A Case Study of Variation Mechanism in an Industrial Product Line

Pengfei Ye[1], Xin Peng[1], Yinxing Xue[2], and Stan Jarzabek[2]

[1] School of Computer Science, Fudan Univeristy, Shanghai, China
{072021110,pengxin}@fudan.edu.cn
[2] School of Computing, National Univeristy of Singapore, Singapore
{yinxing,stan}@comp.nus.edu.sg

Abstract. Fudan Wingsoft Ltd. developed a product Line of Wingsoft Financial Management Systems (WFMS-PL) providing web-based financial services for employees and students at universities in China. The company used a wide range of variation mechanisms such as conditional compilation and configuration files to manage WFMS variant features. We studied this existing product line and found that most variant features had fine-grained impact on product line components. Our study also showed that different variation mechanisms had different, often complementary, strengths and weaknesses, and their choice should be mainly driven by the granularity and scope of feature impact on product line components. We hope our report will help companies evaluate and select variation mechanisms when moving towards the product line approach.

1 Introduction

The goal of this paper is to evaluate strengths and weaknesses of variation mechanisms used in the existing Wingsoft Financial Management System[1] Product Line (WFMS-PL), developed by Fudan Wingsoft Ltd., a small software company in China. We took the following steps in this study. We first analyzed WFMS-PL variant features [7] and presented them as a feature diagram [10]. Then, we studied variation mechanisms in WFMS, namely Java conditional compilation[2], commenting out feature code, design patterns [4], parameter configuration files, and a build tool Ant[3]. Finally, we analyzed how the granularity and scope of features impact on WFMS components affects the effectiveness of variation mechanisms.

We distinguish two types of features according to the granularity of their impact, namely *fine-grained features* affecting many system components, at many variation points, and *coarse-grained features* whose code is usually contained in files that are included into a custom product that needs such features. *Mixed-grained features* involve both fine- and coarse-grained impact. Most of the WFMS features were

[1] WFMS for Shanghai Jiaotong University: http://www.jdcw.sjtu.edu.cn/wingsoft/index.jsp
[2] Java does not formally have conditional compilation, but you can implement the similarfunction: http://c2.com/cgi/wiki?ConditionalCompilationInJava
[3] http://ant.apache.org/

S.H. Edwards and G. Kulczycki (Eds.): ICSR 2009, LNCS 5791, pp. 126–136, 2009.

fine-grained features, managed with conditional compilation and/or manually commenting out the feature code.

Our study shows that different variation mechanisms have different, often complementary, strengths and weaknesses, and their choice should be mainly driven by the granularity and scope of feature impact on product line components. Fundamental differences in capabilities of variation mechanisms justify the use of multiple variation mechanisms. For example, Ant is strong in configuring coarse-grained features, but weak in configuring fine-grained features. Parameter configuration files define environmental variables and variant feature options, but require yet other mechanisms to perform the actual customizations in product line components. Design patterns reduce the coupling in code, making it easier to add, remove or change a variant feature. However, we found only few opportunities to apply design patterns in WFMS. Overloading fields in order to use the same field for different purposes usually helps only in configuring database schema. Conditional compilation is used as the main mechanism to control fine-grained variant features in Java source code, while commenting out feature code is heavily used in HTML and JSP files. In some situations, we suggest possible remedies to weaknesses of variation mechanisms used in WFMS-PL.

Particularly, multiple variation mechanisms must be used to manage each of the mixed-grained features. Our study reveals that while it is natural to match feature granularity with the proper variation mechanism, over time the inter-play between multiple variation mechanisms may be difficult to comprehend.

Variation mechanisms used in WFMS-PL are simple, freely available and commonly used in Software Product Lines (SPL) to complement component/architecture-based approaches. As yet we do not have enough material to compare them with more advanced SPL approaches, such as GEARS[4], Pure[5] or XVCL [6], with possibly better results. We are going to conduct experiments to facilitate such comparison.

