Help Mode.

To compare two statecharts, the most commonly used similarity functions (Nejati et al., 2007; Sokolsky et al., 2006) are:

(1) Node similarity function compares the names of two states s and t. $N(s, t)$
(2) Label similarity function compares the names of two transitions a and b. $L(a, b)$

These two metrics measure the similarity between any two nodes or any two transitions. They can be defined in a number of ways, for example, typographic matching (comparing names using a character matching algorithm) and linguistic matching (comparing names based on their linguistic correlation) (Nejati et al., 2007). The node and label similarity functions can be defined both in binary (complete matching resulting in either 0 or 1) and continuous Real-value domains (partial matching resulting in a real value between 0 and 1 inclusive).

4.4.2. Simulation-based comparison of statecharts

Let the behavior of an ESS be represented as a statechart G and the behaviors of all the proposed z product variants in a product portfolio as statecharts G^z. Suppose we want to determine whether the ESS (represented as G) exhibits the behavior (functionality) of one of the proposed product variants (represented as G^1) or not. For this purpose, a simulation relation will be defined between the starting nodes of the two statecharts. The node s_0 (initial node in G) simulates a node t_0 (initial node in G^1) if any outgoing transition from t_0 to t_1 labeled by, say, x can be matched by a transition from s_0 to s_1, also labeled by x, in such a way that s_1 simulates t_1. If the ESS exhibits the entire behavior of the product variant then the result of simulation will be 1.

Simulation relations are typically defined as recursive functions. A commonly used technique based on such a function is *mq-simulation* proposed by Sokolsky et al. (2006). It has some limitations which make it unsuitable for application in our problem. Firstly, mq-simulation is very strict when comparing two models, i.e., if there is a difference of even one node or edge between two models, it will calculate the similarity between them as *zero*. The mq-simulation metric operates on a binary scale. In our case, we already know that the proposed product variants will vary from the ESS. We want to know the degree of similarity of various proposed product variants with the ESS requiring measurement on a Real-value domain. Hence, we have extended mq-simulation to calculate partial similarity between compared models. Secondly, it is useful to know the differences in behaviors of product variants and the ESS. For this purpose, we have extended the mq-simulation to identify the states where the behavior of a product variant deviates from that of the ESS. This information is recorded in a set of variation points during the simulation process. Formal definition of mq-simulation is presented in Appendix A. A symmetric version of the simulation relation is known as bisimulation.

When G^1 simulates G, we extend the original definition of mq-simulation (Sokolsky et al., 2006) as follows:

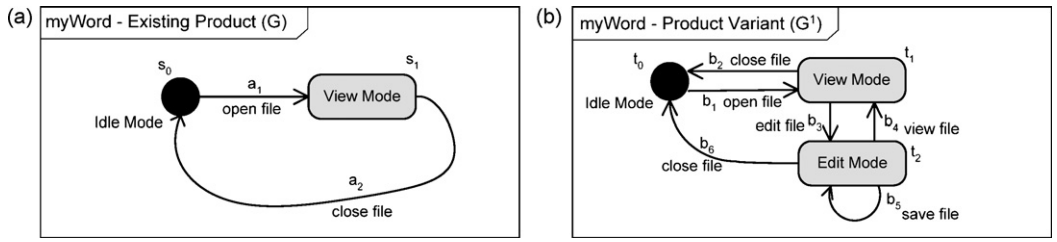

Fig. 3. Statecharts representing simplified behaviors of existing myWord (a) and myWord variant (b).

1. Label matching between any two transitions a and b can also accommodate partial similarity. Note that originally the results were binary {0, 1}.

   L (a, b) results in a value in the range [0, 1].

2. During the simulation process, a set of variation points, initialized as VP = {}, will be updated with the states where mismatch occurs.

With our extended mq-simulation algorithm, for any two given statecharts G and G^1:

G^1 simulate G results in a measure Q(s_0, t_0) with value in the range [0, 1] and a set VP1.

G simulate G^1 results in a measure Q(t_0, s_0) with value in the range [0, 1] and a set VP2.

Definition 8 (mq-simulation): We can say G^1 simulates G up to n if Q(s_0, t_0) = n where n is a real number between 0 and 1 inclusive.

4.4.3. Illustrative example

We explain the concepts of simulation and bisimulation with the help of an example. Consider a hypothetical software system "myWord" which lets its users read text documents. Statechart G in Fig. 3a illustrates a simplified behavior of the ESS (myWord). A variant of myWord has been proposed which contains additional functionality, allowing its users to edit their documents. Statechart G^1 in Fig. 3b illustrates a simplified behavior of the proposed product variant. We apply mq-simulation to show how the two statecharts simulate each other.

First we illustrate the case of G^1 simulates G, i.e., Q(s_0, t_0). According to mq-simulation, Q is defined as a product of node and label similarity functions (Sokolsky et al., 2006).

$$Q(s_0, t_0) = N(s_0, t_0) \times L(a_1, b_1) \times Q(s_1, t_1) \quad (5)$$

The first sub-expression on the right hand side calculates node similarity between the starting states s_0 and t_0. The second sub-expression evaluates the label similarity between the two outgoing transitions a_1 and b_1. The third sub-expression on the right hand side is the recursive definition of the simulation function. Since the name labels of nodes and transitions from the two statecharts exactly match (N and L values are 1), the first two sub-expressions on the right hand side of the above equation result in 1. Hence, Eq. (5) reduces to:

$$Q(s_0, t_0) = Q(s_1, t_1) \quad (6)$$

The only transition going out of state s_1 is to s_0 (the starting state). We consider this as the stopping condition for the recursive call, since we have reached the starting state. Hence, s_1 becomes the end state, in which case, only the node names are matched.

$$Q(s_0, t_0) = N(s_1, t_1) = 1 \quad (7)$$

It shows that the behavior of the ESS is preserved in the product variant or in other words, G^1 has all the behavior of G. We can also observe that the variant of myWord has additional behavior due to new features (Edit Mode).

In our extended mq-simulation model, an additional output of the algorithm is the set of variation points between the two statecharts which in this case will be VP1 = {s_1} indicating that the state s_1 (which is equivalent to t_1 in G^1) is extended to introduce new behavior in the myWord variant.

A similar simulation relation can be established when looking from G to G^1 (Sokolsky et al., 2006). In the case of G (existing myWord) simulates G^1 (myWord variant), we can see from Fig. 3 that G only partially simulates G^1 since it does not have the Edit Mode. Hence, the result of G simulates G^1 will be: Q(t_0, s_0) = 0 according to original mq-simulation algorithm.

