New Set of Metrics for Accessing Usability in Feature Oriented Programming

Amita Sharma, Ph.D
Dept of CS & IT,
The IIS University,
Jaipur, India.

Geetika Vyas
Dept of CS & IT,
The IIS University,
Jaipur, India.

ABSTRACT

Quality and complexity assessment are critical aspects in feature oriented software development. Feature oriented development is part of software product line methodology where software is framed using feature concept. In course of research; object oriented metrics are being used to assess the quality of feature model. But these metrics limit the complexity within the features and complexity across the classes and features relationship are untouched .This state of art proposes a new set of metrics to provide usability and complexity assessment crosswise the feature-class relationship. This will provide better understanding of feature oriented system and help to develop a sustainable system.

Keywords

Feature oriented programming, Object Oriented Metrics, Software Product Line, and Feature-Class Relationship.

1. INTRODUCTION

The modeling foundation of Software Product Line Engineering (SPLE) is the segregation of variant features of all the products which belong to a family. In brief, its aim is to catalog what is common and what differs between products. Feature is a basic term associated with SPL and feature model diagram portrays the product deviation. SPLE is a powerful approach to increase the efficiency of the software engineering process and variety of software system can be developed from a single software product line.

Feature-Oriented Programming paradigm allows decomposition of a program into its constituent features. FOP was designed for Software Product Line thus allowing significant code reuse and the generation of arbitrary programs consisting of user-selected features. Features are structures that extend and modify the structure of a given program in order to meet the user requirement. Feature models first introduced by Kang are used to represent the features available in a product line. In other words they portray all the configurations that a product line can have. 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 focus of this paper is to investigate the usability of classes, and to propose metrics which aim to calculate the usability of classes. The possible benefits are seen behind the research are viewing the further usage of classes in the system, their better maintenance and development.

The rest of the paper is organized as follows: Software product line concept is presented in section II, section III contains Feature oriented programming concepts, section IV contains concepts related to features and feature models, section V contains the available metrics, section VI talks about class diagrams and their relation with feature models, section VII talks about the proposed metrics to calculate usability of classes, section VIII contains the analysis part and the result obtained, section IX concludes the paper.

2. SOFTWARE PRODUCT LINE ENGINEERING

Products being developed for the international market must be adapted for diverse cultural or legal environments, and for different languages, and must provide appropriate user interfaces. Due to cost and time constraints it is not feasible for software developers to develop a new product from scratch for each new customer, and so software reuse has to be increased. Software Product Line Engineering (SPLE) offers a way out to these not so new, but increasingly challenging problems.

SPLE is an approach that develops and maintains families of products keeping track of their common aspects and predicted variability's at the same time [1]. SPLE focuses on reuse and is a viable and important software development paradigm.[2]

3. FEATURE ORIENTED PROGRAMMING

Feature Oriented Programming was designed keeping an eye towards SPLE. The basic idea is to decompose a software system in terms of the features it provides i.e. to modularize the software system into feature model which is a representation of product features and the dependencies of these features. It is used explicitly for systematically defining the commonality and variability [3]. This methodology is recommended because of its ability to produce numerous similar but functionally different programs from the same set of features [4]. This is achieved by simply selecting the desired features. It helps in engineering well-structured software, tailored to the specific user needs and the application scenario. This methodology increases several software quality attributes like adaptability, modularity, traceability, readability, maintainability, extendibility, understandability reusability, flexibility, and ease of evolution.

4. FEATURES AND FEATURE MODEL

"A feature is a structure that extends and modifies the structure of a given program in order to satisfy a stake holder's requirement, to implement and encapsulate a design decision, and to offer a configuration option".

It is a unit of functionality which satisfies a requirement, represents a design decision, and provides a potential configuration option [5]. They are modular entities which encapsulate a particular functionality of the system.

They help in the explanation of commonality and variability during the analysis, design, and implementation phases of software product lines [6]. Classification of features increases the comprehensibility of a product line. By selecting and removing features, software provides different facilities and different configurations. Typically, from a collection of features, many different software systems can be generated that share some common features and at the same time differ in others. A feature model diagram represents all the products of the software product line.

