International Journal of Engineering and Science Invention (IJESI)
ISSN (Online): 2319-6734, ISSN (Print): 2319-6726
www.ijesi.org ||Volume 12 Issue 6 June 2023 || PP 57-63

Enhancement of software quality by the use of various software artefacts to remove code smells.

Indu[1] and Nitika[2]
[1]M.Tech. Scholar, CBS Group of Institutions, Jhajjar, Haryana, India
[2]Assistant Professor, CBS Group of Institutions, Jhajjar, Haryana, India
indudahiya111@gmail.com[1], nitika23@gmail.com[2]

ABSTRACT
Software Product Lines (SPL) are recognized as a successful approach to reuse in softwaredevelopment. Its purpose is to reduce production costs. This approach allows products to bedifferent with respect of particular characteristics and constraints in order to cover differentmarkets.SoftwareProductLine engineering istheproductionprocessin product lines. Itexploits the commonalities betweensoftwareproducts, but alsotopreserve theability to varythe functionality between these products. Sometimes, an inappropriate implementation of SPLduring this process can conduct to code smells or code anomalies. Code smells are consideredas problems in source code which can have an impact on the quality of the derived products ofanSPL.Thesameproblem canbepresentinmanyderivedproducts fromanSPLduetoreuse.A possible solution to this problem can be the refactoring which can improve the internalstructure of source code without altering external behavior. This paper proposes an approachfor building SPL from source code. Its purpose is to reduce code smells in the obtained SPLusing refactoring source code. Another part of the approach consists on obtained SPL's designbasedonreverseengineering.

KEYWORDS
Software ProductLine,Codesmells,Refactoring, Reverse Engineering.

I. INTRODUCTION

Software Product Line (SPL) is a family of related software systems with common and variablefunctions whose first goal is reusability [1]. The SPL approach intends at upgrading softwareproductivity and quality by relying on the similarity that exists among software systems, and bymanaging a family of software systems in a reuse-based way. SPL aims to minimize effort andcost of development and maintenance, to reduce time-to-market and to ameliorate quality ofsoftware [2], [3], [4]. Unsuitable development of a SPLs may give rise to bad programmingpractices,calledcodeanomalies,alsoreferredintheliteratureas"codesmells"[5].

Code smell is often considered as key indicator of something wrong in the system code [5] orundesired code source property.Like all software systems,artifacts of aSPLmay containsseveral code anomalies [6]. Therefore, if these code smells are not systematically removed, theSPL's quality may degrade due to evolution. Code Smells are very-known in classic and singlesoftware systems [7]. However, in the context of SPL, Code Smell is a young topic. [8] proposeda specific SPL's smell, called "Variability Smells". [9] discussed two types of bad smells relatedonSPL:ArchitecturalBadSmellsandCodeBadSmells.[6]and[10]proposeddetectionstrategiesforanomaliesin SPL.

The main goal of this work is to propose a solution to reduce code smells in SPL. Unsuitabledevelopment of a SPLs may give rise to bad practices such as architectural smells and codesmells. Our work tries to reduce development problems through the source code analyze ofproduct variants to detect and correct code smells, identify the variability and build the variabilitymodel of SPL. Detecting and refactoring code anomalies in source code from the start give us achance to develop a SPL with a high quality. Thus, the reverse engineering is a preliminarystrategyfora cleanSPLandtoobtainthe variabilitymodelof SPL.

This paper is organized as follow. Section 2 provides background on code smells, SPL andreverse engineering. Section 3 presents the related work. Section 4 shows the proposed approach.Thelastsectionconcludesandpresentsfuturework.

II. BACKGROUND

2.1 Software Product Lines

The evolution of software development and the growth of product numbers have motivated theemergence of many reuse concepts. Software development communities recognize SPL as asuccessful approach for reuse [11], [12]. This success results from the reduction of productioncosts and time to market. SPL is a software development paradigm that share common feature tosatisfythespecificneedsof particularmarketsegment[13].