When the recursive definition is applied, the outgoing transition b_3 from state t_1 will not have any matching outgoing transition from state s_1, hence it will be matched with the only outgoing transition from s_1, i.e., a_2. This will result in L(b_3, a_2) as zero. Our extended mq-simulation model will consider partial matching between the labels b_3 and a_2 resulting in 50 percent match based on typographic matching, i.e., L(b_3, a_2) = 0.5 resulting in Q(t_0, s_0) = 0.5. It means the existing myWord simulates the product variant up to 50 percent. The set of variation points in this case will be VP2 = {s_1}.

The mq-simulation algorithm can be applied to any model based on directed graphs such as UML activity diagrams, UML collaboration diagrams, etc. Simulation results of the ESS with proposed product variants can be analyzed on the ordinal scale, i.e., using these results the product variants can be ranked based on their behavioral similarity to the ESS.

5. Application of COPE+ on jEdit

jEdit (www.jedit.org) is a popular open-source text editing software system. Its user-base is steadily increasing with frequent feedback and feature requests on the project website (jEdit Project Website, 2009). Observing the increasing user-base and continuous evolution of the jEdit system, we selected it to evaluate our proposed method COPE+. The results are based on the jEdit version 4.0.

5.1. Module 1 – voice of the customer analysis

A total of 109 feature requests were used for illustration in this paper (jEdit Project Website, 2009). These were the unresolved feature requests listed on the jEdit project website as on 1st of February 2009. To keep the number of prioritization objects (available for customers voting) small, we followed a grouping of features given by the domain experts from an existing work (Kuhn et al., 2007). This results in nine feature groups which are called just features again for simplicity reasons. All the analysis in this case study has been performed at the level of these features shown in Table 1.

The users who submit feature requests for jEdit do not prioritize the features using a voting scheme (jEdit Project Website, 2009). Hence, the voting on the features was generated synthetically.

Table 1
jEdit features.

Feature ID	Feature name	Functionality
f(1)	DC	Domain Concepts
f(2)	UI	User Interface
f(3)	RE	Regular Expressions
f(4)	TB	Text Buffers
f(5)	DW	Dockable Windows
f(6)	BS	Beanshell Scripting
f(7)	XR	XML Reader
f(8)	BA	Bytecode Assembler
f(9)	TZ	Tar and Zip Archives

Table 2
Product portfolios from cluster configurations.

Cluster configuration	Product portfolio	Product variants
CC(x)	PP(x)	p(x,1): TB, RE, UI, BA p(x,2): XR, UI, DC
CC(y)	PP(y)	p(y,1): TZ, RE, UI p(y,2): XR, UI, DW, BS
CC(z)	PP(z)	p(z,1): DC, UI, RE, TB, DW, BS, XR, BA, TZ

Votes on the features were randomly generated for 10 assumed customers.

Three cluster configurations of customers were generated by the clustering algorithm. Product portfolios for these cluster configurations are shown in Table 2. Note that certain features appear in all the product variants of a product portfolio, e.g., UI is in both product variants of PP(x). There are certain features that do not appear in any product variant of a product portfolio, e.g., TZ does not appear in any product variant of PP(x). The product portfolio PP(z) consists of a single product p(z, *1*) containing all the nine features for 10 customers.

5.2. Module 2 – structural impact analysis

jEdit v4.0 has 30 source code packages containing 392 Java classes, having about 62 KLOC in total. We used the impact analysis results of Kuhn et al. (2007) for the jEdit version 4.0. They have proposed a technique that clusters source code artifacts (e.g., packages, classes or methods) based on their linguistic similarity. Instead of just clustering source code on keywords, they use semantic knowledge, i.e., the context in which a keyword appears in an artifact. A human expert manually inspects the source code artifacts that are shared amongst various clusters to verify correctness of results. They have identified the number of classes per package of jEdit version 4.0 which will be impacted by each one of the nine features listed in Table 1. Evaluation of *Package Impact Set* $\Phi(i)$ and *Feature Impact Set* $\Gamma(j)$ is shown in Table 3. A number x in a cell (j, i) indicates that feature f(i) will impact x number of classes of package k(j). The last column indicates the total number of classes per package. The last row lists the total number of classes impacted by each feature in ESS's structure.

The set-up of the GA including generation of all the input parameters and objective function were discussed in Section 4.3.2. The two-dimensional array *featureImpact[9][23]* shown in Table 4 represents the structure of an example chromosome based on the data given in Table 3.

A row in the 2D array corresponds to a column in Table 3 and represents package impact set for a feature. The first element of

Table 3
Feature and package impact sets for jEdit software system (taken from Kuhn et al., 2007).

Feature impact set	Package name	Package impact sets									No. of classes impacted in each package
		Φ(1) DC	Φ(2) UI	Φ(3) RE	Φ(4) TB	Φ(5) DW	Φ(6) BS	Φ(7) XR	Φ(8) BA	Φ(9) TZ	
Γ (1)	Lat	1									1
Γ (2)	Commands	1									1
Γ (3)	Doc.	1									1
Γ (4)	Jcmd	2									2
Γ (5)	IO	6									6
Γ (6)	Menu_a	4									4
Γ (7)	Msg.	11				1					12
Γ (8)	Util.	6		2						1	9
Γ (9)	Menu_b	8			3						11
Γ (10)	Quickno	4	2								6
Γ (11)	Browser	6	4								10
Γ (12)	Macos	3	1								4
Γ (13)	Search	8	3	3							14
Γ (14)	Jedit	27	4	5	2	2					40
Γ (15)	Installer	7	1			1				10	19
Γ (16)	Options	4	12		6						22
Γ (17)	GUI	7	26		5	6					44
Γ (18)	Help	1	3								4
Γ (19)	Pluginm	3	4					2			9
Γ (20)	Ref.						1				1
Γ (21)	Collect.						2				2
Γ (22)	Bsh	4	1	5			76				86
Γ (23)	Scrip	1					1				2
Γ (24)	Syntax			11	2						13
Γ (25)	Regexp			27							27
Γ (26)	Buffer	3		10			1				14
Γ (27)	Print				3						3
Γ (28)	Textarea		2	1	4		4				11
Γ (29)	Xml							4			4
Γ (30)	Asm								9	1	10
No. of Classes Impacted in all Packages		118	63	64	25	10	85	6	9	12	

Table 4
Structure of an example GA chromosome based on features' impact on jEdit system.

