Spotting the Phenomenon of Bad Smells in MobileMedia Product Line Architecture

Manjinder Kaur
Asst. Professor, PG Dept. of Computer Science & IT
Lyallpur Khalsa College, Jalandhar
Punjab, India
foundation131288@gmail.com

Parveen Kumar
Asst. Professor, Dept. of Computer Science & Engg
National Institute of Technology, Uttarakhand
Srinagar, India
parveen.it@gmail.com

Abstract*: *Product line engineering a very new engineering method evolved in few past years has proven a very best technique of producing products at rapid way using the concept Reuse of assets, artifacts, and components. Software Product Lines and Software Architectures combined to form a Product Line Architecture (PLA). Many researchers have work on PLA producing different views and experiences. Architectural smells a way of accessing the PLAs, means the identification of bad smells in context of architectures formed using product line. "Why these smells occur" is briefly described and also identification has also been shown using analytical and manual analysis. These smells can be remedied but the process is too large to follow, though small systems can be easily remedied or re-factored using various techniques. Architectural smells are remedied by changing the structure and the behaviors of the internal system elements without changing the external behavior of the system. This paper describes MobileMedia Feature Model creation, Component Model creation as PLA for the assessment of Bad smells.

Keywords: MobileMedia, SPLs, PLA, Bad smells, Feature Model

I. INTRODUCTION

In today's scenario, every organization is following the concept of shifting from single software product development to product lines which is a new trend in the future times. Software Product Line, family of different products which shares same set of core assets or it can be said, a product line consists of multiple systems, which have same architecture and share common core assets with variability among systems [16]. The concept of bad smells is often used to indicate properties that are not defective or unmanageable but introduces the negative effect in systems maintenance. Bad smells are categorized as Code smells and Architectural smells. Code smells are the smells relating the source code or restricted to the implementation level constructs such as classes, methods, parameters or statements. The term code smells was coined by Kent Beck and Architectural smells term was coined by Lippert and Roock [14]. Architectural smells refers to the architectural decisions that adversely or negatively impact system lifecycle properties that are extensibility, testability, reusability and understandability besides this it also affects quality properties such as reliability and performance. Maintainability aspects of PLAs are to be identified and treated whenever possible. To access PLAs, identification of bad smells is one way. Despite having the different methods of evaluation and evolution of SPLs, to the best not many studies has been discussed the issues relating to the smells in PLAs. PLA is a formulation of various products involvement into one system or architectures.

II. RELATED WORK

Researchers are continuously exploring the field with new emerging technologies, using various algorithms, different methods and metrics etc. Alike Product Line Engineering which is new way of reusing the products and this has leaded the researchers and even the market development organizations to achieve a brand new work experience. Architecture smells is a way through which systems can be accessed and evaluated. Many design issues has been founded in the single systems but the question is "Do Architectural smells occur in Software Product Line?" The term Bad smell will be considered in this context, because of budding research drift now days. It hasn't been under knowledge that why these smells occur in the Product Lines? Though less work has been done in this context but also it has not involved the tools for its analysis.

After having the keen observation of the Product Line Architectures, the main objective is to find out the new concepts that are the Identification of Bad smells in context of PLAs. The problem related to the occurrence and Identification of bad smells in context of PLAs has been done. It has been found that Bad smells relating to the code has been done in large extent but in case of Architectures it is recent. In summary, the study workflow will comprise of:

- A SPL sample MobileMedia has been created using tool will be analyzed.
- A Component Model is built in order to extract the conceptual or logical architecture from the code to know the connectivity and relation of various components under features in the architecture.
- Also Analysis has been done regarding the occurrence of smells in order to know "Why they

978-1-4799-5173-4/14/$31.00 ©2014 IEEE

occur in context of PLAs, what's the reason", that is at class level, in packages, in subsystems.

- And finally the Identification of bad smells in the recovered architectures created as Component Model that "how they are occurring".

III. MOBILEMEDIA FEATURE MODEL CREATION:

MobileMedia manipulates media on mobile phones that is music, photo and video, basically a family of multimedia management application for mobile devices and is used in research community of SPLs. In this paper, MobileMedia uses 31 features that are MobileSelection, MobileManagement, Gallery, Internet and Games, which further involve more features as shown in the Figure 4 [17]. Figure 1 represents a Feature Model diagram. A node represents a feature and edges represent dependency between the features. Feature-model diagram also represents constraints on the selected feature when product is build. A white circle (hollow) indicates the optional feature and the black circle (solid) indicates the mandatory feature.