Softwareproductline'sapproachfocusonthesharingofareferencearchitecturebetweenproducts.These productscandifferandthe approachallowsthisvariationwithrespectofparticularcharacteristics and constraints. This difference is the variability present in SPL, whichis the ability of a core asset to adapt to usages in the different product contexts that are within theproduct line scope [14]. Variability must be anticipated and continuously maintained to obtainwished results. The production process of product lines is well known as software product lineengineering (SPLE) which tries to maximize the commonalities and reduce the cost of variations[15]. The SPLE process focuses on two levels of engineering [14]: Domain Engineering (DE) andApplication Engineering (AE). DE focuses on developing reusable artifacts which are used in AEtoconstructa specificproduct.Fig.1presentstheSPLEprocess.

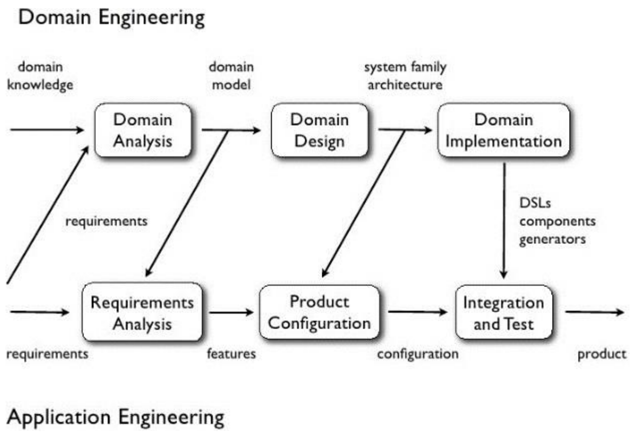

Figure1. DomainEngineeringandApplicationEngineering[14]

2.2 Codesmells

A software system evolves over time. Its evolution is one of the critical phases of the process ofits development. Moreover, the software system changes, moreover the structure of the programdeteriorates. So, complexity increases until it becomes more profitable to rewrite it from thescratch.Whichcaninvolvethreatsonthesoftware quality.

Software system's bad quality is a key indicator of existing bad programming practices, alsoknownintheliteratureassourcecodeflaw,codesmellsorcodeanomalies[5].

Codesmellsareusuallysymptomsoflow-levelproblemssuchasanti-patterns.Theyareindicators of something wrong that structures in the source code [5], their presence can affect inmaintenanceandslowdownsoftwaredevelopment.

In literature, different Code Smells have been defined. For instance, in Fowler's book [5], Beckdefine a list of 22 code smells, for example "Long Method" is a method that is too long and hastoo many responsibilities, so it makes code hard to maintain, understand, change, extend, debugand reuse. "Large Class" is a class contains many fields, methods or lines of code, means that aclass is trying to do too much."Duplicated Code" has negative impacts on software developmentand maintenance. For example, they increase bug occurrences: if an instance of duplicate code ischanged in one part of the code for fixing bugs or adding new features, code may require variouschanges in other parts all over the source code simultaneously; if the correspondents are notchangedinadvertently,bugsarenewlyintroducedtothem[16].

2.3 Reverse Engineering

Reverse Engineering is the process of analyzing a system. The purpose is to identify systemstructure,itscomponentsandtherelationshipsbetweenthem[17].

Reverse Engineering can create representations of the system through transformations between orwithin different abstraction levels. It can also extract design information from source code [17]andmaybeusedtore-implementthesystem.

The reverse engineering process can be done through automated analysis or manual annotations.Thenextstepsconcerntheidentificationofprogramstructureandtheestablishmentoftraceabilitymatrix.

2.4 Refactoring

Refactoring's purpose is to improve the quality of an existing code [5]. This process tries throughthe software system changing to improve its internal structure without having an impact on theexternalbehaviorof thecode.

Refactoring can be a solution for code smells. This process takes as input a source code withproblems and outputs good ones. The resulting code can be reused. The refactoring allows thecode smells identification. Also, it offers the possibility to change the original code containingthesecodesmellsbycoderestructurationtogetanoutputcode withoutcodesmells.

