Examining Usability of Classes in Collaboration with SPL Feature Model

Geetika Vyas and Amita Sharma

Abstract Software product line engineering paradigm focuses on developing families of products keeping track of their common aspects and predicted variabilities. Feature models are often used for depicting the commonalities and variabilities existing in software product lines. Classes are used to program the features of the feature models and thus a significant relationship exists between the two. As software product line focuses on reuse, we have proposed a metric to measure the degree of usability of classes in context of features which are using them. Eclipse FeatureIDE is used to prove the proposed metrics. The aim of the research is to track usability of classes keeping in mind their planned reuse, efficient development and maintenance.

Keywords Software product line engineering · Features · Feature models · Degree of usability · Eclipse FeatureIDE

1 Introduction

Software product lines engineering develops and maintains families of products keeping track of their common aspects and predicted variabilities [1]. It focuses on reusability [2]. It is structured into two main processes: domain engineering (also called engineering for reuse) and application engineering (engineering with reuse) [3]. Features are structures that extend and modify the structure of a given program in order to meet the user requirement. Feature models introduced by Kang are used to represent the features available in a product line. They portray all the configurations a product line can possibly have [4]. The concept of feature is useful for description of commonalities and variabilities not only in the analysis and design but also

Geetika Vyas (✉) · Amita Sharma
The IIS University, Jaipur, India
e-mail: geetika_vyas@yahoo.co.in

Amita Sharma
e-mail: amitasharma214@gmail.com

S.C. Satapathy et al. (eds.), *Proceedings of the International Congress on Information and Communication Technology*, Advances in Intelligent Systems and Computing 438, DOI 10.1007/978-981-10-0767-5_4

implementation phase of software product lines [5]. There exists a significant relationship between classes and features, but no significant work is done in reference to the complexity that exists across the feature–class relationship. A feature in a feature model is supported by class(s) in a class diagram. Software product line paradigm aims reuse; and like features, classes are also reused. Therefore, the relationship between feature model and class diagrams needs to be studied. The core focus of this paper is to investigate the usability of classes. We have proposed a metric to measure degree of usability of classes. Collaboration diagrams are used to check the result generated by the metrics. The proposed metric is beneficial from the point of view wherein we are able to detect the origin point of most vital classes in the feature model, and also detect the most vital features and the least vital features possibly turning dead in the future. Other possible benefits seen behind the research are planned usage of classes in the system, their better development followed by improved maintenance. The rest of the paper is organized as follows: Section 2 contains introduction of feature-oriented programming. Section 3 introduces Eclipse FeatureIDE. Section 4 contains the proposed metrics and its implementation. Section 5 contains the result, analysis and conclusion.

2 Feature-Oriented Programming

Feature-oriented programming paradigm allows decomposition of a program into its constituent features. It was designed for software product line paradigm that allows significant code reuse and the generation of many similar but functionally different programs from the same set of features simply through selection of desired features [6]. The stepwise refinement leads to a layered stack of features. This helps in constructing well-structured software that can be tailored to the specific needs of the user and the application scenario [7, 8].

3 FeatureIDE: Eclipse Plug-in

FeatureIDE is an eclipse-based integrated development environment (IDE). It provides tool support for the feature-oriented design process and implementation of software product lines [9]. Eclipse FeatureIDE provides the most powerful and commercially successful open-source enhanced IDE support for feature-oriented programming implementations [10]. Domain analysis and feature modeling are supported with graphical feature model editor. Feature implementation is supported by variety of composers like AHEAD, FeatureC++, FeatureHouse, AspectJ, DeltaJ, Munge and Antenna building program families. Out of these we have used FeatureHouse which is language independent.

4 Experimental Setup

4.1 Implementation of the Proposed Metrics

In our Previous paper we have proposed metric for degree of usability [11] let us assume an anonymous feature model and implement the proposed metrics over it. Figure 1 contains this feature model, where F1 is the root node. It has three children: F2 (mandatory), F3 (mandatory) and F4 (optional). The parent node F4 has two child features: F5 and F6. Parent nodes F2 and F3 have one mandatory child each F7 and F8, respectively. There are following dummy classes, C1, C2, C3, C4 and C5, used to implement this feature model. The usage of these classes by the features is shown in Table 1.

