Software Product Line: Survey of Tools

By

Qaiser Munir
Muhammad Shahid
Final Thesis

Software Product Line: Survey of Tools

By

Qaiser Munir
Muhammad Shahid

LIU-IDA/LITH-EX-A—10/026—SE

2010-06-07

Supervisor: Kristian Sandahl
IDA, Linköping University

Examiner: Kristian Sandahl
Dedication

We dedicate our work to our families and parents who guided us throughout our lives and studies and are continuously praying for our success.
Abstract

A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission. The main attractive part of SPL is developing a set of common assets which includes requirements, design, test plans, test cases, reusable software components and other artifacts. Tools for the development of software product line are very few in number. The purpose of these tools is to support the creation, maintenance and using different versions of product line artifacts. This requires a development environment that supports the management of assets and product development, processes and sharing of assets among different products.

The objective of this master thesis is to investigate the available tools which support Software Product Line process and its development phases. The work is carried out in two steps, in the first step available Software Product Line tools are explored and a list of tools is prepared, managed and a brief introduction of each tool is presented. The tools are classified into different categories according to their usage, relation between the tools is established for better organization and understanding. In the second step, two tools Pure::variant and MetaEdit+ are selected and the quality factors such as Usability, Performance, Reliability, Memory Consumption and Capacity are evaluated.

Keywords
Acknowledgments

First of all we are thankful to Allah Almighty for all of His blessings, guidance and showing us straight path. Without His blessings we would never be able to complete our work.

We are thankful to our supervisor Kristian Sandahl for giving us opportunity to work under his kind supervision and for sharing his tremendous and insightful research knowledge during the discussions, for his continuous guidance in this thesis work, and for his overall kind attitude and support. With smiling face and awesome jokes he guided us and provided us all necessary knowledge and information.

We are grateful to Naeem Ahmad for giving us motivation, invaluable support and guidance. He remained a continuous source of help, support and guidance for us. Special thanks to Kayhan Ozturk for conducting usability test and Muteer Arshad for moral support.

Last but not least, we are thankful to our parents for praying, their love, financial support and invaluable guidance.

Qaiser Munir
Muhammad Shahid

Linköping, June 2010
Contents

Chapter 1 - Introduction
1.1. Software Reuse ........................................................................................................ 18
1.2. What is a Software Product Line? ........................................................................ 18
1.3. What a Software Product Line is not? .................................................................. 19
1.4. Problems ....................................................................................................................... 19
1.5. Motivation for Thesis ................................................................................................. 20
1.6. Thesis Topics ............................................................................................................... 20
1.7. Method of Work ......................................................................................................... 20
1.8. Contribution ............................................................................................................... 22

Chapter 2 - Theoretical Framework
2.1. An Argument about Software Product Line ............................................................ 23
2.2. Software Product Family, Population and Product Line ........................................ 23
2.3. Organizational Models ............................................................................................. 24
  2.3.1. Development Department .................................................................................. 24
  2.3.2. Business Units .................................................................................................... 24
  2.3.3. Domain Engineering Unit .................................................................................. 25
  2.3.4. Hierarchical Domain Engineering Units ............................................................. 25
2.4. Understanding Fundamental Tasks .......................................................................... 25
  2.4.1. Core Asset Development .................................................................................... 25
  2.4.2. Product Development ....................................................................................... 26
  2.4.3. Variability Management ..................................................................................... 26
  2.4.4. Feature Modeling ............................................................................................... 27
2.5. Industrial Perspective of SPL .................................................................................. 28
  2.5.1. Symbian ............................................................................................................... 29
    2.5.1.1. Company Background .................................................................................. 29
    2.5.1.2. Product Family ............................................................................................ 29
    2.5.1.3. Technology .................................................................................................. 29
    2.5.1.4. Process ......................................................................................................... 30
  2.5.2. HomeAway ........................................................................................................... 30
    2.5.2.1. Company Background .................................................................................. 30
    2.5.2.2. Product Family ............................................................................................ 30
2.5.2.3. Technology .......................................................... 31
2.5.2.4. Process ................................................................. 31
2.6. Advantages .................................................................. 31
2.7. Obstacles .................................................................... 33

Chapter 3 - Analysis of Tools

3.1. What is Software Product Line Tool? .................................................. 35
    3.1.1. Application to Core Asset Development ........................................... 36
    3.1.2. Application to Product Development .................................................. 36
3.2. Categories of SPL Tools .................................................................. 40
3.3. Relationship between SPL Tools ....................................................... 41
3.4. Motivation for the Selection of Tools .................................................. 43

Chapter 4 - Evaluation of Quality Factors

4.1. Usability ............................................................................. 44
    4.1.1. Pure::Variant ........................................................................ 45
    4.1.2. MetaEdit+ ........................................................................... 45
4.2. Performance ........................................................................... 46
    4.2.1. Pure::Variant ........................................................................ 47
    4.2.2. MetaEdit+ ........................................................................... 47
4.3. Reliability ............................................................................. 48
    4.3.1. Pure::Variant ........................................................................ 48
    4.3.2. MetaEdit+ ........................................................................... 49
4.4. Memory Consumption .............................................................. 50
    4.4.1. Pure::Variant ........................................................................ 50
    4.4.2. MetaEdit+ ........................................................................... 50
4.5. Capacity .................................................................................. 51
    4.5.1. Pure::Variant ........................................................................ 51
    4.5.2. MetaEdit+ ........................................................................... 51

Chapter 5 - Discussion

5.1. Usability ............................................................................. 52
5.2. Performance ........................................................................... 53
5.3. Reliability ............................................................................. 53
List of Figures

Figure 2-1: Feature diagram representing e-shop system ................................................. 28
Figure 1-1: Method of Work ................................................................................................ 22
Figure 3-2: Categories of SPL Tools ........................................................................ 41
Figure 3-3: Relationship between SPL Tools .............................................................. 43
List of Tables

Table 3-1: List of Software Product Line Tools ................................................................. 38
Table 4-2: Evaluation Designing ....................................................................................... 45
Table 4-3: Usability Evaluation of Pure::Variant ............................................................ 45
Table 4-4: Usability Evaluation of MetaEdit+ ................................................................. 46
Table 4-5: Small and medium project scale ..................................................................... 47
Table 4-6: System Specification ....................................................................................... 47
Table 4-7: Pure::Variant Performance Measurements ...................................................... 47
Table 4-8: System specifications ...................................................................................... 48
Table 4-9: MetaEdit+ Performance measurement ............................................................ 48
Table 4-10: Reliability measurements of Pure::Variant .................................................. 49
Table 4-11: Reliability measurements of MetaEdit+ ....................................................... 49
Table 4-12: minimum requirements ............................................................................... 50
Table 4-13: System requirements for Pure::Variant ...................................................... 50
Table 4-14: System requirements for MetaEdit+ ........................................................... 51
List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL</td>
<td>Software Product Lines</td>
</tr>
<tr>
<td>AOM</td>
<td>Aspect Oriented Modeling</td>
</tr>
<tr>
<td>OVM</td>
<td>Orthogonal Variability Management</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Software Engineering</td>
</tr>
<tr>
<td>SPLE</td>
<td>Software Product Line Engineering</td>
</tr>
<tr>
<td>IS</td>
<td>Information System</td>
</tr>
<tr>
<td>DSM</td>
<td>Domain-Specific Modeling</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain-Specific Language</td>
</tr>
</tbody>
</table>
Organization of the thesis

**Chapter 1** Introduces about Software Reuse and Software Product Lines. The chapter presents the problems, motivations, topics for thesis, method of working and contribution of the thesis. The chapter also describes what the Software Product Line is and what it is not.