III. RELATED WORK

Common industrial practices lead to the development of similar software products, then they areusually managed to each other using simple techniques, e.g., copy-paste-modify. This is badpractice leading a low software quality, as we mentioned above the "Duplicated Code" codesmell. During the past few years, several studies have investigated two things: how to detect codesmells[18],[19],[20],[21],[22],[23]andhowtocorrect[5],[18],[24]theminasingle software. To the best of our knowledge we found few studies [6], [8], [9], [10], [25], [26] that canbeconsideredrelatedtoourresearch.

[9]performed aSystematicLiterature Review (SLR)to find and classify publishedwork aboutbadsmellsinthecontextofSPLandtheirrespectiverefactoringmethods.Theyclassified70differentbadsmellsdividedinthreegroups:(i)CodeSmells;thataresymptomsofsomethingwronginthesourcecode,(ii)ArchitecturalSmells;thatareanindicationofprobleminhigherlevelsofabstractionand(iii)hybridSmells;thatareacombinationbetweenarchitecturalsmelland code smells. [26] proposed a method to derive metric thresholds for software product lines.Thegoalistodefinethresholdsvaluesthateachmetriccantakeinordertoidentifypotentialproblemsintheimplementationoffeatures.Theyuse4softwaremetrics:LinesofCode(LOC)countsthenumberofuncommented linesofcodeperclass.Thevalueofthismetricindicatesthesizeofaclass.CouplingbetweenObjects(CBO)countsthenumberofclassescalledbyagivenclass.CBOmeasuresthedegreeofcouplingamongclasses.WeightMethodperClass(WMC)countsthenumberofmethodsinaclass.Thismetriccanbeusedtoestimatethecomplexityofaclass.NumberofConstant Refinements(NCR)countsthenumberofrefinementsthataconstanthas.Its valueindicateshowcomplextherelationshipbetweenaconstantanditsfeaturesis.Theirstudyisbasedon33SPLswhicharedividedintothreebenchmarksaccordingtotheirsizeintermsof Linesof Code(LOC).

Benchmark 1 includes all 33 SPLs. Benchmark 2 includes 22 SPLs with more than 300 LOC.Finally, Benchmark 3 is composed of 14 SPLs with more than 1,000 LOC. The goal of creatingthree different benchmarks is to analyze the results with varying levels of thresholds. In term ofthat they illustrate a detection strategy to detect two types of code smells, God Class and LazyClass.Figure2presentsthewaytoidentifyGodClassandLazyClass.

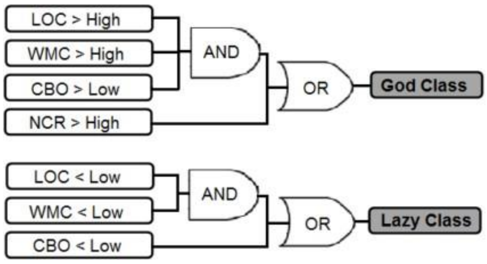

Figure2.CodeSmellsidentification.

Apel et al. [8] proposed bad smell specific to SPLs called variability smell; that is an indicator ofanexistingundesiredpropertyinallkindsof artifactsinanSPL,suchasfeaturemodels.

Fernandes and Figueiredo [6] investigated code anomalies in the context of SPLs, they proposenew detection strategies for well-known anomalies in SPL such as God Class and God Method,ultimately they propose new anomalies and their detection strategies and they propose supportingtoolfortheproposeddetection.

De Andrade et al. [25] conducted an exploratory study that aims at characterizing architecturalsmellsinthecontextofsoftwareproductline.

Abilio et al. [10] proposed means to detect three code smells (God Method, God Class, andShotgun Surgery) in Feature-Oriented Programming source code, FOP is a specific technique todeal with the modularization of features in SPL. They performed an exploratory study with eightSPLs developed with AHEAD; which is an FOP language, to detect code smells in a SPL byusing 16 source code metrics. These metrics corresponds to the detection of three code smellsmentionedabove.Table1presentssome ofthese metrics.