The degree of usability can be defined as the number of times a class is used in different features present in a feature model across the tree. It is obvious that at the root node degree of all the classes will be zero, i.e. at the origin of the class its degree of usability will always be zero. Irrespective of the traversal method the final value of degree of usability of any class will always remain same. Table 1 displays the individual class usage scenario across the feature model. It also shows the calculated value of degree of usability following both methods of traversal, i.e. breath first and depth first.

On the basis of the calculations in the above table we can conclude that classes 1 and 3 have the highest usability. They are used maximum number of times, in comparison to the other classes. The value obtained by this metric is of great worth because it is an indicator of their usage highlighting their importance and subsequent use. Degree of usability can also be derived by classifying abstract and concrete classes. The collaboration diagram generated for this feature model also reflects the same value of usability of each class across each feature. Figure 2 proves our metric, wherein it can be clearly seen that classes 1 and 3 have the maximum reusability.

4.2 Implementation of the Proposed Metric

For implementing the proposed metric, we take the example of the Direct-to-home (DTH) systems. To implement our metric we are taking the broader aspect of DTH.

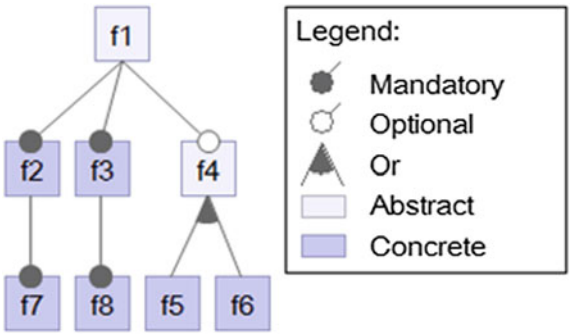

Fig. 1 Anonymous feature model developed using eclipse

Table 1 Calculated values: degree of usability

Name of feature	Name of class	Degree of usability (breath first traversal)	Degree of usability (depth first traversal)
F1 (root feature)	–	–	–
F2	Class 1, 2, 3, 4	d(C1) = 1	d(C1) = 1
		d(C2) = 1	d(C2) = 1
		d(C3) = 1	d(C3) = 1
		d(C4) = 1	d(C4) = 1
F3	Class 1, 3	d(C1) = 2	d(C1) = 4
		d(C3) = 2	d(C3) = 4
F4	Class 1, 2, 4	d(C1) = 3	d(C1) = 6
		d(C2) = 2	d(C2) = 5
		d(C4) = 2	d(C4) = 4
F7	Class 1, 2, 3, 4	d(C1) = 4	d(C1) = 2
		d(C2) = 3	d(C2) = 2
		d(C3) = 3	d(C3) = 2
		d(C4) = 3	d(C4) = 2
F8	Class 1, 3, 5	d(C1) = 5	d(C1) = 3
		d(C3) = 4	d(C3) = 3
		d(C5) = 1	d(C5) = 1
F5	Class 2, 3, 4	d(C2) = 4	d(C2) = 2
		d(C3) = 5	d(C3) = 5
		d(C4) = 4	d(C4) = 3
F6	Class 1, 2, 3	d(C1) = 6	d(C1) = 5
		d(C2) = 5	d(C2) = 4
		d(C3) = 6	d(C3) = 6

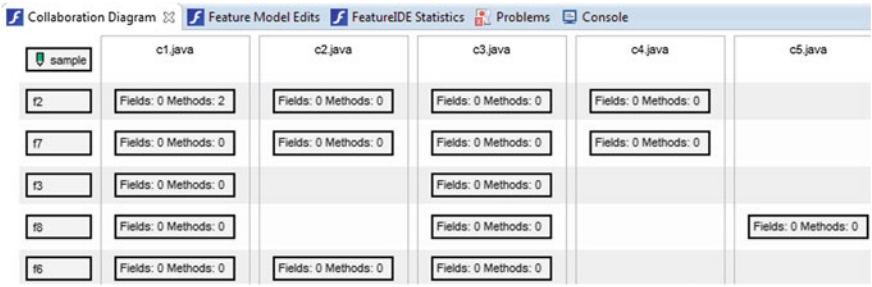

Fig. 2 Collaboration diagram for anonymous feature mode

We are focusing on its limited functionality and services. This television service is the reception of satellite programs with a personal dish installed individually at home. Its network consists of modulators, broadcasting center, encoders, satellites, multiplexers and DTH receivers. Here service provider leases Ku-band