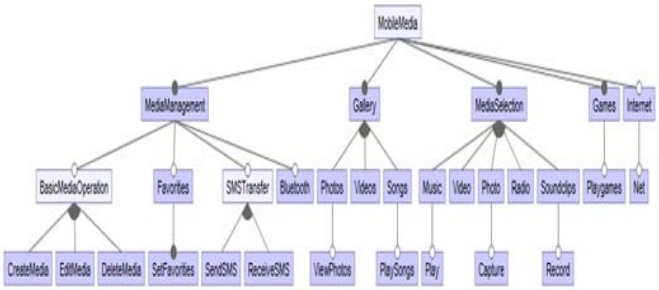

Figure 1: A Feature Model of MobileMedia

A. Creation of feature Model:

First of all a feature project is created by selecting FeatureIDE Project and choosing the composer as AHEAD. When FeatureIDE project will be created it will be displayed on the left side pane of window in the Package Explorer like shown in the figure and model with root and base will be displayed in the Feature Diagram IDE.

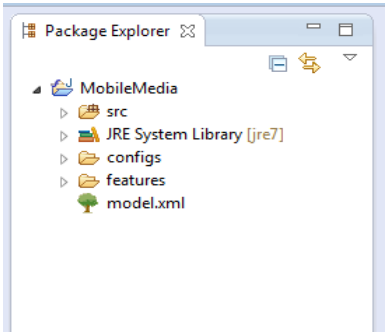

Figure 2: MobileMedia FeatureIDE Project

B. FeatureIDE statistics of the project:

This statistics depicts the composer or generation tool AHEAD (Algebraic Hierarchical Equations for Application Design) being used and the statistics involved under FeatureIDE. AHEAD composer supports the composition of Jak files where Jak extends the Java with Keywords for Feature-Oriented Programming. Number of Abstract feature, Concrete features, Compound features, Terminal features, Hidden features and the Constraints involved in it.

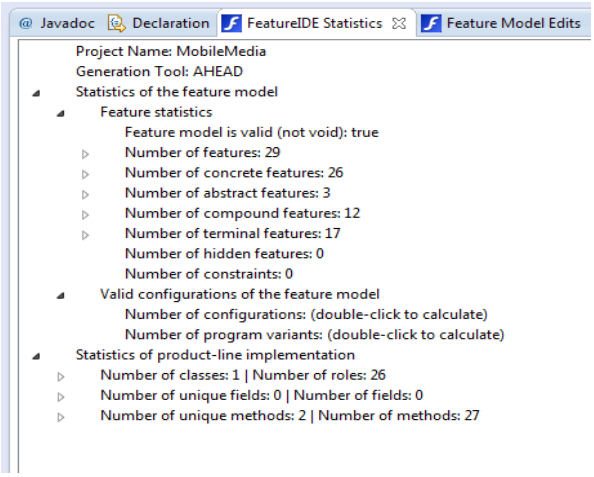

Figure 3: FeatureIDE Statistics

C. Configuration File Creation:

Configuration view displays the number of configurations the user wants to involve in its SPL feature model. It may the main single configuration involving many configurations.

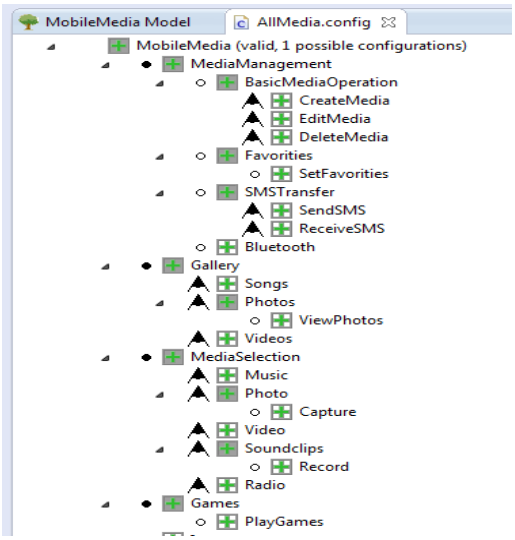

Figure 4: Configuration File

Here MobileMedia involves the main configuration and further its configuration that is feature under it which are refined by the main class involved in the MobileMedia. Refine is the keyword used in

AHEAD composer of FeatureIDE used to specify the refinements of an existing class. The following figure displays the Jak file creation of a single feature which uses the refine to use the main class.