**Chapter 2** Presents arguments about Software Product Line, Software Product Family, Population and Product Line, Industrial Perspective, Advantages and Obstacles of the Software Product Lines. Organizational Models such as Development Department, Business Units, Domain Engineering Unit and Hierarchical Domain Engineering Units are discussed. Fundamental tasks of SPL such as Core Asset Development, Product Development, Variability Management and Feature Modeling are also discussed.

**Chapter 3** Presents Software Product Line Tools and their support for Application to Core Asset Development and Product Development. The chapter also shows some short descriptions of GEARS, MTP, XVCL, Dopler, Pure::variant, Varmod, FAMA-FW, FeatureIDE, MetaEdit+, PuLSE-BEAT, Holmes, FeatureMapper and S.P.L.O.T tools.

**Chapter 4** Evaluates the Quality Factors such Usability, Performance, Reliability, Memory Consumption and Capacity of the Pure::variant and MetaEdit+ tools.

**Chapter 5** Presents the discussion about the obtained results of quality factors for Pure::variant and MetaEdit+ tools in details. The future work is also described in this chapter.
Chapter 1

Introduction

1.1. Software Reuse

Software reuse has become a significant ingredient of software development due to rapid and large amount of software production. The organizations adopt this approach to reduce cost, time to market and to increase the quality of the software. During 1980’s object oriented programming brought reuse in the form of classes [1] and later component based software development introduced reuse in the form of components. These methods did not obtain the original benefits of reuse due to the usage at a very small scale and opportunistic reuse.

Software product line came into being in 1990’s and adopted by several organizations to prove its success. In this approach the reuse is planned, large in scale, wide in range and profitable. The reuse includes artifacts which are costly to develop from scratch and planned in such a way to be used in a family of related products [2].

1.2. What is a Software Product Line?

According to Software Engineering Institute software product line can be defined as:
A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". [2]

The main attractive part of SPL is developing a set of common assets which includes requirements, design, test plans, test cases, reusable software components and other artifacts. The common assets are shared in all the derived products instead of developing for each product separately. The development of individual products from a set of common assets can lead to increase the productivity, quality and decrease the cost and time to market. Because sharing of core assets reduces the development effort which can ultimately decreases the time to market and cost.

Software product line has two major development parts domain engineering and application engineering. Domain engineering is associated with the development of core assets which are common in a family of products. The development of core assets is based on the commonality (common features) and variability (varying features) in the products. Application engineering is the development of individual products by reusing the core assets and adding functionality which is specific to each product.

1.3. What a Software Product Line is not?

Reuse in software product line is different than conventional reuse methods. It is significant not to be confused with the following notions of reuse.

- Different versions of a single product
- Small scale reuse
- Development of a single system with reuse functionality
- Only a set of standards
- Opportunistic reuse

1.4. Problems

The software development in product line is different from conventional software development. We will answer the following questions in this report.

- What are the available tools in product line?
- Classification of tools in different categories?
- Which factors are considered for the selection of tools?
- How to evaluate quality factors for software product line tools
- What is the future of product line tools?
1.5. Motivation for Thesis

Software product line is not more than two decades old, there is lot of research going on in this field but there is almost no this kind of standard document available in the industry and academics, we think that there is a space to contribute.

Another reason is that, this thesis can be beneficial for the organizations which intend to adopt the software product line. The document can be used to understand which software product line tool can be used to solve a specific problem and under which conditions a tool suits well.

Now a day’s all the bigger organizations are moving towards product line for the development of their products as it decreases the time to market and cost. The product line can be the future of the development organizations.

Software product line brings new method for reusability as it is planned reuse not opportunistic, which enhances the reuse.

The development of software can be made more or less like hardware development by selecting different features and components.

1.6. Thesis Topics

In the thesis we will investigate the SPL tools which are used to derive individual products from a set of common assets. There are several tools available in the industry; each having a different approach of developing core assets and sharing it with a number of products. We will investigate two product line industrial tools Pure::Variant and MetaEdit+. First we will cover theoretical concepts and all the available tools in the product line. Secondly we will evaluate the quality factors listed below of the selected tools.

- Usability
- Performance
- Memory consumption
- Reliability
- Capacity

1.7. Method of Work

In order to satisfy the requirements of the thesis we look for research papers published in different international conferences, collection of links dedicated to the software product line and also studied few product line books to get the domain knowledge of the field. We have investigated the reference list available at the end of research papers. We have also explored list
of vendors and a list of research papers related with product line in collection of links devoted to software product line.

We have used Google as search engine and query keywords such as “software product line”, “software product line tools”, “software product line engineering”, “tools for software product line”, “software product line management”, software reuse, feature modeling and variability management in software product lines. To get the details name of the tools has also been queried in search engine to get the relevant information of each tool.

We have used heuristic evaluation method in usability testing and quantitative evaluation methods in testing the performance, reliability, memory consumption and capacity of the selected tools.

![Figure 1-1: Method of Work](image)

21
1.8. Contribution

The thesis can be beneficial for both the academicians and practitioners as it contains both theory and practical part of the software product line. The contribution for the master thesis includes:

- First we cover the theoretical concepts and essential activities of the software product line
- Studied the available software product line supporting tools
- Classification of tools in different categories
- Evaluation of the quality factors for the selected tools
Chapter 2

Theoretical Framework

2.1. An Argument about Software Product Line

As the development of software is different from development of the hardware, Hardware is built by assembling different already built spare parts but the development of software is rather complex. The notion of making software as hardware is built by selecting different parts and components, and then assembling it according to the architecture, can’t be achieved by traditional software development techniques and methodologies. Software product line methodology can be used to develop products by defining a common architecture for a family of related products and then sharing it in all the products instead of developing each product from scratch.

The reuse of the assets and components can also be achieved through software product line. The traditional techniques of software development support reuse of the software assets at a little extent. Object oriented programming and the development of component based software introduce reuse in the form of classes and components which are opportunistic reuse. Software product line introduces proactive approach which supports reuse at a larger extent. According to Bosch reuse of the components is very difficult to adopt and is not common among the organizations. On the other hand reuse of software architecture within the organizational boundaries is also difficult but a reasonable approach to get the potential benefits of reuse [1].

2.2. Software Product Family, Population and Product Line

Software product family consists of many similarities and only few differences in a collection of products. Software product population contains a set of products having many similarities and many differences among them. Software product line is a proactive and planned approach to obtain the maximum reuse of software within product population or product family.
2.3. Organizational Models

Organization models are required to define the construction of organizations which is very significant to implement the software product line successfully. We will describe organizational models which can be applied while adopting software product line. For every model we will present [1] [7] advantages, disadvantages and in what conditions a model is appropriate.