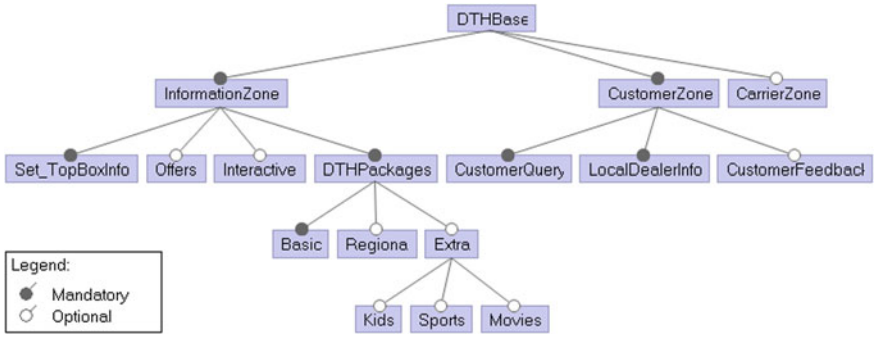

Fig. 3 Contains the feature model

transponders from the satellite. The audio, video and data signals are converted into the digital format and the multiplexer mixes these signals. At the users end, there is a small dish antenna installed and set-top boxes to decode it and viewing of numerous channels. The smallest receiving dish can be 45 cm in diameter. This transmission travels directly to the consumer through a satellite. DTH also offers stereophonic sound effects. Its advantage is that it can also reach remote areas where terrestrial transmission and cable TV cannot penetrate. Along with enhanced picture quality, other benefits are that it allows interactive TV services such as movie-on-demand, internet access, video conferencing and e-mail also. Figure 3 shows the DTH feature model.

Here DTHBase (root feature) has InformationZone (mandatory), CustomerZone (mandatory) and CarrierZone (optional) features. InformationZone has two mandatory features and two optional features, out of which DTHPackages feature has Basic (mandatory) and two features Regional (optional) and Extra (optional) features. Feature Extra has Kids (optional), Sports (optional) and Movies (optional) features. In total this feature model can have 144 valid configurations.

The basic (dummy) classes in this software include CostInfo, CustInfo, LocalDealerInfo and PackageInfo. The java files which use these dummy classes are jak files (extended files of java), also called FeatureIDE files. In later stages, as per need these classes will be refined in order to add new features in the software. These classes are dummy by nature. Implementation needs more effort on the programmer's part.

Using the depth first traversal method, the degree of usability of dummy class CostInfo is as follows:

At Feature InformationZone, d (CostInfo) = 1, (assuming the degree of Class CostInfo1 at DTHBase is 0)
At Feature Set_TopBoxInfo, d (CostInfo) = 2,
At Feature DTHPackages, d (CostInfo) = 3,
At Feature Basic, d (CostInfo) = 4,
At Feature CustomerQuery, d (CostInfo) = 5.

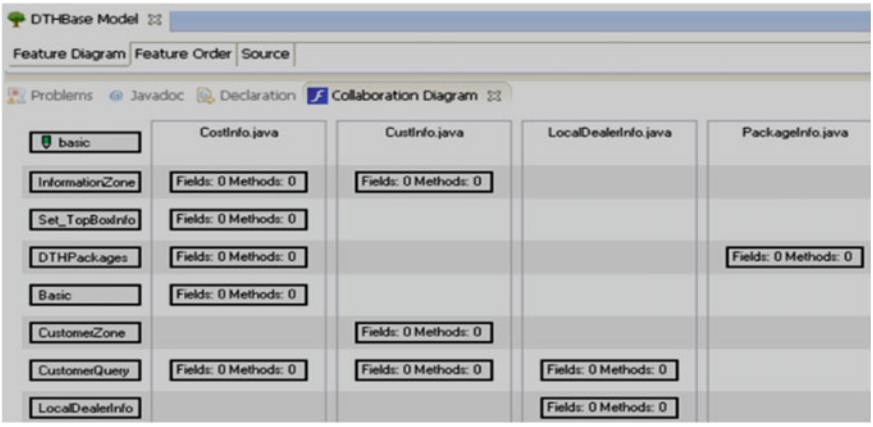

Fig. 4 Collaboration diagram of DTH service feature model

Using the breath first traversal method, the degree of usability of dummy class CostInfo is as follows:

At Feature InformationZone, d (CostInfo) = 1, (the degree of Class CostInfo1 at feature DTHBase is 0)
At Feature Set_TopBoxInfo, d (CostInfo) = 2,
At Feature DTHPackages, d (CostInfo) = 3,
At Feature CustomerQuery, d (CostInfo) = 4,
At Feature Basic, d (CostInfo) = 5.