Table1.Metrics usedtodetectcodesmells [10]

Acronym	Name	Description
NOF	Number of Features	Number of Features which has code artifacts
NCR	Number of Constant Refinements	Number of refinements which a constant has
NMR	Number of Method Refinements	Number of refinements which a method has
TNCt	Total Number of Constants	Number of constants (classes, interfaces - constant)
TNR	Total Number of Refinements	Total of refinements (classes, interfaces - refinement)
TNMR	Total Number of Method Refinements	Total of refinements of a method
TNRC	Total Number of Refined Constants	Total of refined constants
TNRM	Total Number of Refined Methods	Total of refined methods

Considering the discussed related work, we propose an approach aiming to develop an SPL withminimalcodesmellsrisks.

IV. PROPOSEDAPPROACH

The main goals in our study are to (i) investigate the state of the art on code smells in the contextof SPLs as we show above, (ii) propose a solution to decrease code smells in developing softwareproductlines.

Unsuitable development of a SPLs may give rise to bad practices such as architectural smells andcode smells that induce maintenance and development costs problems. Therefore, we propose tobuild an SPL from the scratch using reverse engineering methods, which can help us to detect andcorrectcodesmellsfromthestart.Thus,wecanguarantee greatqualityof SPL.

The main challenge in this task is to analyze the source code of product variants in order to (i)detect and correct code smells,(ii) identify the variability among the products,(iii) associatethem with features and (iiii) regroup the features into a variability model. The proposed approachisobject-orientedlanguageandonlyusesasinputthesourcecodeofproductvariants.

First of all, we use as input source code of product variants then we apply detection strategies forcode anomalies as duplicated code, uncovered code by unit tests and too complex code, after thatwe correct them using an automated bad smell correction technique based on the generation ofrefactoring concepts. Refactoring is a change made to the internal structure of software to rewritethe code, to “clean it up”, to make it easier to understand and cheaper to modify without changingits observable behavior [27].In step 2 and after having a clean code, we are interested in thedetermination of the semantic relations between the names of the classes, the names of themethods and the attributes of all the source codesof the existing products having differentterminologies and not necessary having the same meaning. In term of that we are interested in theharmonization of names, and more particularly in unifying fragments of source codes. Duringunification, we determine the semantic correspondences between the source code elements basedonsemanticknowledgebaseYAGO[28].

YAGO is a semantic knowledge base derived from many data sources like Wikipedia, WordNet,WikiData, GeoNames, and other. Aside YAGO, we will base on Machine Learning methods toget better semantic correspondences between source code elements. In fact, Machine Learningalgorithms can be helpful in the classification of the features. Machine Learning proved hisefficiency in many complex domains like Predictive Analytics [29], image processing [30], andsignal processing… At the end of this step, all names with a semantic relationship would beharmonizedandcanbe furtheranalyzedinthe nextstepofidentifyingcommonalitiesandvariability.Thus,we extractfeaturesby identification of common block (CB) and variationblocks (VB). CB groups the elements present in all the products while VB groups the elementspresent in certain products and not all of them. The role of these blocks is to group subsets toimplementfeatures.Oncethecommonblockandthevariationblocksarecompleted,theextraction of mandatory elements and variation atomic blocks is supported, we associate them tofeatures.Once the common propertiesand variability of productvariants are identified,thefeature model(s) will be constructed.Consequently, we can obtain one ormore than one SPL.OurapproachispresentedinFigure3.