Package impact sets	No. of packages impacted	Package identifiers																					
Φ (1)	22	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	22	23	26
Φ (2)	12	10	11	12	13	14	15	16	17	18	19	22	28	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Φ (3)	8	8	13	14	22	24	25	26	28	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Φ (4)	7	9	14	16	17	24	27	28	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Φ (5)	4	7	14	15	17	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Φ (6)	6	20	21	22	23	26	28	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Φ (7)	2	19	29	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Φ (8)	1	30	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Φ (9)	3	8	15	30	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1

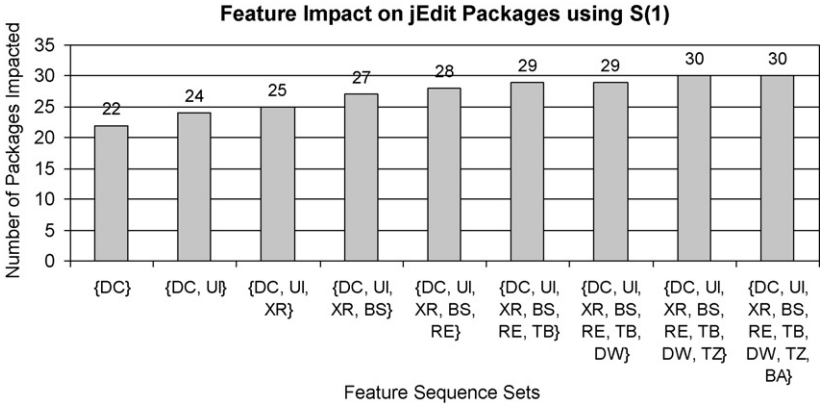

Fig. 4. Impact on jEdit's system structure using sequence S(1).

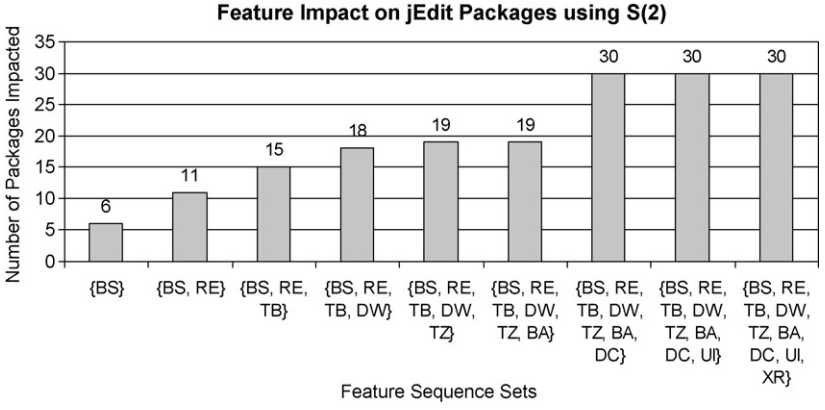

Fig. 5. Impact on jEdit's system structure using sequence S(2).

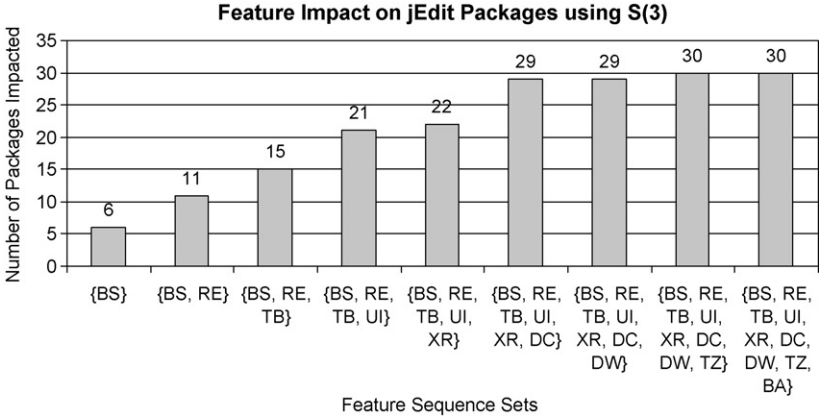

Fig. 6. Impact on jEdit's system structure using sequence S(3).

Table 5
Recommendations to decision maker for portfolio PP(x) based on top three sequences.

Features	Adjacent features						Recommendations			
	S(1)		S(2)		S(3)		Existing features		New features	
							Recommendation	No. of classes impacted	Feature name	Additional packages and classes impacted
Product variant p(x,1)										
TB	RE	DW	RE	DW	RE	UI	Strongly recommended	25	–	–
RE	BS	TB	BS	TB	BS	TB	Strongly recommended	64	BS	+3 packages and 85 classes
UI	DC	XR	DC	XR	TB	XR	Recommended	63	XR	+1 package and 6 classes
BA	TZ	–	TZ	DC	TZ	–	Not recommended	9	TZ	+0 package and 12 classes
Product variant p(x,2)										
XR	UI	BS	UI	–	UI	DC	Strongly recommended	6	–	–
UI	DC	XR	DC	XR	TB	XR	Strongly recommended	63	–	–
DC	–	UI	BA	UI	XR	DW	Recommended	118	–	–

each row represents the total number of packages being impacted by this feature. The remaining elements are the package identifiers. The number −1 is used to fill the rows when there are no further packages to be listed for a given feature. As an example, the 1st row $\Phi(1)$ of Table 4 corresponds to the column of 1st feature (DC) in Table 3. The first element in this row is 22 indicating that this feature impacts a total of 22 packages of jEdit. The subsequent elements in this row list the package identifiers for these 22 packages.

We used the steady-state genetic algorithm with one-point crossover operator for the 2D array chromosome (GAlib, 2009; Wall, 1996). We have not investigated the impact of different parameter settings on the performance, repeatability and scalability of our GA since it goes beyond the scope of this paper. Instead, we have used the following parameter values, which, empirically have proven to be suitable parameter settings for our GA in different experiments (Garousi, 2008; Chang et al., 2001):

Population Size: 100
Crossover Probability: 60%
Mutation Probability: 1%
Number of Generations: 1000

The following top three sequences, based on the GA fitness function value, were generated by the GA and they were selected for evaluation of the product portfolios from module 1. More than three sequences can be selected for evaluation of product portfolios consequently requiring more effort for their analysis.

Sequence S(1) = DC, UI, XR, BS, RE, TB, DW, TZ, BA F(x) = 4.05
Sequence S(2) = BS, RE, TB, DW, TZ, BA, DC, UI, XR F(x) = 4.05
Sequence S(3) = BS, RE, TB, UI, XR, DC, DW, TZ, BA F(x) = 4.02

Figs. 4–6 illustrate the impact on the ESS using the three sequences: S(1), S(2) and S(3), respectively. They highlight the features which impact no additional packages when added in the sequence. For example, when BA, UI and XR are added in S(2), there are no additional packages being impacted. Same is the case when DW and BA are added in both S(1) and S(3).

