A Case for a Uniform Mechanism for Variability Management

Yinxing Xue¹, Stan Jarzabek¹, Pengfei Ye², Xin Peng², Wenyun Zhao²

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

Abstract. Having set up reusable core assets for a software Product Line (SPL), it is a common practice to apply variation mechanisms to manage variant features. As each variation mechanism can handle only certain type of variability, multiple variation mechanisms are often employed, such as conditional compilation, configuration parameters or build tools. Our earlier study of an SPL at Fudan Wingsoft Ltd revealed potential scalability problems of multiple variation mechanisms. As a product line grows, it becomes increasingly difficult to understand features that affect many core assets, at many variation points, and to coordinate changes required at multiple variation points. As a remedy to the above problems, in the follow-up study we replaced variation mechanisms originally used in the Fudan Wingsoft product line, with a single, uniform variation mechanism of XVCL. This paper provides a proof-of-concept that commonly used variation mechanisms can indeed be superseded by a subset of XVCL, in a simple and natural way. We describe the essence of XVCL solution, and evaluate the benefits and trade-offs involved in replacing multiple variation mechanisms with uniform one.

1. Introduction

In previous paper [19], we analyzed a software Product Line (SPL) called Wingsoft Financial Management Systems (WFMS-PL), developed by Fudan Wingsoft Ltd. WFMSes provide web-based financial services for employees and students at universities in China. Following a common practice, Wingsoft set up product architecture, identified core assets for reuse, and then applied a range of common design-time variation mechanisms, such as conditional compilation, commenting out feature code or configuration parameters, to manage product-specific features in core assets. Variation mechanisms mark variation points in core assets to help developers perform customizations, manually or sometimes in automated way.

Features vary in the granularity and in the scope of their impact on core assets: Fine-grained features affect many core assets of an SPL, at many variation points [7]. Code of such features becomes scattered across core assets. A coarse-grained feature, on the other hand, can be contained in a component (e.g., source file) that is included into a custom product when needed. Mixed-grained features involve both fine- and coarse-grained impact.

Coarse-grained features are easier to manage than fine-grained features. Feature granularity depends to some extent on the design of core assets. Good architectural design can change feature
granularity to our favor, increasing the number of coarse-grained features, and reducing the
number of variation points in core assets for the features that remain fine-grained. Variation
mechanisms are needed to manage features that still remain fine-grained or mixed-grained.

Variation mechanism must match feature granularity. Therefore, it is common to use multiple
variation mechanisms, for example, conditional compilation to handle fine-grained features and a
build tool such as Ant to handle coarse-grained features. Such variation mechanisms are easy to
apply, and most of developers are familiar with them. However, as our study revealed [19],
applying multiple variation mechanisms did not scale well, especially in cases of fine-grained
features. While reuse and modification of fine-grained features is inherently difficult, applying
multiple, often poorly compatible variation mechanisms aggravate the problems. In particular, it
becomes increasingly difficult to find and understand already scattered feature code, and to
coordinate changes required at multiple variation points.

As a remedy to the above problems, in the follow-up study we replaced variation mechanisms
originally used in the Fudan Wingsoft product line, with a single, uniform variation mechanism of
XVCL [18]. This paper serves as a proof-of-concept conforming that commonly used variation
mechanisms can indeed be superseded by a subset of XVCL, in a simple and natural way. We also
present an initial evaluation of benefits and trade-offs involved in adopting a uniform variation
mechanism. A practical lesson learned from our study is that in small- to medium-size product
lines, applying multiple variation mechanisms may be a viable solution, as it requires less training,
and variability can be still effectively managed in that way. As the product line grows in size and
the impact of features on core assets becomes more complex, a company may experience
problems, and then moving towards uniform variation mechanism approach may be beneficial.
However, this will require a more systematic approach to reuse, and training of SPL personnel.

The paper is organized as follows: Section 2 summarizes the findings from our earlier study of
multiple variation mechanisms in WFMS-PL [19]. Section 3 describes a single variation
mechanism approach to WFMS-PL, and explains how it alleviates problems of multiple variation
mechanisms. We evaluate the two approaches to variability management in Section 4. Related
work and concluding remarks end the paper.

2. Problems of multiple variation mechanisms

2.1 Overview of the original WFMS-PL

WFMS for Fudan University developed in 2003 evolved into a software Product Line WFMS-
PL [19] with more than 100 customers at major universities in China such as Shanghai Jiaotong
University (can be found at http://www.jdcw.sjtu.edu.cn/wingsoft/index.jsp), Zhejiang University, Harbin
Institute of Technology, Sichuan University, and Chongqing University. First five WFMS product
variants were developed by ad hoc copy-and-paste reuse, and each WFMS was maintained as a
separate product. Despite much similarity, these initial WFMSes were not managed from a
common set of core assets. As the number of customers was growing, ad hoc reuse and
maintenance was becoming more and more taxing on company resources. To address this
problem, Wingsoft set up product architecture and adopted commonly available variation
mechanisms, such as conditional compilation and configuration parameter files, to manage
product variability in core assets. Wingsoft did not use any advanced SPL techniques due to the
practical need for an easy-to-implement, cost-effective SPL, which could be built and reused with minimum training of Wingsoft staff in SPL techniques.

