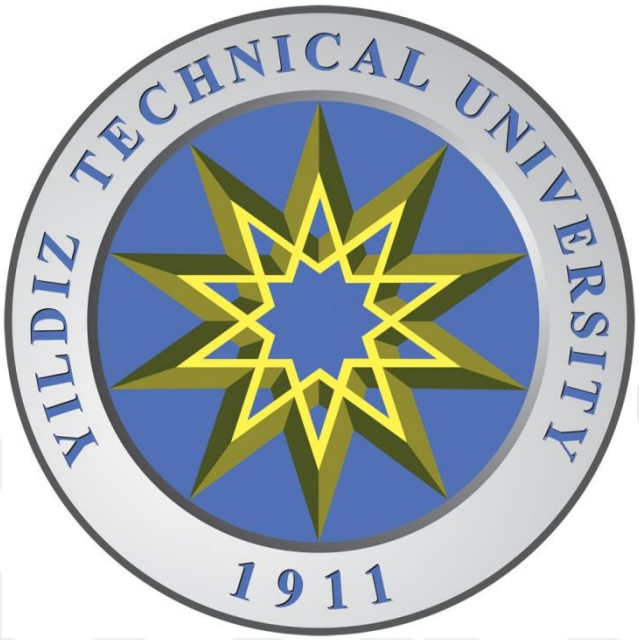

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MASTER THESIS

DECEMBER, 2017

REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

AUTOMATIC DETECTION OF CODE CAUSING NEGATIVE EFFECTS ON SOFTWARE QUALITY

BAYDAA M. MERZAH

MSc. THESIS
DEPARTMENT OF COMPUTER ENGINEERING
PROGRAM OF COMPUTER ENGINEERING

ADVISER
ASSIST.PROF. DR. YUNUS EMRE SELCUK

ISTANBUL, 2017

REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

AUTOMATIC DETECTION OF CODE CAUSING NEGATIVE EFFECTS ON SOFTWARE QUALITY

A thesis submitted by Baydaa M. MERZAH in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** is approved by the committee on 01.12.2017 in Department of Computer Engineering.

Thesis Adviser

Assist.Prof.Dr. Yunus SELCUK

Yıldız Technical University

Approved By the Examining Committee

Assist.Prof.Dr. Yunus SELCUK

Yıldız Technical University _____

Assist.Prof.Dr. Mehmet S. AKTAŞ, Member

Yıldız Technical University _____

Prof.Dr. Selim AKYOKUŞ, Member

İstanbul University _____

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Assistant Professor Yunus Emre SELCUK for his support and encouragement during this research. This thesis would not have been completed without his invaluable guidance and help.

I appreciate the love, caring and support of my parents for their unlimited support and their prayer for me to achieve my goals in the life.

My husband Ammar deserves special thanks for his sensible encouragements, precious support and endless love. He supported me in all stages of my research and my life.

I would like to thank my brother and sisters all my family members for supporting me spiritually throughout writing this thesis and my life in general.

Last but not the least; I take this opportunity to express gratitude to Al-Nahrain University specially my college for their financial support.

December, 2017

Baydaa M. MERZAH

TABLE OF CONTENTS

LIST OF SYMBOLS

$\cap$ Intersection
μ Mean
σ Standard deviation
$\cup$ Union

LIST OF ABBREVIATIONS

ACCO	Average Cyclomatic Complexity of Overridden Methods
ASM	Average Similarity between Methods
ATFD	Access to Foreign Data Metric
BOvR	Base class Overriding Ratio
C	Complexity of the program
CC	Cyclomatic Complexity
E	Number of edges (transfer of control)
fan-in	Local information flow input
fan-out	Local information flow output
FDP	Foreign Data Provider
GoF	Gang of Four
i	intersection
IV	Instance Variables
L	Length of the procedure
LAA	Locality of Attribute Access
N	Number of nodes
P	Number of disconnected parts of the flow
RS	Response set
SIV	Set of Instance Variables

LIST OF FIGURES

Page

LIST OF TABLES

Page

ABSTRACT

AUTOMATIC DETECTION OF CODE CAUSING NEGATIVE EFFECTS ON SOFTWARE QUALITY

Baydaa M. MERZAH

Department of Computer Engineering

MSc. Thesis

Adviser: Assist.Prof. Dr. Yunus SELCUK

The quality of IT software systems outlined by how well it has been designed from the internal and external points of view, also by how well the prerequisites have been met . Ease of developing and maintenance are the targeted factors that ensure successful completion and continuous use. From Steve McCall's perspective[1], software quality characteristics can be categorized for: External and Internal factors .The external quality factors such as reliability, correctness, accuracy, reusability and integrity concern the end user. The external software quality isn't mentioned in this study. The most reveal factors of internal software quality are maintainability, re-usability, flexibility and testability. These factors concern the developers and they ease development and maintenance also.

However, they are affected negativly by bad coding styles in the source code that is known as Code Smells. One of the important princepals in object orienred programming is to build classes with high cohesion and losely coupled. This principle can be violated by one type of code smells known as Feature Envy. In the same context of object-oriented programming, the concept of inheritance has been known as a key feature proposed to increase the amount of software reusability. However, using inheritance is not always the best solution, particularly if it is utilized in improper cases where other types of relationships would be more appropriate. One of the particular issues that violate inheritance principles is the Refused Bequest code smell.These design smells can be detected by expert developers and when they are detected, it is

very important to refactor them to bring up with a better coding design that improves the system's code quality.

Manually searching for code smells in a large code base will take a significant amount of time. Software metrics gives a clear view of the tested software status.Metric based detection technique eases the detection task. Even new developers will have the ability to analyze and detect code flaws automatically without the need for an experts. Metrics values have the important role in modern developing procedures. Therefore automatic detection of code smells can be done in reasonable time and effort. And consecuently reflects on the software quality.

This study aims to detect some types of code smells in Java code. We have used object oriented metrics, static code analysis techniques to detect the code smells.

Keywords: software quality, code smells, refused bequest, feature envy, object oriented metrics, similarity between methods

ÖZET

YAZILIM KALİTESİ ÜZERİNDE OLUMSUZ ETKİLERE NEDEN OLAN OTOMATİK KOD ALGILAMA

Baydaa M. MERZAH

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Yrd.Doç.Dr. Yunus SELCUK

BT yazılım sistemlerinin kalitesi, ana hatları ile iç ve dış bakış açılarına göre ne kadar iyi tasarlandığına ve ön koşulların ne derece iyi karşılandığına bağlı olarak belirtilir. Yazılım sisteminin geliştirilmesinin ve bakımının kolay oluşu, başarılı bir şekilde tamamlanmasını ve sürekli kullanılmasını sağlayan hedef faktörlerdir. Steve McCall'ın bakış açısına göre [1], yazılım kalite özellikleri Dış ve İç Faktörler için kategorize edilebilir: Güvenilirlik, doğruluk, kesinlik, tekrar kullanılabilirlik ve bütünlük gibi dış kalite faktörleri son kullanıcıyı ilgilendirir. Bu çalışmada dış yazılım kalitesinden söz edilmemektedir. İç yazılım kalitesinin en açık göstergeleri, sürdürülebilirlik, yeniden kullanılabilirlik, esneklik ve test edilebilirliktir. Bu faktörler geliştiricileri ilgilendirir ve geliştirme ve bakım işlemlerini de kolaylaştırırlar ancak Kod Kokuları olarak bilinen kaynak kodundaki kötü kodlama stilleri tarafından olumsuz etkilenirler.

Nesneye yönelik programlamadaki önemli ilkelerden biri, yüksek uyum ve düşük bağlaşıma sahip sınıflar oluşturmaktır. Bu ilke, Feature Envy olarak bilinen bir kod kokusu tarafından ihlal edilebilir. Nesneye yönelik programlamada kalıtım kavramı, yazılımın tekrar kullanılabilirliğini artırmak için önerilen önemli bir özellik olarak bilinmektedir. Bununla birlikte, kalıtımın kullanılması her zaman en iyi çözüm değildir, özellikle de diğer türdeki ilişkilerin daha uygun olacağı durumlarda kullanılırsa. Kalıtım ilkelerini ihlal eden kod kokularından biri de Refused Bequest kod kokusudur. Bu

tasarım kokuları uzman geliştiriciler tarafından tespit edilebilir ve tespit edildiklerinde, sistemin kod kalitesini geliştiren daha iyi bir kodlama tasarımıyla yeniden yapılandırılmaları çok önemlidir.

Büyük bir kod tabanında kod kokularının manuel araştırılması önemli ölçüde zaman alacaktır. Yazılım ölçütleri, test edilen yazılım durumunun net bir görünümünü verir. Metrik tabanlı algılama teknikleri algılama görevini kolaylaştırır. Böylece yeni geliştiriciler bile, bir uzmana ihtiyaç duymadan kod kusurlarını otomatik olarak analiz ve tespit etme yeteneğine sahip olacaklardır. Metrik değerlerinin modern geliştirme prosedürlerinde önemli bir yeri vardır. Bu nedenle, kod kokularının otomatik algılanması makul zaman ve çabayla yapılabilir.

Bu çalışma yukarıda belirtilen kod kokularının Java kodunda tespit edilmesini amaçlamaktadır. Refused Bequest ve Feature Envy kod kokularını algılamak için nesneye yönelik metrikleri ve statik kod analiz teknikleri kullanılmıştır. İlgili metrikler ağırlıklandırılarak kullanılmıştır.

Anahtar Kelimeler: Yazılım kalitesi, kod kusurları, Refused Bequest, metodları arasında benzerlik , Feature Envy, nesne dayalı ölçümleri.

CHAPTER 1

INTRODUCTION

1.1 Literature Review

The production of high quality software within short time and reasonable cost is a major aim of software engineering. By translating the quality to a measurable format, the quality control procedure becomes more manageable. So, putting the software quality on a quantifiable basis is a significant work on which many researches and studies have been fulfilled in multiple disciplines and with different applications.

In the International Standard ISO/IEC 9126-1[2], a software quality model is established based on the Factor-Criteria-Metrics (FCM) Quality Model [3]. As reported by the standard, quality is affected by factors and these factors can be assessed by means of criteria. The major factors that evaluating the quality of the software is determined as usability, functionality, reliability, maintainability, efficiency and portability. The relation between these factors are mapped obviously with the related criteria, while the mapping of which criterion is affected by which metric still obscure. The factors defined are influenced by the involved technology, the used hardware, design of software, etc. The maintainability factor is the most design-related factor among all. Since most of the metrics are design-related this makes maintainability to be the most interested factor in the field of internal design quality.

From the software design perspective, nowadays the most prevalent coding technique is OO design. Object-Oriented Design (OOD) is the idea of building the computed system with interactive objects to achieve the required aim. The procedural programming is less maintainable than OO approach [4] and it presents the object concept in addition to important notions like inheritance, polymorphism and encapsulation [5]. OO design improves the modularity of the code compared to functional design. Modularity

increases understandability and as a result the maintenance of the code can be done without any potential difficulties [6]. As OO paradigm has presented new features, new metrics had been introduced to measure these features and concepts. Object-Oriented metrics are useful at summarizing the important aspects and give signs of the internal characteristics of the software [6]. They also help the developers to decide if there is any object has a problem to be refactored or not. MOOD metrics are used for measuring inheritance, polymorphism, and coupling [7] while Chidamber and Kemerer metrics has been developed for class level code measurement [4]. There are many metrics suits in the field of software quality and they will be illustrated in detail in Chapter 2.

As a particular type of software projects, our study is done on object-oriented software projects written in Java programming language. In this study we will present metric-based detection technique which aims to detect special types of code fragments that cause negative effect on the software quality and known as code smells. These smells are: Refused Bequest and Feature Envy smells. By translating the related symptoms to metrics and giving appropriate weight for each metric, then writing a detection algorithm to reform the results with the specified threshold values, the aforementioned smells are detected. These values are computed from analyzing stable versions of group of open source projects.

The remainder of the thesis is organized as follows:

In Chapter 2, software quality models and metrics will be reviewed. A literature review on code smells as general concept. Also a literature review for smells under investigation “Refused Bequest” and “Feature Envy” will be included in this chapter. In Chapter 3, the materials and methods used in the research will be explained in details. In Chapter 4, the obtained results will discuss in detail.

1.2 Objective of the Thesis

This study aims to detect bad code styling that effect negatively on the source code quality. Two types of code smells will be detected in this research: Refused Bequest and Feature Envy. Our detection approach depends on static code analysis with metric-based detection. Some metrics is well known while the others are identified by us. The new defined metrics is the translation of the symptoms related to each smell in our study. Threshold values are used to each metric in order to get more precise detection mechanism.

1.3 Hypothesis

Our research question is: What symptoms of a particular code smells can be formulated as software metrics?

In this study we will use static code analysis approach to analyze object oriented software projects written in Java. For this purpose, we will observe the symptoms of determined code smells and map them into related metrics. We will also observe the normal threshold values or normal ranges of the used metrics. By comparing on the obtained metrics values with the normal values and with the detection algorithm we will enable the developers to detect abnormal values and hence the related code smells automatically without the need for experts' decisions.

CHAPTER 2

GENERAL REVIEW

2.1 Software Quality

IT software systems are coming to be ubiquitous in contemporary life. Users over the world base on interconnected computers, as well as the global information infrastructure, such as the Internet and the World Wide Web (WWW), to accomplish their requirements for data processing, search, storage and information backups, archive and retrieval. All the mentioned needs are come across the boost of strong software bases. This reliance requires the software to be easy to use, to function accurately over the time, and have the ability to develop to meet the future needs easily. Generally, the need for high quality software requires the satisfactions of the development and claims about quality require to be prooven based on pragmatic analyses and measurements. These measurements had been translated to what are known as metrics. Metrics have many applied fields in recent research on software engineering, as we will see in the later section of this chapter.

This chapter introduces various concepts related to quality models, quality metrics and code smells that have negative effects on the code quality.

2.1.1 Quality Models

Software quality models are initiated to define characteristics that affect quality, to set up a group of metrics that measure these characteristics, to gather data that will help to assess the quality of software systems. Since the first presentation of the software quality concept, the researchers in this field have proposed diverse types of software quality models. Each model has its own characteristics and quality perspectives. Many quality models were initiated to assess the factors and sub-factors in field of software

products or systems. All quality models are categorized into three basic paradigms [8], as shown in figure 2.1.

- Legacy based Software Quality Model
- Object-Oriented based Quality Model
- Aspect-Oriented based Quality Model

Categorization Software	Quality Model
Legacy based Quality Model	McCall's Quality Model Boehm's Quality Model Murine & Carpenter Model NEC (Azuma) Model MQ Model Evans & Marciniak's Quality Model Deutsch & Will's Quality Model
Object-Oriented based Quality Model	FURPS Quality Model CMM Model ISO/IEC 9126 Quality Model Dromey's Quality Model Quality Model for Object-Oriented Design (QMOOD) Quality Attributes for COTS component A New Software Quality Model for Evaluating COTS Components
Aspect-Oriented based Quality Model	AOSQUAMO Model REASQ Model AOSQ Model

Figure 2.1: Categorization of software quality models [8]

Among all the previously mentioned models, the most important models which widely used and have a major role in the field of software quality, and within the scope of our study we will focus on a set of standard models which are: McCall's , Boehm's and ISO/IEC 25000 Quality Models, as we will see in this section.

A. **McCall's quality model:** McCall tried to find a common link between developers and users by concentrating on a set of software quality factors that define both the users' points of view and the developers' preferences. This model has, as shown in Figure 2.2, three major perspectives for clarifying the quality of a software product[1]:

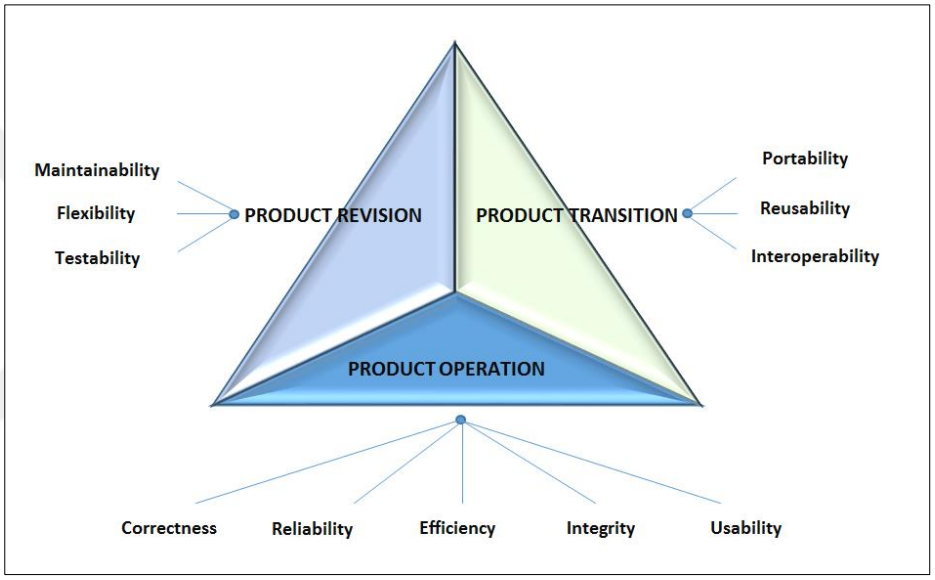

Figure 2.2 McCall's quality factors

Product revision (ability to change at the code level), includes maintainability (the required effort in order to determine the code fragments that have faults and fix them), flexibility (the simplicity of doing any required changes in the operating environment) and testability (the easiness of testing the program ad units and the as integrated parts, to confirm that it is error-free and meets its specified requirements).