These figures also identify the features which have a very high impact, e.g., DC impacts 22 packages when placed at the top of S(1), the same feature impacts 11 additional packages when it is added after BA in S(2). In general, the flat regions of the graphs (adjacent bars with the same height) indicate no additional impact of the feature on the ESS's structure while a sudden jump in the height of a bar (with respect to the height of the previous bar) indicates

Table 6
Recommendations to decision maker for portfolio PP(y) based on top three sequences.

Features	Adjacent features						Recommendations			
	S(1)		S(2)		S(3)		Existing features		New features	
							Recommendation	No. of classes impacted	Feature name	Additional packages and classes impacted
Product variant p(y,1)										
TZ	DW	BA	DW	BA	DW	BA	Not recommended	12	DW	+1 package and 10 classes
									BA	0 Package & 9 Classes
RE	BS	TB	BS	TB	BS	TB	Not recommended	64	BS	+3 packages and 85 classes
									TB	+2 packages and 25 classes
UI	DC	XR	DC	XR	TB	XR	Not recommended	63	XR	+1 package and 6 classes
Product variant p(y,2)										
XR	UI	BS	UI	–	UI	DC	Strongly recommended	6	–	–
UI	DC	XR	DC	XR	TB	XR	Strongly recommended	63	–	–
DW	TB	TZ	TB	TZ	DC	TZ	Not recommended	10	TZ	+2 packages and 12 classes
BS	XR	RE	–	RE	–	RE	Recommended	85	RE	+3 packages and 64 classes

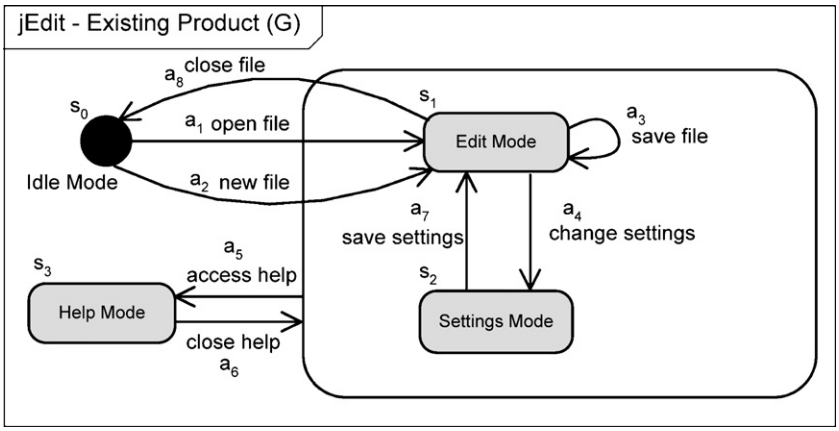

Fig. 7. Statechart representing a simplified subset of jEdit's behavior.

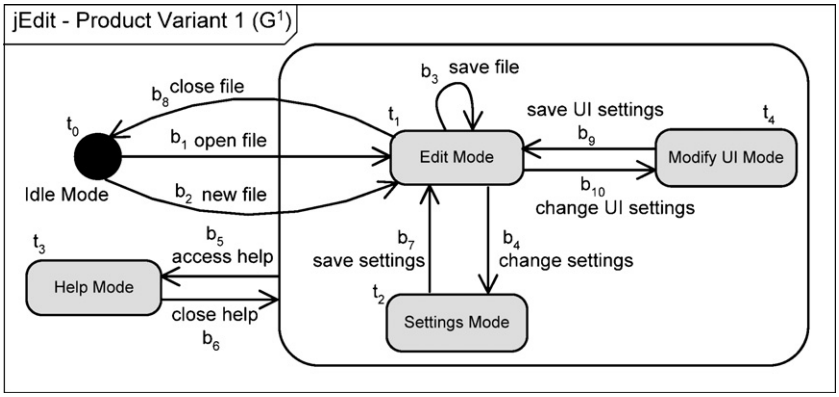

Fig. 8. Statechart representing a simplified subset of p(x,1)'s behavior.

a high impact of the feature on the ESS's structure. The lower the gradient of the graphs, the lesser is the impact of features on the packages. Our GA tries to find sequences of features having a higher number of regions of low structural impact (flat or low slopes).

The human decision maker is presented a set of recommendations based on these top three sequences. This information helps the decision maker to address RQ1. There are two types of recommendations for each feature in a product variant. The first one is for the feature itself, i.e., strongly recommended, recommended or not recommended to be kept in the product variant. If that feature is adjacent to another feature of the same product variant in all the three sequences then it is "strongly recommended", such as feature TB in p(x,1) is adjacent to RE in all the three sequences as shown in Table 5. If a feature is adjacent to another feature in the same product variant in at least one of the sequences (but not in all of the sequences) then it is "recommended" such as feature UI in p(x,1) is adjacent to TB in S(1). If a feature is not adjacent to any feature of the same product variant in all of the sequences then it is "not recommended" such as feature BA in p(x,1). The level of impact of each feature on ESS's structure is also presented to the decision maker, e.g., feature TB in p(x,1) impacts a total of 25 classes in all of the packages that it impacts.

The second type of recommendations suggests new features that can be added in the product variant with minimal or no additional impact on the ESS's structure. If a feature (not in the product variant) is adjacent to any existing feature in a product variant in all

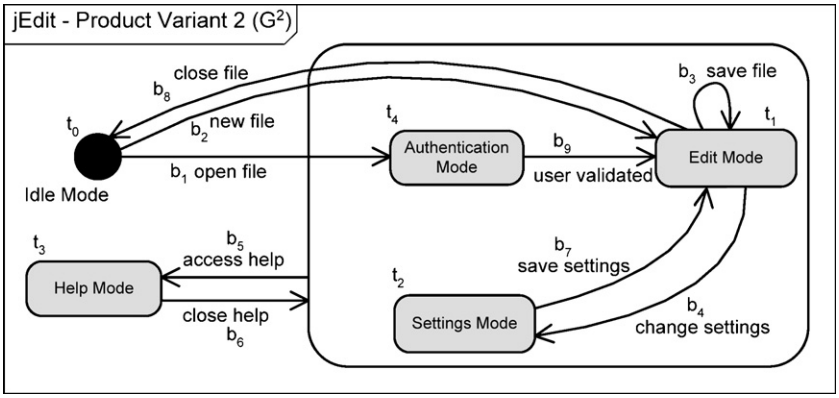

Fig. 9. Statechart representing a simplified subset of p(x,2)'s behavior.

the three sequences then it is "recommended" for inclusion in that product variant. For example BS is adjacent to RE (in p(x,1)) in all the three sequences. It is suggested to the decision maker to add this feature in the product variant with impact on 85 classes and 3 additional packages. Likewise TZ can be added in p(x,1) by impacting 12 classes and without impacting any additional package and XR can be added by impacting 6 classes and 1 additional package in ESS's structure. The impact on ESS's structure is determined by looking at the data in Table 3. Tables 5 and 6 show the recommendations for product variants in product portfolios PP(x) and PP(y).