2.3.1. Development Department

The reusable assets of product line and the actual products are developed and managed in a single department of the organization. The model does not define any specific roles for the product line; any staff member (engineers, architects) can be involved for asset or the concrete product development of the product line. The projects developed in the form of domain engineering and application engineering. The domain engineering is more concerned with the development of the reusable assets or a new version of the reusable assets. The purpose of the application engineering is the development of concrete products which can be delivered to the customer.

The model is suitable for small and consultant organizations having software related staff members up to 30. If the number of staff members increased than 30, there is a need for organizational restructuring. The development department model can be used very efficiently due to effective communication within a single department and ease of use. There is very little overhead involved in managerial and organizational level. When the organization expands the model is not suitable because there is restructuring of organization is involved. The second disadvantage is that the roles are not specified either domain or application engineer which can affect the quality of the product.

2.3.2. Business Units

Every business unit is responsible to develop a one or more than one products. The shared assets are developed in the form of domain engineering projects, the domain engineering development team members belong to all the business units involved in the development of the product line. Any business unit can evolve and extend the functionality of the reusable assets, the business unit responsible for the evolution of the reusable assets test the evolved or extended functionality before the use of other business units.

The approach is applicable when the number of staff members is between 30 and 100. The domain engineering unit is important to reduce n-n communication between all the business units to one-n communication between domain and product engineering units. The model is better than the development department which supports up to 100 software engineers. The drawback of the approach is that all the business units are involved in the development of the concrete
products so there is no clear concentration on the reusable assets which can affect the timely and reliable evolution of the assets in product line.

2.3.3. Domain Engineering Unit

The domain engineering model can be implemented by two ways; single domain engineering and multiple domain engineering units. In the first case only a single organizational unit is responsible for the development and evolution of the reusable assets (architecture and components). In multiple domain engineering units, one domain unit is responsible for design and evolution of the software architecture and for every architectural component or a group of components a component unit is responsible which manage the evolution and design of the component. The difference between the two approaches is that in the second approach the level of specialization increases and the product engineering unit needs to communicate with several domain engineering units.

The domain engineering approach is more applicable for organizations which the size of the domain engineering units is large, so it’s better to divide a larger unit into multiples to concentrate on the group of components for the product line. The advantage of the approach is that it reduces the communication from n-n to one–n way. In addition it supports a large number of software engineering staff than the other stated models.

2.3.4. Hierarchical Domain Engineering Units

The domain engineering units which have further specialization in product line use the upper most product line level assets as a base from where to find their own product line. The reusable shared set of assets at the top level is called as platform.

The model is suitable in conditions when there is high variability among products, and bigger organizations having long term existing products. The stated approach is capable to incorporate complicated and large product line families. The negative aspect includes, change in top most level assets require change in the specialized domain units which need to be balance delicately and it requires a lot of effort so that each unit can evolve independently.

2.4. Understanding Fundamental Tasks

2.4.1. Core Asset Development

The core asset base consists of all the artifacts which can be reused in the product line. The purpose of the development of core assets is to design a production plan for reusable assets from where the individual products can be instantiated. The output of the core asset development is the
product line scope, production plan and core asset base. Following concepts are used in the development of core assets.

**Product Constraints:** Determine the common and varying features in the products that will make a product line, and how different products behave. Predict about which functions the products must have in the future. Find out the bounds of performance and quality requirements of concrete products.

**Decision Model and Product Decision:** It describes the common and varying features for products in the product line.

**Production Strategy:** The process of how to assemble and configure core assets and the varying features in the products. Product decisions are used during production to configure the variation points and which assets are used as input.

### 2.4.2. Product Development

The development of each concrete product depends upon the scope of the product line, production strategy, core assets and the product’s description which is specific to every product.

- The scope of the product line determines whether it is possible to build the product from the core assets or not
- The production strategy determines how the core assets are utilized to develop the actual products
- Core assets are the artifacts from where the actual products are build
- Product description contains varying features which are specific to each product (common features are considered as part of core assets in product line)

### 2.4.3. Variability Management

One of the most attractive features of software product line is variability, which can be explained that a large number of products belonging to the same product line sharing some or most of the common characteristics but there are certain aspects which vary from other products in a product line. In order to manage this variability between the products the term software product line variability management is used. We will present three techniques to manage variability in product line.

**Aspect Oriented Modeling (AOM):** Aspect Oriented Modeling separates the functionalities and the crosscutting relationships [12]. In this technique functionalities are encapsulated in aspect, and the crosscutting relationships are encapsulated in aspect relation rules.

**Orthogonal Variability Management (OVM):** There are two methods to represent the variability in software product lines. The first method is to integrate the variability in existing models and the other is using a dedicated variability model in which variability is not combined
with the existing models [8]. The OVM uses the second approach for variability management in product line. The information which OVM documents regarding variability is [8]:

**Variation Point:** “This documents the variable item or variable property of an item.”

**Variant:** “This documents the possible instances of variation point.”

**Variability Constraints:** “These can be constraints on variability because product management decided.”

**Visibility Constraints:** “This discerns between internal and external variability. Only external variability is visible to the customer.”

### 2.4.4. Feature Modeling

Feature is the distinct aspect, quality or characteristic of a system. All the products in the product lines have some common and varying features. Feature model is the compact implementation of all the products in Software Product Lines in terms of their features. Feature modeling is the activity that identifies external visible characters of the products in the domain and organizes them in feature models. The basic goal of feature modeling is to identify commonalities and differences among all the products in Software Product Lines [10].

The features of the products are classified and identified in terms of Capabilities, Domain technologies, Implementation techniques and Operating environment. Capabilities are the visible characteristics that can be identified as distinctive services, operations and non-functional characteristics. Domain technologies represent the way of implementing these distinct services or operations. Implementation techniques are the general functions and techniques which are used to implement the services and operations. Operating environment is the environment in which applications are used. The common features among products are modeled as mandatory features whereas non-common features are modeled as optional or alternatives features. Optional features are used when selectable features have to model whereas alternative features are modeled when only one feature to be selected among different features [11].

Feature modeling is achieved with the use of feature diagram. Feature diagram is an AND-OR tree like hierarchy of features which focuses on conceptual and structural aspects of relationships among features. The relationships in feature diagram are of three types, Composed-of relationship, Implemented-by relationship and Generalization/Specialization relationship. When there exists part-whole relationship between a feature and sub-features then Composed-of relationship is used, when a feature is implemented to other sub-feature then Implemented-by relationship is used and when a feature is generalization of sub-feature then Generalization/Specialization relationship is used [11]. The root of the feature represents Software Product Line. The relationships between root and child can be established with the use of mandatory, optional and alternative notations [9].
The relationships between parent feature and child feature (sub-feature) can be divided in Mandatory, Optional, Alternative and OR relationships [9] [10].

Mandatory: If the child feature is mandatory then it will present in all the products in which its parent features appear. According to the Figure 2-1 all the products of E-Shop will contain Catalogue, Payment and Security features [9].

Optional: If child feature is set to be optional then it can be optionally present in all the products in which its parent features appear. According to the Figure 2-1 Search is the optional feature for the products [10].

Alternative: If in the child features, only one feature is allowed to be selected in which its parent features appear then the relations is set to alternative. In the Figure 2-1 only High or Standard can be selected at one time [9].