Product transition (the ability to adapt to multiple environments) and (the functional properties), is all about portability (the required effort to convey a program from one environment to another), reusability (the simplicity of reusing software in a different context) and interoperability (the effort required to pair the system with anthers).

Quality of product operations depends on correctness (the degree of accomplishing the specification), reliability (the degree of accuracy), efficiency (the optimum use of resources, e.g. processor time, storage), integrity (the protection of the program from unauthorized access) and usability (the easiness of using the software).

Moreover, the model listed three crucial quality perspectives in a hierarchy of factors, criteria and metrics as illustrated in Figure 2.3:

- Factors: 11 factors are used to characterize the view of the users and summarize the external view of the software.
- Criteria: 23 quality criteria used to design the software as seen by the developers and describe the internal structure of the software
- Metrics (To control): They are used to control the design steps by providing a scale and method for measurement.

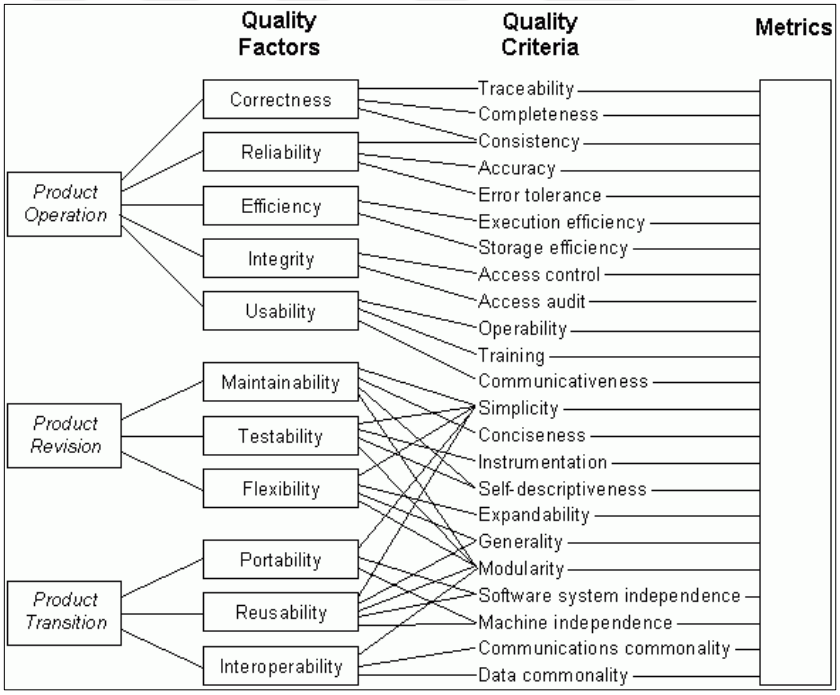

Figure 2.3 detailed hierarchy of McCall's quality model

B. Boehm's Quality Model

In 1978 Boehm presented a new quality model [9]. Boehm combined all the previously defined models and illustrated their shortcomings that assess the quality of software in quantitative and automatic means. His model attempts to define software quality qualitatively by clarifying a set of metrics and attributes. Boehm's model has a

similarity with McCall Quality Model in the point of presenting a hierarchical model. This model organized in the following format and as shown in Figure 2.4:

- High-level: 3 characteristics represent the high level requirements to evaluate the software, which are: General Utility, As-is Utility, Maintainability.
- Intermediate level characteristics.
- Primitive characteristics.

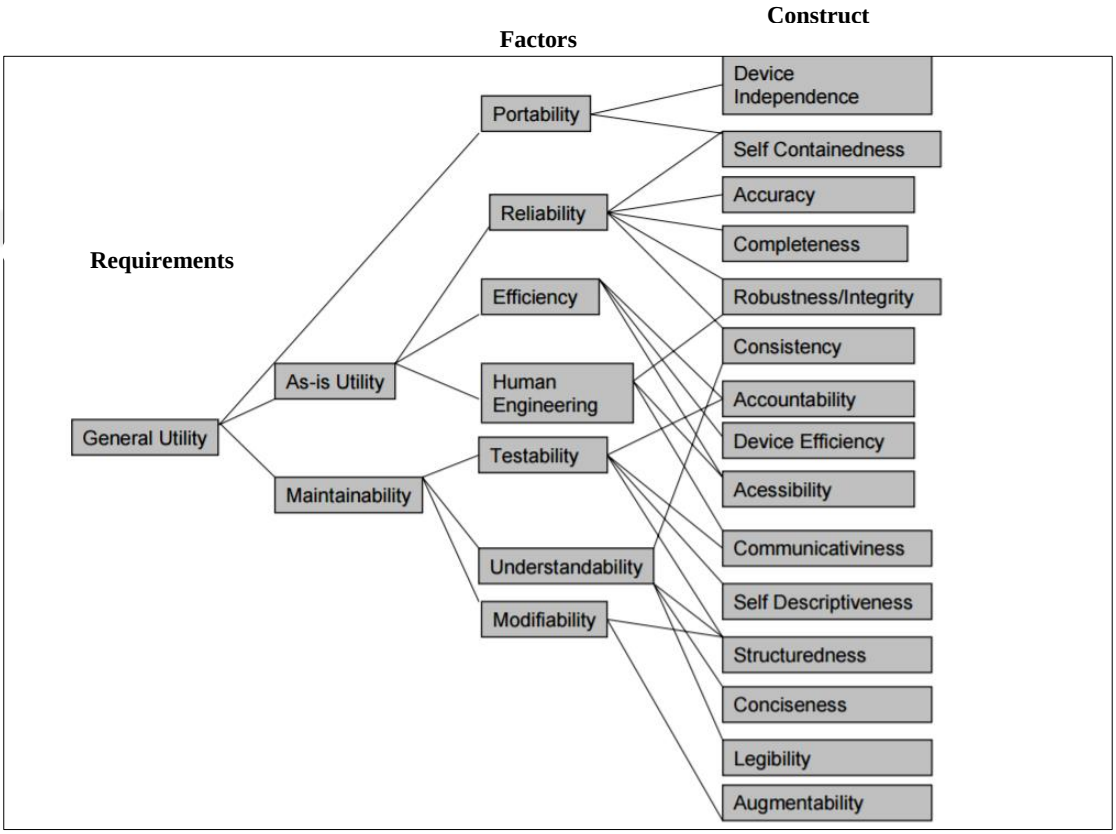

Figure 2.4 Boehm's quality model characteristics

Each high level characteristic correlated to set of factors, and these factors are fragmented to primitive constructs.

C. ISO/IEC 25000 (SQuaRE)

The series of standards ISO/IEC 25000, also known as SQuaRE (The Systems and software Quality Requirements and Evaluation) adopt the purpose of designing a comprehensive framework for evaluating the quality of software products. This model was the consequence of the gradual development of the previously published standards; particularly from ISO/IEC 9126, which explained the evaluation of software product as

a quality model, and ISO/IEC 14598, which defines the process for software product evaluation. This family of standards ISO/IEC 25000 composed of five parts [10]:

- **ISO/IEC 2500n – Quality Management Division**: define all common models, terms and definitions referred further by all other standards from SQuaRE family.
- **ISO/IEC 2501n – Quality Model Division**: present quality models in details for computer systems and software products, quality in use, and data.
- **ISO/IEC 2502n – Quality Measurement Division**: include a software product quality measurement reference model, mathematical definitions of quality measures, and empirical guidance for their application. Presented measures apply to software product quality.
- **ISO/IEC 2503n – Quality Requirements Division**: assists in determining the requirements of the quality. These requirements can be used in the process of quality requirements induction for a software product to be developed or as a feed in for an evaluation process.
- **ISO/IEC 2504n – Quality Evaluation Division**: supply requirements, advices and suggestions for software product evaluation.

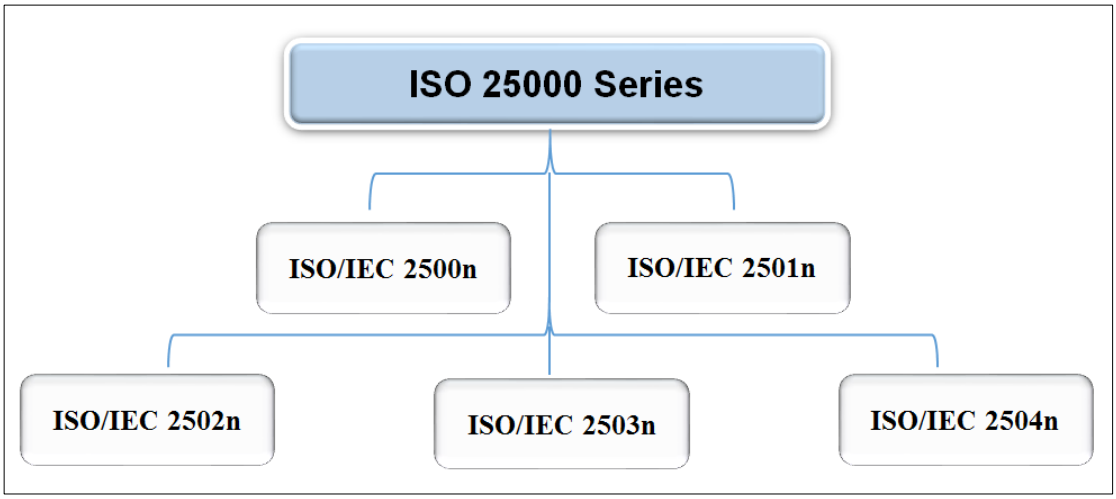

Figure 2.5 ISO 25000 Series

As with other standards, ISO/IEC 25000 describes what to evaluate but does not determine how to apply it. In other words, it does not mention any information about the thresholds for the metrics to evaluate them in the use, nor does it illustrate how to class these metrics in order to set an ideal value to a software product [11].

2.1.2 Software Quality Metrics

In the software production lifecycle as whole, the topic of software metrics is valuable. Software metrics give a clear review on the examined project status. Metrics measure the internal software structure and the software development process, also there is a set of metrics specialized to measure the project's status from the external view. It also enables the developers to gain the quality level before the final submission of the product [12]. In this study we will deal with the metrics which are defined at code level. This helps us to characterize the project's code and evaluate the design and by trying to detect code flaws some suggestions will be provided from the most significant refactoring solutions in the literature [13].

Generally software metrics characterize software engineering aspects such as product (source code, design and test case), process (like analysis, coding and design) and people (like the efficiency developers or the productivity of an individual designer) [14]. From the most general overview, the metrics are two groups which are software quality metrics and object oriented metrics. The software quality metrics can be classified into four main groups [15]:

A. Size Related Metrics
B. Complexity Metrics
C. Halstead Metrics
D. Quality Metrics

In figures 2.6 and 2.7, we can see that each set has its own metrics suite:

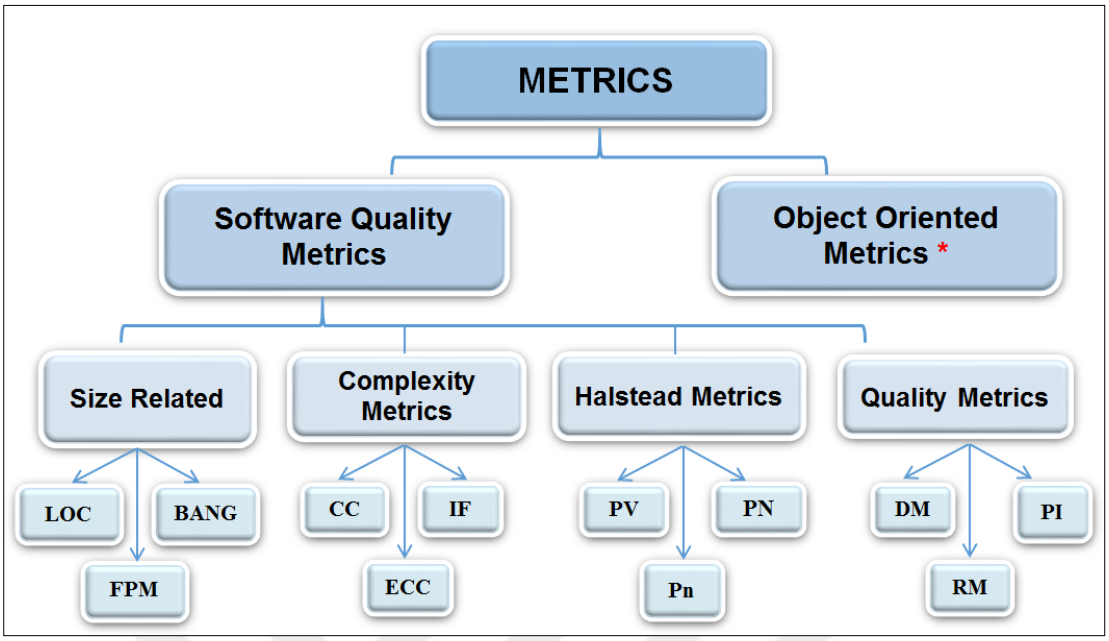

Figure 2.6 Metrics types

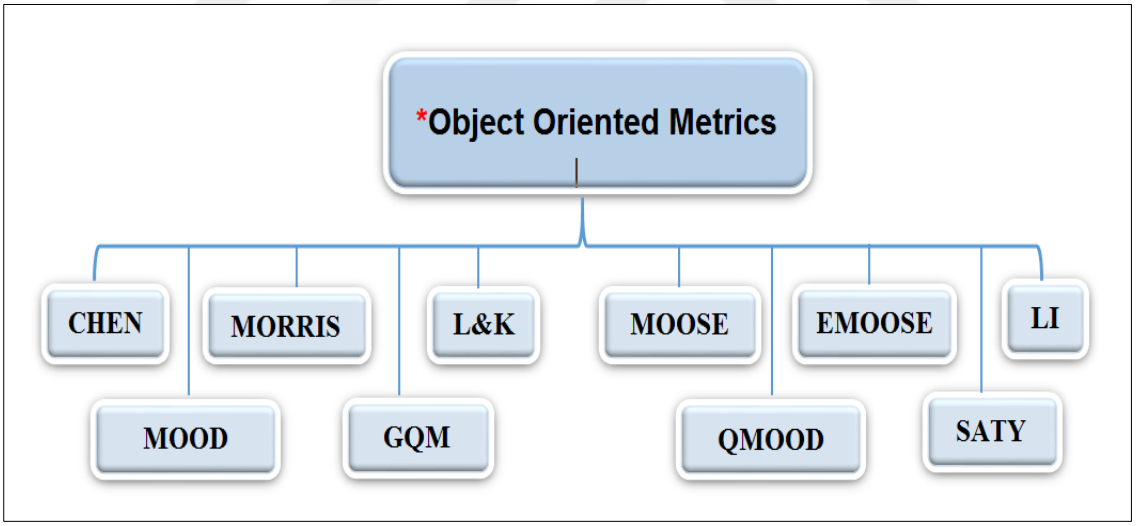

Figure 2.7 Object oriented metrics

1. **Software Quality Metrics**

A. **Size Related Metrics** :

This set of metrics is used to measure the size of the software. It has three main types:

i. **Line of code (LOC):** this metric is used to count the number of lines in the source code of the program. Some researchers built this metric by counting the number of lines include the whitespaces and comments, while the others just count the number of actual code statements. Generally, it is used to measure the complexity and productivity of the examined code.

ii. **Function Point Metrics (FPM):** This metric proposed by Albrecht [16], it is used to estimate the required lines of code during the software development lifecycle depending on the inputs, outputs and inquiries. Thus, the estimation of the needed effort for the development process becomes easily.

iii. **Function Bang:** DeMarco [17] proposed that the size of a software product can be estimated from the elements of a Structured Analysis (SA) description during the phase of requirements specification phase. For precise measurement, DeMarco classified the systems into three groups: function-strong, data-strong, and hybrid systems. This classification is based on the ratio RE/FF, where RE is the number of relationships in the retained data model and FP means the number of functional primitives (bottom-level processes) in a dataflow diagram (DFD). If the ratio RE/FP has a ratio less than 0.7 the system is function-strong. If the ratio RE/FP has a value more than 1.5 the system is data-strong, else the system is a hybrid one. A function metric for function-strong systems, called a Function Bang, is counted from a DFD based on the complexity of the dataflow and the types of operations (functions) operating on these dataflows.