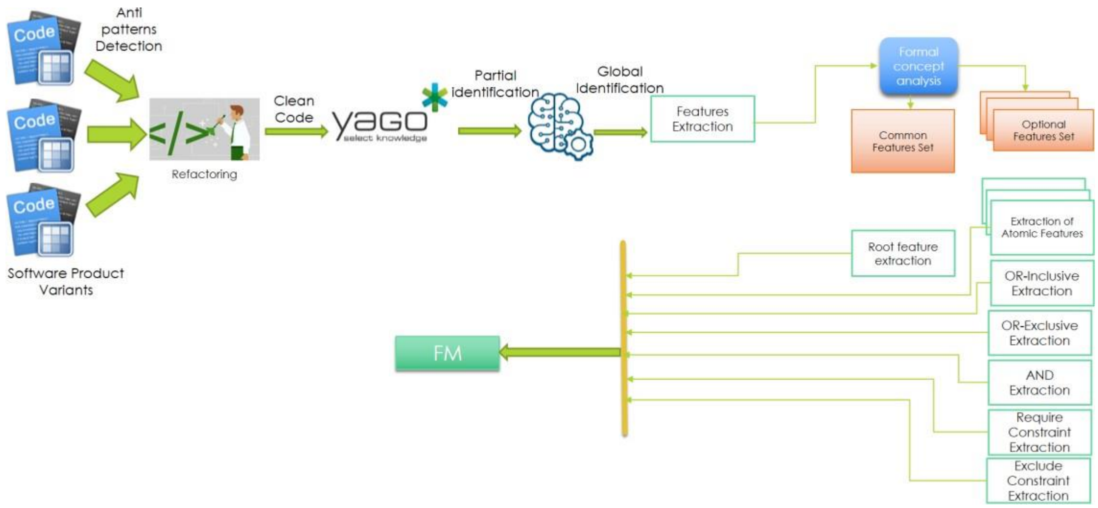

Figure3.ProposedApproach.

V. CONCLUSIONS

Softwarereuseisanimportantchallengeinsoftwareengineering.SoftwareProductLineisoneof the technique used to ensure the success of this challenge. The obtained products can containreused parts or components. These parts can include some problems in their source code moreknownasCodeSmells.Theseproblemscanpropagate betweenthedifferentproducts.

A solution to avoid the Code smells in source code, is refactoring which can improve the internalstructure of software system by trying to find the problem and avoid it using some restructurationtechniques.

In this paper, we try to present an approach which combines refactoring to eliminate code smellsand reverse engineering to propagate modifications to the design level. Our purpose is to obtain asoftwareproductline modelfreefromcodesmells.

Our future works will be the refinement of the different parts of the approach. Also, we willchoosetheappropriatetoolstouseinourprototype.

REFERENCES

[1]. Lee, S.hyun. & Kim Mi Na, (2008) "This is my paper", ABC *Transactions on ECE*, Vol. 10, No. 5,pp120-122.
[2]. Gizem, Aksahya & Ayese, Ozcan(2009)*Coomunications & Networks*,Network Books,ABCPublishers.
[3]. Rincón, L. F. et al., 2014. An ontological rule-based approach for analyzing dead and false optionalfeatures in feature models. In Electronic Notes in Theoretical Computer Sciences, Vol. 302, pp 111–132.
[4]. Jacobson, I. et al., 1992. Object-oriented software engineering: a use case driven approach. Addison-Wesley,USA.
[5]. Xue, Y., 2011. Reengineering legacy software products into software product line based on automaticvariability analysis. Proceedings of the 33rd International Conference on Software Engineering, NewYork,USA,pp. 1114–1117.
[6]. Ouali, S. et al., 2012. From Intentions to Software Design using an Intentional Software Product LineMeta-Model.Proceedingofthe8thInternationalConferenceonInnovationsinInformationTechnology,AlAin, UAE.
[7]. Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston,MA,USA.
[8]. Fernandes, E. and Figueiredo, E., 2017. Detecting Code Anomalies in Software Product Lines".Proceedings of 7th Brazilian Conference on Software: Theory and Practice, Maringa, Brazil, pp. 49–55.
[9]. Zhang, M. et al., 2011. Code Bad Smells: A Review of Current Knowledge. In Journal of SoftwareMaintenanceandEvolution:ResearchandPractice,WileyOnlineLibrary,pp. 179-202.
[10]. Apel,S.etal.,2013.Feature-OrientedSoftwareProductLines:ConceptsandImplementation.Springer.
[11]. Vale, G. et al., 2014. Bad Smells in Software ProductLines:A Systematic Review.Proceedings ofthe 8th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), Brazil,pp. 84-94.
[12]. Abilio, R. et al., 2015. Detecting Code Smells in Software Product Lines-An Exploratory Study.Proceeding of the 12th International Conference on Information Technology - New Generations, pp.433–438.
[13]. Ouali, S. et al., 2011. Framework for evolving software product line. In International Journal ofSoftwareEngineering&Applications,Vol.2, No.2,pp. 34-51.
[14]. Weiss, D. M. and LAI, C. T. R., 1999. Software Product-Line Engineering: A Family-Based SoftwareDevelopmentProcess.Addison-Wesley.
[15]. Pohl,K.andMetzger,A.,2006.SoftwareProductLinetesting.InCommunicationoftheACM,pp78-81.
[16]. Czarnecki,K.andEisenecker,W.,2000.GenerativeProgramming:Methods,Tools,andApplications.Addison-Wesley.