In the past years, there were some case studies on variation mechanism. These studies, however, usually focused on variation implementation with certain techniques like AspectJ [13], FOP [1] or XVCL [6]. The industrial case study presented in [17] aims at architecture-based variability realization in large companies. In this paper, we analyze a real product line using a mixed set of light-weight variability mechanisms in a small company. We believe Wingsoft choice of variation mechanisms was typical, and other small companies may find our experiences reported in this paper useful when moving towards the product line approach.

2 An Overview of WFMS

WFMS was developed in 2003 and evolved to an SPL with more than 100 customers today, including major universities in China such as Fudan University, Shanghai Jiaotong University, Zhejiang University. During its evolution, Wingsoft set up product architecture and was adopting variation mechanisms such as Java conditional compilation, Ant, parameter configuration files, and design patterns to manage product variability.

[4] GEARS: http://www.biglever.com/
[5] Pure::Variants: http://www.pure-systems.com/

The core assets of the WFMS-PL were designed and implemented by few *domain engineers*. Domain engineers sometimes also played the role of an *application engineer*s responsible for initial, program-level customization of core assets for a custom product. *Service engineers*, familiar with financial business but with little or no programming knowledge, did final customer-side customizations and deployment, using readable parameter configuration files. Usually, application engineers only provided in-office application-specific implementations, and responded to requests of service engineers. Domain engineers maintained WFMS products for many customers delegating routine work to service engineers.

WMFS consists of four subsystems, namely Financial Management Subsystem (FMS), Salary Management Subsystem (SMS), Reward Management Subsystem (RMS), and Tuition Management Subsystem (TMS). We selected the TMS for our case study, as it involved types of variability and variation mechanisms that were representative of the whole WFMS. TMS is a web-based portal for students to pay online their tuition fee, with functions such as login, fee browsing, online payment, payment detail generation and bank settlement. The code of TMS is 25% of the whole WFMS system, comprising 58 Java source files, 99 JSP web pages, and several configuration files.

A WFMS feature diagram is shown in Fig. 1. The minimum and maximum choices of OR-features are shown as numbers surrounded by square brackets. 80% of the 32 variant features can be selected for custom TMS. However, there are also some feature interactions. For example, the selection of *InitPayMode* depends on the number of selected variant features under *FeeItemSelection* and the selection of *Settlement* depends on whether selected banks require settlement. TMS features include fine-, coarse, and mixed-grained features.

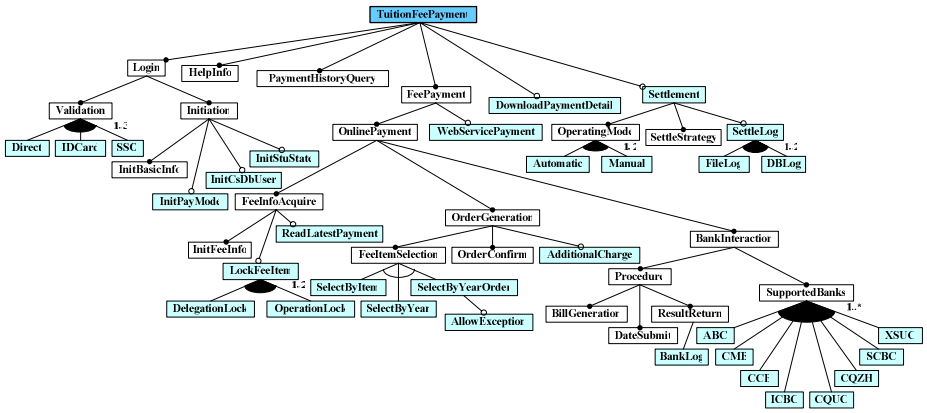

Fig. 1. The feature diagram of TMS

3 Variation Mechanism in TMS

3.1 Review of Variation Mechanism in TMS

Five variation mechanisms shown in Table 1 were used to manage variant features in TMS. In the table, column "# Features" indicates the number of features whose

customizations involved a given technique. Ant, conditional compilation and commenting out variant feature code were most commonly used.