B. Complexity Metrics: This set of metrics has different means depending on the presenter; all of them have the same aim which is finding the degree of complication for the source code. The following are the main metrics used in this field:

i. **Cyclomatic Complexity (CC):** McCabe's cyclomatic complexity is a software quality metric that measures the complexity of a software program. Complexity is deduced from finding the number of all distinct paths in the source code of the program. The more complicated code which have a high value. The complexity of

the code means that it is difficult to understand and hard to maintain. Cyclomatic complexity is derived from the dataflow graph of a program as follows:

Cyclomatic complexity (CC) = E - N + 2P (1.1)

ii. **Extended Cyclomatic Complexity (ECC)**: Myers [18] has extended the concept of cyclomatic complexity to cover some aspects not mentioned in McCabe's notion. He proposed that decision points with multiple conditions are more complicated than decisions with one condition. In this extension, complexity is measured as an interval rather than a single value. The lower bound of the interval is the number of decisions plus one or the cyclomatic complexity of the program. The upper bound of the interval is the number of individual conditions plus one. Specifying the complexity as an interval accounts for both the decisions and the conditions in a program.

iii. **Information Flow**: Kafura and Henry [19] presented their metric to measure the complexity of a program as information flow. This metric depends on counting the number of local information flows input (fan-in) and flows output (fan-out). The following formula (2) defines the metric :

$$C = [L] * [(fan - in) - (fan - out)]2 \quad (1.2)$$

C. **Quality Metrics**: this type of metrics don't have fixed format or particular type, it depends on the requirements of the product. At the end of the requirements specification phase, the project's manger determines the quality level and chooses the appropriate metrics to evaluate the satisfaction of the user's needs. Generally, few types of this metrics are available like: Defect Metric, Reliability Metrics and Maintainability Metrics.

2. Object Oriented Metrics

Object-oriented programming has powerful features that characterize its work, such as encapsulation, information hiding, polymorphism inheritance and dynamic binding. These features simplify the reusability and unit-based development processes. However, they might cause some types of bad styling code fragments that are not easy to detect by traditional testing techniques. Traditional testing techniques, such as functional testing and branch testing, are not feasible to diagnose OO problems. To overcome these scarcities, it is important to adopt an object-oriented testing technique that takes these features with cost balancing into account. Object-oriented metrics have been studied and proposed as good detectors for different types of code smells, as we will see in the

later sections. In this section we will focus on one popular OO metrics suites in the literature.

i. **Metrics for object-oriented software engineering (MOOSE):** This suite proposed by Chidamber and Kemerer (CK) et al. [20]. This metrics have led to a considerable amount of interest and are widely used in many studies in the field of object-oriented software since it had been published until now. The CK metrics suite consists of six metrics that evaluate the characteristics of the object-oriented design are:

ii. **Weighted Methods per Class (WMC):** This metric measured by computing the total complexity of all the methods in a class. A large number of methods in a class may have a potentially larger impact on the derived classes since the methods in the parent will be inherited by the child.

iii. **Depth of Inheritance Tree (DIT):** This metric is used to find the maximum length from the root node to the node in the lowest level (leave) in the hierarchy of the code. The complexity of a class can be represented by DIT. Thus, a system which has a lot of inheritance layers will be hard to understand. Also, it will be an indication to the reusability of many methods.

iv. **Number of children (NOC):** this metric is defined as the directly derived leaves in the classes' hierarchy. These points are used to know the number of subclasses that will inherit the methods of the base class. The great value of this metric means that base class has improper abstraction.

v. **Coupling between Objects (CBO):** is used to count the number of classes that related to the current class. The abundant coupling drops the modularity of the class making it less responsive to the reusability. Also it will increase the sensitivity and difficulty to the changes during the code maintenance.

vi. **Response for class (RFC):** this metric is used to find the response set of the class by the combination of two sets: the first is the set of methods called by appropriate one method (M) and the second is set of all methods in the class (Ri), as illustrated in the following equation (3). Large value makes the testing and debugging of the object more complicated.

$$RS = \{ M \} \cup \text{all } i \{ Ri \} \quad (1.3)$$

vii. **Lack of Cohesion in Methods (LCOM):** This metric is used to count the number of null intersection methods pairs minus the number of similar method pairs used in the class. The null intersection methods have no shared instance variables, while the similar methods have minimum of one common instance variable. It is used for measuring the pairs of methods within a class using the same instance variable.

2.2 Code Smells

The concept of code smell is introduced by Fowler [21] as signs of internal design flaws within the software. He defined 22 kinds of code smells. Each type has its own symptoms and different effects on the code development process. Therefore, the detection of code smells has become a mandatory technique to enclose the code issues that may affect negatively on the software quality by causing problems for further development and maintenance ([6], [21], and [22]). Accordingly, the consensus is that all types of code smells need to be detect firstly and then refactored to deny or diminish such issues [23].

In this section we will have a general review on the code smells sorts and the main reasons to cause them, then devote our review on the types under investigation in the thesis work, which are Refused Bequest and Feature Envy code smells.

Since bad smells have an impact on code quality, it is mandatory to have a look on the main causes led to these smells to appear into the software design. Suryanarayana [24] mentioned in his book the prime reasons that leads to code smells occurrence, as shown in Figure 2.8.

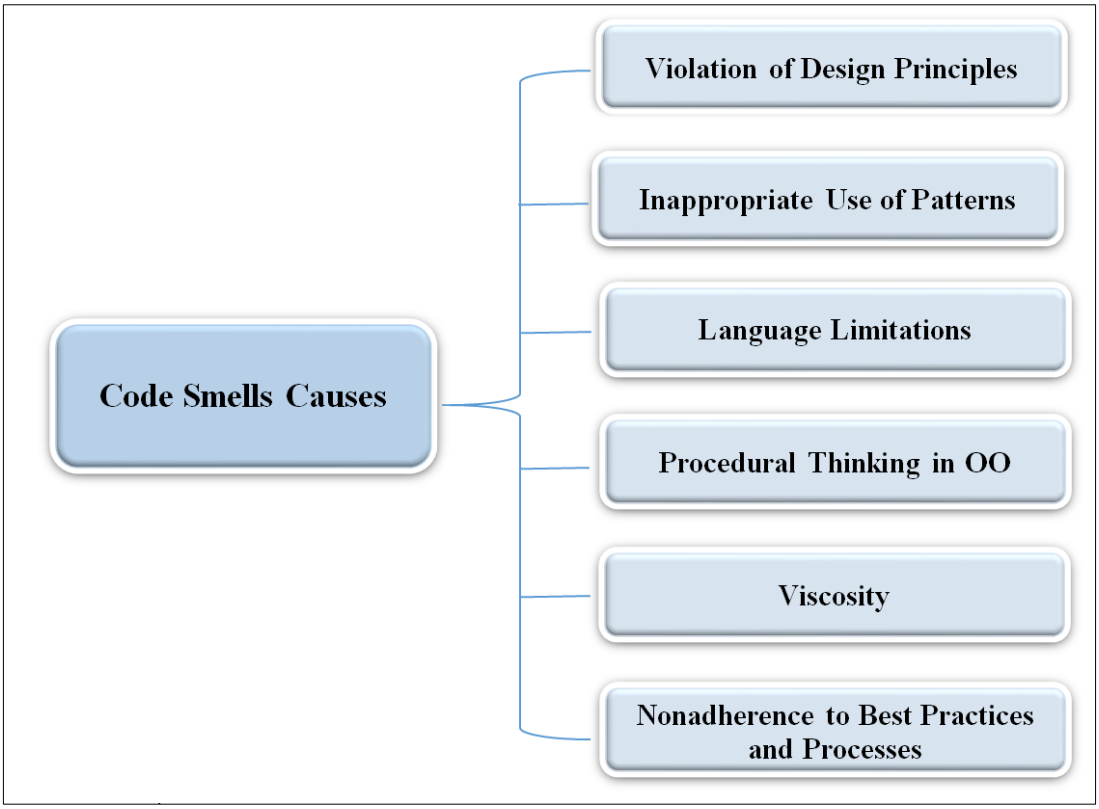

Figure 2.8 General causes of code smells

2.2.1 Code Smells Categories

According to Mantyla et al [25], some code smells defined by Fowler [21] have common features and can be grouped to gather. He classified them into seven main groups as:

i. **Bloaters:** They represent code fragments that have abnormal size that it cannot be handled in an appropriate manner. The following smells can be included under this set: Large Class, Primitive Obsession, Long Method, Data Clumps and Long Parameter List.

ii. **Object-Orientation Abusers:** This type of smells is related to the violation of object oriented programming principles or applying these rules in improper manner. This group has the following types: Switch Statements, Refused Bequest, Temporary Field, Parallel Inheritance Hierarchies and Alternative Classes with Different Interfaces.

iii. **Change Preventers:** The code structure in this category has the problem of difficult modification. This category has two code smells: Shotgun Surgery and Divergent Change.

iv. **Dispensables:** This type can be defined as an unnecessary part of code which should remove or giving additional responsibilities to improve its status. It has the following kinds: Data Class, Duplicate Code, Lazy Class and Speculative Generality**.**

v. **Encapsulators:** This set's elements shared the violation of OO encapsulation rule but in different ways. It has the following two elements: Message Chains and Middle Man.

vi. **Couplers:** This type of issues caused because the classes are highly coupled while the OO aim is to have loosely coupled classes. It include: Feature Envy and Inappropriate Intimacy.

vii. **Others:** This category includes the two remaining smells: Comments and Incomplete Library Class. They are do not fit into any of the categories above.

In the rest of this chapter, we will have a constraint overview on the types of code smells that will be study in our research which are Refused Bequest and Feature envy. Also we will discuss the detection strategies used in the literature to detect these types.

2.2.2 Refused Bequest Code Smell

In the context of object-oriented programming, the concept of inheritance has been known as a key feature proposed to increase the amount of software reusability. However, using inheritance is not always the best solution, particularly if it is utilized in improper cases where other types of relationships would be more appropriate. One of the particular issues that violate inheritance principles is the Refused Bequest code smell. It is related to an inheritance hierarchy where a subclass does not obligate the interface inherited from its parent class.

More precisely, Refused Bequest smell is present if the inherited functionality by the sub-class is not actually used or specialized by means of overriding. The appropriate refactoring in this case is the "Replace Inheritance with Delegation" [21] which dictates to transform an inheritance relationship into composition where the sub-class has a reference to an object of the base class and uses only the needed functionality. This refactoring agrees with the GoF suggestion "Favour Composition over Inheritance" [26]. We can infer from that Refused Bequest smell can't appear in abstract classes or interfaces, so we excluded them during our analysis.

There are two main approaches for Refused Bequest code smell detection as well. These approaches are based on either static code analysis or a combination of static and dynamic code analysis[27].

A) **Static Code Analysis Detection**

Stefan Slinger proposed a detection strategy employing the static code analysis of the source code [28]. In this approach, a parser analyses the Java source code files and produces abstract syntax trees. When the parser is done, an analyzer (visitor) traverses the abstract syntax trees, collects smell aspects and stores them in a repository. For refused bequest identification, smell aspects such as information on parent classes, methods, fields, and the methods and fields that are used by a class are needed. A Grok script is then executed on the smell aspect or fact repository to identify the classes that contain refused bequest code smells. This detection strategy combines appropriate code metrics with definition of threshold values proposed by Lanza and Marinescu [6].

In a paper by Marinescu, a set of rules and software metrics are identified by static analysis of selected software projects' source code. These software metrics are compared with threshold values to identify design flaws or code smells [29].

B) **Combination of Static and Dynamic Analysis Based Approach**

The necessity to analyse hierarchy clients to identify original intent of a generalization has been emphasized by P.F.Mihancea in his paper [30]. A suite of metrics which quantify the uniformity of clients' calls with respect to the services provided by a hierarchy had been proposed. Then these metrics are used to depict class hierarchy and to recognize if there are any anomalies in the design. He evaluated the approach on two medium-sized projects and found that the approach does actually aid to make the characterizing of the essence for the base class with respect to interface reuse.

Amin Milani Fard et al. proposed an approach that uses a metric-based algorithm, and combines static with dynamic analysis to detect these smells in JavaScript code [31]. Due to the dynamic nature of JavaScript, static code analysis alone is not sufficient. Therefore, in addition to static code analysis, dynamic analysis is also employed in the paper to monitor and infer information about objects and their relations at runtime. JavaScript objects, their types, and properties are inferred dynamically by querying the browser at runtime. Finally, based on all the static and dynamic data collected, code smell is detected using the metrics.

The approach proposed by T. Tourwe et al was making use of logic meta-programming to differentiate improper interfaces [32]. In this paper, all direct subclasses of a base class are identified and then all 3 possible subset of these identified classes are used to recognize inappropriate interfaces.

A combination of dynamic and static analysis is used to identify refused bequest by Elvis Ligu et al. [33]. In that paper, introduction of intentional error in sub-classes' non-overridden inherited methods is used to identify client's usage of super class methods. Measuring symptom severity on a smell thermometer can underline suspicious hierarchies that warrant the need to be refactored.

2.2.3 Feature Envy Code Smell

In the object oriented programming context, misplacing the members of the class is one of the main flaws. Also, distributing the responsibilities among classes must be in the right way and avoid making the class responsible for tasks that should be manipulated by other classes. In the same context, classes must be loosely coupled and highly cohesive. The code smell that violates the previous mentioned rules is known as Feature Envy. Feature Envy code smell is a symptom of improper association between classes. It occurs when a method within one class is more concern in some members of other classes that it is currently defined in. The higher the coupling between classes, the higher the number of classes are influenced when changes are needed to be done in the system. In strongly coupled classes even a tiny deliberate modification could result in a long series of unpredicted changes in a lot of classes. Consequently, the interdependence between classes should be remained to the lowest value if possible.

In the field of Feature Envy detection, many studies had been presented. Tsantalis and Chatzigeorgiou presented a method of detection by computing the similarity degree between a class and method using entity sets [34]. The entity set of a method have the members that it accesses whereas the entity set of a class include all members that be owned by the class (excluding getters and setters). The inner and outer entity distances are calculated. Inner entity distances should be as low as possible to attain high cohesion; while the distance of the outer entity should be as large as possible for low coupling. Their entity placement metric of a class is the ratio of average inner to average outer entity distances. If the ratio is high, the class may not be cohesive or it is highly coupled with other class.

The approach proposed by Sales et al. [35] involves evaluating dependency sets for a given method m in a class C. Two average similarity coefficients are calculated**.** The first is the average similarity identifies the dependencies between m and the remaining methods in the class. The second is the average similarity determines the dependencies between m and methods in another class Ci. If the second average is greater than the first average, then m has high similarity to methods in in the class Ci than its current class C and Ci could be the appropriate class for m.

Oliveto et al. [36] submitted another technique to detect Feature Envy bad smell by identifying method friendships. They adapted the viewpoint of classes and methods are quite similar to sets of people. In this approach, the degree of similarity among methods in the system is determined and friendships among these methods are ranked.

Dexun et al. [37] proposed weight based distance metrics theory to show up the candidates of Feature Envy. They calculated distance between entities (attributes and methods) and classes. The weight based distance metric depends on the multiple invoking relationships between each two entities. They compared their results with simple distance based approach.

Another approach which is related directly to work will be discussed later in the discussion chapter.

CHAPTER 3

MATERIALS AND METHODS

The importance of producing software with high quality has been mentioned in the previous chapters. Many quality models are being used to evaluate the quality level and many of them rely on software metrics. Many approaches had been presented to detect the code fragments that cause negative effect on the product's quality. The simplicity of the used technique simplifies the assessment process for the developers. By connecting the information gained from the previous chapters the following question may come in the light:

"How to evaluate the quality of the software from the developers' perspective?"

Software development is usually done under time pressure. So developers need simple and accurate techniques to detect code smells and find the appropriate refactoring methods to bring up the quality to the required level. Since metrics measure the internal software structure and the software development process, this helps us to characterize the software projects and evaluate the design.

In this chapter, the methodology of detecting the code smell types which were mentioned in Chapter 2 will be explained. The tools that helped us in the measurements of the selected metrics will be illustrated. A general description for the software projects which are used as the test code in the work will be presented.

3.1 Methodology

In our study, we used object oriented metrics and static code analysis. Threshold values for the metrics are calculated from a pool of selected open source software projects written in Java. We selected a group of stable releases of each software project as training set and build a data set of metrics for each project separately. These projects are

from different domains to support the variety of actual metric values. Software projects had been selected from Qualitas Corpus Index [38]. Table 3.1 shows the training set and their details.