Hierarchically arranged features of a feature model can be classified as [7]:

- Mandatory
- Alternative feature group
- Optional
- Or feature group
- Excluded
- Includes

Fig. 1 shows a sample feature model of e-shopping. This modified feature model explains the whole process of online shopping and its major features like catalog, payment, security and search. The steps undertaken by the website visitor are : to login, view catalogue or use search option, select items and add them to cart, select suitable payment mode, do payment and log off. For this purpose each user has a unique id and an account which holds his address, order status and other details. The website owner has the authority to add/modify/delete a product(s)/ user(s) and to keep track of order details and delivery of the same.

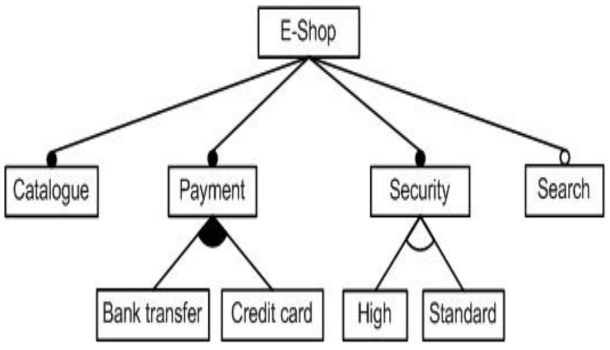

Fig. 1. Sample feature model for e-shopping.

5. METRICS

A variety of traditional metrics are available for assessing software quality and complexity. These metrics are software measurement units, which measure the degree to which a given system, component or process possesses a given attribute [8]. They use numerical ratings to measure various domains like the complexity and reliability of source code, its length and quality of the development process and the performance of the application when completed. But as SPL has specific characteristics these measures to evaluate the quality of individual products cannot be applied easily to evaluate software product lines.

By the help of object-oriented metrics one can measure the effectiveness of object-oriented techniques in respect to the design of a system as in the case of software product line paradigm [9]. TABLE I contains few measures.

TABLE I MEASURES FOR SPL FEATURE MODEL

Measure type	Measure name
Size measure	Number of features (NF)
	Number of top features (NTop)
	Number of leaf features (NLeaf)
Structural complexity measures	Cyclomatic complexity (CC)
	Cross tree constraints (CTC)
	Ratio of variability (RoV)
	Coefficient of connectivity-density (CoC)
	Flexibility of configuration (FoC)
Length measure	Depth of tree (DT)

These measures are useful to assess the quality of feature models. But these metrics limit the complexity within the features and complexity across the classes and features relationship is untouched. These metrics do not suffice in controlling the usability or complexity of the whole system.

6. CLASS DIAGRAMS AND FEATURE MODELS

Class diagrams represent the static view of an application which includes their features, constraints and relationships - associations, generalizations, dependencies, etc. They are not only used for visualizing, documenting and describing different aspects of a system but also for constructing executable code of the software application. They describe the attributes and operations of a class and also the constraints imposed on the system.[10]

A significant relationship is seen between classes and features. A feature in a feature model is supported by class(s) in a class diagram. For example if there exists a generalization/ specialization relationship between two classes that supports two different features respectively, this generalization/ specialization relationship between two classes' also maps to feature dependency between two features.

Therefore we need to study the relationship between feature model and class diagrams. For this purpose a sample class diagram is constructed based on the e-shop feature model, shown in figure 2. This modified class diagram broadly showing the major classes to be used in the implementation of the process.