Java conditional compilation and commenting out code: In Java, conditional compilation is realized with *final-boolean* variables. If a *final-boolean* variable's value is *false*, then the code in the statements under *if* is not compiled into the generated bytecode file. The effect is similar to *#define* and *#ifdef* C/C++ preprocessor directives [15]. Fig. 2 illustrates the usage of *final-boolean* variables to manage variant features in TMS class *FeatureConfiguration*. The limitation of Java's conditional compilation is that it can only be used in inner-method statements. It cannot handle the inclusion or exclusion of class methods or attributes. In WFMS, such cases were handled by manually commenting out the code that was not required in a given product variant. Commenting out was also used in non-Java files such as JSP files or SQL script. The main reason for such practice was that the engineers at Fudan Wingsoft could not find flexible tools to manage variability in these files at that time.

Table 1. Feature numbers for variability techniques used in TMP

# Techniques	# Features
Conditional compilation & comment	31
Ant	19
Overloading fields	13
configuration items	12
Design Pattern & reflection	3

```
1  public class FeatureConfiguration {
2      // Configuration items
3      public static final boolean DelegationLock = true;
4      public static final boolean OperationLock = true;
5  }
```

```
1  public class FeeInfo {
2      . . .
3      public void initInfo(FeeUser user, boolean isPaidFeeInfo)
           throws Exception {
4          //get each year's fee items
5          for( int i=0; i < yearTemp.size(); i++ ) {
6              If ( FeatureConfiguration.DelegationLock
                   && FeatureConfiguration.OperationLock )
7                  // Code when both features are selected
8              else if ( FeatureConfiguration.DelegationLock )
9                  // Code when delegationLock is selected
10             else if ( FeatureConfiguration.OperationLock )
11                 // Code when operationLock is selected
12         }
13     }
14 }
```

Fig. 2. Managing variant features with Java's final-boolean mechanism

Design patterns and reflection: [14] has described the use of the abstract factory pattern in SPL. It also extended this concept using the *dynamic abstract factory pattern*, in which concrete factories were adapted to support new concrete products at run-time by adding *Register* and *UnRegister* operations to Abstract Factory for each

```
1  public class FeeOrder {
2      private Initializer initializer;
3      public init(FeeUser user, FeeInfo info, HttpServletRequest request) {
4          Class c;
5          try{
6              c = Class.forName( user.getPayMode() );
7              initializer = (Initializer) c.newInstance();
8              initializer.init ( . . .);
9          } catch(Exception e ) {
10             e.printStackTrace();
11         }
12     }
13 }
```

Fig. 3. Reflection used in strategy pattern

abstract product. Although the most frequently used design patterns in TMS were *AbstractFactory* with *FactoryMethod* and *Strategy*, reflection mechanism, instead of operations returning name of product, was also used to dynamically instantiate proper concrete instances according to configuration options so that the specific class names could be abstracted from the source code. Then Ant could be used to control the inclusion and exclusion of a strategy subclass. Fig. 3 shows the use of *Strategy* Pattern in TMS.

Overloaded Fields: Variant features affect TMS database schema. Overloading table fields helps to contain some of those impacts. For example, a table may have several fields named *spec_1*, *spec_2* ... *spec_n*, and the same filed may be used to store bank card number in one product variant and ID card number in another one. There were also tables and fields that make sense for some product variants, but are useless for others. With overloading fields, all the products could share the same DB schema, but still support different data structures required for variant features. Overloading fields was adopted for WFMS-PL database.

```
1  <project name="webfee"  basedir="." default="main">
2     <target name="copy-src" depends="create-folders">
3        <!-- Copy java classes of Feature DownloadPaymentDetail -->
4        <copy todir="${src.dir}">
5           <fileset dir="${core-src.dir}/${DownloadPaymentDetail}"/>
6        </copy>
7     </target>
8     <target name="copy-webpage"
9        depends="create-folders">
10       <!-- Copy webpages of Feature DownloadPaymentDetail -->
11       <copy todir="${web-root.dir}">
12          <fileset dir="${core-webpage.dir}/${DownloadPaymentDetail}" />
13       </copy>
14    </target>
15 <project>
```