Figure 1. Architecture of WFMS

The architecture of WFMS is shown in Figure 1. There are four subsystems, namely Financial Management Subsystem (FMS) that manages all the university income and expenses, Salary Management Subsystem (SMS) that manages salary of employees, Reward Management Subsystem (RMS) that manages rewards for employees and students, and Tuition Management Subsystem (TMS) that manages student tuition fees.

We selected the Tuition Management Subsystem (TMS) for our study, as it involved types of variability and variation mechanisms that were representative for the whole WFMS. The code of TMS is 25% of the whole WFMS system, comprising 58 Java source files, 99 JSP web pages, and several configuration files.

As illustrated in Figure 2, TMS is a web-based portal for students to pay their tuition fee online, with functions such as login, fee browsing, on-line payment and history query. In addition, TMS also provides accounting services (e.g., report generation and bill settlements) that interface universities with banking systems. TMS adopts traditional 3-tier architecture, namely user interface, business logic and database access tier.

Figure 2. The architecture of TMS

Feature impact on TMS’s core assets and feature interactions. TMS components are derived from corresponding TMS core assets which have been instrumented with variation mechanisms to accommodate variant features required in custom products. In TMS feature model, there are 32 variant features and 9 mandatory features. Table 1 shows the impact of TMS features on core assets: #VP – the number of variation points at which a feature affects core assets, #Size – LOC of feature implementation, #AF – the number of affected core assets. We see that most TMS features have fine-grained impact on core assets.

Feature dependencies constrain legal combinations of features that can be implemented in any custom product. For example, selection of one feature may entail selection of yet other feature. In TMS, feature ABC represents a bank ABC and feature Settlement means that a bank needs settlement for each payment. As bank ABC requires settlement for each payment therefore
whenever we select feature ABC, we must also select Settlement. Inter-dependent features tend to interact with core assets at the same variation points, and may also affect each other code.