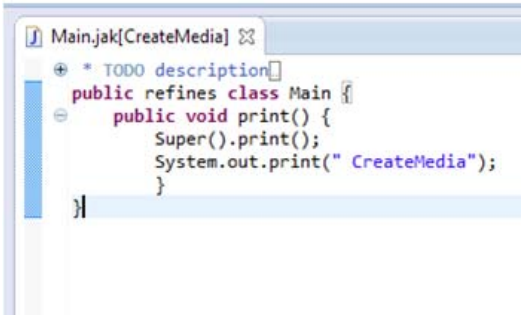

Figure 5: Source Code showing a how a class Main Jak is refined by another lower level class

IV. COMPONENT MODEL CREATION:

From the documentation and considering domain knowledge in order to evaluate the source code, manual analysis has been performed. It is a static implementation view of system i.e. organization of components of a particular moment. ArchStudio is being used for the development of the PLA of MobileMedia. It is an open-source software and systems architecture development environment of integrated tools for modelling, visualizing, analyzing and implementing software and systems architectures. Component Diagrams under ArchStudio which is in ADL form that is xADL based language graphical representation are used to model physical aspects of a system (like executables, libraries, files, documents etc) which resides in a node. It doesn't describe functionality of system but it describes components used to make those functionalities. From several visualizations for xADL models such as Archipelago, ArchStudio's graphical editor, ArchEdit provides visualizations as symbol graphs like box-and-arrow models; Archipelago is used for the creation of Component Model of MobileMedia. Figure 5 displays the MobileMedia SPL component model or can be described as MobileMedia PLA as defined from the software product lines. There are six components Selector, GUI, MobileMedia, Gallery, Manager and Create/Delete with In and Out interfaces doing or providing their own functioning to one another. Components provide accurate objects for each provided interface; provided/required each interface has a name, component is asked by the framework "give me the object that equivalents to this provided interface". The component model creation in Archipelago provides the xml file as xADL describing the functioning and linking of components to one another.

Manager Component is playing main role as acquiring or receiving inputs from Gallery component, selecting options from the Selector Component, content handling from the GUI Component, creation and deletion from the Create/Delete Manager Component and sending features functionalities and other operations from other components to the MobileMedia Component.

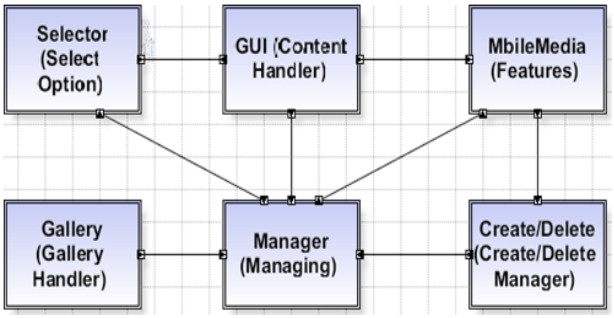

Figure 6: MobileMedia SPL Component Model

GUI Component is the content handler which is handling the whole graphical user interface functionalities of the components as the features functioning overall. MobileMedia Component as a main component is holding or receiving the features functioning from the Manager Component.

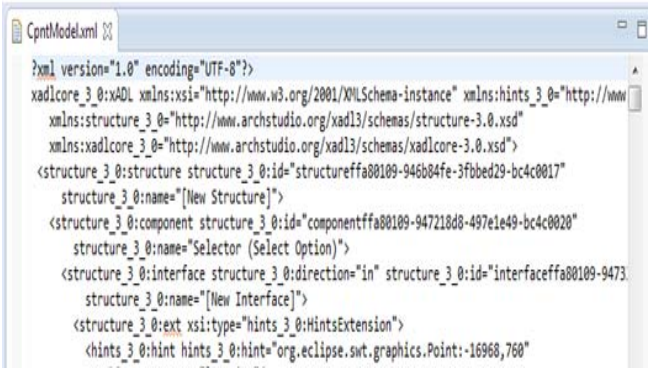

Figure 7: xADL file as .xml of the component model created

III. OCCURRENCES OF BAD SMELLS:

Bad smells as Architectural smells besides code smells can frequently be identified but these will demand large refactoring. The architectural smells don't always certainly indicate there is a problem but they end to places in the system's architecture that will need to be analyzed further. Architecture smells can occur at various levels as:

A. Class level:

Smells at this level refers to the fundamental relations between individual classes. The longer the cycles involved in the classes it will have much more stronger smells than the shorter ones and if they are sharing the same classes, then the situation will be more complicated and lead to unruly chaos. It may be because of Obsolete Classes, Static Cycles in classes, Subclasses do not redefine methods etc.