Fig. 4. Using Ant to include optional features

Ant: An important class of configuration parameters was managed by Ant. [3] used Ant and configuration files to differentiate variability in process and variability in product. Lower layers always override the *build.xml* file of upper layers, by which only the *build.xml* of the leaf layer will take effect. By reading layer orders in configuration files, the leaf layer can be determined. In WFMS, Ant was useful as a

```
1  <webFee>
2      <paymode>PayByItem</paymode>
3      <bank-info>
4        <bankList>
5          <bank>ICBC</bank>
6          <bank>CCB</bank>
7          <bank>CMB</bank>
8        </bankList>
9        <ICBC>
10         <bankUrl>http: //mybank.icbc.com.cn/servlet/co...</bankUrl>
11         <keyPath>C: /apache-tomcat-5.5.25/webapps/...</keyPath>
12         <keyPass>12345678</keyPass>
13         <merchantid>440220500001</merchantid>
14       </ICBC>
15     </bank-info>
16     <DownloadDetail>true</DownloadDetail>
17 </webFee>
```

Fig. 5. Using configurations files

variation mechanism for coarse-grained variant features as Ant can be used to control the inclusion/exclusion of not only java source files but also JSP files and security certificates. For instance, optional feature *DownloadPaymentDetail* of TMS was managed by Ant as shown in Fig. 4. This feature was implemented by a Java class and a JSP file. The inclusion of this feature in a customized product was implemented by moving the relevant files from the path of core asset to the path of *javac* command in the Ant configuration file.

Parameter Configuration Files: In TMS, self-defined configuration files were also employed as a variation mechanism, as shown in Fig. 5. Configuration files contained both data and control parameters. Data parameters, such as URLs of banking services and key path, were also widely used in single products. Control parameters feature to be selected for a custom product, while other variation mechanisms were used to perform the actual selection. For example, the parameter *paymode* in Fig. 5, indicating the right sub-class to be initialized, worked together with the reflection and the *strategy* pattern shown in Fig. 3 (see the underlined part). Another example of parameters working with other variation mechanisms is the parameter *DownloadDetail* in Fig. 5. A simple tool was implemented to read this parameter and generate Ant script shown in Fig. 4 if the value is *true*.

3.2 Summary of Variation Mechanism in TMS

Fig. 6 shows which variant features were managed using which variation mechanisms. We do not show overloading fields which was used for variants in database table schema only and did not overlap with other mechanisms. More than 80% features (26 among 32) were managed by more than one variation mechanism: 13 features were managed by three mechanisms and three features by four mechanisms. Design patterns were always used together with other mechanisms. Another interesting observation is that almost all features involved the use of conditional compilation and/or commenting out feature code, as in WFMS, like in many other SPLs, we saw many fine-grained features.