Table 3.1 Software projects training set

Name	Status	Full Name	Domain	NOC	Release date	Latest Release	Selected Release
ArgoUML	Active	ArgoUML	Diagram generator	2560	15.12.2011	0.34	0.34
JasperReports	Active	JasperReports	Diagram generator	1844	20.07.2010	6.4.0	6.2.2
Velocity	Active	Velocity Engine	Diagram generator	261	10.05.2010	1.7	1.6.4
Springframework2	Active	Spring Framework	middleware	3089	20.10.2010	2.5.6	2.0.8
Struts	Active	Struts	middleware	1074	16.08.2010	2.3.32	2.3.24
Tomcat	Active	Tomcat	middleware	1739	11.08.2010	9.0.0	7.0.42

After we collected the metrics from these six big projects, now we need for a robust technique to have a standard threshold values. We proposed to determine the minimum and maximum threshold values by the Equation 3.1 where STDV is the standard deviation.

$$\mu - 2\sigma < \text{Threshold} < \mu + \sigma \quad (3.1)$$

Some metrics values were collected readily by available tool [39] while the new defined metrics were computed partially by the tool and partially by manual computations. The definition of the metrics used in our approach and their threshold values will be explained later in this chapter.

3.2 Detection Approach

Within the scope of the thesis, we selected two types of code smells which are directly related with the violation of object oriented programming concepts. As mentioned in the previous chapter, the code smells under investigation are Refused Bequest and Feature Envy.

Our detection approach depends on object oriented metrics. Each code smell has a set of symptoms that differentiate it from other types. In the following section we will illustrate the symptoms for each code smell in our study and try to translate them to corresponding metrics.

3.2.1 Refused Bequest Detection

Before we start the explanation of the metrics used to detect Refused Bequest smell, first we will give an idea about the background to one of the new defined metrics. This is the similarity between methods.

Similarity of methods is based on how related they are on usage of instance variables of the class. The same notion is used in our own approach as well. The similarity of two sets can be determined by comparing the ratio for number of elements in their intersection to the number of elements in their union. This definition of similarity can be expanded to similarity between methods [20]. Consequently, the value of similarity between two methods can be calculated by finding the ratio for the set of common instance variables used by both methods to the set of total number (union) of instance variables used by the two methods [40]. Two methods have high similarity ratio if the number of elements in the intersection of the set of common instance variables is high. Consider two methods M1 and M2. Let {SIV1} and {SIV2} be the set of instance variables used by M1 and M2, respectively. We define:

$$|SIV|_u = \{SIV1\} \cup \{SIV2\} \tag{3.2}$$

$$|SIV|_i = \{SIV1\} \cap \{SIV2\} \tag{3.3}$$

That means; $|SIV|_u$ is the total number of instance variables used by the two methods and $|SIV|_i$ is the number of mutual instance variables used by both methods. Then, method similarity (MS) between the two methods can be defined as:

$$MS = \frac{|SIV|i}{|SIV|u} \tag{3.4}$$

The maximum similarity value between two specified methods is 1 while the minimum value is 0 where MS value of 0 refers no common instance variables shared by the two methods and MS value of 1 refers to the two methods used the same instance variables. As an example, consider two methods M1 and M2 where {SIV1} = {x, y, z} is the set of instance variables used by method M1, and {SIV2} = {z, t, e} is the set of instance variables used by method M2. Then, we have: {SIV1} U {SIV2} = {x, y, z, t, e} and {SIV1} ∩ {SIV2} = {z} resulting in |SIV| u = 5 and |SIV| i = 1. The Similarity between these two methods then becomes MS =1/5 = 0.2. This gave us the basis for formulating the new metric presented in our approach. We propose to use this approach in refused bequest detection strategy, by calculating the similarity between the superclass methods and the overridden methods in the derived class. Then we compute the average for the computed values. The new metric's name is ASM (Average Similarity of Overriding Methods). If the value is less than the threshold value and with a combination with other predefined metrics then the class is a candidate of refused bequest. We propose a metrics based smell detection strategy for object-oriented software systems.

There are two main symptoms should be available in a subclass in order to be a Refused Bequest candidate. They had been explained by [30] in details. These are:

- **Sub-class doesn't use superclass's bequests**. This symptom can be translated to multiple metrics. In our approach we used the following metrics: **BOvR** and **ASM.**

- **Sub-class is too long and complex**. In this study this symptom is translated to the following metric: **ACCO.**

BOvR metric represent the ratio of the number of overridden methods from the base class to the total number of methods in the sub-class. The ACCO metric is defined by us. It computes the average cyclomatic complexity of the overridden methods in the sub-class. The ASM metric represents the average of similarity of overridden methods in the sub-class. Thresholds for each metric had been illustrated in Table 3.2. The values are calculated from the software projects that were listed in Table 3.1.

Table 3.2 Metrics' Threshold Values

Metric Name	Threshold (+)	Threshold (-)	Average
ASM (Average Similarity between Methods)	1	0	0,5
ACCO (Average Cyclomatic Complexity of Overridden methods)	9,431	0	4,716
BOvR (Base- class Overriding Ratio)	0,332	0	0,166

According to the values in Table 3.2, we propose to mark a class as a Refused Bequest instance if its ASM or ACCO value exceeds their respective thresholds. The BOvR metric is used to support the ACCO metric in the detection in case it has a value below the threshold value. The entire calculation process is shown in Figure 3.1.

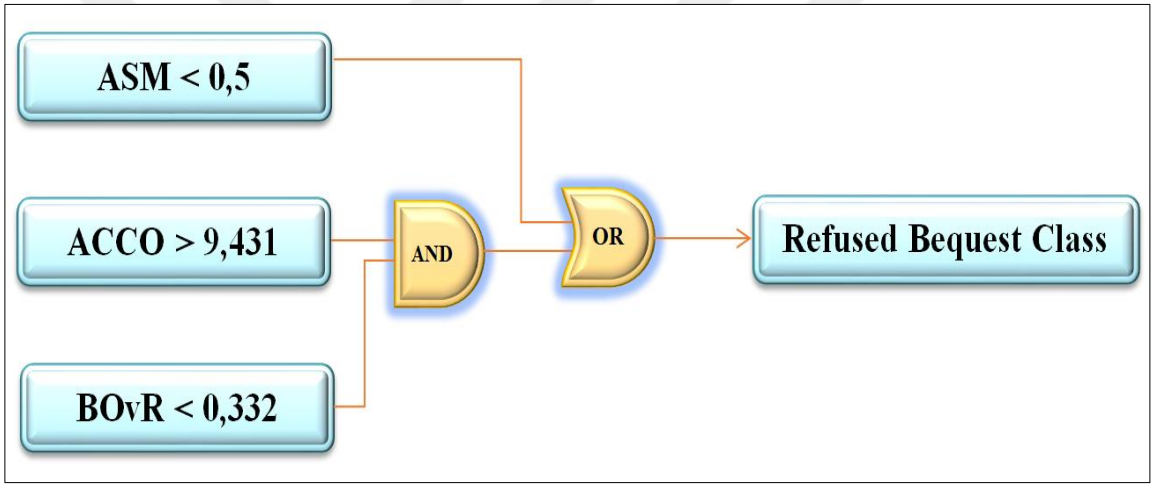

Figure 3.1 Proposed method of detecting refused bequest instances

3.2.2 Feature Envy

From all the approaches of Feature Envy detection presented in Chapter 2, we used a modified version of the metric based detection mechanism presented by Marinescu [6] in our work. Our proposal is a combination of object oriented metrics with some implementation rules which has been used to form the final detection strategy. The most important symptoms in this method are:

- Method uses directly attributes from unrelated classes that are accessed directly or by invoking accessor method. This symptom is translated to Access to Foreign Data metric **(ATFD).**
- The attributes from external classes used by the method is more than the attributes it has. The Locality of Attributes Accesses **(LAA)** is the equivalent metric to this symptom.
- The envied attributes may be belonging to many other classes or just from particular classes. In case of using attributes from multiple classes this will act as a controller [41]. But in case of the envied attributes are from particular few classes this will be an indication of method misplacing and an indication of Feature Envy. This sign can be measured by Foreign Data Provider **(FDP)** metric.

This detection mechanism takes into consideration all the details and conditions that make the inspected method as a Feature Envy candidate. But the weakness of this approach is the ambiguity of the actual threshold values for the used metrics. We tried to find standard values for them but they are not available in the literature. In our thesis we analysed open source software projects as mentioned previously in this chapter and we used the standard static formula to produce a standard threshold values for the used metrics and make them as standard values for the next studies. The threshold values are shown in Table 3.2 and our detection approach is given in Figure 3.2

Table 3.2 Metrics' Threshold Values

Metric Name	Threshold (+)	Threshold (-)	Average
ATFD (Access to Foreign Data).	3,598	0	0,521
LAA (Locality of Attribute Access)	1,373	0,492	0,933
FDP (Foreign Data Provider)	1	0	0,5

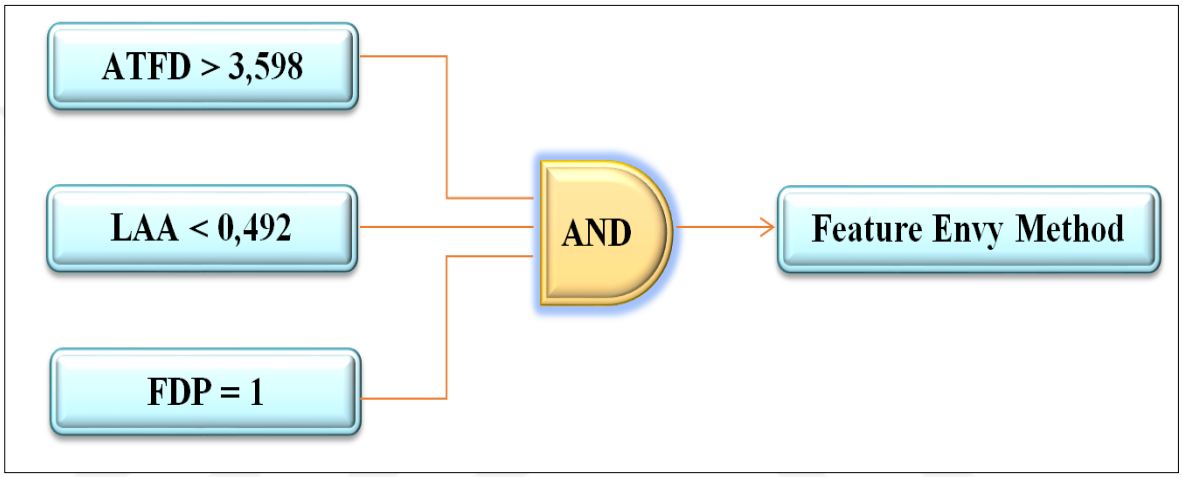

Figure 3.2 Proposed method of detecting feature envy instances

3.3 Attributes of the Software Under Test

To apply our methodology, we selected an open source Java project. This project is known as Sweet Home3D version 4.0 distributed under GNU General Public License [42]. It is public domain software. It is free internal design structure application that helps the designer to draw the plan of your house, arrange furniture on it and visit the results in 3D. It can be run under Windows, Mac OS X 10.4 to 10.12, Linux and Solaris. It is available in 25 different languages. The selected version reported that have bugs and had been fixed in later releases. It has multiple releases; the last one was the 5.4 on 31-01-2017. Figure 3.3 are showing an example from the designs achieved by the application published on the application's website.

Figure 3.3 design examples

The general attributes for Sweet Home 3D version 4.0 is illustrated in Table 3.3. The metrics in Table 3.3 had been collected by metrics 1.3.6 plugin [43] for Eclipse.

Table 3.3 General attributes of sweet home 4.0 software

Metric Name	Metric Value
Lines of Code (LOC)	76572
Number of Methods (NOM)	4154
Number of Overridden Methods (NORM)	411
Number of Classes (NOC)	376
Number of children	144
Number of Packages (NOP)	9
Depth of Inheritance Tree (DIT)	6
Number of Hierarchies	37

3.4 Tools used during the Analysis Process

Software tools are used during our work to collect the metrics used in the detection technique. These tools facilitated the metrics values gathering process. Each tool is specialised in a set of metrics. Also they can be either a stand-alone applications or as plugins. Now, let's explain briefly each tool used in the work.

1- **iPlasma**: Integrated Platform for Software Modelling and Analysis. It is a free stand-alone tool presented by LOOSE Research Group for the first time in 2005 [44] and the last updated version 6.1 was in 2012. The internal structure of the tool appears in figure 3.4.

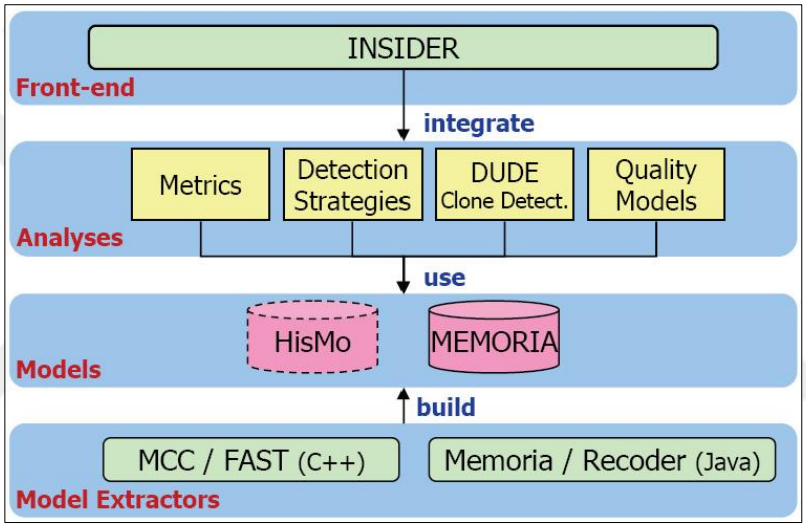

Figure 3.4 Internal structure of the iPlasma

In our work we used the front-end (INSIDER) to upload the java project which we want to analyse. It can be run by using the command prompt of windows and execute a batch file to run the insider graphical user interface (GUI) .Then we selected the group's category as classes, methods or any other available groups; depending on the target under investigation and the metric needed in the study, as shown in figures 3.5 - 3.7.

In our work, we selected to group the source code as classes to obtain the some metrics of Refused Bequest, and in the other phase we used the methods' group to get the metrics of Feature Envy. Figure 3.5 shows the INSIDER front-end with the uploaded Java source code for the Sweet Home 3D 4.0.