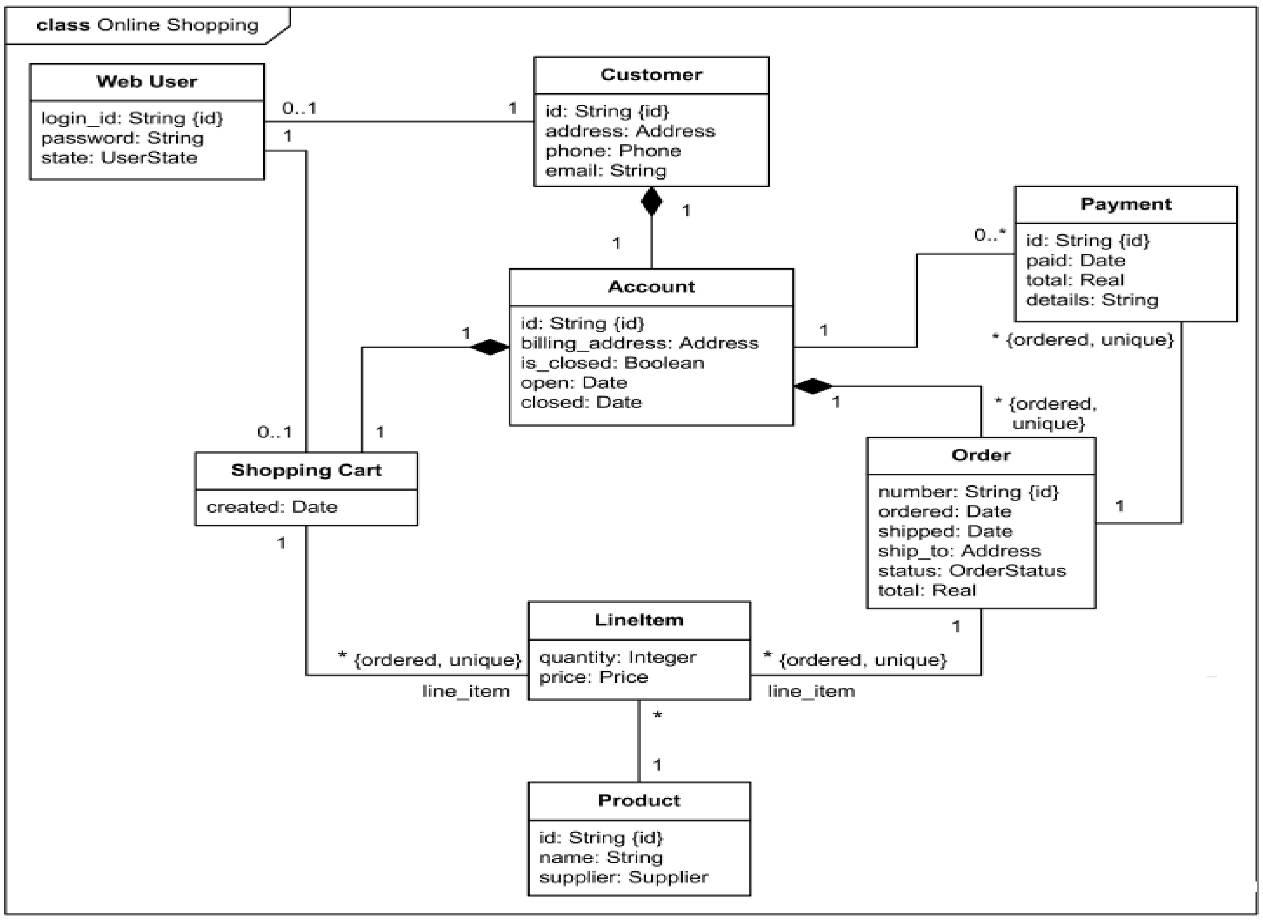

Fig. 2. Sample class diagram for e-shopping.

SPLE talks about reuse and like features, classes are also reused. This reuse can be of two types wherein the class is reused with slight or no modification, as per need.

By observing the class diagram in figure 2, we see that here also few classes are used extensively at more than one place i.e. reused. For example the class order is used at the customer end as well at the website manager end. This class also shows its effect in the database classes (not shown in the diagram) in the way that shipment/delivery of an order leads to changes in the total available number of that item. Few other classes are also reused.

SPLE and feature models focus on reusability and thus reusability in reference to classes should also be kept in mind as this will minimize the programmer's effort while coding for the commonality of the product line. This usage of classes can be determined by the number of occurrences or repetition of one class in more than one feature or its sub-features. Metrics thus developed will help reduce the implementation time, prior testing and reuse will eliminate bugs and localize code modifications when a change in implementation is required. Along with measuring the degree of usability of classes, these metrics will also help to trace the flexibility of the classes because ultimately it is the flexibility which will make further usage possible.