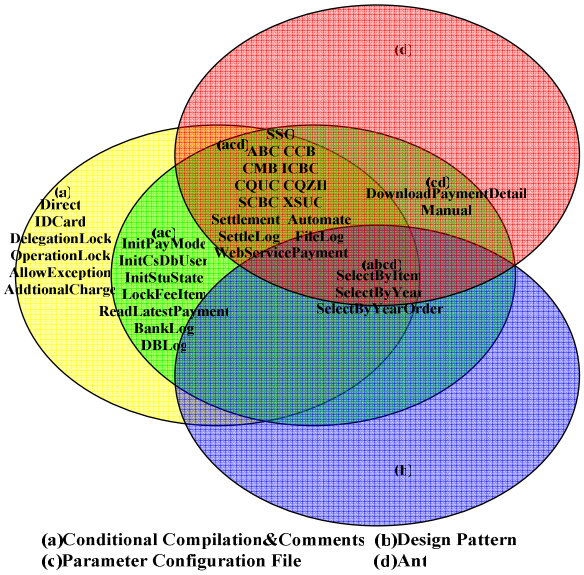

Fig. 6. Variation mechanisms per feature

For mixed-grained features, a combination of several variation mechanisms was usually used. For example, when we include a source files for a selected variant feature with Ant, we still need conditional compilation to configure corresponding caller in the base code. Table 2 summarizes the usage, scope, merits and drawbacks of various variation mechanisms used for WFMS-PL variant features. It is interesting to compare traditional variation mechanisms, such as described in this paper, and more advanced ones such as XVCL [6], AspectJ [13] or FOP [1]. It also will be the next step of our work.

4 Evaluation of the WFMS-PL and Possible Improvements

Projecting experiences from TMS study, we now evaluate WFMS-PL variability management strategies from the perspectives of feature granularity, ease of applying, readability and managing the consistency between WFMS core assets and custom products. On overall, we found that the current WFMS-PL strategies for variability management properly matched the feature granularity. The newcomers to the team could easily understand when and how to apply them. However, over time when the impact of many features was accumulating, the readability of the WFMS-PL has suffered, and it was becoming difficult to trace features to code and manage features consistently. The detailed reasons and suggested remedies are given in the following paragraphs.

Table 2. Summary of variation mechanism in WFMS-PL

Variation Mechanism	Usage	Scope	Merits	Drawbacks
Conditional compilation & Comments	Using *Final-Boolean* mechanism in Java and natural language comments	*Final-Boolean* method is only used on inner-method statements. Comments can be adapted to all places	Easy to learn and use	All maintain works and configurations are manual. Sub-paths explode as the number of tangled features increases.
Design Pattern	To gain good modularization in OO source code Used together with Java Reflection and Ant	Class or method level Best to gain class level flexibility in OOP part of a product line.	Providing Elegant code, High readability, Good Extendibility.	Scope of application is narrow and always need the aids of other techniques.
Overloading Fields	Making all the customized products share the same attribute in database	Database table schema	Avoiding troubles to change the name of attributes	Hard to maintain, easy to cause confusion about the meaning of the fields
Configuration File	Implementing the configuration of various parameters according to the variant feature selection and environmental change	Giving parameters of the variant feature selections and the environment	Good mechanism to do feature configuration	It needs to cooperate with other methods and introduces the non-traceability issue. Many inter-dependent configuration parameters
Ant	Conditionally compiling java source files and building deployments	System level customization Dealing with all kinds of files	Powerful and popular build tool, flexible to deliver product variants	Only file-level variants

Feature granularity: Feature granularity is a critical factor that guides selection of variation mechanisms as feature characteristics must be matched by the capabilities of a variation mechanism(s) used to manage a given feature. In WFMS-PL, fine-grained features were managed by conditional compilation in Java code, and with commenting out code section in other WFMS artifacts. Ant was used to manage coarse-grained features at the level of package or class inclusion/exclusion level. Design pattern played the role of a class or method extension mechanism. In Table 3, we show the number of WFMS variation points for each feature impact granularity level. Fine-grained impacts of features required small changes in Java expressions, statements, method signatures, comments (in Java code, JSP or HTML), database table scripts, and parameter configuration files. Medium-grained impacts required changes of Java methods or attributes, changes of database table scripts, and changes of configuration items in parameter configuration files. Coarse-grained impacts required inclusion or exclusion of product-specific source files.

