Using similarity metrics for mining variability from software repositories

Mannion, Mike; Kaindl, Hermann

Published in:
Proceedings of the 18th International Software Product Line Conference

DOI:
10.1145/2647908.2655964

Publication date:
2014

Document Version
Peer reviewed version

Link to publication in ResearchOnline

Citation for published version (Harvard):
Mannion, M & Kaindl, H 2014, Using similarity metrics for mining variability from software repositories. in *Proceedings of the 18th International Software Product Line Conference.* vol. 2, ACM, New York, pp. 32-35. https://doi.org/10.1145/2647908.2655964

Using Similarity Metrics for Mining Variability from Software Repositories

Mike Mannion
Executive Group
Glasgow Caledonian University
70 Cowcaddens Road, Glasgow
+441413313285
m.a.g.mannion@gcu.ac.uk

Hermann Kaindl
Vienna University of Technology, ICT
Gußhausstr. 27-29
A-1040 Vienna, Austria
+43 1 58801-38416
kaindl@ict.tuwien.ac.at

ABSTRACT

Much activity within software product line engineering has been concerned with explicitly representing and exploiting commonality and variability at the feature level for the purpose of a particular engineering task e.g. requirements specification, design, coding, verification, product derivation process, but *not* for comparing how *similar* products in the product line are with each other. In contrast, a case-based approach to software development is concerned with descriptions and models as a set of *software cases* stored in a repository for the purpose of searching at a product level, typically as a foundation for new product development. New products are derived by finding the most similar product descriptions in the repository using *similarity metrics*.

The new idea is to use such similarity metrics for mining variability from software repositories. In this sense, software product line engineering could be informed by the case-based approach. This approach requires defining and implementing such similarity metrics based on the representations used for the software cases in such a repository. It provides complementary benefits to the ones given through feature-based representations of variability and may help mining such variability.

Categories and Subject Descriptors

D.2.13 [Reusable Software]

General Terms

Design.

Keywords

Product lines, commonality and variability, feature-based representation, case-based reasoning, similarity metrics

1. INTRODUCTION

In Software Product Line Engineering (SPLE), as a software product line grows to several tens or even hundreds of products, different product groups in the product line are often overseen by different product managers managing in different markets using different teams of engineers. Consequently it can be very difficult to monitor and fully understand the degree of similarity different products have with each other. The following scenarios are common.

If a product manager's reward and recognition are based on successful sales, products often gain new features ("feature creep") as managers strive to be successful even if it means straying into different markets and taking market share from other colleagues' products. So it can be helpful for a product line manager to know how the products in a product portfolio are becoming more or less similar to each other. Secondly when entering a new target market, it can be helpful to know what products in the current portfolio might be closest to the product descriptions that are believed to work for the new market, and hence adapted. Thirdly, from a product line platform architect's perspective, the value of a product platform decreases as the amount of commonality reduces and a decision is sometimes required about when to break one product line platform into more than one and to support multiple product lines. Each of these scenarios can have a significant impact on market positioning as well as product line development and maintenance efficiency. Over time significant parts of product line models can become less efficient and effective as a vehicle for product derivation and need re-engineering. This paper argues that we can learn lessons from Case-Based Reasoning (CBR) for anticipating this re-engineering task.

In the next sections we sketch both *feature-model based* and *case-based development*, to make the paper self-contained. Then we briefly contrast these approaches explaining where the use of similarity metrics may have a role to play in mainstream SPLE. Finally, we discuss a few ideas of using similarity metrics for mining variability (also based on the literature).

2. PRODUCT LINE DEVELOPMENT USING FEATURE MODELS

Most software product line engineering projects include the significant task of identifying and describing the key features of each product in the product line. The set of product descriptions is captured in a single *feature model* that contains all common and variant features of the software product line at different levels of abstraction. It can be helpful to organize a feature model as a forest, in which the features are related to each other in parent-child relationships where the children can be said to elaborate the detail of a parent feature [1]. Feature model representations are often some combination of text-based, logic-based, or set-algebraic based.

In principle, a feature model has proved to be an enduring concept in software product line development because it is straightforward conceptually and visually to model commonality and variability, to add additional information to each feature, and to view and navigate between different levels of the forest. The value of a feature model lies in the cleanliness and efficiency with which it can be used to derive the features of a new product that satisfy the constraints in the feature model.