Or-relationship: The child features may have in the OR-relationship with the parent feature when at least one feature has to be selected in the given features. It can also be possible to select all the features which are in the OR-relationship but at least one feature has to be selected. In the Figure 2-1 Bank transfer and Credit card are the OR-relationship features. Both of the features can be selected but it is mandatory to select at least one of the features between these two.

For the development of reusable core assets for product lines variability and commonality from domain perspective are very essential. For the identification and organization of variability and commonality from domain perspective feature modeling is an efficient and effective method [11].

2.5. Industrial Perspective of SPL
In this section we have presented two case studies from the industry to show that how they implement the software product line to overcome their existing problems, increasing the productivity and to reduce the time to market. For each case study first we described the company background, product family, technology and at the end we have presented the process they follow.

2.5.1. Symbian

The product developed by the company is not an ending product but it’s a framework which is used to develop products related to wireless information devices such as communicators and smart phones [1].

2.5.1.1. Company Background

The market of mobile phones is covered by Nokia, Ericsson and Motorola because these are the major producers of mobile devices. Symbian was founded in 1998 from the software unit of Psion Software. Symbian is the producer of operating system and establishing the standards for mobile phones, smart phones and wireless information devices. The Symbian operating system is based on the Psion’s EPOC (electronic piece of cheese).

2.5.1.2. Product Family

The two major categories of products are hand held PC’s and smart phones. The hand held PC’s are becoming more towards wireless communication functionality. And the smart phones are getting more PC like functionality. There are three identified device families; a landscape display communicator, a portrait display communicator and a smart phone family. Communicators are information centric devices with voice functionality and smart phones are voice centric devices with information capability. These devices provide functionality like internet, fax, email, and SMS. Symbian has two families of smart phones one having a fix display and another with a switchable display between landscape and portrait display. And each family has a list of specific devices.

2.5.1.3. Technology

EPOC is a suite of applications used in wireless information devices and hand held battery powered computers. It also consists of connectivity software used for synchronization between computers and servers. EPOC consists of four key parts.

Core Components: The core components are run time kernel, multi tasking support, power management, support for client server architecture, API’s for data management and graphical user interface.
Communication Components: Communication components give drivers, API’s and protocols for data exchange.

Language: It includes a run time environment of java to JDK specification with AWT. It also includes run time and development environment for OPL.

Applications: Engine and graphical user interface which implements end user application. Separation of engine and graphical user interface favors the adoption of engine in several device families.

2.5.1.4. Process

Symbian makes two types of projects one is domain engineering and another is device family requirements definition (DFRD). Domain engineering is responsible for developing new versions of EPOC components and building new components. And DFRD is responsible for building next version of EPOC for a particular DFRD.

2.5.2. HomeAway

As in the previous case study we first present the company background, HomeAway, then product family and the technology which the company uses. Finally we describe the evolution process of the HomeAway product line [6].

2.5.2.1. Company Background

HomeAway is the worldwide leader of online vacation rentals. The company connects home owners, property managers with travelers who seek the space, value and services of vacation rental homes as an alternative to hotels. The company has largest and more diverse selection of homes in more than 100 countries. More than 50 million travelers visit the HomeAway each year to select the vacation rental homes. HomeAway gives vacationers privacy, space and prices unmatched by hotels.

2.5.2.2. Product Family

HomeAway includes HomeAway.com, VRBO.com, CyberRentals.com, AlVacations.com, GreatRentals.com, TripHomes.com, Holiday-Rentals.co.uk (UK), HolidayRentals.fr (France) and FeWo-direkt.de (Germany). The BigLever Software Inc. implements product line approach for HomeAway. The set of reusable assets for the above mentioned family of products can be web portals, content management systems (CMS), graphical user interface functionality, adding and removing properties, menus and web pages etc. The varying features may be language, geographical location and brand etc.
2.5.2.3. Technology

HomeAway applied the 3 tiered software product line methodology and the Gears framework to transition from conventional software development to product line development. Configuration management trigger methods are used to manage the change in variation places and for notification. Gears enable HomeAway for separate and automatic production, build, test and deployment of the sites.

2.5.2.4. Process

HomeAway achieved its technical and business benefits by using 3 tiered mechanism and Gears software product line engineering technology in 60 days. New features and products can be added in the sites without disturbing other sites with no overhead. Different new sites can be extended from the family of products on the bases of user profiles and interactions.

2.6. Advantages

After studying different case studies presented in section 2.5 and in [4] we have found several benefits of Software product line [2] [3] [5].

1. **Decrease time of development:**

   In the initial stages of product development, product lines take more time as compared to development of similar products using singular system. The reason behind this is product line requires some efforts for planning, architecture and realization of the infrastructure. But when infrastructure is developed, new product can easily be developed by reusing this infrastructure due to which it takes less time to develop new products.

2. **Reduce number of defects:**

   Product lines highly reduce the number of defects. The companies which are using product lines have reported that reduction in the defects rate as 96%. The high percentage shows the effectiveness of product lines for defect reduction. This capability of product lines enable engineers to spent their times on the development of the new products rather than wasting it to find and fix bugs and errors.

3. **Decrease in development cost:**
As already mentioned that by using product line techniques new products can easily be developed by using infrastructure reusability so it decreases development cost of the products. On the other hand, developing new products using singular system require that more the products more would be effort and cost. So product line decreases the development cost of the products.

4. Decrease development effort per product:

For the development of similar products using a singular system expected to require almost constant effort. Product lines require some initial efforts for planning, architecture and realization of the product line infrastructure and initially take some more time as compared to other systems but when the infrastructure is developed it is very easy to reuse it to develop new products. So product lines decrease development effort per product as compared to other systems.

5. Increases productivity:

Software product line reduces effort and cost to develop and deploy of the similar software products due to which productivity of the software engineers increases. The companies that use product lines reported that increase in the improvement of productivity range between a factor of 2 to 3.

6. Decrease time to market:

Before the release of the first product, product line requires some more planning and development effort and hence takes some more time. But after the release of the first product it requires less development time and development effort to develop new products so new products can be introduce to market more quickly as compared to other systems.

7. Utilizes human resources:

As mentioned before that product lines reduce effort and cost of development and deployment of the similar products. Moreover product lines reduce number of defects up to the rate of 96%. Due to these reasons time and efforts of the human resources can be saved which can be spent on the development of other products and/or to accomplish other tasks. So with the use of product lines human resources can be utilized in an effective ways.

8. Large scale reuse:
The organizations which are developing similar products in the same applications domain, software product lines not only reduce their development and maintenance cost but also shortens the time to market and increase the productivity of these organizations. From the definition of the software product line, it can be observed that software product line use common sets of core assets for achieving reusability. Core assets are the reuse software artifacts which are so generic that they support the development of different products in the software product lines. These core assets can be requirement specifications, components, architectures, test cases or any document which can be generic enough to support reusability of the products.

9. Increase customer satisfaction:

Software product lines increase the customer satisfaction. The product line through its quality of mass customization capability addresses the customer satisfaction. The product lines not only focus on how each product satisfies the needs of the customer but how well each product satisfies the needs of the customer.

10. Decrease labor cost:

As mentioned before effort and cost of development and deployment of the similar products can be reduced by using product lines and new products can easily be developed by using infrastructure reusability. Due to these reasons labor cost also decreases because of the fact that time of development and deployment of the new products decreases by using product lines which tends to decrease in the labor cost.