### Table 1. Variant features of TMS

<table>
<thead>
<tr>
<th>TMS Feature Name</th>
<th>#VP</th>
<th>#Size</th>
<th>#AF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>The direct login mode</td>
</tr>
<tr>
<td>IDCard</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td>The login mode with ID number</td>
</tr>
<tr>
<td>SSO</td>
<td>5</td>
<td>17</td>
<td>3</td>
<td>The single sign-on mode</td>
</tr>
<tr>
<td>InitPayMode</td>
<td>9</td>
<td>16</td>
<td>4</td>
<td>Initiation of the payment mode</td>
</tr>
<tr>
<td>InitCsdbUser</td>
<td>10</td>
<td>18</td>
<td>5</td>
<td>Initiation of user account of finance database</td>
</tr>
<tr>
<td>InitStuState</td>
<td>10</td>
<td>37</td>
<td>3</td>
<td>Initiation of the student’s state</td>
</tr>
<tr>
<td>WebServicePayment</td>
<td>18</td>
<td>2144</td>
<td>17</td>
<td>Providing web service of payment</td>
</tr>
<tr>
<td>LockFeeItem</td>
<td>16</td>
<td>108</td>
<td>5</td>
<td>Locking fee items from being paid</td>
</tr>
<tr>
<td>DelegationLock</td>
<td>8</td>
<td>10</td>
<td>2</td>
<td>Locking fee items whose payments are delegated to bank</td>
</tr>
<tr>
<td>OperationLock</td>
<td>3</td>
<td>39</td>
<td>2</td>
<td>Locking fee items that are waiting for payment results</td>
</tr>
<tr>
<td>ReadLatestPayment</td>
<td>10</td>
<td>90</td>
<td>6</td>
<td>Showing the result of last payment</td>
</tr>
<tr>
<td>SelectByItem</td>
<td>6</td>
<td>288</td>
<td>6</td>
<td>Selecting fee items by each item</td>
</tr>
<tr>
<td>SelectByYear</td>
<td>6</td>
<td>176</td>
<td>6</td>
<td>Selecting fee items by the year of item</td>
</tr>
<tr>
<td>SelectByYearOrder</td>
<td>6</td>
<td>603</td>
<td>6</td>
<td>Selecting fee items by the year order</td>
</tr>
<tr>
<td>AllowException</td>
<td>6</td>
<td>69</td>
<td>3</td>
<td>Allowing exceptions in SelectByYearOrder mode</td>
</tr>
<tr>
<td>AdditionalCharge</td>
<td>7</td>
<td>15</td>
<td>3</td>
<td>Storing additional charge</td>
</tr>
<tr>
<td>ABC</td>
<td>18</td>
<td>1451</td>
<td>18</td>
<td>The Agriculture Bank of China</td>
</tr>
<tr>
<td>CCB</td>
<td>17</td>
<td>1043</td>
<td>17</td>
<td>The Construction Bank of China</td>
</tr>
<tr>
<td>CMB</td>
<td>11</td>
<td>784</td>
<td>11</td>
<td>The Commercial Bank of China</td>
</tr>
<tr>
<td>ICBC</td>
<td>24</td>
<td>1339</td>
<td>24</td>
<td>The Industry and Commercial Bank of China</td>
</tr>
<tr>
<td>CQUC</td>
<td>6</td>
<td>783</td>
<td>6</td>
<td>Office of Chongqing University in charge of fee payment</td>
</tr>
<tr>
<td>CQZH</td>
<td>8</td>
<td>1038</td>
<td>8</td>
<td>Chongqing Sub-Branch of the Bank of China</td>
</tr>
<tr>
<td>SCBC</td>
<td>7</td>
<td>546</td>
<td>7</td>
<td>Sichuan Sub-Branch of the Bank of China</td>
</tr>
<tr>
<td>XSUC</td>
<td>6</td>
<td>844</td>
<td>6</td>
<td>Office of Southwest Normal University in charge of fee payment</td>
</tr>
<tr>
<td>BankLog</td>
<td>13</td>
<td>111</td>
<td>11</td>
<td>Log interaction information with the banks</td>
</tr>
<tr>
<td>DownloadDetail</td>
<td>3</td>
<td>136</td>
<td>3</td>
<td>Downloading payment details</td>
</tr>
<tr>
<td>Settlement</td>
<td>13</td>
<td>487</td>
<td>13</td>
<td>Bank settlement</td>
</tr>
<tr>
<td>Automatic</td>
<td>4</td>
<td>216</td>
<td>3</td>
<td>Automatic operation mode</td>
</tr>
<tr>
<td>Manual</td>
<td>1</td>
<td>48</td>
<td>1</td>
<td>Manual operation mode</td>
</tr>
<tr>
<td>SettleLog</td>
<td>9</td>
<td>153</td>
<td>4</td>
<td>Log settlement info.</td>
</tr>
<tr>
<td>FileLog</td>
<td>7</td>
<td>103</td>
<td>6</td>
<td>Log information in files</td>
</tr>
<tr>
<td>DBLog</td>
<td>6</td>
<td>79</td>
<td>4</td>
<td>Log information in database</td>
</tr>
</tbody>
</table>

Table 2 shows that 18 out of 32 TMS variant features interact with other features. Feature dependencies and interactions further complicate the way feature code is tangled with core assets at variation points. For example, if we use conditional compilation to manage the impact of feature interactions on core assets, then relevant variation points will include many options to choose from during customization, with each option catering for a specific type of feature interaction. This complicates analysis, maintenance and reuse of both core assets and features.

### Table 2. Feature interactions

<table>
<thead>
<tr>
<th>Interacting features</th>
<th>Dependency rule</th>
<th>Interaction in source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct, IDCard</td>
<td>N/A</td>
<td>1 vp</td>
</tr>
<tr>
<td>InitPayMode, SelectByItem, SelectByYear, SelectByYearOrder,</td>
<td>Number of the selected features among SelectByItem, SelectByYear and SelectByYearOrder decides the selection of InitPayMode. If only one of the three features is selected, InitPayMode should not be selected.</td>
<td>N/A</td>
</tr>
<tr>
<td>WebServicePayment, InitCsdbUser</td>
<td>N/A</td>
<td>1 vp</td>
</tr>
<tr>
<td>WebServicePayment, SelectByItem</td>
<td>N/A</td>
<td>1 vp</td>
</tr>
<tr>
<td>ReadLatestPayment, OperationLock</td>
<td>N/A</td>
<td>1 vp</td>
</tr>
<tr>
<td>ABC, ICBC, AdditionalCharge</td>
<td>If ABC or ICBC is selected, AdditionalCharge should be selected.</td>
<td>N/A</td>
</tr>
<tr>
<td>ABC, CMB, CQZH, Settlement</td>
<td>If ABC or CMB or CQZH is selected, Settlement should be selected.</td>
<td>N/A</td>
</tr>
<tr>
<td>BankLog, DBLog</td>
<td>N/A</td>
<td>1 vp</td>
</tr>
</tbody>
</table>
2.2 Variation mechanisms applied in TMS-PL

Wingsoft adopted simple, freely available variation mechanisms for TMS-PL [19]. Different variation mechanisms have different, often complementary, strengths and weaknesses, and their choice is mainly driven by the granularity and scope of feature impact on core assets. For example, conditional compilation is suitable for fine-grained features and a build tool such as Ant - for coarse-grained features. If capabilities of a variation mechanism match the nature of feature impact, then applying a variation mechanism to handle a feature is easy. At Wingsoft, experienced domain engineers were selecting the right variation mechanism for features at hand.