COPE+ had recommended removing BA from p(x,1) (see Table 5). The human decision maker (the first author) looked at the structural impact results in Table 3 which confirmed that feature BA does not share any packages with other features in the product variant p(x,1). Therefore, he accepted the recommendation to remove BA from p(x,1). The decision maker also accepted to include XR in product variant p(x,1) at the cost of impacting six classes and one additional package (see Table 5). The updated feature set of p(x,1) is TB, RE, UI, and XR. The recommendation to include BS in p(x,1) was not accepted by the decision maker because of the impact on a large number of classes in ESS's structure. The composition of product variant p(x,2) remains unchanged as suggested by COPE+.

All the three features of p(y,1) and one of the features of p(y,2) from portfolio PP(y) are not recommended by COPE+ (see Table 6). Therefore, the decision maker removed the portfolio PP(y) from the set of candidate portfolios. No recommendations are given for product portfolio PP(z) since it already contains all the nine features.

With two candidates remaining for possible transitioning of the ESS: PP(x), a product line containing two product variants and PP(z), a single product containing all the nine features, the decision maker decides to conduct further analysis before selecting one of the two options. In the following section a similarity analysis is performed between the ESS and the two product variants in portfolio PP(x).

5.3. Module 3 – similarity analysis

In this example, G (Fig. 7) is the UML statechart representing a simplified subset of behavior of the existing jEdit system in terms of its "mode" of functionality. Note that we are modeling the system usage and not looking into internal details on how the features are implemented. In Fig. 7, states are shown as nodes (rounded rectangles) while arrows represent transitions with label a_i as events triggering those transitions. Event labels are shown on the edges. We will compare the behavior of each proposed product variant of PP(x) with the existing jEdit system through simulation. The statechart representations are generated manually by analyzing the flow of events in the system's GUI.

5.3.1. Simulating product variant p(x,1)

One of the new functionality for product variant p(x,1) is related to customization of user interface (Table 2: Feature UI). For brevity, only one new feature is being considered to demonstrate the simulation algorithm. Fig. 8 represents behavior of p(x,1) as statechart G^1 including this additional functionality. Simulation results in both the directions can be considered while evaluating the proposed product variants. In our opinion, the results for G simulates G^1, i.e., the evolving software system (ESS) simulates the behavior of the proposed product are important since the ESS has to be modified (evolved) into the proposed product variant.

ESS (represented as G) simulates product variant p(x,1) (represented as G^1) results in a similarity measure of $Q(t_0, s_0) = 0.66$ which means the existing jEdit system simulates the behavior of p(x,1) up to 66%. This is because of the presence of new events b_9 and b_{10} as well as a new state t_4 in p(x,1). The variation point set VP1 = $\{s_1\}$, meaning s_1 state (Fig. 7) in the existing jEdit system becomes a variation point in p(x,1) initiating new behavior which may include triggering new events or calling new states. The detailed application of the simulation model is given in Ullah (2009a).

5.3.2. Simulating product variant p(x,2)

One of the new functionality for product p(x,2) is related to file management (Table 2: Feature DC). Only one new feature is being considered to demonstrate the simulation algorithm. This translates to the statechart G^2 in Fig. 9 representing behavior of p(x,2). The result of ESS (represented as G) simulates product variant p(x,2) (represented as G^2) is 0.75. It is because of the new state t_4 and a new event b_9 in p(x,2). The variation points set VP2 = $\{s_0\}$ (see Fig. 7). The detailed application of the simulation model is given in Ullah (2009a).

Human decision maker is presented the simulation results of product variants which help her to address RQ2. Product variant p(x,2) performs better than p(x,1) with respect to simulation results. The final decision to accept one or all the product variants in portfolio PP(x) lies with the decision maker, who can, in addition to simulation results, also consider other factors such as relative importance of customer segments, long and short term goals of the organization, market competition, etc. The decision maker can also initiate further iterations of COPE+ by negotiating with customers to change their feature preferences to generate another set of solutions in the next iteration.

6. Discussions

6.1. Decision support for jEdit software system

Method COPE+ addresses the complex problem of evolving a single software system into a portfolio of product variants. As we mentioned earlier, the goal of COPE+ is not to propose best possible product portfolios. This is impossible given the uncertainty in available data and a huge number of other criteria that typically influence such decisions. Ebert and Smouts (2003) present a list of tricks which should be followed and traps that need to be avoided if an organization wants to successfully transition its ESS to a product line. The first trick is to "strongly coordinate marketing and engineering". One of the traps to avoid is transitioning to product line development by just looking at the business case without going into the detailed structure of the ESS.

The three product portfolios PP(x), PP(y) and PP(z) were defined through cluster analysis using votes of customers on features. Such candidate portfolios can also be generated through other techniques such as market analysis or product manager's personal judgment. It is difficult to predict the practicability of any of these portfolios without including consideration for ESS's structure. Therefore, choosing any one of the candidates without evaluating it against ESS's structure will be considered as an ad hoc decision. COPE+ provided the support to the decision maker by recommending specific changes in the business case (product portfolios proposed by customers) to align it with the ESS's structure.

To answer the fundamental question proposed in Section 2, the decision maker will look at the alignment between the customers' and structural views. For example, in Table 5 for product portfolio PP(x), majority of the features are recommended (except BA) which is not the case for PP(y) as shown in Table 6. COPE+ provided further support to the decision maker by identifying the similarity level of product variants with the ESS. Based on this information transitioning the ESS to product portfolio PP(x) is a good choice out of the three candidates. The single product option PP(z) is a good choice if majority of features in all the other portfolios suggested by customers are not recommended based on structural analysis

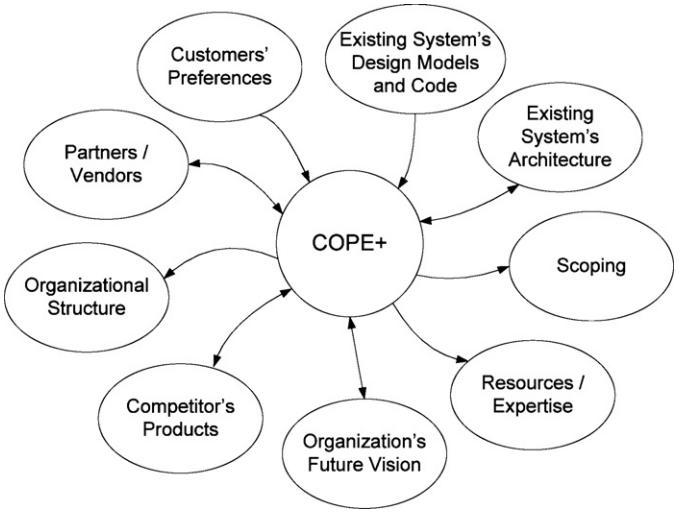

Fig. 10. Positioning of COPE+ in key criteria for product evolution decision making.

or the product variants have very low level of similarity with ESS based on behavioral analysis.