2.7. Obstacles

There are some hurdles and obstacles involved in the development of software product line which need to be coped to make the software product line successful.

- Investors are reluctant to invest because product line as a software development technique is still under development phase.
- Only targeted to big industry (aerospace, automobile, military, mobile phone) , need some case studies from small industry
Open source community should be involved in the development of product line tools and techniques except few bigger organizations.

There are some management and organizational risks involved which needs to mitigate.

The area need equal attention from both academics and industry.

Lack of availability of tools, a PhD student in Linkoping University recently presented his thesis which shows a proof of the unavailability of the tools in the software product line [28]. The problem has also been identified in “Software Product Line Practices and Patters” written by P. Clements and L. Northrop [13].
Chapter 3

Software Product Line Tools

In this chapter we will present the available tools which are used for the development of software in product line. The goal of this chapter is to observe which tools are available and give a brief introduction for each tool.

3.1. What is Software Product Line Tool?

There are several activities in software development which are automated with the support of CASE (computer aided software engineering) tools to convert the customer requirements into a useful product. There are numerous tools available to help the automation process of different phases of software development e.g. analysis, design, implementation and maintenance [13]. The selection of the tool depends on the business requirements and the technical demands of the product [13].

Tools for the development of software product line are very few in number [13]. The purpose of tools in the product line is to support the creation, maintenance and using different versions of product line artifacts (core assets and application products). This requires a development environment that supports the management of asset and product development, processes and sharing of assets among different products [13]. Tools for software product line support the following features in a product line environment [13] [14].

- The tool supports the structure of the organizations which intend to implement the product line and the development process of the product line.
- Characterize the common and varying features of the products in the product line where it is necessary (requirements, architecture, testing etc).
- Manage traceability and dependency between products and assets for a product line to manage the evolution and maintenance using configuration management capabilities.
- The development process is based on the architecture which supports various purposes of the product line architecture.
• Maintain a stable product line production operation by producing application products from their specific requirements.
• Support technical and management practices e.g. metrics and scoping.
• Maintain a criteria to determine either a product is a member of the product line or not, tailoring a product to specific requirements without violating the scope of the product line.

3.1.1. Application to Core Asset Development

The tool support the development of core assets and the production strategies used to derive products from core assets. Tools are required in the development of core assets and product line practices for instance [13].

• Product line scoping practices are supported by tools that collect and show the common and varying features of products in the product line.
• Tools are needed in the product line requirement engineering to represent the requirements for products in the product line.
• Tools are required in the product line architecture to portray the products in the product line without violating the product line scope definition.
• Tools are required in the product line component development to develop and test components to check the consistency with common and varying features.
• Tools are needed in the production plan practices used to create strategies and methods that support the performance of the development of the products.

3.1.2. Application to Product Development

The tool support in the product development depends on the extent of automation implicated in the production of specific products. The development of products can be entirely or partly automated. The strategies of product development are mentioned in the production plan. The production plan may vary depending on which activities are automated in the product development [13].

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Organization</th>
<th>Industrial/Academics</th>
<th>Technique</th>
<th>Customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gears</td>
<td>Biglewer Software</td>
<td>Industrial</td>
<td>Feature Modeling</td>
<td>Salion, HomeAway, LSI Log</td>
</tr>
<tr>
<td>MTP</td>
<td>Prosperity Height Software</td>
<td>Industrial</td>
<td>Text Based Adaptable Components</td>
<td>------</td>
</tr>
</tbody>
</table>

http://www.biglever.com
<table>
<thead>
<tr>
<th>Tool</th>
<th>User</th>
<th>Type</th>
<th>Feature</th>
<th>Domain</th>
<th>Company/Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure::Variant</td>
<td>Pure Systems GmbH</td>
<td>Industrial</td>
<td>Feature Modeling</td>
<td>Audi, Danfoss Drivers, Daimlers Chrysler Research</td>
<td></td>
</tr>
<tr>
<td>XVCL</td>
<td>National University of Singapore</td>
<td>Academic</td>
<td>Frame Technology</td>
<td>SES Systems Pte Ltd</td>
<td></td>
</tr>
<tr>
<td>Varmod</td>
<td>University of Duisburg-Essen, Germany</td>
<td>Academic</td>
<td>Orthogonal Variability Modeling</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>MetaEdit+</td>
<td>MetaCase</td>
<td>Industrial</td>
<td>Domain Specific Modeling</td>
<td>Nokia, Philips</td>
<td></td>
</tr>
<tr>
<td>FeatureIDE</td>
<td>University of Megdeburg, Germany</td>
<td>Academic</td>
<td>Feature Modeling</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>FAMA-FW</td>
<td>University of Servilla, Spain</td>
<td>Academic</td>
<td>Analysis of Feature Modeling</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>Dopler (Toolsuit)</td>
<td>Johannes Kepler University, Australia</td>
<td>Academic</td>
<td>Product Line Scope</td>
<td>Seimens VAI</td>
<td></td>
</tr>
<tr>
<td>Pulse-Beat</td>
<td>Fraunhofer Institute Experimental Software Engineering, University of Kaiserlautern, Germany</td>
<td>Academic</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Holmes</td>
<td>University of Alberta</td>
<td>---</td>
<td>SPL Life Cycle Support</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>FeatureMapper</td>
<td>Technische University, Dresden</td>
<td>Under development</td>
<td>Model Driven Development and Product Line</td>
<td>---</td>
<td></td>
</tr>
</tbody>
</table>
GEARS

GEARS [15] is the software product line tool developed by the BigLever Software Inc. GEARS is the software product line (SPL) life cycle framework which provides all the capabilities of SPL tool throughout the software development life cycle. The Gears feature model language gives access to software architects to represent the similar and varying features in the form of a model which distinguishes the products in the portfolio. For each product the feature profile is created separately in the portfolio.

MTP

Meta Programming Text Processor [16] is a tool used to define and derive text based adaptable components which are developed by the Prosperity Height Software. Adaptable component consists of software artifacts or a set of programs from which instances of the family can be derived. MTP notation indicates the sets of decision which differentiates the producible products. The decisions can be used to derive instances using MTP tool and to fulfill the varying needs of different instances.

XVCL

XML based Variant Configuration Language [17] is a technology for enhanced reusability and maintenance developed by the National University of Singapore. XVCL is a variability management mechanism used to manage the different products which may vary in requirements, design etc. The unique feature of the tool is while evolving the reusable assets it changes the reusable assets in products where it is required without disrupting other products variants.

Dopler

The tool [18] is developed by the Johannes Kepler University based on the feature specifications which represents individual products in product line to automatically generate the software. The approach used by the tool introduces business decision making in the SPL which includes individual product features, stakeholder needs, variability management, architecture essentials and makes configuration of the product by the non technical persons for instance sales people can configure a product from the feature specifications. The tool uses DecisionKing, ProjectKing and Configuartion Wizard for variability modeling, product derivation and product customization including evolution of assets respectively.