B. ***Package level:***
This smell can occur in and between packages, like in various programming languages concepts that is clubbing of related classes existing. A package can contain a number of classes and can be nested syntactically occurring because of unused packages, cycles between packages, too small or too large packages etc.

C. ***Subsystem:***
Smells occurs In and Between subsystems when there are large systems, when packages are bundled in so called subsystems. This smell may occur if there are no subsystems involved, if system is too large or small, or it is having too many subsystems, cycles in subsystems, overgeneralization involved etc.

D. Layers:
Clubbing of layers is subsystem often found into larger systems and they serve as structure of the system. Number of smells can emerge in as well as in between layers. These smells can occur if no layer exists, cycles between layers, strict layers violated, too many layers etc.

IV. IDENTIFICATION OF BAD SMELLS IN PLAs:

Bad smells or Bad architectural smells in the architecture occurs, when poor structure of architecture-level constructs (i.e. Components, Connectors and Interfaces) causes a reduction in the system maintainability properties. An automated analysis that is identification of bad smells in context of architectures or PLAs is performed using Structure101, a tool which takes source code of MobileMedia SPL as input, and produces the output as number of models after Lexical interpretation. Before the two products focuses on two separate problems, first hitting codebases into shape with the LSM in Restructure101, then keeping it in shape with Structure101 XS detection and Architecture diagrams. The concept of LSM is used as Restructure101 for the refactoring purpose i.e. simulating restructuring actions in the LSM. Once the code-base's physical structure reflects the modular hierarchy by using Restructure101 but using Structure101 it will keep by defining Architecture Diagrams that at the same time communicate the intended structure and define constraints on dependencies and visibility so that violations are flagged at compile and build time. Restructuring (in Restructure101) means simulating restructuring actions in the LSM until having model, exporting the action list to IDE, and doing the refactoring work. Results in the end are that the code-base has the same structure as the one modelled. Doing the work means implementing the new structure in some physical way, using packages, Maven projects, namespaces, assemblies, or whatever. Figure's below depicts the representations in a higher abstraction level, such as Decomposition View, LSM view, Dependency graph. The output provided by tool i.e. Decomposition view or model shows the hierarchy of the entities. The implementation of MobileMedia SPL is supported by the main entity that further calling other main procedures or main function i.e. print and the other one is int. MediaManagement entity performs the main functioning and provides the functionalities to Main and other entities are also interrelated with it

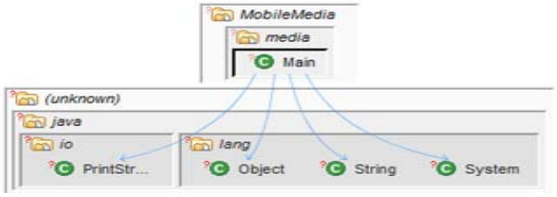

Figure 8: Decomposition view

Figure 9 is also the Decomposition view of the SPL; it is just more detailed view or model presenting the functionalities.

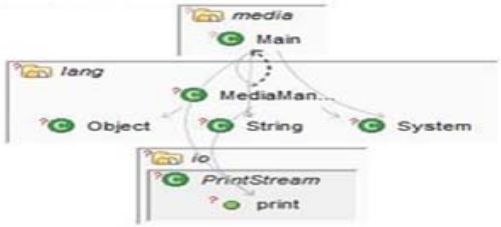

Figure 9: Decomposition View in detail

LSM (Levelized Structure Map), the visual model which can browse, filter and manipulate using mouse actions, using toolbar and context menu. Every item depends on at least one item on the level immediately below it items in the same row do not depend on each other, and items on the lowest level do not depend on any other items at the same scope as items in the LSM are levelized into rows, or levels. The LSM is always levelized so that an immediate and accurate representation when the code changes occurs or when applying restructuring actions. This arrangement conveys a lot of dependency information so that the specific item-to-item dependency arrows can be hidden without loss of context. The Figure displays the dependencies between the entities that are the packages and the classes involved in which are accessed from the MobileMedia SPL.