Our study [19] also revealed potential problems of multiple variation mechanisms: One feature was often handled by a number of variation mechanisms which had to be properly synchronized. 26 among 32 variant features were managed by more than one variation mechanism, 13 features - by three mechanisms, and three features - by four mechanisms. Examples below illustrate what was involved in managing such features, and hint at complications that are bound to arise as the size of the system and the number of inter-dependent variant features grow.

Feature PayByItem (Figure 3 and Figure 4) is a mixed-grained feature managed by configuration parameters that parameterize reflection and strategy design patterns. In Figure 4, method `user.getPayMode()` in the 6th line returns the value of a configuration parameter that indicates the payment mode. According to that value, the design pattern generates the specific code skeleton for feature PayMode.

```
<webFee>
  <paymode>PayByItem</paymode>
  <bank-info>
    <bank-list>
      <bank>ICBC</bank>
      <bank>CCB</bank>
      <bank>CMB</bank>
    </bank-list>
    <ICBC>
      <bank-url>http://mybank.icbc.com.cn...</bank-url>
      <key-path>C:/apache-tomcat-5.5.25/...</key-path>
      <key-pass>12345678</key-pass>
      <merchant-id>440220500001</merchant-id>
    </ICBC>
  </bank-info>
  <download-detail>true</download-detail>
</webFee>
```

Figure 3. Managing PayByItem with configurations parameters
Coarse-grained impact of feature PayByItem was managed by build tool Ant (Figure 5). Using parameters, Ant could include/exclude source files that were relevant/irrelevant to a given feature. Ant script set the target path for the source code to be included in the final build. Feature PayByItem has a related JSP file selFeeItem.jsp, which was included by the command `<fileset dir>` in the 12th line.

```
<project name="webfee" basedir="." default="main">
  <target name="copy-src" depends="create-folders">
    <!-- Copy java classes of Feature PayByItem -->
    <copy todir="${src.dir}"/>
  </copy>
  <target name="copy-webpage" depends="create-folders">
    <!-- Copy webpages of Feature PayByItem -->
    <copy todir="${web-root.dir}/">
      <fileset dir="${core-webpage.dir}/${PayByItem}"/>
    </copy>
  </target>
</project>
```

Fine-grained features were managed by Java conditional compilation and commenting out feature code in core assets. Java conditional compilation is easier to maintain than commenting out code, but its application is limited to Java source code. Wingsoft developers used commenting out technique for non-Java core assets such as DB scripts or JSP scripts. Both variation mechanisms were indispensable for TMS core assets.

Fine-grained features are the main source of problems for scalability of the above approach to managing SPL variability. Scattered impact of fine-grained features brings forth the difficulties to keep multiple variation mechanisms in sync one with another. Feature PayByItem in Figure 3 illustrates the problem. Inter-related configuration parameters control both Ant and Java conditional compilation. If the payment mode is switched from PayByItem to PayByYear, then the Ant script must be changed accordingly, and variation points controlled by Java conditional compilation the commented out code in DB scripts and JSP scripts have to be modified accordingly. There are many examples of such interactions between variation mechanisms in the original TMS core assets. Its maintenance and evolution entails the accurate understanding of multiple variation mechanisms, and familiarity with variant features and core assets.

As the size of the system grows, the above inconveniences aggravate. These observations encouraged us to experiment with a strategy that employs a single variation mechanism with capabilities to manage all the variability situations found in TMS core assets in a uniform and tractable way.

3. Single variation mechanism approach to TMS core assets

XVCL [18][8], based on Frame Technology [1], is a language-independent variation mechanism for SPLs.

---

**Figure 4. Managing PayByItem with reflection and strategy patterns**

**Figure 5. Managing PayByItem with Ant**
3.1 Variation mechanism of XVCL

XVCL encapsulates core assets in meta-components called *x-frames*. Each variation point in core assets is marked with a suitable XVCL command, such as such as <adapt>, <insert-before>, <insert>, <insert-after> and <break>, to enable customizations. SPL variant features are formally mapped into all the relevant variation points in core assets by means of XVCL parameters and commands. The SpCification x-frame, called SPC, sets values of XVCL parameters according to feature selection. XVCL Processor interprets x-frames starting from the SPC (Figure 6), traverses x-frames, propagates customization information (parameters) to them, adapting visited x-frames accordingly, and emitting code for a custom product. XVCL mechanisms allow us to manage features with fine-, coarse- and mixed-grained impact on core assets. Due to its language-independence, any type of SPL core assets including Java code, JSP files, DB scripts, textual documents, test cases or even UML models in XMI can be consistently customized for any legal selection of features required in a custom product.