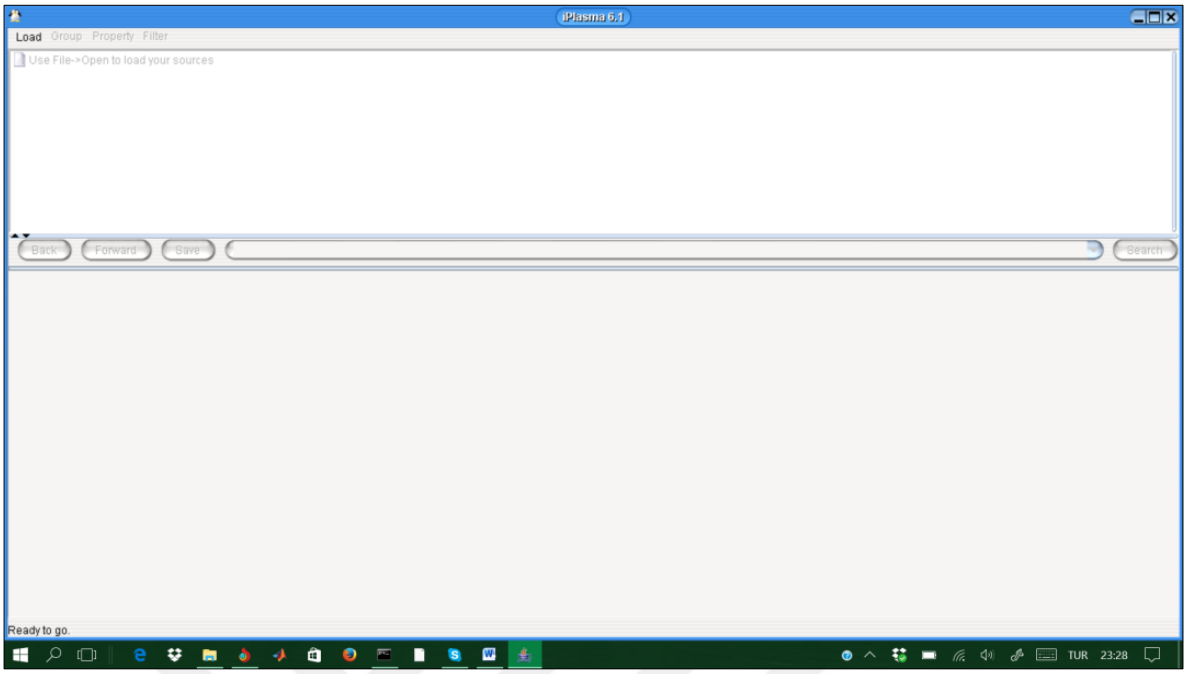

Figure 3.5 INSIDER's front-end

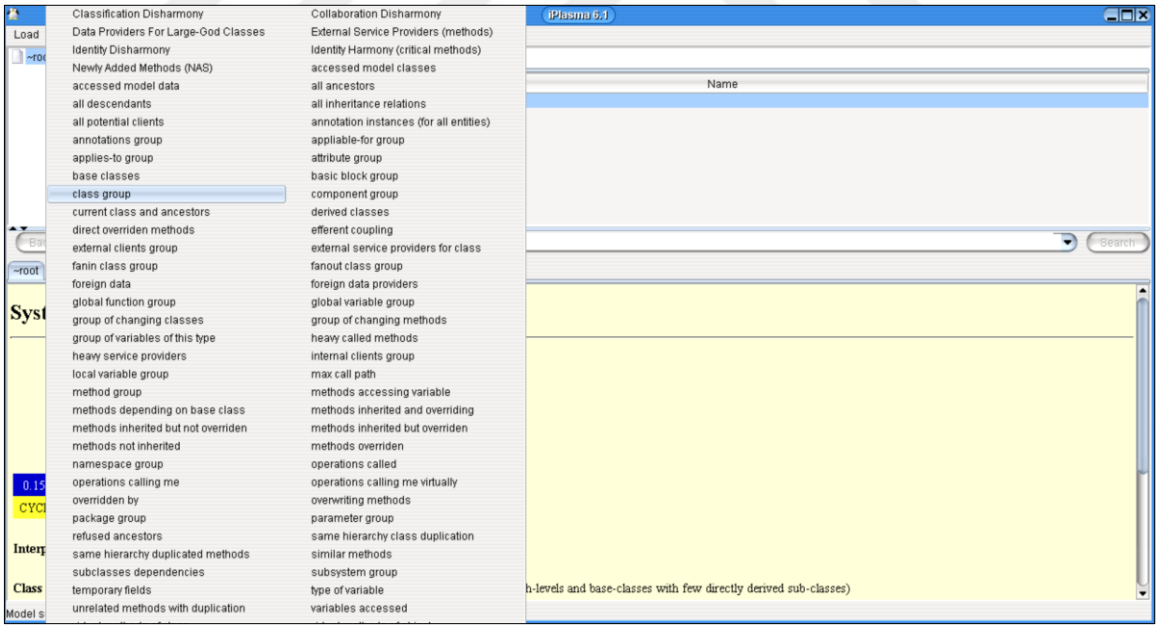

Figure 3.6 Selecting the class group

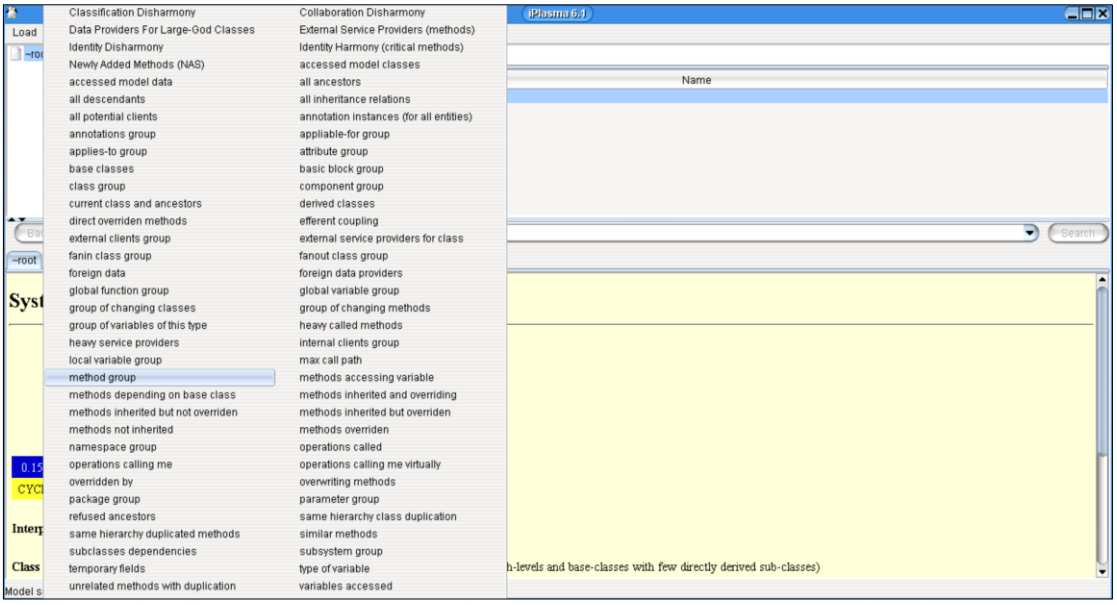

Figure 3.7 selecting the methods group

This tool has an important role for us to understand the inheritance hierarchal tree of the test project. By using the system complexity overview option as shown in Figure 3.8, we can have a clear view of the available inheritance hierarchies in the source code. The edges represent the inheritance relationship [6]. There are many other meanings related to the node size and colour, edge width and colour, but they are beyond the scope of our study, it can be found in [6] for more details.

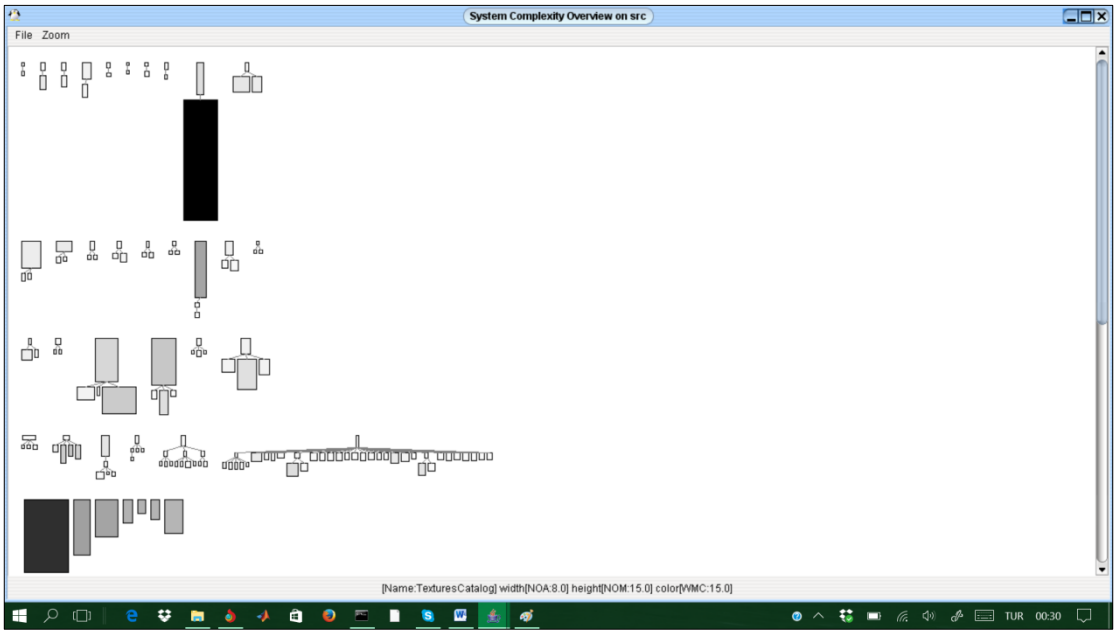

Figure 3.8 the inheritance trees of the tested code

2- **Metrics 1.3.6 plugin:** by Frank Sauer is an open source metrics calculation and dependency analyzer plugin for the Eclipse IDE. It measures various metrics and detects cycles in package and type dependencies. It provides two types of views; one for the layered package graph view and the other for the dependency graph view as shown in Figure 3.9 .This plugin is used in our work to find the metrics in Table 3.3.

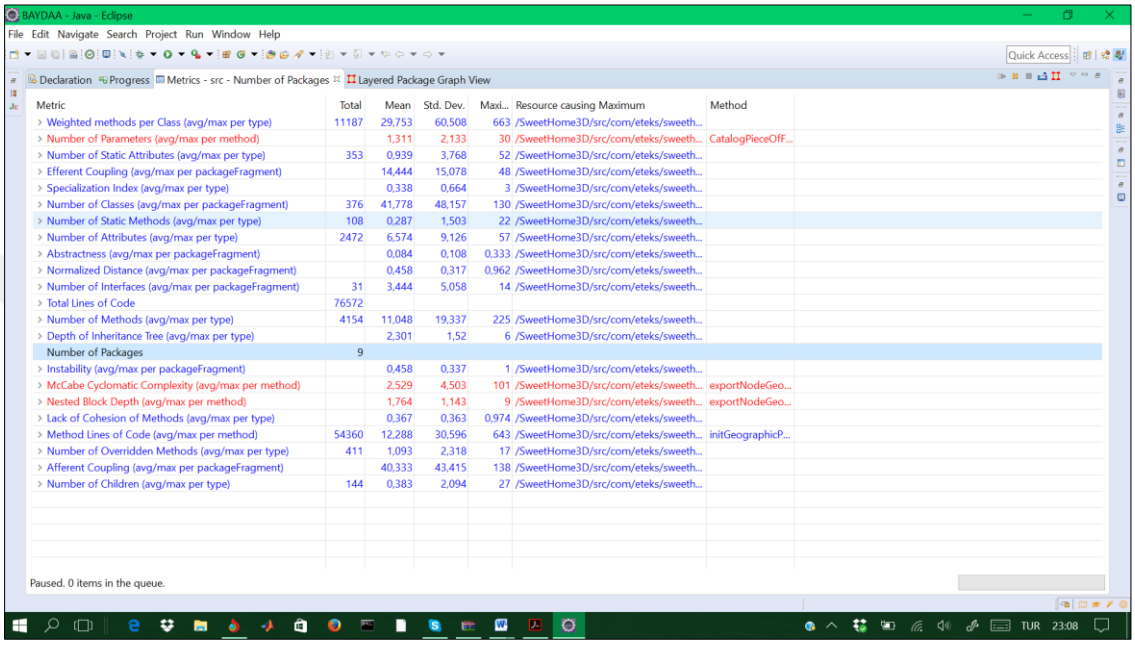

Figure 3.9 Metrics obtained by metrics 1.3.6 plugin

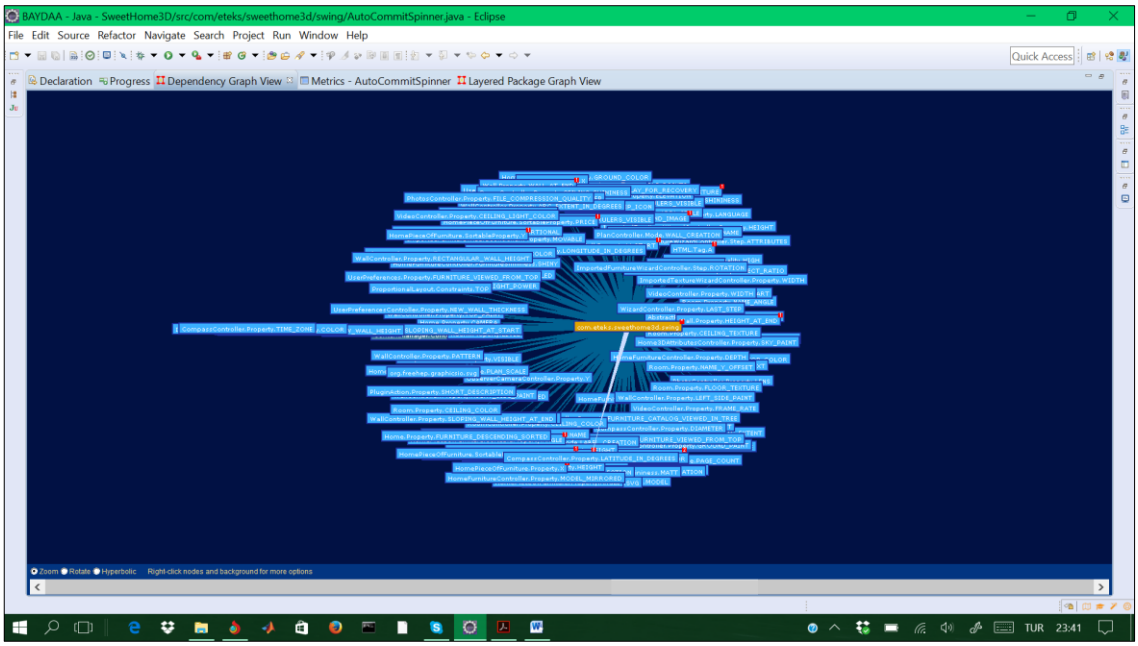

Figure 3.10 Dependency graph view

CHAPTER 4

RESULTS AND DISSCUTION

In the previous chapter we discussed our approach theoretically. In this chapter we present the results we have obtained by applying our methodology on some other software projects. The results will be presented in two subsections and the next subsection will contain discussions.

4.1 Refused Bequest's Results and Discussion :

Our target in this level is to examine all the sub-classes in the test project's source code. The key idea behind the proposed detection technique lies in the detection of whether the average similarity of the overridden methods in a subclass in a given hierarchy is less than the threshold value. In addition, our approach takes other metrics into account to detect and evaluate the suspicious classes. The first metric is Base Class Overriding Ratio (BOvR) which measures the ratio of the number of overridden methods from the base class to the total number of methods in the tested sub-class. This gives us an idea about the amount of reused code from the parent class. The other factor that may have an effect on the inheritance properties is the Average Cyclomatic Complexity of Overridden methods (ACCO). In normal cases; when the sub-class inherits and overrides the methods these methods be like the original version but with the variables or parameters of the sub-class, or adding additional functionality related with the nature of the sub-class. So; the ACCO should be near to the upper limit of the standard threshold value.

Our test is done on open source Java project Sweet Home 3D 4.0, as mentioned in section 3.3 of chapter 3. We presented the general code properties in Table 3.3, but the general metrics for each separated hierarchy is illustrated in Table 4.1. The class diagrams of some hierarchies which indicated as Refused Bequest candidate will be presented in later sections.

Table 4.1 The Metrics of each hierarchy

Hierarchy Number	Hierarchy Root Name	Root Class Type	Number of Classes	DIT
1	HomeApplication	Abstract	3	1
2	FurnitureController	Class	2	1
3	NullableSpinnerNumberModel	Class	2	1
4	VisualTransferHandler	Class	2	1
5	AutoCommitSpinner	Class	2	1
6	ResourceAction	Class	2	1
7	Camera	Class	2	1
8	TexturesCatalog	Class	2	1
9	FurnitureCatalog	Class	2	1
10	RecorderException	Class	2	1
11	UserPreferencesChangeListener	Abstract	3	1
12	AbstractPhotoController	Abstract	3	1
13	HomeController	Class	3	2
14	FrameGenerator	Class	3	1
15	PieceOfFurnitureTopView	Abstract	3	1
16	LocatedTransferHandler	Abstract	3	1
17	ScaledImageComponent	class	3	1
18	FileContentManager	Class	3	1
19	CatalogPieceOfFurniture	Class	3	1
20	CameraControllerState	Abstract	3	1
21	PointWithAngleMagnetism	Class	3	1
22	URLContent	Class	4	1
23	UserPreferences	Abstract	4	1
24	HomePieceOfFurniture	Class	4	1
25	WizardController	Abstract	4	1
26	ModifiedPieceOfFurniture	Class	4	1
27	Object3DBranch	Abstract	5	1
28	ModelPreviewComponents	Class	5	2
29	AbstarctDecoratedAction	Class	5	2
30	WizardControllerStepState	Abstract	4	1
31	BackgroundImageWizardStepState	Abstract	4	1
32	ImportFurnitureWizardStepState	Abstract	5	1
33	ImportedTextureWizardStepState	Abstract	3	1
34	ControllerState	Abstract	27	2

Table 4.1 (cont'd)

35	**AbstarctModeChangeState**	**Abstract**	**6**	**1**
36	AbstarctWallState	Abstract	3	1
37	AbstarctRoomState	Abstract	3	1

According to [33], when a designer employs a generalization relationship and places an abstract class or an interface on the root of the hierarchy, the intention is rather clear: The goal is to apply the Dependency Inversion Principle and essentially to allow polymorphic behavior where the public interface of the base abstract class (or interface) is implemented by a corresponding subclass. In these cases it is theoretically impossible to encounter a Refused Bequest symptom, since the same benefit cannot be achieved by other means. In other words, it is clearly evident that the employed generalization is on purpose, well-designed and constitutes an "is-a" relationship. Also, the methods in the interfaces have no body of code to compare with overridden version in the sub-class. On the other hand, overriding the abstract methods of the abstract super-class is mandatory, so the sub-class will be obligated to inherit and reuse these methods. In this case there will be no big chance to refuse its parent's bequest. So, we excluded the hierarchies in which the base class is either an interface or an abstract class.