LSM in Figure 10 shows a class can't be moved into package, as a class is created under a package. It will help the user to understand the model of architecture that means it simulates restructuring actions in the LSM.User or Architecture can easily understand how the system is working and how much new alteration

is needed in the system, can be added like classes, packages or other modules.

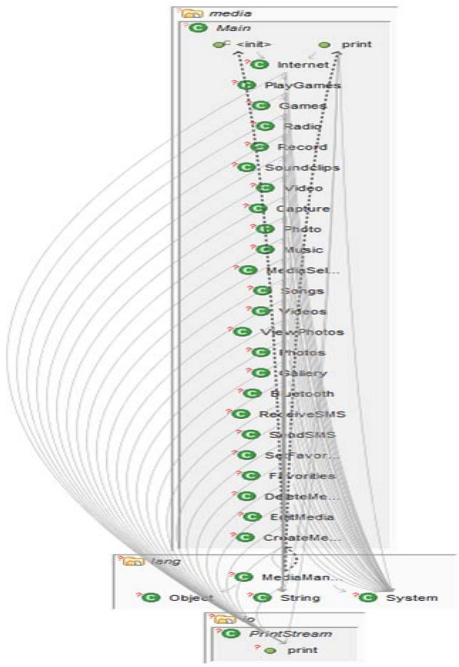

Figure 10: (Levelised Structure Map) of MobileMedia SPL showing physical organization

Now, when architecture is to be build from dependency graph, it's not possible because it only contains low-level code items (methods, fields, etc.) as shown in Figure 11.

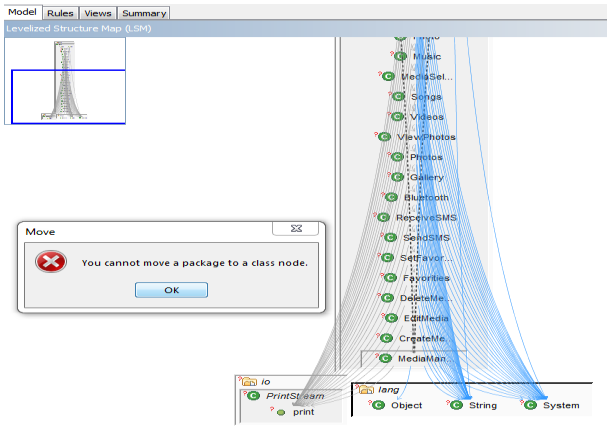

Figure 11: LSM showing package can't be moved to a class node

In Figure 12, an architecture model has been displayed behind the message box which is created under the Rules tab. The prompt message box is shown to the user if he links the cells in the same hierarchy.

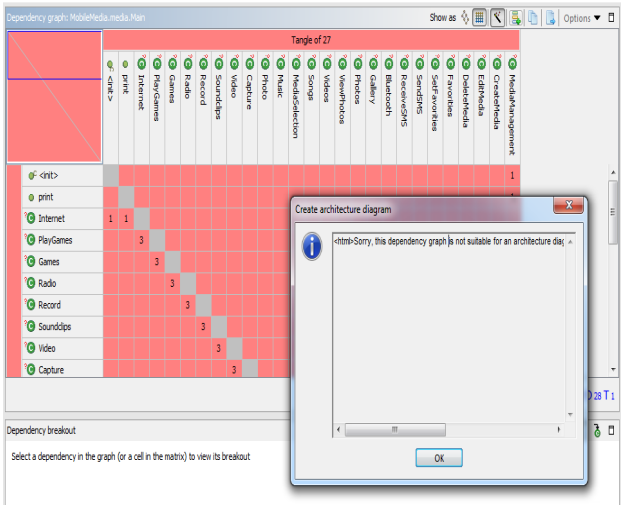

Figure 12: Dependency Graph is not suitable for architecture diagram

Types of Architectural smells found in context of MobileMedia PLA

A. Connector Envy:

Connector Envy, a smell occurs to that component which covers too much functionality with concern to connections. Various types of services involved as: Communication, conversion, Coordination and Facilitation. As mentioned previously, the Manager component contains the implementation of functionalities, as the MobileMedia component triggers what is displayed.