3.2 TMS core assets instrumented with XVCL

Figure 6 provides a snapshot of the WFMS core assets in XVCL representation, and Figure 7 expands some x-frames to highlight the workings of XVCL. SPC specifies which features we need in a custom WFMS product by setting values for XVCL parameters that correspond to selected features. Values of those parameters propagate to x-frames below, navigating configuration and detailed customizations of core assets and features accordingly. Level 2 x-frames define architecture-level customizations, in terms of configuration of core assets for a custom WFMS product. Some of the coarse-grained feature impacts are also addressed at Level 2. Level 3 x-frames contain the actual code of core assets and features, instrumented with XVCL commands to enable customization of fine-grained features.

Features that we want to select for a custom product are assigned non-empty string values, while features to be de-selected are assigned empty string values. Therefore, SPC shown in Figure 7 selects features IDCard and SSO (related to Login), and feature PayByItem (related to Paymode) for a custom product.

<select> commands mark variation points in x-frames below SPC. The value of an XVCL parameter that controls <select> identifies an <option> to be processed. <select PayByItem> in x-frame OnlinePayment at Level 2 illustrates a simple variation point affected by one feature only, namely PayByItem. If feature PayByItem is selected, then the Processor emits feature code to the custom product; otherwise, <select> has no effect.
between features and WFMS core assets in XVCL.

selected features, at all the relevant variation points. XVCL parameters set in SPC create a bridge

IDCard

features. Note that "IDCard+SSO" is processed whenever the two interacting features

option

to v’s value, as assigned in respective <set> command. The value of Login, <set> to be a
directing processing to the <select>, corresponding to the particular combination of selected
features. Note that <option “IDCard+SSO”> is processed whenever the two interacting features
IDCard and SSO are selected. Symbol ‘+’ is a separator.

XVCL parameters formally link together customizations of all the core assets affected by selected features, at all the relevant variation points. XVCL parameters set in SPC create a bridge between features and WFMS core assets in XVCL.

Figure 7. Detailed view of WFMS core assets in XVCL

<select Login> in x-frame FeeUser at Level 3 marks a variation point affected by three features, namely IDCard, SSO and Direct. Notation @v, where v is an XVCL parameter, means reference to v’s value, as assigned in respective <set> command. The value of Login, <set> to be a concatenation of the three XVCL parameters corresponding to these features, controls <select>, directing processing to the <option> corresponding to the particular combination of selected features. Note that <option “IDCard+SSO”> is processed whenever the two interacting features IDCard and SSO are selected. Symbol ‘+’ is a separator.

XVCL parameters formally link together customizations of all the core assets affected by selected features, at all the relevant variation points. XVCL parameters set in SPC create a bridge between features and WFMS core assets in XVCL.
3.3 One variation mechanism instead of many

Our example of Figure 7 also illustrates how a single variation mechanism can successfully provide capabilities of many variation mechanisms. In x-frame FeeOrder, \texttt{@PayMode c = new @PayMode()} replaces configuration files and Strategy pattern in the original WFMS core assets. Here, we need a parameterized name of the class. Java generics support parametric types, but not class names. In the original WFMS core assets, Strategy pattern and configuration parameter stored in a configuration file were used to mitigate the problem (Figure 4). Strategy pattern reads the name of a required class from the configuration file.

In the original WFMS core assets, architecture-level configurations of core assets and coarse-grained features were done by Ant. For example, if we select feature PayByItem, Ant’s command \texttt{<fileset dir >} in Figure 5 includes file selfFeeItem.jsp into the custom product. The same is achieved by \texttt{<adapt>} placed under \texttt{<select>} in x-frame OnlinePayment. Of course, Ant has more capabilities than XVCL’s \texttt{<adapt>}, but in this context only Ant’s asset configuration capabilities are used.

In x-frames DBSchema and FeeUser, we see how XVCL’s parameters and \texttt{<select>} replace conditional compilation and commenting out feature code. For example, feature InitPayMode affects DBSchema and FeeUser and is managed by conditional compilation and commenting out technique. Manual modification of the conditional compilation or comments has to be done to include/exclude features. In XVCL on the other hand, variation points are inter-lined and customizations are automated.