Pure::variant
Pure::Variant [19] is developed by the Pure Systems GmbH and supports all the phases of software development from requirements specification to test cases and maintenance. The modules of the tool are designed to fulfill needs of the embedded systems. The requirements are expressed in the form of feature models which depict the individual products in the SPL. The representation of the feature models depends on the stakeholder developer, user, customer etc. The design solution of the feature model is performed and included in the family of models which can be further processed automatically. There are several levels for family models and the highest level is made by components, which presents the solution of one or more functional features in the form of logical parts of the product (classes, functions etc).

Varmod

VARMOD [20] is developed by the University of Duisburg-Essen to manage the variability in software product line. The tool uses the orthogonal variability modeling (OVM) approach. Variability management creates and validates the relationship between varying products, variation points and varying assets. The VARMOD tool environment includes VARMOD-EDITOR, VARMOD-MODELER and VARMOD-DEVELOPER used for orthogonal variability models, assigning of core assets to specific variants and individual product development respectively.

FAMA-FW

FAMA-FW [21] is produced by the University of Seville, and is a framework used for automated analysis of feature models using logic representation techniques.

FeatureIDE

FeatureIDE [22] is an Eclipse based plugin developed by the University of Magdeburg that supports the development of a set of programs using AHEAD architecture.

MetaEdit+

MetaEdit+ [23] is developed by the meta-CASE a well known organization for domain specific modeling. The tool is an integrated environment for constructing and using domain specific solutions. It provides support for multi-user, multi-platform environment, distributed sources of data, language support and integration with third party tools.

PuLSE-BEAT

Pulse-beat [24] is a decision support tool for product line systems. The tool has three different phases; categorizing the characteristics which is executed by domain and product line experts. The second phase is about to gather the information and performed by the domain experts. The third phase is about scoping which is performed by the product line experts. The decision is made on the above three requirements.
Holmes

The tool is developed [25] by the University of Alberta and supports all the phases of software product line. The Holmes supports activities such as describe product line, classify market segment and differentiate users and investigate the relationship among the products. The modeling, designing and development of the product line are also included in the Holmes using an integrated method. The Holmes uses a critiquing system for its users to provide semantic support.

FeatureMapper

The tool is under development in Technische Universität Dresden [26]. The tool approach is to combine the model driven development (MDD) and software product line engineering (SPLE).

S.P.L.O.T

Software product line online tool [27] provides a web based reasoning and configuration for product line systems. The system is developed in java with HTML based engine for user interfaces. The system also provides a database for feature models which can be helpful for product line researchers and practitioners.

3.2. Categories of SPL Tools
3.3. Relationship between SPL Tools

Software reuse is the process of creating software artifacts from the existing systems. The idea is to create new software systems using existing software systems rather than developing the software systems from scratch. The type of artifacts can be module-level implementation structures, design structures, documentation, specifications, transformations and so on. Software reuse is needed to achieving better, more quickly and lower cost software.

For managing and developing relationships between the tools we divided the tools in two categories; Software Reuse and Software Product Lines. Software Reuse category contains the tools which are mainly used or developed for reusability purpose whereas Software Product Lines category contains the tools which are used or developed for product lines. The distribution of these tools and their relationships are shown in the figure 3-2.
Figure 3-3 Tools Relationship diagram

<table>
<thead>
<tr>
<th>Tool</th>
<th>Icon</th>
<th>Tool</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gear</td>
<td><img src="icon" alt="Gear" /></td>
<td>MetaEdit+</td>
<td><img src="icon" alt="MetaEdit+" /></td>
</tr>
<tr>
<td>Pure::variant</td>
<td><img src="icon" alt="Pure::variant" /></td>
<td>Doppler</td>
<td><img src="icon" alt="Doppler" /></td>
</tr>
<tr>
<td>FeatureIDE</td>
<td><img src="icon" alt="FeatureIDE" /></td>
<td>Pulse-Beat</td>
<td><img src="icon" alt="Pulse-Beat" /></td>
</tr>
<tr>
<td>S.P.L.O.T</td>
<td><img src="icon" alt="S.P.L.O.T" /></td>
<td>Holmes</td>
<td><img src="icon" alt="Holmes" /></td>
</tr>
<tr>
<td>MTP</td>
<td><img src="icon" alt="MTP" /></td>
<td>FeatureMapper</td>
<td><img src="icon" alt="FeatureMapper" /></td>
</tr>
<tr>
<td>XVCL</td>
<td><img src="icon" alt="XVCL" /></td>
<td>FAMA-FW</td>
<td><img src="icon" alt="FAMA-FW" /></td>
</tr>
</tbody>
</table>
3.4. Motivation for the Selection of Tools

The software product line tools are very few in number and we want to evaluate two tools belonging to two different categories shown in figure 3.2. For the accomplishment of this task we first contact the most well known tool in the software product line Gears but the tool was not available for academic evaluation. We select the Pure::Variant another commercial tool in the product line and we did not find any problem of getting the evaluation copy of the tool.

We also face some installation problems with an open source tool FeatureIDE due to the reason another commercial tool famous for domain modeling MetaEdit+ is selected to evaluate its quality factors.

The second motivation to select the tools is that, it was easy to get the help material and white papers about the tools.

We also want to experiment with the evaluation methods to understand the maturity of the tools.
Chapter 4

Evaluation of Quality Factors

In this chapter we will explore the quality factors of the selected tools. The results are based on the authors’ observations by running the tools for several hours and observing their behavior from different aspects of quality factors.

The experiments have been performed on the authors’ site and used windows systems with 1000 Mega Bytes of RAM.

4.1. Usability

Heuristic evaluation is an informal method of usability inspection; its purpose is to identify the problems with the user interfaces of a particular application. The evaluation is carried out by assigning tasks to the users to determine at what extent the user interface is according to the intended user’s preferences and requirements. There were two scenarios designed for each tool to measure the usability of the tools. The first scenario was about to create a simple project application and adding some relevant functionality in the project, but the second task was rather difficult than the first task. Each author was working on a separate tool and was responsible for the collection of results. We have two sets of data for each tool. There was one evaluator involved in the experiment except authors. To obtain the second data point the authors exchange their roles from observer to evaluator and evaluator to observer with each other. It is important to mention that authors were completely unaware of the tools when they were behaving as evaluators.
An oral description of the tool and a written documentation of the tasks have been provided to the evaluators before the experiments.

<table>
<thead>
<tr>
<th>Experimenter</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluator1</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>Evaluator2</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

*Table 4-2: Evaluation Designing*

### 4.1.1. Pure::Variant

<table>
<thead>
<tr>
<th>Observed Element</th>
<th>Task1</th>
<th>Task2</th>
<th>Task1</th>
<th>Task2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Spent on Task</td>
<td>4</td>
<td>6</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>(minutes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Completed?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mistakes</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Good/bad features called</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>by Evaluator</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of help Menus</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Table 4-3: Usability Evaluation of Pure::Variant*

**Observations:**

The first evaluator took less time to complete the task1 than the second evaluator due to the familiarity with eclipse platform. It is also observed during the experiment that evaluator learn very quickly from the graphical illustrations provided in the help material instead of the written text. The evaluator2 took less time to complete the second task due to the expertise in the software product line.