To facilitate dealing with classes' names during the analysis phase we proposed to give appropriate latter to each class, start from C1 to C55 as illustrated in Table 4.2.

Table 4.2 Original and Given Names of the Classes

No.	Original class Name	Given Letter
1	FurnitureController	C1
2	PlanController	C2
3	NullableSpinnerNumberModel	C3
4	NullableSpinnerLengthModel	C4
5	VisualTransferHandler	C5
6	FurnitureCatalogTransferHandler	C6
7	AutoCommitSpinner	C7
8	NullableSpinner	C8
9	ResourceAction	C9
10	ControllerAction	C10
11	Camera	C11
12	ObserverCamera	C12
13	TexturesCatalog	C13
14	DefaultTextureCatalog	C14
15	FurnitureCatalog	C15
16	DefaultFurnitureCatalog	C16
17	RecorderException	C17
18	InterruptedRecorderException	C18
19	HomeController	C19
20	HomePluginController	C20
21	HomeAppletController	C21
22	ScaledImageComponent	C22
23	ScaledImagePreviewComponent	C23
24	OriginalImagePreviewComponent	C24
25	FileContentManager	C25
26	AppletContentManager	C26
27	FileContentManagerWithRecordedLastDirectories	C27
28	CatalogPieceOfFurniture	C28
29	CatalogDoorOrWindow	C29
30	CatalogLight	C30
31	PointWithAngleMagnetism	C31
32	WallPointWithAngleMagnetism	C32
33	RoomPointWithAngleMagnetism	C33
34	URLContent	C34
35	HomeURLContent	C35
36	ResourceURLContent	C36

Table 4.2 (cont'd)

37	TemporaryURLContent	C37
38	HomePieceOfFurniture	C38
39	HomeDoorOrWindow	C39
40	HomeFurnitureGroup	C40
41	HomeLight	C41
42	ModifiedPieceOfFurniture	C42
43	ModifiedDoorOrWindow	C43
44	ModifiedLight	C44
45	ModifiedFurnitureGroup	C45
46	ModelPreviewComponents	C46
47	AbstractModelPreviewComponent	C47
48	RotationPreviewComponent	C48
49	AttributePreviewcomponent	C49
50	IconPreviewcomponent	C50
51	AbstarctDecoratedAction	C51
52	PopupMenueItemAction	C52
53	MenueItemAction	C53
54	ToolBarAction	C54
55	Buttonaction	C55

The calculation of the ASM metric was done manually on our test code. It had been started from the calculation of similarity between methods concept [41] [42]. In this study we focused on the similarity between the original methods in the base class and the overridden methods in the sub-class. The similarity between methods for some hierarchies of the tested code is shown in the following Tables (Table 4.3- 4.16). Colors indicate overridden methods and their base versions. For example, one method of a base class denoted with m1 is overridden by the method denoted by m*1 and both methods have the color green.

Table 4.3 Similarity between methods for hierarchy 2

C1-C2	Original Methods	
Overridden Methods	m1	m2
m*1	0.5	1
m*2	0	0

From Table 4.3 we can see that m1 is overridden by (m*1) with similarity equal to 0. 5 and m2 is overridden by (m*2) with similarity also equal to 0. The ASM for C2 is equal to 0.25.

Table 4.4 Similarity between methods for hierarchy 7

C11-C12	Original Methods							
Overridden Methods	m1	m2	m3	m4	m5	m6	m7	m8
m*1	**0.25**							
m*2	0.25	**1**						
m*3	0.25	1	**1**					
m*4	0.167	0.33	0.33	**1**				
m*5	0.2	0.5	0.5	0.25	**1**			
m*6	0.2	0.5	0.5	0.25	0.33	**1**		
m*7	0.2	0.5	0.5	0.25	0.33	0.33	**1**	
m*8	Null	Null	Null	Null	Null	Null	Null	**Null**

In the previous Table there are 8 overridden methods most of them has a high similarity values except (m*8) and (m8) have no instance variables. The ASM for C12 is equal to 0.893.

Table 4.5 Similarity between methods for hierarchy 13

C19-C21	Original Methods			
Overridden Methods	**m1**	**m2**	**m3**	**m4**
m*1	**0.667**			
m*2	0.4	**1**		
m*3	0.33	0	**0**	
m*4	0.33	0.167	0	**0.5**

From Table 4.5 we can see that the sub-class C21 has four overridden methods from the base class C19 with ASM equal to 0.542.

Table 4.6 Similarity between methods for hierarchy 18

C22-C23	Original Methods
Overridden Methods	m1
m*1	1

C22-C24	Original Methods
Overridden Methods	m1
m*1	1

In the hierarchy number 18 there are two sub-classes **C23** and **C24** with one overridden method in each of them. The **ASM** has a value of **1** for both sub-classes.

Table 4.7 Similarity between methods for hierarchy 19

C25-C26	Original Methods			
Overridden Methods	m1	m2	m3	m4
m*1	1			
m*2	0	1		
m*3	0	0	0.333	
m*4	0	0	0.333	0.5

C25-C27	Original Methods	
Overridden Methods	m1	m2
m*1	Null	Null
m*2	Null	Null

In the hierarchy 19 there are two sub-classes **C26** and **C27**. In **C26** there are four overridden methods with **ASM** equal to **0,708**. While in **C27** there are two overridden methods but they don't use the instance variables. So, the result is Null.

Table 4.8 Similarity between Methods for Hierarchy 20

C28-C29	Original Methods
Overridden Methods	m1
m*1	Null

C28-C30	Original Methods
Overridden Methods	NA

Hierarchy 20 also has two sub-classes C29 and C30. C29 has one overridden method with ASM equal to Null; because the overridden methods do not have instance variables. C30 has no overridden methods, so the ASM metric is not applicable in this case and has a value of NA.

Table 4.9 Similarity between Methods for Hierarchy 22

C31-C32	Original Methods
Overridden Methods	NA

C31-C33	Original Methods
Overridden Methods	NA

From Table 4.9 we can see that we have two sub-classes **C32** and **C33**. These sub-classes don't have any overridden methods from their parent class and the value of **ASM** is **NA**.

Table 4.10 Similarity between Methods for Hierarchy 23

C34-C35	Original Methods
Overridden Methods	NA

C34-C36	Original Methods
Overridden Methods	NA

C34-C37	Original Methods
Overridden Methods	NA

Hierarchy 23 has 3 sub-classes **C35**, **C36** and **C37**. All of them have no overridden methods and the **ASM** equal to **NA**.

Table 4.11 Similarity between methods for hierarchy 25-C39

C38-C39	Original Methods					
Overridden Methods	m1	m2	m3	m4	m5	m6
m*1	1					
m*2	0	-				
m*3	0	0	1			
m*4	0	0	0	1		
m*5	0	0	0	0	0.333	
m*6	0	0	0	0	0	0.333

Table 4.12 Similarity between methods for hierarchy 25-C41

C38-C41	Original Methods			
Overridden Methods	**m1**	**m2**	**m3**	**m4**
m*1	**1**			
m*2	0.5	**0.5**		
m*3	0.5	0.2	**1**	
m*4	0.5	0.2	1	**1**

Table 4.13 Similarity between methods for hierarchy 25-C40

C39-C40	Original Methods																	
Overridden Methods	m1	m5	m10	m11	m12	m13	m14	m15	m16	m17	m18	m19	m20	m21	m22	m23	m24	m25
m*1	**1**																	
m*5	0	**1**																
m*10	0	0	**0**															
m*11	0	0	0	**1**														
m*12	0.3	0	0	0.5	**1**													
m*13	0	0	0	0	0	**1**												
m*14	0	0	0	0	0	0	**1**											
m*15	0	0	0	0	0	0	0	**1**										
m*16	0	0	0	0	0	0.5	0	**0**	**0.5**									
m*17	0	0	0	0	0	0	0	0	0	**1**								
m*18	0	0	0	0	0	0	0	0	0	0	**0**							
m*19	0	0	0	0	0	0	0	0	0	0	0	**0**						
m*20	0	0	0	0	0	0	0	0	0	0	0	0	**0**					
m*21	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**0**				
m*22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**			
m*23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**		
m*24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	
m*25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**

In hierarchy 25 we have three sub-classes C39, C40 and C41. C39 has six overridden methods with ASM equal to 0.611. C40 has eighteen overridden methods with ASM equal to 0.694 and C41 has four overridden methods with ASM value equal to 0.875.

Table 4.14 Similarity between methods for hierarchy 27

C42-C43	Original Methods
Overridden Methods	m1
m*1	1

C42-C44	Original Methods
Overridden Methods	m1
m*1	1

C42-C45	Original Methods
Overridden Methods	m1
m*1	1

We can see in hierarchy 27 there are three sub-classes with one overridden method in each and with ASM value of 1.

Table 4.15 Similarity between methods for hierarchy 29-C47, 48, 49, 50

C46-C47	Original Methods
Overridden Methods	NA

C46-C48	Original Methods
Overridden Methods	m1
m*1	Null

C46-C49	Original Methods
Overridden Methods	NA

C46-C50	Original Methods	
Overridden Methods	m1	m2
m*1	0	2
m*2	1	1

For hierarchy 29 we have four sub-classes C47, C48, C49 and C50. C47 and C49 have no overridden methods with ASM equal to NA. C48 has one overridden method but it doesn't use the instance variables, so the ASM equal to Null. C50 has two overridden methods and the ASM equal to 0.5.

Table 4.16 Similarity between methods for hierarchy 30

C51-C52	Original Methods
Overridden Methods	m1
m*1	**1**

C51-C53	Original Methods
Overridden Methods	m1
m*1	**1**

C51-C54	Original Methods
Overridden Methods	m1
m*1	**1**

C51-C55	Original Methods
Overridden Methods	m1
m*1	**1**

Finally, hierarchy 31 has four sub-classes **C52**, **C53**, **C54** and **C55**. Each one has one overridden method with **ASM** equal to 1.

During the calculations of the ASM metrics for all the hierarchies in our test project, we noticed that there are different cases for it. There are two important factors that have great effect in the application of this metric. These are the availability of instance variables in the base class and the availability of overridden methods in the sub-class. In some cases this metric is not applicable because either no overridden methods in the sub-class or there are no instance variables in the base class. Table 4.17 shows the details of these cases.

Table 4.17 ASM metric cases

Case No.	IVs	Overridden Methods in the Sub-class	ASM Status
Case -1-	Available	Available	Applicable
Case -2-	Available	Not available	NA
Case -3-	Not available	Available	NA
Case -4-	Available	Available(no IVs in the overridden methods)	Applicable = Null
Case -5-	Available (Not used)	Available(no IVs in the overridden methods	Applicable = Null

The Null value means that ASM metric has met its requirements and it is applicable, but it has no value. This state can be happened if the overridden methods don't use the instance variables or the original methods in the base class don't use the available instance variables.

The other two metrics ACCO and BOvR are collected and combined with the ASM for our test code and the results are shown in Table 4.18.

Table 4.18 Metrics Values of the Hierarchies

Hierarchy Number	Hierarchy	Class Type	ASM	ACCO	BOvR
1	**FurnitureController**	**Base-Class**	-	-	-
	-PlanController	sub-class	0.25	2.2	0.0089
2	**NullableSpinnerNumberModel**	**Base-Class**	-	-	-
	-NullableSpinnerLengthModel	sub-class	NA (case-2-)	0	0
3	**VisualTransferHandler**	**Base-Class**	-	-	-
	-FurnitureCatalogTransferHandler	sub-class	NA (case-2-)	Null	0
4	**AutoCommitSpinner**	**Base-Class**	-	-	-
	-NullableSpinner	sub-class	NA (case-3-)	Null	0
5	**ResourceAction**	**Base-Class**	-	-	-
	-ControllerAction	sub-class	NA (case-5-)	1	0.333
6	**Camera**	**Base-Class**	-	-	-
	-ObserverCamera	sub-class	0.893	1.167	0.4
7	**TexturesCatalog**	**Base-Class**	-	-	-
	-DefaultTextureCatalog	sub-class	NA (case-2-)	Null	0
8	**FurnitureCatalog**	**Base-Class**	-	-	-
	-DefaultFurnitureCatalog	sub-class	NA (case-2-)	Null	0
9	**RecorderException**	**Base-Class**	-	-	-
	-InterruptedRecorderException	sub-class	NA (case-2-)	Null	0
10	**HomeController**	**Base-Class**	-	-	-
	-HomePluginController	Sub-class	NA (case-2-)	Null	0
	-HomeAppletController	Sub-class	0.542	1.25	0.667
11	**ScaledImageComponent**	**Base-Class**	-	-	-
	-ScaleImagePreviewComponent	Sub-class	1	2	0.25
	-OriginalImagePreviewComponent	Sub-class	1	2	0.25
12	**FileContentManager**	**Base-Class**	-	-	-
	-AppletContentManager	Sub-class	0.708	4	0.667
	-FileContentManagerWithRecordedLastDirectories	Sub-class	Null (case-4-)	4.5	0.667
13	**CatalogPieceOfFurniture**	**Base-Class**	-	-	-
	-CatalogDoorOrWindow	Sub-class	Null (case-4-)	1.5	0.1
	-CatalogLight	Sub-class	NA (case-2-)	2	0

Table 4.18 (cont'd)

		Class			
	-WallPointWithAngleMagnetism	Sub-class	NA (case-2-)	Null	0
	-RoomPointWithAngleMagnetism	Sub-class	NA (case-2-)	Null	0
15	**URLContent**	**Base-Class**	-	-	-
	-HomeURLContent	Sub-class	NA (case-2-)	Null	0
	-ResourceURLContent	Sub-class	NA (case-2-)	Null	0
	-TemporaryURLContent	Sub-class	NA (case-2-)	Null	0
16	**HomePieceOfFurniture**	**Base-Class**	-	-	-
	-HomeDoorOrWindow	Sub-class	0.611	1.71	0.5
	-HomeFurnitureGroup	Sub-class	0.694	1.97	0.4186
	-HomeLight	Sub-class	0.875	1.2	0.5
17	**ModifiedPieceOfFurniture**	**Base-Class**	-	-	-
	-ModifiedDoorOrWindow	Sub-class	1	1	1
	-ModifiedLight	Sub-class	1	1	1
	-ModifiedFurnitureGroup	Sub-class	1	3	0.5
18	**ModelPreviewComponents**	**Base-Class**			
	-AbstractModelPreviewComponent	Inner-class	NA (case-2-)	Null	0
	-RotationPreviewComponent	Sub-class	Null (case-4-)	Null	0.125
	-AttributePreviewcomponent	Sub-class	NA (case-2-)	Null	0
	-IconPreviewcomponent	Sub-class	0.5	1	1
19	**AbstarctDecoratedAction**	**Base-Class**	-	-	-
	-PopupMenueItemAction	Sub-class	1	6	1
	-MenueItemAction	Sub-class	1	5	1
	-ToolBarAction	Sub-class	1	2	1
	-Buttonaction	Sub-class	1	3	1

The cases where the ASM metric of case -2-, which means the sub-class is not recommended being Refused Bequest candidate, because it accepts all the inherited bequests of its parent class without any specialization or any change in its functionality. This can be considered as a sign of design problem or abuse for the inheritance. In case of the ASM is applicable we need to see the metrics and compare the results with our

threshold values. The following Table shows us the Refused Bequest candidate classes and we will explain each case separately with its class diagram.

Table 4.19 Refused bequest candidate classes

No.	Sub-Class Name	ASM	ACCO	BOvR
1.	NullableSpinnerLengthModel	NA	Null	0
2.	NullableSpinner	NA	Null	0
3.	ControllerAction	Null	1	0.333
4.	DefaultTextureCatalog	NA	0	0
5.	DefaultFurnitureCatalog	NA	0	0
6.	InterruptedRecorderException	NA	0	0
7.	HomePluginController	NA	0	0
8.	CatalogLight	NA	2	0
9.	WallPointWithAngleMagnetism	NA	Null	0
10.	RoomPointWithAngleMagnetism	NA	Null	0
11.	HomeURLContent	NA	Null	0
12.	ResourceURLContent	NA	Null	0
13.	TemporaryURLContent	NA	Null	0
14.	RotationPreviewComponent	Null	Null	0.125
15.	AttributePreviewcomponent	NA	Null	0
16.	FurnitureCatalogTransferHandler	NA	0	0
17.	FileContentManagerWithRecordedLastDirectories	Null	4.5	0.667
18.	CatalogDoorOrWindow	Null	1.5	0.1
19.	AbstractModelPreviewComponent	NA	0	0
20.	PlanController	0.25	2.2	0.0089

In order to clarify the obtained results and make the results more clear; we will discuss the ASM cases and explain one example from each case by using the class diagram.

- **Case -1-** : in this case the two main requirements to calculate the ASM metric was met. We can see that clearly in the hierarchy number 11 from Table 4.18, as shown in the class diagram in Figure 4.1.