To answer RQ1, module 2 uses a genetic algorithm to generate sequences for implementation of features in the ESS's structure. Top k sequences having low impact on ESS's structure are selected for further analysis. For the jEdit example, three sequences were selected. The outputs of modules 1 and 2 are used to answer RQ1. For the jEdit example, product portfolio PP(x) was selected because the features in its product variants impacted cohesive sets of packages as compared to the product variants in portfolio PP(y). Consequently, the effort to transition the ESS to PP(x) is lower than the same for transitioning to PP(y). To answer RQ2, module 3 analyzes the product variants in the selected product portfolio through simulation to determine their level of behavioral similarity with the ESS.

6.2. Applicability of the method COPE+

We position COPE+ as shown in Fig. 10 amongst some of the important criteria influencing the decision to evolve the ESS to a product line. The criteria shown in Fig. 10 are not meant to be exhaustive. They are taken from the case studies reported in SPL literature (Clements and Northrop, 2001; Ebert and Smouts, 2003; Linden et al., 2007). The inward arrows indicate that *Customers' Preferences* on product features and information about *ESS's Design Models and Code* are used as inputs to determine product portfolio for addressing customers' clusters. The interaction of COPE+ with the remaining seven bubbles shows that results of COPE+ can be used as an initiator in decision-making process in these areas. Bidirectional arrows signify that results of COPE+ can influence the decision-making process in the other areas as well as the decisions in those areas can be considered by the decision maker while selecting a portfolio from the candidate portfolios suggested by COPE+.

We are aware that development organizations have a good knowledge about the architectures of their systems, and that they also perform formal or informal scoping. They have expertise in certain domains. They also know about their competitors' products and continuously interact with their partners and vendors to develop products. COPE+ can be imagined as a black hole where development organizations rarely go despite having all these data. COPE+ can use their customers' preferences and system structure information through design models and code to generate a first-level base plan. This plan can become a starting point for decision making in all the other areas shown in Fig. 10. For example, the architects can look at the results of COPE+ and determine whether the existing architecture is suitable for evolution into the proposed product portfolio. The architectural knowledge can also be used as a criterion by the decision maker while selecting a product portfolio. Feature partitioning is extremely difficult in certain types of system architectures. For example, large legacy systems with monolithic structures are difficult to partition along feature boundaries because of millions of lines of code embedded in file structures connected through a spaghetti jungle (Wu et al., 2005). The decision maker can compare the feature sets of product portfolios generated by COPE+ with the same for competitor's products while selecting a product portfolio. Results of COPE+ can be shared with vendors and partners for product planning and their inputs can be used by the decision maker to improve the results of COPE+. Similarly, results from COPE+ can be used as an input for defining the organization's future vision, organizational structure and resource planning.

The results proposed by COPE+ should be checked for their correctness and viability from both the customers' and system's perspectives. For the example software system in this paper, we applied FODA (Kang et al., 1990; Kang, 2009), a popular domain analysis method to identify inconsistencies of proposed product variants with the target domain of a text editing system (interested reader is referred to Ullah, 2009b for details). Following the procedure proposed by FODA, all the applications (product variants in our case) are compared with the domain model to identify inconsistencies. Domain experts then resolve the inconsistencies by either updating the application or taking the inconsistent application out of the target domain. Domain analysis showed that product p(x,2) does not support all the features of the text editing system domain. Domain experts along with product manager and other stakeholders will look into product p(x,2) more closely to resolve these issues. Resolution of such issues is out of scope of this paper and the proposed method COPE+, but can be looked into in future works.

On the other hand, it is not easy to quantitatively measure, customers' satisfaction. If a metric was available for this purpose, we would have applied it to determine whether the customers prefer the product portfolio or a single product containing all the features. Since we were unable to find such a metric, we analyzed the customers' votes on product features to determine customers' acceptance of the results. The analysis showed that normalized votes per customer are significantly higher for p(x,1) and p(x,2) (0.81 and 0.79, respectively) as compared to the single product option p(z,1) (0.48). With the product portfolio, the customers (in respective segments) get the features that they highly desire (expressed through voting). When all the customers are offered a single product containing all the features then the average normalized vote values are lower because higher level of variation amongst customers' preferences on features. This is one indicator of customer acceptance level for the proposed product portfolio (Ullah, 2009b). A detailed cost–benefit analysis will identify whether the benefits outweigh the costs attached with this evolution strategy. This cost benefit analysis is subject of a future work.

6.3. Limitations

There are different limitations related to the proposed method.

(1) The first one refers to the use of artificial data for customers' input on proposed features. In our opinion it does not limit the applicability of the approach.
(2) The method requires availability of data related to customers, features, ESS's structure which may be difficult to obtain for some systems.
(3) Preparation of statechart models may require significant effort for large systems. Reverse-engineering techniques can auto-

mate most (if not all) of this process (van Zeeland, 2009). The overall effort to apply COPE+ can be reduced by providing tool support that is able to connect the three modules for transfer of data and results.
(4) It is not possible to represent all types of behavior of a system using a statechart. For example, we only considered the behavior of the system initiated by the user through a menu. Other types of models and techniques are needed to conduct a comprehensive behavioral comparison between product variants and the ESS.
(5) Technological relationships amongst features (such as coupling) have not been considered while generating solutions. Also note that COPE+ does not explicitly incorporate the interdependencies amongst features, i.e., how strongly a feature is desired depending on the availability of other features in a product. This is an issue that future work will focus on.

6.4. Assumptions and threats to validity

The assumptions and threats to validity of the work presented in this paper are listed below.

(1) One of the key assumptions is related to structural analysis of ESS. We have argued that reducing the number of packages being impacted by features of a product variant will reduce the effort to transition the ESS to the proposed product line. While it seems intuitive, it may not always be true.
(2) For the jEdit system, we have used feature impact analysis results from Kuhn et al. (2007) which can be a threat to the validity of conclusions.
(3) Real customers were not involved for input on features. Therefore, no feedback was available to determine the level of acceptance for the results generated by COPE+.
(4) It is also assumed that all the customers have equal weights (importance) which does not hold in most real-world situations.