In practice, the construction and use of a feature model is a highly complex process. Over time, a product line can evolve to have tens, hundreds or occasionally even thousands of products; sometimes one product consists of another product which has its own product line. In feature model construction and maintenance, maintaining a precise and detailed understanding of the model, beyond a certain threshold, of what features are in what products, what features are similar across different products, or what products have become similar to other products, becomes increasingly difficult.

In product derivation when a feature model is large and complex, the corresponding number of feature selections and their interdependencies is also large and complex, and selection errors often occur e.g. selected features do not meet product market needs, or do not satisfy feature model constraints. Resolving these errors is usually achieved either by modifying the selection choices made or redesigning the feature model.

Feature model re-engineering often occurs following one or more automated analysis approaches. Benavides et al. [2] showed that the purposes of such analysis varied and included determining:

- if a specific product configuration satisfies the constraints of the feature model
- if there are *any* product configurations that satisfy these constraints, how many products there are
- how many products satisfy a given set of features
- whether there are anomalies in the feature model itself e.g. contradictions, redundancy
- the degree (expressed as a numeric value) to which a feature model has variable or common features.

Whilst the purposes varied, the technical approach had two principal steps:

(i) the input parameters (e.g. a feature model and/or a partial configuration) are translated into a specific representation e.g. propositional logic, constraint programming, description logic or ad–hoc data structures

(ii) off-the–shelf solvers or specific algorithms are used to undertake the analysis and provide the results as an output.

The outputs of these approaches usually provide some additional information about the strengths and weaknesses of the commonality and variability structure in the underlying feature model, which often causes it to be re-engineered to some extent. However, much of this information is often at such a fine level of granularity, that re-engineering one part of the model can generate problems elsewhere: a case of not seeing the wood for the trees. Additional tools are required that can provide more than one lens onto large scale feature models from different perspectives.

Other work on re-engineering has been situated within a reverse engineering context or a domain engineering context rather than the explicit purpose of maintaining an existing product line. In [3], a model comparison tool, EMF Compare, is presented that assumes product model descriptions have been written in a Common Variability Language (CVL) and implements a process for constructing a generic product line model by matching commonality and variability points in different product models. In [4], a method is described for detecting changes to features of different product variants during evolution using a differencing algorithm. This algorithm casts the problem as a set of pairwise comparisons across *N* product variants, to find a maximum common subgraph from two typed attributed graphs (TAG), in which each TAG consists of three types of graph nodes, RootFeature, LeafFeature and CompositeFeature, and each feature has three properties i.e. name, a parent feature, and a (possibly empty) set of sub-features.

In their review of software product line evolution approaches Laguna and Crespo [5] discovered that much of the work to date can be categorized into reengineering of (typically object-oriented) legacy code and requirements; specific aspect-oriented or feature-oriented refactoring into SPLs, and refactoring for the evolution of existing product lines. They discovered that whilst there were many published examples of industrial reengineering of legacy systems, there were far fewer examples of product line refactoring.

3. PRODUCT LINE DEVELOPMENT USING CASE-BASED REASONING

In cognitive science, Gentner [6] set out a structure-mapping theory for analogy that argued that analogy is characterized by the mapping of relations between objects, rather than the attributes of objects, from base to target, and that greater weighting is given to higher-order relations. Case-based Reasoning is grounded in cognitive science and is an automated approach to problem solving that is based on retrieving the most similar previous case to the problem to be solved. New product development is then grounded in adapting this case to build a solution. In many CBR applications usually the retrieved products are the *k* most similar to the target problem (“k nearest neighbour” retrieval) or simply k-NN (e.g. [7]). Alternatively, the retrieved products may be those whose similarity to the target problem exceeds a predefined threshold. In some CBR applications, the product case file may also include products that whilst similar in principle did not work as expected in practice.

There are many ways to shape the product case file, to represent each product and to measure similarity. Choices made depend on the application context, the problem to be solved, the task to be performed, and the user class. Similarity matching is achieved by comparing some combination of *surface* features i.e. those provided as part of its description (typically represented using attribute-value pairs), *derived* features (obtained from a product’s description by inference based on domain knowledge) and *structural* features (represented by complex structures such as graphs or first-order terms). Depending on the complexity of the representation used, an overall similarity measure is computed from the weighted similarity measures of different elements.