Figure 4.1 Case-1- class diagram

Figure 4.1 shows us that the base class `ScaledImageComponent` has two instance variables, which is one of the requirements to apply the ASM metric. For the first sub-class `ScaleImagePreviewComponent` there is one overridden method from its base class `paintComponent ()`. The ASM of the sub-class is equal to 1. Also, for the second sub-class has an ASM equal to 1. Depending on our approach, that means they applied the inheritance principle in a good way.

- **Case -2-:** In this case one of the requirements needed for ASM calculation is missing. There are no overridden methods in the sub-class. This can be considered as the sub-class utilizes all bequests from its parent. An example of this case is shown in Figure 4.2. As we can see that the sub-class `FurnitureCatalogTransferHandler` doesn't reused any overridden methods from its parent.

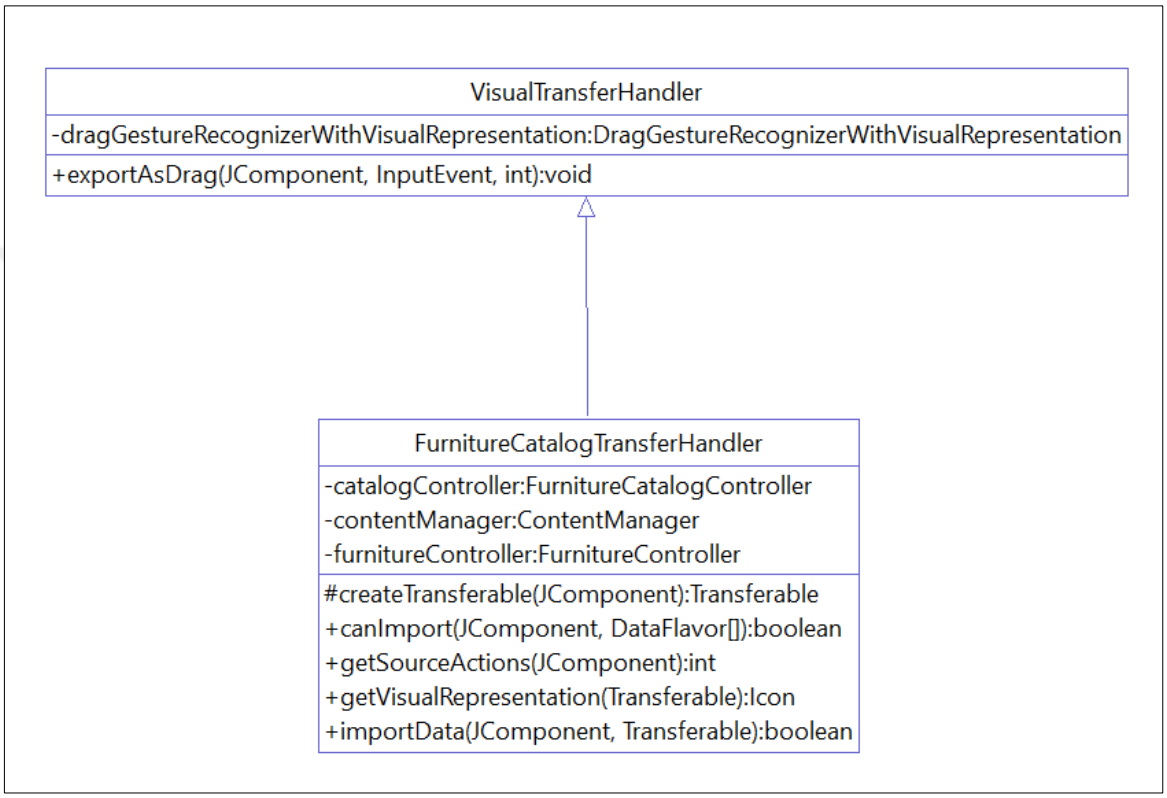

Figure 4.2 Case-2- class diagram

- **Case -3-:** this case missed the availability of instance variables in the base class. That means even if there is an overridden method it will be impossible to compute the ASM metric. We can see an example in the sub-class `AutoCommitSpinner` in figure 4.3.

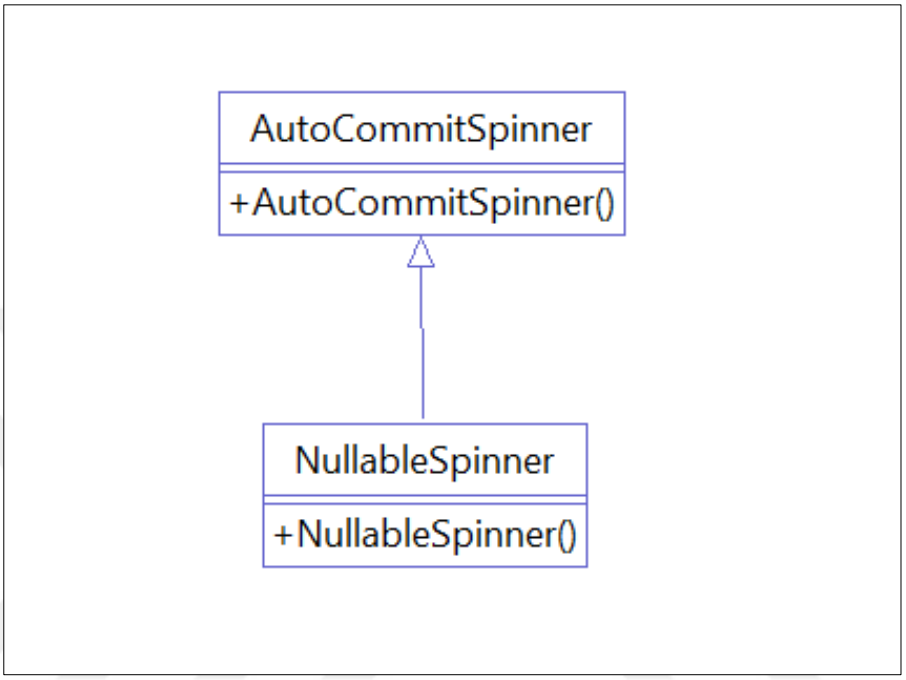

Figure 4.3 Case-3- class diagram

- **Case -4- :** This case is a special case of case-1-, where the two requirements are available but we couldn't obtain a real value for our metric. This case happens when there are overridden methods in the sub-class but they don't use the instance variables of the base class neither directly nor indirect way.

Figure 4.4 show us the sub-class `FileContentManagerWithRecordedLastDirectories` which overridden the method `getLastDirectory ()` and `setLastDirectory ()`, but they don't use any instance variable. In this case the ratio of intersection set of the instance variables to the set of union will be empty set. We proposed to put a Null as a value for the ASM in this case.

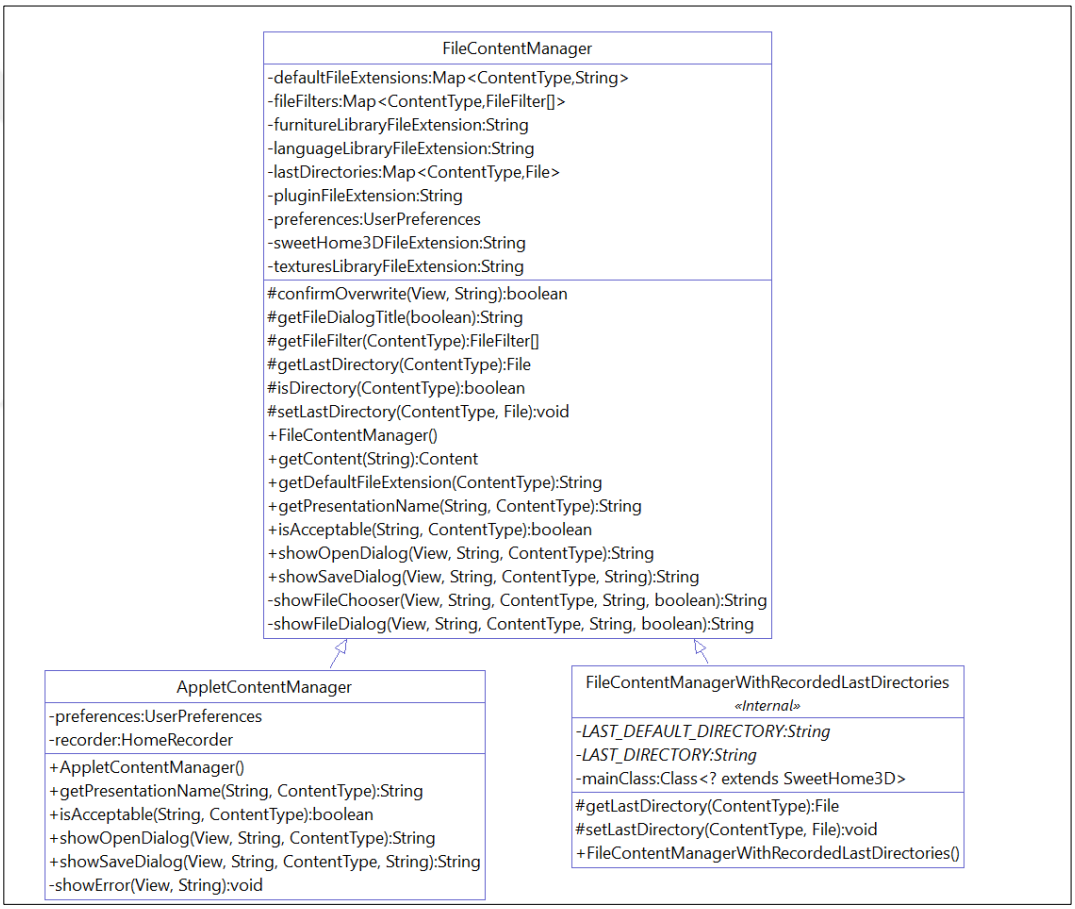

Figure 4.4: Case-4- class diagram

- Case -5- : In the last case, we indicated that the original methods of the base class don't use the instance variables of its class. So, the intersection and the union sets will be empty. Consequently, this will effect on the calculation of our ASM metric. In this case we proposed to give a Null value.

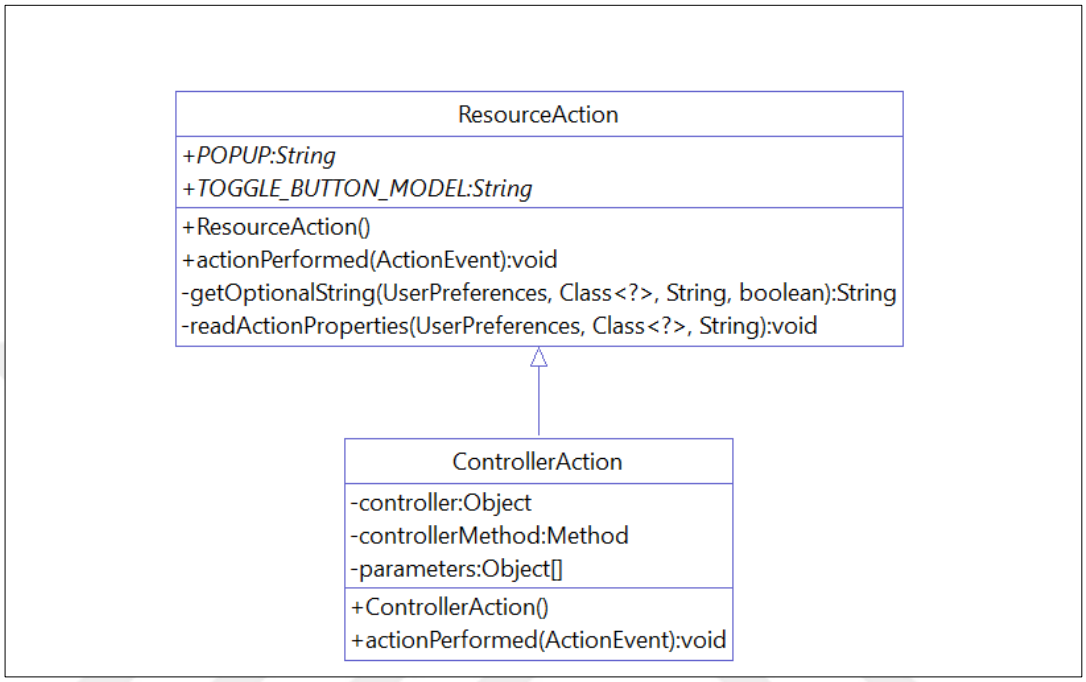

Figure 4.5: Case-5- class diagram

Figure 4.5 shows us that sub-class `ControllerAction` overridden the method `actionPerformed ()`, but when we investigated the code of the base class and the sub-class we found that no instance variables used in the original version of the method.

It is important to mention that we don't use the zero value in case -4- and case-5-; because this value is given when the two requirements of the ASM metric are met but there is no similarity between methods.

When we analysed the results in Table 4.19, we noticed that some cases of Refused Bequest classes even that their ASM value are Null or NA they have a clear values for the ACCO metric. We can conclude from that the sub-class in this case doesn't override methods from its parent, while it overrides methods from other classes or interfaces.

Another important note, some sub-classes have a Null or NA values, while the BOvR metric has a numerical value not equal to zero. This can be translated as; even there is no similarity between the methods in the base class with the overridden version in the child class but they apply the case-4- and case-5- where the main requirements are met but the application was in wrong way.

We evaluated our results with another two tools. These tools are iPlasma [38] and Ptidej tool [46]. The results are shown in Table 4.21. All the instances detected by Ptidej tool as Refused Bequest candidates are detected by our approach. While not all the candidates detected by our metrics are able to be detected by Ptidej and iPlasma tools. These results give as a high confidence with our proposed metrics and our detection mechanism. The number of detected instances for each tool is shown in the following Table.

Table 4.20: number of detected instances in each tool

Our Approach	Ptidej	iPlasma
6	13	1

Table 4.21 Results of different tools

NO.	Sub-Class Name	Our Approach	Ptidej	iPlasma	Is Code Smell?
1.	PlanController	✓	✓	×	✓
2.	NullableSpinner	✓	✓	×	✓
3.	ControllerAction	✓	✓	×	✓
4.	DefaultTexturesCatalog	×	✓	×	×
5.	DefaultFurnitureCatalog	×	✓	✓	×
6.	InterruptedRecorderException	×	✓	×	×
7.	HomePluginController	×	✓	×	×
8.	FileContentManagerWithRecordedLastDirectories	✓	×	×	✓
9.	CatalogDoorOrWindow	✓	✓	×	×
10.	CatalogLight	×	✓	×	×
11.	HomeURLContent	×	✓	×	×
12.	ResourceURLContent	×	✓	×	×
13.	TemporaryURLContent	×	✓	×	×
14.	HomeDoorOrWindow	×	×	×	×
15.	RotationPreviewComponent	✓	×	×	✓
16.	HomeAppletController	×	✓	×	×

4.2 Feature Envy's Results and Discussion:

At this level our aim is to search about a method which is interested in the data of another class more than the class it belongs to. It happens when the source of the envied data comes from only one or two classes. This refers to method misplacing. During the metrics collecting phase we grouped the source code of our test code as methods and then we obtained the metrics for each method. After completing the previous phase, now the data set is ready to filter by our threshold values on the data. Our methods and metrics data set initially has 4660 methods. We proposed to filter them by removing the constructors. Depending on the nature of the constructors it just used to initialize the attributes of the class it belongs to, so it rarely to find a constructor is a Feature Envy candidate. After applying this filter the data set became 3335 methods. The final results are shown in Table 4.22.