7. Future work

There are several directions for future work. A prototype tool is being planned to automate the COPE+ method. The existing method lacks cost–benefit analysis of proposed product evolution strategies. The reason for not including economic and cost estimation models in this paper is because they are outside the scope defined. The focus of this paper is to provide decision support (for evaluation and selection of product portfolios) using the structural and behavioral knowledge of the system under analysis. The economic analysis including the cost and benefits of proposed product portfolios is presented in Ullah et al. (2010). SIMPLE (Bockle et al., 2004) is applied to determine the cost of proposed product portfolios and benefit models from management information systems (MIS) discipline are applied to determine the return on investment. A detailed comparative analysis of our method with the works in marketing and MIS is also presented in Ullah et al. (2010).

Further case studies with real customers are expected to validate the results especially from the customers' side. In addition, we are also planning to remove some of the constraints and assumptions currently being applied in COPE+. For example, presently all the customers are being considered as equally important. Although it is relatively simple to adjust the model to remove this constraint, most of the existing clustering algorithms do not support assignment of different weights to customers reflecting their importance levels.

COPE+ currently generates horizontally differentiated product variants. Such product variants share some features with each other but every product variant has some unique features as well. This is because of the non-overlapping customers' clusters being generated by the clustering algorithm. In future, we plan to generate vertically differentiated product variants which closely correspond to the horizontally differentiated product variants proposed by COPE+ for comparing cost and benefits across two types of segmentations (Nault and Wei, 2005). Vertically differentiated product variants are created by adding features to the smallest (in terms of number of features) product variant. Hence, the smallest product is contained within the next higher level product and so on with one super-product that contains all the features.

Acknowledgements

The authors would like to thank Barrie Nault for his valuable comments and suggestions. This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (Discovery Grant no. 250343-07 for Günther Ruhe). Vahid Garousi is supported by the NSERC Discovery Grant no. 341511-07 and also by the Alberta Ingenuity New Faculty Award no. 200600673.

Appendix A. Extremal quantitative simulation

Given two statecharts $G^1 = (S^1, L^1, T^1, t_0)$ and $G = (S, L, T, s_0)$, extremal quantitative simulation (or mq-simulation for short) (Sokolsky et al., 2006) is recursively defined. For example, G^1 simulates G is defined by the function $Q(s_i, t_i)$. There are two possible scenarios when evaluating this function.

1. There is no outgoing transition from s_i:

$Q(s_i, t_i)$ is equal to the results of the name matching function for the nodes s_i and t_i.

2. There is at least one outgoing transition from s_i:

For every outgoing transition x from s_i to s'_i, $Q(s_i, t_i)$ is equal to the product of the following three sub-expressions:

a. Result of the name matching function for the nodes s_i and t_i.
b. Result of the label matching function for the transition x from s_i to s'_i and the transition y from t_i to t'_i. s'_i is the child (also called successor state) of s_i and t'_i is the child of t_i. In case of multiple transitions originating from t_i, the one resulting in the highest value of the label matching function is considered.
c. Recursive call to the simulation function with the children of nodes s_i and t_i represented as s'_i and t'_i.

More formally, the mq-simulation function can be defined as:

$$Q(s_i, t_i) = \begin{cases} N(s_i, t_i) & \text{if there is no outgoing transition from } s_i \\ N(s_i, t_i) \cdot \prod \max\{L(x, y) \times Q(s'_i, t'_i) : \\ \qquad \text{subject to all children } t'_i \text{ of } t_i\} \end{cases} \quad (A.1)$$

References

Arnold, R., 1996. Software Change Impact Analysis. IEEE Computer Society Press, Los Alamitos, USA.
Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, second ed. Addison-Wesley Professional.
Bockle, G., Clements, P., McGregor, J., Muthig, D., Schmid, K., 2004. A cost model for software product lines. Lecture Notes in Computer Science 3014, 310–316.
Carlshamre, P., 2002. Release planning in market-driven software product development. Requirements Engineering Journal 7 (3), 139–151.