The ReDSeeDS project (http://www.redseeds.eu) developed a specific similarity metric including textual, semantic and graph-based components [8]. For similarity matching, it compares requirements *representations* (usually in requirements specifications or models) rather than requirements per se [9]. It even permits reuse given a *partial* requirements specification and without the need to develop a “complete” requirements specification first [10]. The specification of these new requirements can be facilitated, since the retrieved software product contains related requirements, which may be reused as well and the implementation information (models and code) of (one of) the most similar problems can then be taken for reuse and adapted to the newly specified requirements. There are also well-defined reuse processes for this approach, even tightly connected with tool support (in parts) [11].

The value of CBR lies in its conceptual simplicity and modelling flexibility, and hence the efficiency with which it can be used to identify existing products that satisfy the requirements of new products. In practice, the modelling flexibility becomes a hindrance as the number of products and their complexity significantly increases such that similarity matches reduce and/or retrieval times increase. In any consideration of re-engineering the product case file, retrieval computation time versus retrieval precision versus cost of re-engineering are usually the key issues.

4. CONTRASTING THESE APPROACHES

SPLE and CBR have both been established to support reuse related to software families but address it differently. Table 1 summarizes these differences and provides an overview. In SPLE, the premise underlying the construction of a single large feature model is that the rigour and consistency of the model structure is used to directly derive new products from existing product elements. The cost of model construction and maintenance is large. In CBR, in contrast, the premise is that each product is constructed by effectively "cutting and pasting" from the nearest product that has already been built. The cost of constructing a set of product descriptions is small though it does rely on consistent representations of products to enable similarity matching algorithms to work effectively.

In SPLE, precision about feature naming and identity, feature description and feature relationships, across the product line, are essential to enable commonality and variability to be exploited correctly in the feature model. In CBR, less precision is required for product descriptions because similarity metrics can be used to identify similar features. CBR applications require vocabularies to support text-based similarity matching.

Table 1. Overview

	SPLE	CBR
Model Structure Construction	Complex	Straightforward
Model Content	Detailed, Precise	Good-enough, Supported by vocabularies
Product Derivation	Constrained facilitated automated product derivation	Adaptation of automated retrieval of similar cases

In SPLE, the new product that is required is specified by a product engineer being presented with a number of product feature options that are wholly consistent with the feature model that has been constructed. While this method can work, many requirements engineers can feel constrained by this approach if it does not reflect their way of thinking and it is easy to get lost in the detail of choices. In CBR, the new product that is required is partially specified up-front, and this partial specification is then used to retrieve similar products which can then be amended. This enables an engineer to focus on key features, and then make judgments on what else is required or not.

So, while feature models represent commonality and variability explicitly, CBR relies on similarity metrics for identifying similar software cases at the time of reuse. Table 2 shows that these approaches have different key properties and trade-offs between costs of making software artefacts *reusable* and benefits for *reusing* them.

Table 2. Costs vs. benefits

	Costs of making *reusable*	**Benefits for *reuse***
Feature-model based	Substantial	Facilitates automated product derivation
Case-based	Negligible	Facilitates finding similar cases for reuse

5. SIMILARITY METRICS FOR MINING VARIABILITY

5.1 Mining Variability in Feature Models

Evolutionary algorithms can be used to reverse engineer feature models. However identifying parents can be problematic. In [12, 13] fitness functions deployed over representative feature sets from publicly available case studies could generate feature models that denoted proper supersets of the desired feature sets with only a small number of generations but often contained surplus features. Reducing the surplus took longer, requiring more generations, and balancing precision and efficiency of fitness function combinations remains an open question.

5.2 Mining Variability without Feature Models

In [14, 15] a recommender system is presented that relies on data mining techniques to construct a product line feature model from descriptions of a set of discrete products. It uses an incremental diffusive clustering (IDC) algorithm (that deploys a *k*-nearest neighbour machine learning method using a cosine similarity metric) to identify features to be placed in a feature pool. An analyst prepares an initial product feature profile that is converted to term vector form, and then compared with the term vector representation of each feature in the feature pool using cosine similarity. Features are then ranked according to their similarity to the product description, and presented to the analyst for confirmation.