### 4.1.2. MetaEdit+
<table>
<thead>
<tr>
<th></th>
<th>Evaluator1</th>
<th>Evaluator2</th>
<th>Evaluator1</th>
<th>Evaluator2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Spent on Task (minutes)</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Task Completed/Not Completed</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mistakes</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Good/Bad Features called by Evaluator</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Use of Help Menu</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

*Table 4-4: Usability Evaluation of MetaEdit+*

**Observations:**

Both of the evaluators took same time to accomplish the task. The second evaluator did more mistakes than first due to having no expertise in product line while performing task. The evaluators faced problem when adding roles to states and buttons while domain modeling in task due to complex interface.

**4.2. Performance**

The performances of the tools are measured by running the tools under a certain workload and observed how the tools behave. The functionality used to test the performance of the tools varies from one tool to another. For the pure::variant tool the performance is measured in terms of number of features and for MetaEdit+ tool the performance is determined on the number of states. Apart from this the performance varies as the size of project increases.

To achieve the performance testing we started running small projects to medium level projects containing several features models and number of states. There are two types of response time calculated; measured response time and experimental response time. The measured response time is the actual time responded by the system against a particular functionality. The experimental response time is calculated by subtracting the reaction time from the measured response time. We have used the stop watch to calculate the measured response time of the system. The -0.2 seconds is considered as the reaction time while switching the hand from the application to the stop watch. Along with this the tools performance is also dependent on system resources (on which the tests are performed). These are the assumptions made while performing the test.

\[
\text{Measured Response Time} = \text{MTR} - \text{RT}
\]
Experimental Response Time (ERT) = MTR-RT

<table>
<thead>
<tr>
<th>Project Type</th>
<th>No. of States/Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>4 – 10</td>
</tr>
<tr>
<td>Medium</td>
<td>11 – 19</td>
</tr>
</tbody>
</table>

*Table 4-5: Small and medium project scale*

### 4.2.1. Pure::Variant

The experiment has been conducted on the author’s site with the following specifications of the system.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Microsoft Windows 7 Professional</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Pentium Dual CPU @ 1.6GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
</tr>
<tr>
<td>System type</td>
<td>Laptop</td>
</tr>
</tbody>
</table>

*Table 4-6: System Specification*

### Observations:

<table>
<thead>
<tr>
<th>Functionality Name</th>
<th>No of Features</th>
<th>Measured System Response Time (Seconds)</th>
<th>Reaction Time (Seconds)</th>
<th>Experimental Response Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop Selection</td>
<td>5</td>
<td>1.1</td>
<td>-0.2</td>
<td>0.9</td>
</tr>
<tr>
<td>Car Example</td>
<td>13</td>
<td>1.3</td>
<td>-0.2</td>
<td>1.1</td>
</tr>
<tr>
<td>Weather Station</td>
<td>15</td>
<td>3.2</td>
<td>-0.2</td>
<td>3.1</td>
</tr>
</tbody>
</table>

*Table 4-7: Pure::Variant Performance Measurements*

The experimental results show that the system works well for both small and medium level projects but as the number of feature models increases the system response time also increases.

### 4.2.2. MetaEdit+

47
The experiment is conducted on the author’s site with the following specification of the system.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Microsoft Windows XP (SP-2)</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Pentium Dual CPU @ 1.6GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
</tr>
<tr>
<td>System type</td>
<td>Laptop</td>
</tr>
</tbody>
</table>

*Table 4-8: System specifications*

**Observations**

<table>
<thead>
<tr>
<th>Functionality Name</th>
<th>No. of Features</th>
<th>Measured System Response Time (Seconds)</th>
<th>Reaction Time (Seconds)</th>
<th>Experimental Response Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Watch</td>
<td>5</td>
<td>0.8</td>
<td>-0.2</td>
<td>0.6</td>
</tr>
<tr>
<td>Restaurant Finder</td>
<td>10</td>
<td>1.1</td>
<td>-0.2</td>
<td>0.9</td>
</tr>
<tr>
<td>Conference Registration</td>
<td>12</td>
<td>1.7</td>
<td>-0.2</td>
<td>1.5</td>
</tr>
</tbody>
</table>

*Table 4-9: MetaEdit+ Performance measurement*

The experimental results show that as the number of states increases the response time of the system also increases.

### 4.3. Reliability

The reliability of a software application is calculated by running the software for thousands of hours and changing the observer after a specific number of hours running in real scenarios. Due to resource constraints and software limitations we cannot have thousand of execution hours. To measure the reliability we calculate the failure ratio of the tools. We run the tools for a number of hours and observe behavior of the tools. We run the tools in a segment of 5 hours (maximum) and write down in a separate excel sheet the time of running, functionality performed and the number of crashes found. After a certain time period we calculate the number of hours running and the identified faults for each tool separately.

#### 4.3.1. Pure::Variant
<table>
<thead>
<tr>
<th>No. of Features</th>
<th>No of Hours Under Work</th>
<th>No of Crashes found</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>23</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

*Table 4-10: Reliability measurements of Pure::Variant*

Observations:

During the experiment we have found only one fault where the system goes in an infinite state while executing the application. The system is supposed to show some sort of execution instead of an infinite condition. The results show that if we run the system for 23 hours there are chances that the system may show unexpected behavior.

4.3.2. MetaEdit+

<table>
<thead>
<tr>
<th>No. Of States/Objects</th>
<th>No. Of Hours</th>
<th>No. Of Crashes/faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>28</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

*Table 4-11: Reliability measurements of MetaEdit+

Observations:

The tool has been run 28 hours and we did not find any crash or failure in the system and its behavior was as intended.
4.4. Memory Consumption

We have gone through all the white papers and manuals provided on the websites to get information about the resource utilization and memory consumption, installation packages, supporting applications and plug-ins, additional required platforms, system resources and operating systems. In some cases we have contacted the vendor organizations to get information about specific attributes of the memory consumption.

4.4.1. Pure::Variant

The following are the minimum requirements for the tool to be installed on any machine.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Operating Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse 3.0-3.2</td>
<td></td>
</tr>
<tr>
<td>JRE (Java run time environment)</td>
<td></td>
</tr>
<tr>
<td>RAM  256 MB</td>
<td></td>
</tr>
<tr>
<td>X86 based systems</td>
<td></td>
</tr>
</tbody>
</table>

*Table 4-12: minimum requirements*

4.4.2. MetaEdit+

MetaEdit+ is platform independent tool therefore it works on all major platforms. The details of platforms, hardware and operating systems of the MetaEdit+ tool are shown in Table 4-4. For a single computer the software takes 40MB space on the hard disk and uses 40MB memory while running. For multi-user systems the server requires 20MB of memory.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Hardware</th>
<th>Operating Systems</th>
</tr>
</thead>
</table>

*Table 4-13: System requirements for Pure::Variant*
### Table 4-14: System requirements for MetaEdit+

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Processor/Platform</th>
<th>Operating System Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS X</td>
<td>Mac (PowerPC or Intel)</td>
<td>Mac OS X 10.3.4 or later (Panther, Tiger, Leopard)</td>
</tr>
<tr>
<td>Linux</td>
<td>Pentium</td>
<td>Any modern distribution e.g. Ubuntu, SuSE, RedHat(i386, Kernel 2.2 or later, glibc 2.1.3, or later. X11R6)</td>
</tr>
<tr>
<td>HP-UX</td>
<td>HP PA-Risk, e.g. HP 9000 Series 700</td>
<td>HP-UX 11.x or later</td>
</tr>
<tr>
<td>Solaris</td>
<td>Sun SPARC workstation</td>
<td>Solaris 5.6 or later</td>
</tr>
</tbody>
</table>

#### 4.5. Capacity

The capacity of the tools is determined by calculating the number of bytes occupied by the amount of features/states created in a particular application and up to what extent the system fulfill its requirements. The hardware and software are constant in testing the capacity of the workload for each tool but the size of the functionality varies such as number of features in case of pure::variant and number of states in MetaEdit+. The goal of this test is to determine that in a specific workload whether the system meets its requirements of the size or not.