3.4 Feature queries

Variability management with single variation mechanism has also this advantage that tools can be implemented to help developers analyze, reuse and maintain features and core assets. To reuse or maintain features, developers must be able to locate and analyze all the variation points at which features affect core assets and each other. Variation points for fine-grained features spread through many core assets. Using a single, uniform variation mechanism such as XVCL allowed us to implement query tools that can help in feature code location and analysis.

Developers specify features of interest in FQL (Feature Query Language) [9]. The tool evaluates queries and displays the results. In FQL, we can ask queries such as “which base components are affected by feature \( f \) and at which variation points?”, “which features interact with feature \( f \)?”, “in which base components and at which variation points feature \( f_i \) interacts with feature \( f_j \)?”. FQL is an SQL-like notation. We write queries in terms of XVCL elements.

Figure 8 shows a query to locate all the variation points at which feature InitPayMode affects WFMS-PL core assets.

```sql
declare x-frame x; option o;
select x, o
where o.f-names = “InitPayMode”
and Contains (x,o)
```

Figure 8. Finding code of feature InitPayMode

Figure 9 shows a query that finds all the variation points at which features IDCard and SSO interact one with another.
We refer the reader to [9] for details of query-based feature analysis.

4. Evaluation

What is the impact of replacing multiple variation mechanisms with a single one on SPL productivity? To answer this question we conducted lab studies and collected inputs from Fudan Wingsoft Ltd. regarding the original WFMS core assets developed by Wingsoft using multiple variation mechanisms, and core assets in XVCL. We also conducted a short course in which we explained the original WFMS-PL by Wingsoft, and the one in XVCL. The course was attended by representatives from 40 companies involved in SPL projects. After the course, we conducted a survey and their feedback is included in our evaluation. Below, we comment on productivity during domain engineering (i.e., building and evolving core assets), and product derivation.

4.1 Domain engineering effort

The original WFMS core assets were built by gradual re-engineering of existing WFMSes. Core components and their interfaces were stabilized first, and then variation mechanisms were used to prepare them for ease of customization, as described in Section 2. While it is difficult to precisely determine the effort to build core assets, we obtained some relevant information from Wingsoft engineers who were involved in re-engineering. Selecting suitable variation mechanisms for various features was not difficult for experienced engineers. Also, each step of applying variation mechanisms was quite simple. New staff joining the Wingsoft team had no difficulty to understand the variation mechanisms used in WFMS core assets and their role. Some problems could be observed during evolution of the WFMS core assets. When multiple variation mechanisms were used together to accomplish a variability task, it might not be clear how to find all the relevant variation points, and understand the exact interplay between variation mechanisms. Still, given the size of WFMS core assets and relatively small number of features, the solutions adopted by Wingsoft team were considered to be adequate for the purpose.

To get insights into the effort of replacing multiple variation mechanisms with XVCL, two students re-engineered the original WFMS core assets into XVCL representation. One NUS student was an XVCL expert, and the other student from Fudan University was a WFMS expert, also participating in maintenance of the original WFMS core assets. It took two weeks for them to replace multiple variation mechanisms with XVCL in core assets for TMS subsystem. Applying XVCL was greatly simplified, as core assets were already in place, and students preserved most of the variation points. The main task was to work out overall XVCL controls and then to replace multiple variation mechanisms with XVCL commands at respective variation points.

Evolution of core assets involves adding new features, and modifying features. The effort to evolve core assets depends on the number of variation points involved in change, and the complexity of finding, analyzing, changing variation points and tracing the impact of change.
While the number of variation points in both solutions is almost the same, evolution of XVCL solution is easier than evolution of the original solution. This is due to uniform treatment of features, formal links between all the variation points relevant to a given feature, and feature query system.

4.2 Product derivation and maintenance effort

Deriving new products includes reuse of existing features, modifying features, and implementing extra features required by customers. As before, the effort of each such task depends on the number of variation points involved in change, and the complexity of finding, analyzing, changing variation points and tracing the impact of change.

Table 3 summarizes statistics relevant to product derivation effort. “Managed variation points” means variation points that have to be revised manually when reusing or modifying features. “Managed variation points” is a subset of all the variation points at which feature affects core assets. For example, among core assets affected by feature \textit{InitPayMode} are Java files and DB schema files. To reuse this feature in the original WFMS-PL, all affected files need be manually changed. In the XVCL solution, once we \texttt{set} value of XVCL parameter \textit{InitPayMode} in \texttt{SPC} (Figure 7), all the customizations for feature \textit{InitPayMode} spark from there, can be found by feature queries, and automatically performed by the XVCL Processor. Feature \textit{InitPayMode} requires one managed variation point.