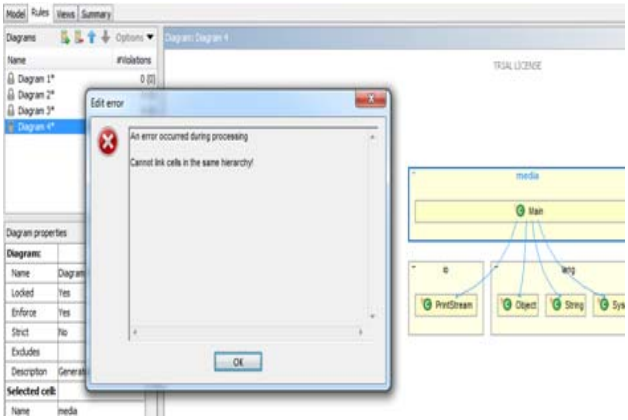

Figure 13: Architecture displays cells in the same hierarchy cannot be linked

Connector Envy smell is identified in the MobileMedia component, due to its extensive coordination and communication links with the other components. MobileMedia contains an operation that transfers control set by the provided events to the Manager component through several conditional statements. So the direct method invocation represents a negative effect to the

component's reusability in that the dependability of these components is increased.

The MobileMedia Component imports several libraries that are used to implement the graphical interface of generated products and implements a listener of events, which are sent through GUI interactions. Furthermore, the Manager component is presenting high communication and facilitation services by interacting with every other component in the system. To improve the component's maintainability, it is possible to designate the connection attributes to a specialized entity, but in this case the performance of operations could be affected since all features are concentrated within the functional part of the component. Product construction capabilities and connector responsibilities when combined represents a reduction in testability because application interaction functionalities can't be tested separately.

B. Scattered parasitic Functionality

Scattered Parasitic Functionality smell is categorized as existence of high-level concern that realizes across multiple components dealing with modifiability as at least one component addresses multiple concerns making a bottleneck. In a scattered concern, if all components are composed of implementation then the resulting component would realize orthogonal functionalities. The GUI implements the triggered GUI attributes in Manager and supports construction of visual components arranged by MobileMedia as referring the java native library to handle interfaces. It doesn't representing a high-level concern; changes in such related components would not represent impact widely. In SPLs, scattered parasitic functionalities represent aspects that are difficult to realize, giving the widespread effect among products.

C. Ambiguous Interfaces

Ambiguous Interface smell occurs when a component is offering only a single and general entry-point such as interface referred as ambiguous or uncertain. Despite the fact, the component offering and processing multiple services, an ambiguous interface will provide only one public method. This smell reduces understandability as ambiguous interface doesn't disclose what services a component is offering. Before using the services to the particular component, the user of that component needs to examine the implementation of internals. Through one general action listener, all the features are implemented in the component and accessed. This smell is common in SPLE due to natural characteristics. A generic entry-point could be overloaded by functionalities that are different, thus require very different handlers for effective management.

D. Extraneous Adjacent Connector

When two connectors of different types are used to link a pair of components, this smell occurs. Using the concept of procedure calls makes the control transfer explicit therefore maximizing the understandability. Basically, senders and receivers of events are unaware of each other as how event connectors can be easily replaced or updated, thus leading to the increased adaptability and reusability. Understandability is also affected when two components are connected with common connector type because it is difficult to understand under which situations additional communication is occurring between the affected components.

Manager Component in SPL used is providing functionality services to construct the MobileMedia instance (product). The occurrence of smell would be represented by procedure call due to the use of event listeners by triggering the feature execution. However, the only interaction between MobileMedia and Manager Components is through the events provided by the GUI.

E. Feature Concentration

Feature concentration smells is specific to the SPL context and centralizing the characterization of the SPL features in one architectural entity. Relating to the Scattered Parasitic Functionality and Ambiguous Interfaces smells in different functionalities has been implemented in a single construct which might offering one general entry point. Coupling and Cohesion are the design decisions driven as metrics, between different entities. Making the similar design decisions in a SPL with several hundreds of features would represent drastic impacts on this PLA understandability and maintainability capabilities throughout the lifecycle.

The identification of smells is relevant because points of improvement can be identified and refactor in the architecture level regardless of changes in scope of SPL [1]. Pointing the identification of smells is totally dependent on the architecture design specification. It is a difficult task to recover design decisions which influences the detection of properties that can be remedied.