Fine-grained features trigger most of the problems. Conditional compilation and commenting out feature code was used to manage fine-grained impacts. A big problem is how to trace variant features down to the many variation points relevant to them. This problem aggravates when multiple variation mechanisms are used to manage a given feature. Some feature enhancements involve changes at many variation points that must be properly coordinated. WFMS engineers often encountered the problems of inconsistent product release, e.g., a product variant was deployed with incorrect database schema.

Fine-grained and coarse-grained features were most common. We try to analyze the reasons as follows: Coarse-grained impacts are easy to configure. Whenever possible, domain engineers tried to contain the variant feature code in separate files which could then be included into custom products that required those features. Wizards could be implemented to allow application/service engineers to easily include such features into

custom products. Fine-grained impacts are attributed to the variability of the business flow or logic of the application itself and of the programming languages. However, fine-grained features made it more difficult to consistently manage the overall product configuration. They often affected coarse-grained features and the new variation points had to be injected into the source code of coarse-grained features.

Table 3. The number of variation points per impact granularity level

Granularity	#Java	#JSP	#Conf. File	#DB Schema	#Total
Finest	14	0	0	0	14
Fine	67	43	3	3	116
Medium	18	0	7	5	30
Coarse	40	57	9	0	106
#Total	139	100	19	8	266

Ease of application: Customizations of core assets by configuring parameters and database schemas could be managed by service engineers who were in charge of deployment of a customized product on the customer site. Service engineers were familiar with general financial domain, user requirements, and basic deployment operations, but did not know much about programming and internals of the WFMS-PL. Wizard-supported parameter configuration files provided an easy-to-use configuration capability for service engineers.

Ease of application without involvement of any unconventional or proprietary techniques was the most important reason for Fudan Wingsoft to adopt simple and commonly available variation mechanisms for WFMS-PL. This reduced the learning curve and the staff training cost, important factors for any small or middle-sized company. Unless current mechanisms were found totally ineffective, Wingsoft would be chary of adopting new ones. WFMS-PL was constructed in lightweight, reactive way, which was in line with the company's benefits so far.

Readability: Design patterns and Ant did not hinder readability, but conditional compilation, commenting our feature code and overloading fields made code difficult to understand for applications engineers and even domain engineers. In our project, 30% of code in class *FeeOrder,* 20% of code in *FeeInfo* and 35% of code in *FeeUser* was managed by Java conditional compilation. Given that there are no other techniques to manage fine-grained features, this problem is very hard to solve. If we keep the code of variant feature code embedded in the base code, the code is bound to become hard to read. One can consider Aspect-Oriented Programming (AOP) [13] or Feature-Oriented Programming (FOP) [1] to separate features from the base code, but these approaches pose new problems as demonstrated in [11] and [12]. To improve the readability of the code, a promising approach is to resort to visualization tool's support such as CIDE [12] or [8].

Traceability and extensibility: Traceability between features and their respective variation points has to do with both feature reuse and evolution. Here are some examples of problems:

- Each feature may be addressed at many variation points scattered through many SPL core components. To reuse or modify the feature we must find and analyze code at all these points.
- One SPL core component is usually affected by many features that may be managed by different possibly overlapping variation mechanisms. To reuse or modify the feature we must understand interactions among these mechanisms.

Variation mechanisms described in this paper provide a workable but not perfect solution for traceability problems. Table 4 shows features that involved several variation mechanisms, with their respective variation points spread across different WFMS-PL core components. How to manage these variation points consistently was the issue of traceability. The difficulty in traceability also brought in the problem of product extensibility at those variation points.

Table 4. The number of variation points in example features

Variant Feature	Preprocessing	Conf. Files	Ant	Total
WebService-Payment	6	2	2	10
ABC	2	1	3	6
CCB	1	1	2	4
CMB	2	1	2	5
ICBC	1	2	3	6

Yet another traceability problem that has to do with two-way propagation of changes between SPL core components and customized product variants [15]. This problem often hinders reuse, and we believe it is difficult to address it in the frame of variation mechanisms described in this paper. A meta-level representation of the SPL core components paves the way for more effective solutions to these problems [9]. They can capture and manage synchronously the overall impact of features on SPL core components [5].