Text-based similarity metrics using term vectors can be enhanced by semantic and graph-based components [8]. Another approach is to measure similarity in terms of behavior, i.e., state changes in response to external stimuli [16]. In [17], a neural network self-organizing map (SOM) is used to identify and create a similarity structure between products which can then be searched to identify the most appropriate existing product upon which to base the development of a new product. The SOM algorithm is run over a set of product line requirements that have been converted to requirement data vectors in a consistent subject-object-verb format.

6. CONCLUSION

Broadly the focus in SPLE is on model precision and development efficiency, whereas the focus in CBR is identifying the most similar product available and adapting it. These approaches can be complementary. One way would be to explore the greater use of similarity metrics within SPLE.

7. REFERENCES

[1] Mannion, M. and Kaindl, H., 2008 Using Parameters and Discriminants for Product Line Requirements. *Systems Engineering*, 11(1), 61–80.

[2] Benavides, D., Segura, S., Ruiz-Cortes, A., 2010 Automated Analysis of Feature Models 20 years Later: A Literature Review. *Information Systems*, 35(6):615–636.

[3] Zhang, X., Haugen, O., Moller-Pedersen B., 2011 Model Comparison to Synthesize a Model-Driven Software Product Line, In *Proceedings of 15th International Conference on Software Product Lines*, 90–99.

[4] Xue, Y., Xing, Z., Jarzabek, S., 2010 Understanding Feature Evolution in a Family of Product Variants, *Proceedings of 17th Working Conference on Reverse Engineering*, 109–118.

[5] Laguna,M., Crespo, Y., 2013 A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming*, 78(8):1010–1034.

[6] Gentner, D., 1983 Structure-Mapping: A Theorerical Framework for Analogy, *Cognitive Science*, 7, 155–170.

[7] Cover, T M., Hart P E., 1967 Nearest Neighbour Pattern Classification, *IEEE Trans on Information Theory*, 13:21-27.

[8] Bildhauer, D., Horn, T., Ebert, J., 2010 Similarity-driven software reuse. In *Proceedings of CVSM'09*, IEEE, 31–36.

[9] Kaindl, H. and Svetinovic, D., 2010 On confusion between requirements and their representations. *Requirements Engineering*, 15, 307–311.

[10] Kaindl, H., Smialek, M., and Nowakowski, W., 2010. Case-based Reuse with Partial Requirements Specifications. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE 2010)*, 399–400.

[11] Kaindl, H., Falb, J., Melbinger, St. and Bruckmayer, Th., 2010 An Approach to Method-Tool Coupling for Software Development. In *Proceedings of the Fifth International Conference on Software Engineering Advances (ICSEA 2010)*, IEEE, 101–106.

[12] Benavides, D., Felfernig, A, Galindo, J.A., Reinfrank, F., 2013 Automated Analysis in Feature Modelling and Product Configuration, *Proceedings of the 13th International Conference on Software Reuse (ICSR 2013)*, 160–175.

[13] Lopez-Herrejon, R.E., Galindo, J.A., Benavides, D., Segura, S., Egyed, A., 2012 Reverse Engineering Feature Models With Evolutionary Algorithms: An Exploratory Study, *Search-Based Software Engineering*, LNCS 7515, 168–182.

[14] Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., Mobasher, B., 2013 Supporting Domain Analysis through Mining and Recommending Features from Online Product Listings, *IEEE Trans on Software Engineering*, 39(12), 1736–175.

[15] Dumitru, D., Gibiec, M., Hariri, N., Cleland-Huang, J., 2011 Mobasher, B. Castro-Herrera, C., Mirakhorli, M., On-Demand Feature Recommendations Derived from Mining Public Software Repositories, In *Proceedings of 33rd International Conference on Software Engineering*, 181–190.

[16] Reinhartz-Berger, I., Sturm, A., Wand, Y., 2011 External Variability of Software: Classification and Ontological Foundations. In *Proceedings of ER'11*, Springer-Verlag Berlin Heidelberg, LNCS 6998, 275–289, 2011.

[17] Feldhusen, J., Milonia, E., Nagarajah, A., Neis, J., Schubet, S., 2012. Enhancement of adaptable product development by computerised comparison of requirement lists, *International Journal of Product Lifecycle Management*, 6(1).