#### 4.5.1. Pure::Variant

The tool creates a separate XML file for each feature model and save it in the workspace. All the child features in a particular feature model are created in the same XML file. There is 1.5 kb memory size required for each feature model. The tool manages the feature models as a file based database. The capacity of the feature models depends as the system requirements support the tool.

#### 4.5.2. MetaEdit+

The objects/states are stored in a repository directory. The repository directory contains *areas*, *users* and *backup* subdirectories. The repository directory consists of two files *manager.ab* and *trid*. The *manager.ab* contains the names, paths and encoded passwords of all the *areas* in that repository. For each object/state approximately 1 KB memory size is required. The capacity of the objects/states depends as the system requirements support the tool.
In this chapter we will analyze the obtained results of the quality factors. And discuss possible reasons behind the results which differentiate or resemble the results of the tools. We will also present the future work of the software product line and the selected tools.

5.1. Usability

The new users can skip the learning cycle of the tool and can start working without hesitation if they are familiar with the platform. The experimental results show that the tool based on the existing platform can be easier to learn and use instead of a tool which is not based on existing platforms due to familiarity with the platform as shown in table 4-3. We have seen in both cases that the domain expertise can also help the user to learn quickly than the non experts. Both the users have completed the task1 in same time period and did same mistakes in task2 due to complex interface of metaedit+ as shown in table 4-4. The graphical illustrations are more attractive to the users due to easy to learn and remember capabilities than the written text, which
have been seen in case of pure::variant. MetaEdit+ can be made user friendly by adopting standard drawing frameworks.

The number of bad features is higher than the number of good features called by the evaluator in pure::variant while performing task2 which show a mismatch between functionality and evaluator’s mindset.

5.2. Performance

There can be different dimensions of measuring the performance testing of the tools such as how quickly the system respond while executing a particular application having a specific amount of functionality, switching from one workspace to another, query execution and report generation, scalability and stability. Due to the limitation of resources we were able to evaluate the system response time against query execution.

The experimental results show that the response time of a system depends on the number of features or number of states in the given system. The response time increases as the workload on the tools increases and vice versa. There is a 0.2 seconds difference as shown in table 4-7 in the response time when number of features increase from 5 to 13 and 1.9 seconds difference when features increase from 13 to 15 due to inter dependency between different features. Each feature has some attributes which creates dependency between features.

There is a 0.3 seconds difference in the response time as shown in table 4-10 when number of states increase from 5 to 10 and 0.6 seconds difference when states increase from 10 to 12, which also show a dependency between different states and other connectors which establish a connection between different states. The response time can also depend on the system specification, network and bandwidth, higher the system specifications the more early system will respond.

The performance testing is more concerning about whether the tools are fast enough or not, to respond the user in defined time period and what need to plan for future if the work load increase than a limited amount and what if tools show an unexpected behavior. The performance shortcomings can be resolved in the next version of the tools.

5.3. Reliability

Reliability testing determines the effectiveness of the system and can be helpful to make the system more secure and free from failures. The system will be more reliable if it has less chances of failure to occur. The selected tools were run for at least 23 hours and different size and type of functionality has been performed to identify the faults.
The failure ratio metric is used to determine the reliability of the selected tools. During the experiment as shown in table 4-11 we have identified a single fault in pure::variant by running it in 23 hours and executing different functionality. The tool showed an infinite waiting state and session was terminated forcefully. There can be two possible reasons of the unexpected tool’s behavior. The functionality performed on the tool did not synchronize well with the tool’s internal structure which kept the tool in a waiting state infinitely. The second possible reason can be communication problem between the functionality and the tools interface.

The MetaEdit+ tool as shown in table 4-12 was run for 28 hours and we did not find any fault or unexpected behavior while testing the different sort of functionality on the tool. The tool work as it was intended, all the interfaces and connections were working properly. The tool can be said as a reliable tool for domain modeling purposes as we did not find any fault for a number of running hours.

Reliability has a bigger impact on software product line because different components and feature models/state models are used to instantiate the individual products from a set of common assets. The reliability of every component and feature model can increase/decrease the reliability of the overall product.

5.4. Memory Consumption

There is no extra memory or other specifications required by the pure::variant except Eclipse platform which makes it easy for experimenters and organizations to adopt. There is 20-40 MB of memory is required to install the metaedit+ on any machine.

5.5. Future Work

The software product line has several challenges in its maturity phase such as

- variability management
- lack of variability management standards
- lack of tool support

The product line needs equal attention from the industry and academics to bring new methods for variability management and to improve the existing approaches and come up with professional tools to support the entire development life cycle of product line. The usability problems of the metaedit+ can also be overcome to make the tool easy to use for new developers and users.
Conclusion

The research is going on from last two decades in Software Product Line. The development of common assets includes requirements, architecture, design, test plans, test cases, reusable software components and other artifacts. The individual products are developed from a set of common assets which can lead to increase the productivity, decrease the development effort, cost and time to market.

The development of software product line is different than conventional software development which demands tool support for product line. There are very few tools available for software product line. The area needs more attention from the industry as there are more tools available in the academics as compared to the commercial category. The tool support is also significant as there are no standards available to manage the variability among family of products.

The pure::variant is a pure product line tool which supports all the phases of the product line development process and provide integration for the standard case tools. The metaedit+ is a domain specific modeling tool which can be used for software product line development purposes. The heuristics evaluation methods can be suitable for the usability evaluation of the product line tools when there are some resource constraints. The list of available tools and their description and evaluation results of the quality factors for the selected tools can be useful for the researchers, academicians, professionals and vendor organizations to learn, improve or upgrade the functionality of the tools.
References


[13]. P. Clements and L. Northrop, Software Product Lines: Practices and Patterns,


Abstract
A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission. The main attractive part of SPL is developing a set of common assets which includes requirements, design, test plans, test cases, reusable software components and other artifacts. Tools for the development of software product line are very few in number. The purpose of these tools is to support the creation, maintenance and using different versions of product line artifacts. This requires a development environment that supports the management of assets and product development, processes and sharing of assets among different products.

The objective of this master thesis is to investigate the available tools which support Software Product Line process and its development phases. The work is carried out in two steps, in the first step available Software Product Line tools are explored and a list of tools is prepared, managed and a brief introduction of each tool is presented. The tools are classified into different categories according to their usage, relation between the tools is established for better organization and understanding. In the second step, two tools Pure::variant and MetaEdit+ are selected and the quality factors such as Usability, Performance, Reliability, Memory Consumption and Capacity are evaluated.

Number of pages: 60

Keywords