As another example, core assets affected by feature \textit{Settlement} include seven Java files, four JSP scripts, one configuration file, and one file containing DB schema. To reuse feature \textit{Settlement} in the original WFMS-PL, we must change code at eight variation points handled by conditional compilation, comments and Ant. We have 13 variation points, and eight managed variation points. The location of managed variation points as well as relationship among them is not formally captured, therefore must be communicated via external documentation or re-discovered when needed. In XVCL solution, for the same feature there are also 13 variation points, but only three managed variation points (XVCL parameters for \textit{Settlement} and for two dependent features). All the variation points are inter-linked via relevant XVCL parameters \texttt{set} in \texttt{SPC}, and reuse of the feature is automated by the XVCL Processor.

<table>
<thead>
<tr>
<th>Table 3. Managed variation points</th>
</tr>
</thead>
<tbody>
<tr>
<td>#variation points</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>Original WFMS core assets</td>
</tr>
<tr>
<td>XVCL WFMS core assets</td>
</tr>
</tbody>
</table>

4.3 Other inputs from Wingsoft and course participants

Comments on code readability. Both representations applied variation mechanisms to embed fine-grained feature code in the code of core assets at relevant variation points. In some core assets, 50% of code was feature code. Wingsoft engineers were concerned about readability and maintainability of the code, but they were also pleased with XVCL’s ability to mark traces of customizations relevant to a given feature and ease of finding all the variation points relevant to a given feature.

Comments on copyright protection. In the original WFMS core assets, feature code is embedded in core assets, but only some features are needed in a custom product. Wingsoft
engineers often included unnecessary feature code into a custom product because of time involved in feature removal, and also because such code might be useful in future enhancements of a custom product. When extra functionality is contained in files that are released in readable form (e.g., JSP or XML configuration files), this practice can sometimes create copyright problems: The extra functionality may be used in the ways not intended by a company that developed code, or copied to enhance products of other customers without proper agreement. Such cases happened in Fudan Wingsoft experience.

In XVCL, unwanted features are never included into a custom product, as the job of feature manipulation is automatically done by the XVCL Processor, therefore does not affect productivity of engineers. Other than protecting copyrights, such precise and flexible control over feature inclusion/exclusion to/from custom products also matters in situations when we need to build highly optimized products, for example embedded software.

4.4 Evaluation summary

Overall, it was felt that for small-to-medium systems such as WFMS (around 50KLOC), adopting multiple variation mechanisms is still practical. Variation mechanisms used in the original WFMS are simple and known to most of engineers. They came into engineers’ mind naturally, could be applied on the fly during core asset design, with minimum disruption of conventional programming. Multiple mechanisms provide an elementary infrastructure for SP support. Handled by the experienced engineers, the original WFMS core assets serves well for the derivation of almost 100 product variants.

As the size of core assets and the number of variant features grows, and feature interactions get more complicated, problems may show up. Feature reuse and maintenance may become more complex because of the many variation points at which feature code needs be understood. Manual customizations become time-consuming and error-prone, even for skilled domain engineers. Then, it may be worth to consider migrating to a uniform variation mechanism such as XVCL.

In XVCL, for a feature reused as-is we need small number of managed variation points, at which we \(<\text{set}>\) XVCL parameters for that feature and its dependent features (in SPC). All the variation points for a given feature are formally linked to XVCL parameter representing that feature. The ability to locate and analyze traces of customizations for each feature helps developers reuse and modify features with less errors and unwanted side-effects as compared to working with the original WFMS core assets. Reuse is automated by the XVCL Processor.

However, adoption XVCL is not without pitfalls, some of which XVCL shares with other variation mechanisms. Much of the feature code still remains tangled with core assets, affecting readability. This is a big problem, but so far alternative approaches based on specification-based variation points such as AOP [13] or FOP [2] failed to provide an effective solution to fine-grained feature management in SPLs [11][12] (we comment further in Related Work section). XVCL’s feature queries can help developers identify variation points relevant to various features, but the actual feature modifications are not easy if the number of variation points is large. Training must be provided for the team to learn a new technique. The correctness of transformations from XVCL representation to code can be checked only during compilation.
5. Related WORK

Managing variability is the essence of software Product Line (SPL) practice [3]. Variation mechanisms are one of the enablers of reuse-based derivation of products from reusable core assets. Productivity gains due to reuse to much extent depend on the effectiveness of product derivation. The importance of having adequate variation mechanism fitting specific needs of given SPL is stressed by SPL proponents and practitioners [4][1][10][17]. Deelstra et al. [5] identify the weakness of variation mechanisms as one the obstacles that impedes implementing reuse strategies via the SPL approach in some industries.

Karhinen et al. [10] analyzed problems of managing variability solely at the level implementation level, for example, using conditional compilation or configuration management tools. Authors’ experiences from Nokia projects indicate that managing features with #ifdefs while technically feasible, is inherently complex, error-prone and does not scale. They proposed to use design means to manage variability. Similar problems with conditional compilation were also reported in FAME-DBMS [15].