5 Conclusion

In this paper, we evaluated strengths and weaknesses of variation mechanisms used in Wingsoft Financial Management System Product Line (WFMS-PL), developed by Fudan Wingsoft Ltd. Feature characteristics must be matched by the capabilities of variation mechanism(s) used to manage a given feature. Fundamental differences in capabilities of variation mechanisms justify the use of multiple variation mechanisms. Our study confirmed that different variation mechanisms have different, often complementary, strengths and weaknesses. Their choice should be mainly driven by the granularity and scope of feature impact on product line components. In some situations, we suggested possible remedies to weaknesses of variation mechanisms used in WFMS-PL. Our study revealed that while it was natural to match feature granularity with the proper variation mechanism, over time the inter-play between multiple variation mechanisms may become difficult to comprehend.

Variation mechanisms used in WFMS-PL are simple, practical, commonly used in SPLs to complement component/architecture-based approaches. We hope our report

will help companies to make more informed decisions when moving towards the product line approach. In follow-up study, we compare the original WFMS-PL with a representation built with XVCL [6] as a variation mechanism, and plan to extend our study to other SPL approaches.

Acknowledgement. This work was supported by Fudan University grant (the National Natural Science Foundation of China under Grant No. 60703092, the National High Technology Development 863 Program of China under Grant No. 2007AA01Z125, and Shanghai Leading Academic Discipline Project under Grant No. B114) and National University of Singapore grant R-252-000-336-112.

References

1. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering 30(6) (2004)
2. Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, H., Pohl, K.: Variability Issues in Software Product Lines. In: van der Linden, F.J. (ed.) PFE 2002. LNCS, vol. 2290, p. 13. Springer, Heidelberg (2002)
3. Díaz, Ó., Trujillo, S., Anfurrutia, F.I.: Supporting Production Strategies as Refinements of the Production Process. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 210–221. Springer, Heidelberg (2005)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley Professional, Reading (1995)
5. Jarzabek, S.: Effective Software Maintenance and Evolution: Reuse-based Approach. CRC Press Taylor & Francis, Boca Raton (2007)
6. Jarzabek, S., Bassett, P., Zhang, H., Zhang, W.: XVCL: XML-based variant configuration language. In: ICSE 2003 (2003)
7. Jarzabek, S., Ong, W.C., Zhang, H.: Handling Variant Requirements in Domain Modeling. Journal of Systems and Software 68(3) (2003)
8. Jarzabek, S., Zhang, H., Lee, Y.P., Xue, Y., Shaikh, N.: Increasing Usability of Preprocessing for Feature Management in Product Lines with Queries. Accepted for ICSE 2009 poster
9. Jirapanthong, W., Zisman, A.: XTraQue: Traceability for Product Line Systems. Software and System Modeling 8(1) (2009)
10. Kang, K.C., Lee, J., Donohoe, P.: Feature-oriented Product Line Engineering. IEEE Software 19(4) (2002)
11. Kästner, C., Apel, S., Batory, D.: A Case Study Implementing Features Using AspectJ. In: SPLC 2007 (2007)
12. Kästner, C., Apel, S., Kuhlemann, M.: Granularity in Software Product Lines. In: ICSE 2008 (2008)
13. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
14. Linden, F., Schmid, K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer, Heidelberg (2007)
15. Pohl, K., Böckle, G., Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
16. Spencer, H., Collyer, G.: #ifdef Considered Harmful, or Portability Experience with C News. In: Summer 1992 USENIX Conference (1992)
17. Svahnberg, M., Gurp, J., Bosch, J.: A Taxonomy of Variability Realization Techniques. Software: Practice and Experience 35(8) (2005)