V. CONCLUSION

Code smells helps developers to locate when and where source code needs to be refactor. Similarly, Architectural smells helps architects when and where to refactor them. Smells in context of SPLs is a fresh topic of discussion. Therefore, the overall discussion has been done on occurrence and identification of architectural smells in context of SPL and results has been produced. As a future work, interesting discussion would be how to effectively remedy the occurrence of smells either through specific manual operations or using tools to

support corrections. Though various tools has been used in this dissertation work for the identification of smells but as a current topic in the Software Product Line Engineering, less work has been done in context of identification of smells in SPLs or it can be said PLAs.

REFERENCES

[1] Hugo Sica de Andrade (2013) *"Software Product Line Architectures: Reviewing the Literature and Identifying Bad Smells"* MASTER THESIS SOFTWARE ENGINEERING ADVANCED LEVEL 30 HP, (pp. 1-99)

[2] Rick Flores, Charles Krueger, Paul Clements, (2012) *"Mega-Scale Product Line Engineering at General Motors"*, Software Product Line Conference Best Paper Award in Industrial Track), BigLever Software, (pp. 259-268).

[3] Ganesh B. Regulwar, Raju M. Tugnayat, (2012) *"Bad Smelling Concept in Software Refactoring*", ICMTS, (pp. 56-61).

[4] Sadiq1 Mohd., Rahman2 Abdul, Mohammad Asim2, Shabbir Ahmad3, Ahmad1Javed (2010) *"esrcTool: A Tool to Estimate the Software Risk and Cost"*, Second International Conference on Computer Research and Development, IEEE, (pp.886-890).

[5] Ankit Chaudhary, Basant K Verma, (Member IEEE), Jagdish L. Raheja, (2010*) "Product Line Development Architectural Model"*, IEEE, (pp. 749-753).

[6] Joshua Garcia, Daniel Popescu., George Edwards and Nenad Medvidovic, (2009) *"Identifying Architectural Bad Smells"* 13th European Conference on Software Maintenance and Reengineering, IEEE, (pp. 255 – 258).

[7] Joshua Garcia., Daniel Popescu, George Edwards, and Nenad Medvidovic (2009) *"Toward a Catalogue of Architectural Bad Smells"*, QoSA '09 Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems, IEEE (pp. 146–162).

[8] Sofia Azevedo, Ricardo J. Machado, Dirk Muthig, Hugo Ribeiro (2009)" *Refinement of Software Product Line Architectures through Recursive Modeling Techniques"*, Confederated International Workshops and Posters, ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK, (pp 411-422).

[9] Christian. Kastner, Thomas Thum, Gunter Saak, Janet Feigenspan,. Thomas Leich, Fabian Wielgorz, Sven Apel, (2009) *"FeatureIDE: A Tool Framework for Feature-Oriented Software Development"*, Software Engineering, 2009. ICSE, IEEE 31st International Conference, (pp. 611 – 614).

[10] Hojin Cho, Jin-Seok Yang (2008) *"Architecture Patterns for Mobile Games Product",* Advanced Communication Technology, ICACT, 10th International Conference, (pp.118-122).

[11] Joachim Bayer, Oliver Flege, Cristina Gacek (2007) "*Creating Product Line Architectures*", International Workshop IW-SAPF-3, (pp 210-216).

[12] María Cecilia Bastarrica, Nancy Hitschfeld-Kahler, Pedro O. Rossel (2006), *"Product Line Architecture for a Family of Meshing Tools",* 9th International Conference on Software Reuse, ICSR, (pp. 403-406),

[13] Klaus Pohl, Günter Böckle, Frank van der Linden (2005) "*Software Product Line Engineering"* Foundations, Principles, and Techniques, Springer

[14] Martin Lippert, and Stefan Roock. (2004) "*Refactoring in Large Software Projects (How to Successfully Execute Complex Restructurings)",* (pp. 31-84).

[15] Colin Atkinson, Joachim Bayer and Dirk Muthig, (2000) *"Component-Based Product Line Development: The KobrA Approach",* SPLC, (pp. 289-310)

[16] Manjinder Kaur, Parveen Kumar (2014) "*Systematic Review on Software Product Line Engineering (SPLE)",* International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 1.

[17] Manjinder Kaur, Parveen Kumar (2014) *"MobileMedia SPL creation by FeatureIDE using FODA",* Global Journal of Computer Science and Technology, Volume 14, Issue 3.