[17]. Thiel,S.andHein,A.,2002.ModelingandUsingProductLineVariabilityinAutomotiveSystems.InIEEESoftwareVol.19, No.4, pp. 66-72.
[18]. Hotta,K.etal.,2012.AnEmpiricalStudyontheImpactofDuplicateCode.AdvancesinSoftwareEngineering.
[19]. Chikofsky,E.J.andCross,J.H.,1990.Reverseengineeringanddesignrecovery:Ataxonomy.INIEEESoftware,Vol.7, pp.13–17.
[20]. Moha, N. et al., 2010. DECOR: A Method for the Specification and Detection of Code and DesignSmells.Transactions onSoftwareEngineering,Vol.36,No.1, pp.20-36.
[21]. Sjoberg, D. et al., 2013. Quantifying the effect of code smells on maintenance effort. SoftwareEngineeringIEEETransactions, Vol. 39,No.8, pp. 1144–1156.
[22]. Van Emden, E. and Moonen, L., 2002. Java quality assurance by detecting code smells. ProceedingWorkingConf.ReverseEngineering,IEEEComputerSocietyPress,pp.97—107.
[23]. Marinescu, C. et al., 2005. Iplasma: An integrated platform for quality assessment of object-orienteddesign. Proceedings of the 21st IEEE International Conference on Software Maintenance, Budapest,Hungary.
[24]. Liu, X. and Zhang, C., 2016. DT: a detection tool to automatically detect code smell in softwareproject.Proceedingsofthe4th Int.Conf.Mach.Mater.Inf.Technol.Appl.,vol.71,pp.681–684.
[1] Fontana,F.etal.,2012.AutomaticDetectionofBadSmellsinCode:AnExperimentalAssessment.InJournalofObjectTechnology.
[2] Campbell,D. and Miller, M., 2008. Designing refactoring tools fordevelopers. Proceedings of the2ndWorkshoponRefactoringTools,NewYork, NY,USA.
[3] De Andrade, H. S. et al., 2014. Architectural bad smells in software product lines. Proceedings of the1stInternationalConferenceDependableSecur.CloudComput.Archit.,pp.1–6.
[4] Vale, G. and Figueiredo, E., 2015. A Method to Derive Metric Thresholds forSoftware ProductLines.Proceedings29thBrazilianSymposiumonSoftwareEngineering, pp.110–119.
[5] Regulwar,G.B.andTugnayat,R.M.,2012.BadSmellingConceptinSoftwareRefactoring.InternationalProceedingsofEconomicsDevelopmentandResearch,Vol.45,pp. 56–61.
[6] Rebele, T. et al., 2016. YAGO: a Multilingual Knowledge Base from Wikipedia, Wordnet, andGeonames.Proceedingofthe15th InternationalSemanticWebConference,Kobe,Japan.
[7] Demsar, J. et al., 2004. Orange: From experimental machine learning to interactive data mining.Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery inDatabases,Pisa, Italy.
[8] Ebrahimi,K.S.etal.,2013.Combiningmodalityspecificdeepneuralnetworksforemotionrecognitioninvideo.Proceedings of the 15thACM onInternationalconference onmultimodalinteraction,Sydney,Australia.