Chang, C., Christensen, M., Zhang, T., 2001. Genetic algorithms for project management. Annals of Software Engineering 11, 107–139.
Clements, P., Northrop, L., 2001. Software Product Lines: Practices and Patterns, third ed. Addison-Wesley Professional.
Doval, D., Mancoridis, S., Mitchell, B., 1999. Automatic clustering of software systems using a genetic algorithm. In: Proceedings of the 9th International Conference of Software Technology and Engineering Practice. Pittsburgh, USA, pp. 73–81.
Dzidek, W., Arisholm, E., Briand, L., 2008. A Realistic evaluation of the costs and benefits of UML in software maintenance. IEEE Transactions on Software Engineering 34 (3), 407–432.
Ebert, C., Smouts, M., 2003. Tricks and traps of initiating a product line concept in existing products. In: Proceedings of the 25th International Conference on Software Engineering, Portland, USA, pp. 520–525.
Eisenbarth, T., Koschke, R., Simon, D., 2003. Locating features in source code. IEEE Transactions on Software Engineering 29 (3), 210–224.
Frank, E., Massy, W., Wind, Y., 1972. Market Segmentation. Prentice-Hall Inc, Englewood Cliffs, NJ.
Fritsch, C., Hahn, R., 2004. Product line potential analysis. In: Proceedings of the 8th International Software Product Line Conference, LNCS 3154, pp. 228–237.
GAlib, 2009. http://lancet.mit.edu/ga.
Garousi, V., 2008. Empirical analysis of a genetic algorithm-based stress test technique for distributed real-time systems. In: Proceedings of the Genetic and Evolutionary Computation Conference, Atlanta, USA, pp. 1743–1750.
Gomaa, H., 2004. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley.
Harman, M., Mansouri, S., Zhang, Y., 2009. Search based software engineering: a comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03. Department of Computer Science, King's College, London, UK.
Helferich, A., Herzwurm, G., Schockert, S., 2005. QFD-PPP: product line portfolio planning using quality function deployment. In: Proceedings of the 9th International Conference on Software Product Lines, Rennes, France.
Helferich, A., Schmid, K., Herzwurm, G., 2006. Product management for software product lines: an unsolved problem? Communications of the ACM 49 (12), 66–67.
jEdit Project Website, 2009. http://sourceforge.net/projects/jedit.
Kang, K., 2009. FODA: twenty years of perspective on feature models. In: 13th International Software Product Line Conference, Keynote address, San Francisco, USA.
Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S., 1990. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
Kishi, T., Noda, N., Katayama, T., 2002. A method for product line scoping based on a decision-making framework. In: Proceedings of the 6th International Software Product Line Conference, LNCS 2379, pp. 348–365.
Kuhn, A., Ducasse, S., Girba, T., 2007. Semantic clustering: identifying topics in source code. Information and Software Technology 49 (3), 230–243.
Linden, F., Schmid, K., Rommes, E., 2007. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer-Verlag, New York, USA.
Mancoridis, S., Mitchell, B., Rorres, C., Chen, Y., Gansner, E., 1998. Using automatic clustering to product high-level system organizations of source code. In: Proceedings of the 6th International Workshop on Program Comprehension, Ischia, Italy, pp. 45–52.
McBride, R., Zufryden, F., 1988. An integer programming approach to the optimal product line selection problem. Marketing Science 7 (2), 126–139.
Mitchell, M., 1996. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA.
Moorthy, K., 1984. Market segmentation, self-selection, and product line design. Marketing Science 3 (4), 288–307.
Moraes, M., Almeida, E., Meira, S., 2009. A systematic review on software product lines scoping. In: Proceedings of the 6th Experimental Software Engineering Latin American Workshop, Sao-Carlos, Brazil.
Nault, B., Wei, X., 2005. Product Differentiation and Market Segmentation of Information Goods. Social Sciences Research Network.
NDepend, 2010. http://ndepend.com.
Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., Zave, P., 2007. Matching and merging of statechart specifications. In: Proceedings of the 29th International Conference on Software Engineering, Minneapolis, USA, pp. 54–64.
Northrop, L., Clements, P., 2007. A Framework for Software Product Line Practice Version 5.0. Software Engineering Institute, Pittsburgh, USA.
Object Management Group (OMG), 2009. UML 2.2 superstructure specification. Available from: http://www.omg.org/spec/UML/2.2/Superstructure/PDF.
Riebisch, M., Streitferdt, D., Philippow, I., 2001. Feature scoping for product lines. In: Proceedings of the International Workshop on Product Line Engineering – The Early Steps: Planning, Managing and Modeling, Germany.
Rittel, H., Webber, M., 1984. Planning problems are wicked problems. In: Cross, N. (Ed.), Developments in Design Methodology. Wiley, Chichester, pp. 135–144.
Rohatgi, A., Hamou-Lhadg, A., Rilling, J., 2009. Approach for solving the feature location problem by measuring the component modification impact. IET Software 3 (4), 292–311.
Rommes, E., 2003. A People oriented approach to product line scoping: enabling stakeholder cooperation with user scenarios. In: Proceedings of the International Workshop on Product Line Engineering, Germany, pp. 284–303.
Ruhe, G., 2003. Software engineering decision support – a new paradigm for learning software organizations. Advances in learning software organization. Lecture Notes in Computer Science (Springer) 2640, 104–115.
Saliu, M., Ruhe, G., 2007. Bi-objective release planning for evolving software systems. In: Proceedings of the 6th Joint Meeting of European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, Dubrovnik, Croatia, pp. 105–114.
Sanfeliu, A., Fu, K., 1983. A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man and Cybernetics 13 (3), 353–362.
Schmid, K., 2002. A comprehensive product line scoping approach and its validation. In: Proceedings of the 24th International Conference on Software Engineering, ACM, New York, USA, pp. 593–603.
Schmid, K., John, I., 2002. Developing, validating and evolving an approach to product line benefit and risk assessment. In: Proceedings of the 28th Euromicro Conference, Dortmund, Germany, pp. 272–283.
Sokolsky, O., Kannan, S., Lee, I., 2006. Simulation-based graph similarity. In: Hermanns, H., et al. (Eds.), In: TACAS 2006: Proceedings, LNCS, vol. 3920. Springer, pp. 426–440.
Sundman, J., 2007. Impact analysis with rational architecture management tools. Available from: http://www.ibm.com/developerworks/rational/library/apr07/sundman/index.html.
Turban, E., Aronson, J., Liang, T., 2004. Decision Support Systems and Intelligent Systems. Prentice-Hall.
Ullah, M., Ruhe, G., 2007. One product versus product line: decision support based on customer needs analysis. In: Proceedings of the 11th International Conference on Software Product Lines, vol. 2, Kyoto, Japan, pp. 174–183.
Ullah, M., Ruhe, G., Garousi, V., 2009. Towards design and architectural evaluation of product variants: a case study on an open source software system. In: Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering, Boston, USA, pp. 141–146.
Ullah, M., Wei, X., Nault, B., Ruhe, G., 2010. Balancing business and technical objectives for supporting software product evolution. International Journal of Software Engineering and Computing, submitted.
Ullah, M., 2009a. COPE+ and its application. Technical Report SERG-2009-01. University of Calgary, Canada.
Ullah, M., 2009b. A method for design and evaluation of product variants. Technical Report SERG-2009-03. University of Calgary, Canada.
van Zeeland, D., 2009. Reverse-engineering state machine diagrams from legacy C-code. Master's Thesis. Department of Mathematics and Computer Science, Eindhoven University of Technology.
Wall, M., 1996. A genetic algorithm for resource-constrained scheduling. Ph.D. Thesis. Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, USA.
Wu, L., Sahraoui, H., Valtchev, P., 2005. Coping with legacy system migration complexity. In: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, Shanghai, China, pp. 600–609.

Muhammad Irfan Ullah is a PhD candidate in the Department of Computer Science of University of Calgary, Canada. His MS and BSc degrees were from Lahore University of Management Sciences and University of Engineering and Technology Taxila, respectively. Irfan's research interests are in the areas of software architectures, software release planning and software product lines. He has received several awards and scholarships during his academic career.

Günther Ruhe holds an Industrial Research Chair in Software Engineering at the University of Calgary. His interdisciplinary research in the areas of product release planning and project management includes mathematical optimization, software measurement, process modeling and simulation as well as empirical methods. He received a doctorate degree in Mathematics with emphasis on Operations Research from Freiberg University, Germany and a doctorate degree in Computer Science from University of Kaiserslautern, Germany. From 1996 to 2001, he was deputy director of the Fraunhofer Institute for Experimental Software Engineering in Germany. He is the author of three books and several book chapters. Ruhe has published more than 160 reviewed research papers at journals, workshops and conferences. He is the Associate Editor of the IST journal and a member of the Editorial Board of a number of journals. Ruhe is a member of the ACM, the IEEE Computer Society, and the Informatics Society GI in Germany.

Vahid Garousi is an Assistant Professor of Software Engineering and an Alberta Ingenuity new faculty at the University of Calgary, leading the Software Quality Engineering (SoftQual) research group. He received a PhD in software engineering from Carleton University in 2006. His MSc and BSc degrees were from the University of Waterloo in 2003, and Sharif University of Technology in 2000, respectively. Vahid has been on the program or organization committees of many international, IEEE and ACM conferences. He is a member of the IEEE and the IEEE Computer Society, and is a registered professional engineer in Canada. His research interests include: model-driven development, testing and quality assurance, and applications of optimization and evolutionary computation to software testing.