Architectural/component approach to SPL applies design means to manage variability, in the attempt to modularize features as far as this is possible. Still, in most of application domains many features remain fine-grained, with their impact scattered through core assets [12][19][10]. Such feature must be managed with additional variation mechanisms such as described in Section 2 and Section 3 of this paper.

Research community proposed Feature-Oriented Programming (FOP) [14] as an approach to feature management for SPL reuse. FOP is based on feature modularization, and a mechanism for feature composition into a base program. One of the motivations of FOP is to support SPLs. Mixin technique [16] has been widely used for FOP, with AHEAD [2] being its most advanced realization. AHEAD provides powerful solution for feature management in many situations, but is not be geared for fine-grained features. A number of authors also proposed Aspect-Oriented Programming (AOP) [13] as a variation mechanism. Using AOP, features are modularized as aspects (advices and introductions) and then weaved (feature composition) into a base program. A recent study has revealed difficulties in using AspectJ as a FOP realization technique [11].

In view of the above findings, Kastner et al. [12] relaxed the requirement for feature modularization, and revisited the idea of keeping feature-related code together with the code of core assets. They proposed a tool called CIDE (Colored IDE) to visualize feature code in core assets. CIDE helps programmers find and manipulate feature code. As CIDE works on an abstract syntax tree, it cannot handle some feature impacts. In contrast to CIDE, XVCL uses programming language-independent preprocessing-like program representation to achieve similar goals.

In CIDE, annotations may be validated to preserve language rules, while XVCL commands may be placed in arbitrary program points, leading to syntactic errors in the generated code. Aligning XVCL representation with constructs of the underlying programming language is possible and can alleviate some problems, but aligning is not enforced by the method. In addition, in some cases XVCL deliberately breaks such alignment for better flexibility in feature management. Validating meta-level transformations is the strength of CIDE, while simplicity, language-independence and the ability to handle any variability situations are strengths of XVCL.
6. Conclusions

This study was conducted jointly by Fudan Wingsoft Ltd., a software company based in Shanghai, researchers at Fudan University and National University of Singapore (NUS). Our earlier study of a Wingsoft product line revealed that applying multiple variation mechanisms may not scale well to larger software Product Lines (SPL). In particular, it becomes increasingly difficult to find and understand feature code scattered through core assets (so-called fine-grained features), and to coordinate changes at multiple inter-related variation points. Multiple variation mechanisms also hinder readability, reuse and evolution of core assets heavily affected by fine-grained features.

In this paper, as a remedy to the above problems, we presented an approach based on a single, uniform variation mechanism of XVCL, capable of managing both fine-grained features, as well as features whose impact requires customizations at the product architecture/component level. We evaluated the XVCL-based product line representation in lab experiments, and in Fudan Wingsoft Ltd, a company that initially used multiple variation mechanisms and then applied XVCL.

Overall, for small-to-medium systems such as WFMS (around 50KLOC), multiple variation mechanisms are still practical. Handled by experienced engineers, the original WFMS core assets serves well for the derivation of almost 100 product variants. Common variation mechanisms can be applied with minimum disruption of conventional programming. Multiple mechanisms provide an elementary infrastructure for SPL support.

However, as the impact of features on core assets accumulates and gets more complex, understanding and synchronizing multiple, poorly compatible variation mechanisms may become difficult. We may have much manual, repetitive and error-prone work to do during reuse and evolution of core assets and features.

This weakness of multiple variation mechanism approach is the strength of a uniform variation mechanism approach such as XVCL. XVCL captures customization knowledge in human-readable and machine-executable (by XVCL Processor) form. Therefore, feature reuse is simplified and automated. Knowledge of prior customizations helps in designing customization required by new features during SPL evolution. However, the design of reusable core assets with a uniform variation mechanism requires more skill and effort. Both common variation mechanisms and XVCL keep fine-grained feature code embedded in core assets which hampers readability. XVCL tried to alleviate this problem with feature queries that navigate developers to all the variation points relevant to features. This is only a partial remedy to feature scattering.

In the future, we plan to extend the scope of XVCL application to other subsystems of the WFMS. We will look for groups of similar components that can be represented as XVCL generic, adaptable structures and re-design WFMS core assets to take advantage of that. In that way, we may be able to substantially reduce the size of core assets, as well as the number of variation points, alleviating problems that seem to be inherent in approaches to managing variability such as described in this paper. We will formulate methodological guidelines for effective application of the XVCL approach by engineers at Fudan Wingsoft and in other companies. Based on these experiences, we plan to develop technology transfer guidelines. XVCL Workbench simplifies application of XVCL and increases reuse benefits. Building a production-quality XVCL Workbench is an essential part of our technology transfer plan.
References