Developing metrics for tracing usability of classes in feature will help programmers to effectively manage various issues like building, distribution, installation, configuration, deployment, maintenance. These metrics are also worthy because if these issues are not considered, features may appear to be reusable from feature model point of view, but will not be reused practically.

7. PROPOSED METRICS

The degree of usability can be defined as the number of times a class is used in different feature's present in a feature model across the tree. It is obvious that at the root node degree of all the classes will be zero, put in other words at the origin of the class it's degree of usability will always be zero.

For our calculation let's assume a feature model where, F1 is the root node. It has three children: F2, F3 and F4. The parent node F4 has two children: F5 and F6. Parent nodes F2 and F3 have one child each F7 and F8 respectively. (F denotes feature, C denotes the class, and d(C) denotes the degree of usability.)

Across the feature model the class usage scenario is as follows:

F1 has C1, C2, and C3

F2 has C1, C2, C3, and C4.

F3 has C1, C3.

F4 has C1, C2, and C4.

F5 has C2, C3, and C4.

F6 has C1, C2, C3, and C6.

F7 has C 1, C2, C3, and C4.

F8 has C1, C3, and C5.

8. ANALYSIS AND RESULTS

As we can see that parent feature F1 has classes 1, 2, 3. Feature2 has classes 1, 2, 3 and a new class C4. So in feature F2 classes 1, 2, and 3 are reused or extended. Their degree of usability can be calculated as follows:

At F2, d (C1) =1, (assuming the degree of C1 at F1 is zero)

At F2, d (C2) =1, (assuming the degree of C1 at F1 is zero)

At F2, d (C3) =1, (assuming the degree of c1 at f1 is zero)

At F2, d (C4) =0

Likewise degree of usability can be calculated for all the classes used in the feature model, which will be 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 calculated value will help us monitor the usage of each class i.e. the most used class and the least used class. This will ultimately benefit the programmer.

9. CONCLUSION

SPL is a paradigm focusing on reuse based engineering of software's. This is attained by capturing the various variability and commonalities existing between the configurations of the target domain. Since classes are used to implement all the features, their usability should also be studied. The (re) use of classes needs to be quantified in order to track the extent of class usability and flexibility.

Our current work is general by nature and is in its initial stages. Our proposal still needs validation. We are currently working upon the theoretical and empirical validation by studying variety of feature models and the classes used for implementing them. This further experimentation will validate our work and will help us draw the final conclusions.

10. REFERENCES

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

[2] G. K. Hanssen, "Opening Up Software Product Line Engineering", Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering 2010, ACM, New York, USA, Pages 1-7.

[3] S. Apel and C. Kastner, "An Overview of Feature-Oriented Software Development," J. Object Technology (JOT), Volume 8, No. 5, 2009, Pages 49-84.

[4] A. Sharma, S.S. Sarangdevot, "Investigating the Application of Feature-Oriented Programming in the Development of Banking Software Using Eclipse-FeatureIDE Environment", International Journal of Computer Science & Technology, March 2011,Volume 2, Issue 1 ,Pages 53-57.

[5] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report Cmu/Sei-90-Tr-021 , 1990. Available at-http://www.Sei.Cmu.Edu/Library/Abstracts/Reports/90tr021.Cfm.

[6] D. B. Cuevas, "On The Automated Analysis of Software Product Lines using Feature Models-A Framework for Developing Automated Tool Support", The Department of Computer Languages and Systems, Spain, June 2007

[7] E. Bagheri, D. Gasevic, "Assessing the Maintainability of Software Product Line Feature Models Using Structural Metrics", Springer, September 2011, Volume 19, Issue 3, Pages 579-612.

[8] I.Sommerville, Software Engineering, Pearson Education, India, 2008.

[9] N.Fenton, S.L.Pfleeger, "Software Metrics: A Rigorous and Practical Approach", 2nd Ed., International Thomson Computer Press, 1996.

[10] J. Rumbaugh, M. R. Blaha , "Object Oriented Modeling and Design" , Prentice Hall, 1991.