Thus we can conclude that the degree of usability of dummy class CostInfo, irrespective of the traversal method, is 5 and is the highest. To check whether the metric is returning the correct value, we refer to the collaboration diagram generated by Eclipse FeatureIDE. Once we define the FeatureIDE files, FeatureIDE generates a collaboration diagram which shows the collaboration of all classes with feature. Figure 4 contains the collaboration diagram for this example. The columns in the diagram contain the classes and rows contain the features which are using these classes. It clearly depicts that class CostInfo is the most referred class. Out of the four dummy classes it is the most frequently used one. Through the diagram also we come to the conclusion that the degree of usability of dummy class CostInfo is 5.

5 Analysis and Conclusion

A significant relationship is seen between features and classes. The strong association between these two leads us to relate the core focus of SPL in both the respects, i.e. to discuss usability of features and classes as well. The available measures in literature limit the complexity within the features. The complexity across the classes and features relationship remains untouched. Available metrics do not suffice in

controlling the usability of the whole system. The metric proposed in our paper is generating the degree of usability of various classes used in the example of DTH services. The collaboration diagram of the example also proves that the metrics are returning values which are true from the practical point of view. The classes which have highest usability theoretically have the same usability practically also. The calculated value thus obtained by our metric will help us check the usage of each class. This will ultimately benefit the programmers, practitioners and researchers in better understanding of classes. It will also help in improved control and development of the product line. It will help determine the best ways for the maintenance of classes which are an integral part of the whole process. Our current work is generalized by nature and is in its initial stages. Our proposal still needs validation. We are currently working upon the theoretical and empirical validations by studying variety of feature models and the classes used for implementing them [12]. We will also apply the metric over more examples to calculate accurate results. Further experimentation will validate our work and help us draw the final conclusions.

References

1. Clements, P., Northrop, L., Software Product Lines: Practices and Patterns, Addison-Wesley, Boston (2001).
2. Montagud, S., Insfran, E., Abrahão, S., A systematic review of quality attributes and measures for software product lines, Software Quality Control, Vol. 20, Issue 3–4, pp. 425-486, (2012).
3. Pohl, K., Bockle, G., Linden, F., Software Product Line Engineering: Foundations, Principles and Techniques, Springer-Verlag New York, Inc. Secaucus, NJ, USA, (2005).
4. Kang, Kyo, C., Lee, Jaejoon, Kim, Kijoo, Kim, Jounghyun, G., Euiseob, S., Moonhang, H., FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering (Springer Netherlands), vol. 5, pp. 143–168, (2004).
5. Bagheri, E., Gasevic, D., Assessing the Maintainability of Software Product Line Feature Models Using Structural Metrics, Software Quality Journal, vol. 19 (3), pp. 579–612, (2011).
6. Apel, S., Kastner, C., An Overview of Feature-Oriented Software Development, J. Object Technology (JOT), vol. 8, No. 5, pp. 49–84, (2009).
7. Sharma, A., Sarangdevot, S.S., Investigating the Application of Feature-Oriented Programming in the Development of Banking Software Using Eclipse-Featureide Environment, International Journal of Computer Science & Technology, Vol. 2, Issue 1, pp 53–57,(2011).
8. Rumbaugh, J., Blaha, M., R., Object Oriented Modeling and Design, Prentice Hall, (1991).
9. Leich, T., Apel, S., Marnitz, L., Tool Support for Feature-Oriented Software Development - FeatureIDE: An Eclipse-Based Approach, OOPSLA Workshop on eclipse technology eXchange (ETX), San Diego, USA, (2005).
10. Kastner, C. FeatureIDE: A Tool Framework for Feature-Oriented Software Development, Proc. 31st Int'l Conf. on Software Engineering (ICSE), Vancouver, Canada, IEEE, (2009).
11. Sharma, A., Vyas, G., New Set of Metrics for Accessing Usability in Feature Oriented Programming, International Journal of Computer Applications, vol. 81(11), pp. 19–22, (2013).
12. Siegmund, J.; Siegmund, N.; Apel, S., "Views on Internal and External Validity in Empirical Software Engineering," in Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, vol. 1, no., pp. 9–19, 16–24 May 2015.