Table 4.22 Results of feature envy

No.	Method Name	ATFD	LAA	FDP
1	addAreaSidesGeometry	8	0	1
2	getAreaOnFloor	7	0	1
3	updateView	6	0	1
4	updateViewPlatformTransform	5	0	1
5	updateWall	4	0	1
6	paintComponent	4	0.43	1
7	computeTransform	5	0.29	1
8	updateShininessRadioButtons	4	0	1
9	propertyChange	7	0.12	1
10	paintRoomNameOffsetIndicator	5	0	1
11	paintRoomAreaOffsetIndicator	4	0	1
12	paintWallsOutline	7	0	1
13	paintWallResizeIndicator	11	0	1
14	paintPieceOFFurnitureIndicators	8	0	1
15	paintDimensionLineResizeIndicator	5	0	1
16	paintLabels	5	0.17	1
17	paintWallAlignmentFeedback	16	0.11	1
18	equalsWallPoint	4	0	1
19	paintDimensionLineAlignmentFeedback	16	0.16	1
20	equalsDimensionLinePoint	4	0	1
21	getPageFormat	7	0	1
22	setBackFaceShown	6	0.4	1
23	getValueAt	5	0.17	1
24	createComponents	6	0	1
25	createComponents	7	0	1

Table 4.22 (cont'd)

26	compareCameraLocation	7	0	1
27	getApplicationOrLibraryUpdateMessage	8	0	1
28	updateProperties	8	0.2	1
29	storeCamera	9	0.1	1
30	alignPieceOfFurnitureAlongSides	5	0	1
31	doReverseWallsDirection	16	0	1
32	splitSelectedWall	10	0.17	1
33	getReferenceWall	4	0	1
34	adjustPieceOfFurnitureSideBySideAt	37	0.08	1
35	isIntersectionEmpty	10	0	1
36	isIntersectionEmpty	5	0	1
37	joinNewWallEndToWall	4	0	1
38	getPieceOfFurnitureRotatedNameAt	7	0.12	1
39	moveItems	6	0	1
40	moveWallStartPoint	11	0	1
41	moveWallEndPoint	11	0	1
42	reverseDimensionLine	5	0	1
43	doAddWalls	5	0.29	1
44	postWallResize	4	0.2	1
45	postDimensionLineResize	4	0.2	1
46	moveWallPoints	10	0	1
47	pressMouse	23	0	1
48	setMode	5	0	1
49	setMode	5	0	1
50	moveMouse	4	0.33	1
51	moveMouse	8	0.2	1
52	moveMouse	5	0.44	1
53	setMode	5	0	1
54	enter	5	0.17	1
55	pressMouse	5	0.17	1
56	getDimensionLineAngle	4	0.33	1
57	moveMouse	4	0.2	1
58	setMode	5	0	1
59	moveMouse	4	0.33	1
60	moveMouse	4	0.33	1
61	moveMouse	4	0.33	1
62	moveMouse	4	0.33	1
63	equalsWallPoint	4	0	1
64	setMode	6	0	1
65	getWallAngleInDegrees	10	0	1
66	showWallAngleFeedback	27	0	1

We can see from the Table above that there are duplicated methods. These methods have the same name but they are in different classes. They are overridden methods for an abstract class `ControllerState`. The same thing is with `setMode ()` method and from the same abstract class.

If we compared the results of the ATFD and LAA metrics (FDP is fixed for all) we can see that when the value of LAA increase (greater than 0.2) the value of ATFD decreased. But; when it reaches near the zero value the value of ATFD increased.

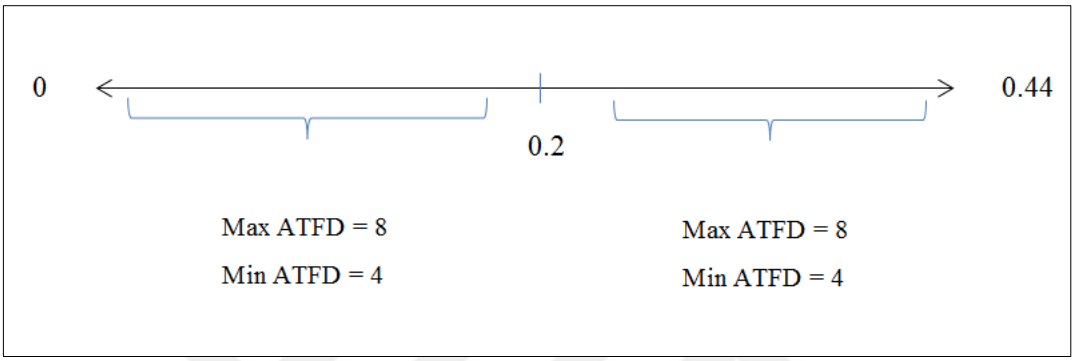

Figure 4.6 Relation between ATFD and LAA metrics

We selected two other tools to detect Feature Envy methods. These tools are iPlasma tool [38] and JDeodorant eclipse plugin [47].The number of detected Feature Envy cases for each tool is available in the following Table.

Table 4.23 Number of detected cases in each tool

Our Approach	JDeodorant	iPlasma
66	87	149

Table 4.24 Results of different tools

No.	Method Name	Our Approach	iPlasma	JDeodorant	Is Code Smell?
1	addAreaSidesGeometry ()	✓	✓	✓	✓
2	getAreaOnFloor ()	✓	✓	✓	✓
3	updateViewn ()	✓	✓	✓	✓
4	updateViewPlatformTransform ()	✓	✓	✓	✓
5	updateWall ()	✓	✓	×	✓
6	paintComponent ()	✓	×	×	×
7	computeTransform ()	✓	✓	×	✓
8	updateShininessRadioButtons ()	✓	✓	×	✓
9	propertyChange ()	✓	✓	×	✓
10	paintRoomNameOffsetIndicator ()	✓	✓	✓	✓
11	paintRoomAreaOffsetIndicator ()	✓	✓	✓	✓
12	paintWallsOutline ()	✓	✓	×	✓
13	paintWallResizeIndicator ()	✓	✓	×	✓
14	paintPieceOFFurnitureIndicators ()	✓	✓	×	✓
15	paintDimensionLineResizeIndicator ()	✓	✓	×	✓
16	paintLabels ()	✓	✓	×	✓
17	paintWallAlignmentFeedback ()	✓	✓	×	✓
18	equalsWallPoint ()	✓	✓	✓	✓
19	paintDimensionLineAlignmentFeedback()	✓	✓	×	✓
20	equalsDimensionLinePoint()	✓	✓	✓	✓
21	getPageFormat()	✓	✓	×	✓
22	setBackFaceShown()	✓	×	×	×
23	getValueAt()	✓	✓	×	✓
24	createComponents()	✓	✓	×	✓
25	createComponents ()	✓	✓	×	✓
26	compareCameraLocation()	✓	✓	✓	✓
27	getApplicationOrLibraryUpdateMessage()	✓	✓	×	✓
28	updateProperties()	✓	✓	×	✓
29	storeCamera()	✓	✓	✓	✓
30	alignPieceOfFurnitureAlongSides()	✓	✓	✓	✓
31	doReverseWallsDirection()	✓	✓	×	✓
32	splitSelectedWall()	✓	✓	×	✓
33	getReferenceWall()	✓	✓	×	✓
34	adjustPieceOfFurnitureSideBySideAt()	✓	✓	×	✓
35	isIntersectionEmpty()	✓	✓	✓	✓
36	isIntersectionEmpty()	✓	✓	✓	✓
37	joinNewWallEndToWall()	✓	✓	✓	✓
38	getPieceOfFurnitureRotatedNameAt()	✓	✓	×	✓
39	moveItems()	✓	✓	×	✓
40	moveWallStartPoint()	✓	✓	×	✓
41	moveWallEndPoint()	✓	✓	×	✓
42	reverseDimensionLine()	✓	✓	✓	✓
43	doAddWalls()	✓	✓	✓	✓
44	postWallResize()	✓	✓	×	✓
45	postDimensionLineResize()	✓	✓	×	✓
46	moveWallPoints()	✓	✓	×	✓

Table 4.24 (cont'd)

47	pressMouse()	✓	✓	✕	✓
48	setMode()	✓	✓	×	✓
49	setMode()	✓	✓	×	✓
50	moveMouse()	✓	✓	×	✓
51	moveMouse()	✓	✓	×	✓
52	moveMouse()	✓	✓	×	✓
53	setMode()	✓	✓	×	✓
54	enter()	✓	✓	×	✓
55	pressMouse()	✓	✓	×	✓
56	getDimensionLineAngle()	✓	×	✓	✓
57	moveMouse()	✓	✓	×	✓
58	setMode()	✓	✓	×	✓
59	moveMouse()	✓	✓	×	✓
60	moveMouse()	✓	×	×	×
61	moveMouse()	✓	×	×	×
62	moveMouse()	✓	×	×	×
63	equalsWallPoint()	✓	✓	✓	✓
64	setMode()	✓	×	×	×
65	getWallAngleInDegrees()	✓	✓	×	✓
66	showWallAngleFeedback()	✓	✓	×	✓
67	DimensionLine()	×	×	✓	×
68	getDoorOrWindowShapeAtWallIntersection()	×	✓	✓	✓
69	getDoorOrWindowSashShape()	×	✓	✓	✓
70	addSelectObjectMenuItems()	×	×	✓	×
71	moveDimensionLinePoint()	×	×	✓	×
72	getTextureCoordinates()	×	×	✓	×
73	getSunDirection()	×	×	✓	×
74	getPieceBoundingRectangleWidth()	×	✓	✓	✓
75	getPieceBoundingRectangleHeight()	×	✓	✓	✓
76	computeRoomBorderGeometry()	×	✓	✓	✓
77	getFurnitureComparator()	×	×	✓	×
78	toggleCameraSelection()	×	×	✓	×
79	isPieceOfFurniturePartOfBasePlan()	×	×	✓	×
80	sortFurniture()	×	×	✓	×
81	addComponent3DRenderingErrorObserver()	×	×	✓	×
82	getOptionalLocalizedString()	×	×	✓	×
83	createNavigationPanel()	×	×	✓	×
84	setPlanRulersVisible()	×	×	✓	×
85	cloneHomeInEventDispatchThread()	×	×	✓	×
86	getOptionalString()	×	×	✓	×
87	getTextures()	×	×	✓	×
88	getMinX()	×	×	✓	×
89	getMaxX()	×	×	✓	×
90	getMinY()	×	×	✓	×
91	getMaxY()	×	×	✓	×
92	getObserverCameraMinimumElevation()	×	×	✓	×
93	getRoomSideLength()	×	×	✓	×
94	getRoomSideAngle()	×	×	✓	×

Table 4.24 (cont'd)

95	getPaintedItems()	✕	✕	✓	✕
96	moveHomeItemsToLevel()	✕	✕	✓	✕
97	alignPieceOfFurnitureAlongLeftOrRightSides()	✕	✓	✓	✓
98	updateOpenRecentHomeMenu()	✕	✕	✓	✕
99	doAddFurniture()	✕	✕	✓	✕
100	doToggleBackgroundImageVisibility()	✕	✕	✓	✕
101	isPieceOfFurnitureVisibleAtSelectedLevel()	✕	✕	✓	✕
102	getDetecTableRoomsAtSelectedLevel()	✕	✕	✓	✕
103	getDetecTableWallsAtSelectedLevel()	✕	✕	✓	✕
104	postPieceOfFurnitureWidthAndDepthResize()	✕	✓	✓	✓
105	selectLevelFromSelectedItems()	✕	✕	✓	✕
106	computeRoomPartGeometry()	✕	✕	✓	✕
107	getHeaderRenderer()	✕	✕	✓	✕
108	createLockUnlockBasePlanButton()	✕	✕	✓	✕
109	addColorListener()	✕	✕	✓	✕
110	addIconYawListener()	✕	✕	✓	✕
111	savePhoto()	✕	✕	✓	✕
112	deleteLastRecordedCameraLocation()	✕	✕	✓	✕
113	doDeleteFurniture()	✕	✕	✓	✕
113	toggleFurnitureSort()	✕	✕	✓	✕
114	toggleFurnitureVisibleProperty()	✕	✕	✓	✕
115	writePreferences()	✕	✕	✓	✕
116	deleteCameras()	✕	✕	✓	✕
117	doDeleteWalls()	✕	✕	✓	✕
118	doAddRooms()	✕	✕	✓	✕
119	doDeleteRooms()	✕	✕	✓	✕
120	doAddDimensionLines()	✕	✕	✓	✕
121	doDeleteDimensionLines()	✕	✕	✓	✕
122	doAddLabels()	✕	✕	✓	✕
123	doDeleteLabels()	✕	✕	✓	✕
124	postPieceOfFurnitureHeightResize()	✕	✓	✓	✓
125	getVisibleItemsAtSelectedLevel()	✕	✕	✓	✕
126	doDeleteItems()	✕	✕	✓	✕
127	addSizeListeners()	✕	✕	✓	✕
128	goToCamera()	✕	✕	✓	✕
129	alignFurnitureSideBySide()	✕	✓	✓	✓
130	addRooms()	✕	✕	✓	✕
131	createWall()	✕	✕	✓	✕
132	selectItems()	✕	✕	✓	✕
133	addDimensionLines()	✕	✕	✓	✕
134	addLabels()	✕	✕	✓	✕

We examined and analysed some cases of our detected results as illustrated in this section.

Looking at our test code (the code of the methods is available in appendix B) we find the method `addAreaSidesGeometry` in the class `Ground3D` is a Feature Envy candidates. It doesn't use data from its definition class with LAA equal to zero. The method invokes much more methods and access fields from the class `HomeTexture` than its own class with value of ATFD equal to 8.

We can see another case in our test code. The method `updateViewPlatformTransform` is available in the class `HomeComponent3D`. From our obtained metrics we can see that it is using data of the class `Camera` more than its own class with ATFD equal to 5. Also, it doesn't use any data from the class it belongs to.

We have another case from our test code. There is a method `adjustPieceOfFurnitureSideBySideAt` is allocated in the class `PlanController`. It has 37 attributes from the class `HomePieceOfFurniture` with ATFD equal to 37, while the number of attributes from its class is equal to 3 with LAA of 0. 08.

When we analysed the relation between the metrics as shown in Table 4.25, it is so clear that ATFD and FDP is related positively (their values increase together as a linear relationship). While the relation between LAA and the other two metrics is negatively related. That means when the value of LAA metric increases the other two metrics should be decreased linearly.

Table 4.25 Correlation between metrics

	ATFD	**LAA**	**FDP**
ATFD	1		
LAA	-0.5805	1	
FDP	0.709703	-0.75583	1

LAA metric has an important role in the detection of Feature Envy candidates. It is the ratio of the local data used by the method from its class to the total number of data used including data from foreign class. We noticed that all methods with LAA equal to zero are undetectable by JDeodorant plugin, while they are detectable by our approach and iPlasma tool. This is a defect point in the use of JDeodorant tool. In other hand, iPlasma tool detected Feature Envy methods more than the other two approaches, while they may not be actual Feature Envy instances. The ambiguity of the threshold values is the main reason for these cases. It used meaningful thresholds and this is not accurate in the final results. We enhanced this approach by computing numerical threshold values depending on statistical formula. The comparison with clear values is easier and more accurate. These values can be set a standard for the future work.

4.3. General Discussion

This thesis work proposes a method for detecting Feature Envy code smell and another method for detecting Refused Bequest code smell. Code smell detection is challenging because mapping metrics to code smell symptoms is not always straightforward task. These mappings can lead to false-negative and false-positive detections. A manual code review is needed to validate detected code smell instances but manual review can also miss some true-positive code smell cases.

For the results in Table 4.21 and Table 4.24, we computed the confusion matrices and we found the accuracy for each tool.

Table 4.26 Confusion matrix of our approach results

		Actual Refused Bequest	
		Refused Bequest	**NOT Refused Bequest**
Our Approach	**Refused Bequest**	5	1
	Not Refused Bequest	0	10

Table 4.27 Confusion matrix of ptidej tool results

		Actual Refused Bequest	
		Refused Bequest	**NOT Refused Bequest**
Ptidej Tool	**Refused Bequest**	3	10
	Not Refused Bequest	2	1

Table 4.28 Confusion matrix of iPlasma tool results

		Actual Refused Bequest	
		Refused Bequest	**NOT Refused Bequest**
iPlasma Tool	**Refused Bequest**	0	1
	Not Refused Bequest	5	10

Table 4.29 Accuracy table of refused bequest results

	Our Approach	**iPlasma Tool**	**Ptidej Tool**
Accuracy	0. 9375	0.625	0. 25
Recall	1	0	0. 6
Miss.Rate	0. 0625	0.375	0.75
Fall out	0. 09	0.091	0. 91

Table 4.30 Confusion matrix of our approach

		Actual Feature Envy	
		Feature Envy	**NOT Feature Envy**
Our Approach	**Feature Envy**	62	4
	Not Feature Envy	9	60

Table 4.31 Confusion matrix of iPlasma tool

		Actual Feature Envy	
		Feature Envy	**NOT Feature Envy**
iPlasma	**Feature Envy**	68	0
	Not Feature Envy	3	63

Table 4.32 Confusion matrix of JDeodorant tool

		Actual Feature Envy	
		Feature Envy	**NOT Feature Envy**
JDeodorant	**Feature Envy**	27 TP	60 FP
	Not Feature Envy	41 FN	6 TN

Table 4.33 Accuracy table of feature envy results

	Our Approach	**iPlasma Tool**	**JDeodorant Tool**
Accuracy	0. 91	0. 978	0. 246
Recall	0. 873	0. 958	0. 397
Miss.Rate	0. 097	0. 022	0. 754
Fall out	0. 0625	0	0. 909

We compared the results in Table 4.32, that our approach has the highest accuracy among the other tools. Which means it is able to detect the actual Refused Bequest candidates precisely and with misclassification error equal to 0.0625. Our approach is more accurate than the other tools .iPlasma tool came at the second place with accuracy equal to 0.625. It has a high error rate equal to 0.375. The Ptidej tool is the lowest accuracy with value of 0.25 and with highest error rate value reach to 0.75.

Table 4.33 show us the accuracy for the tools used to detect Feature Envy code smell. The highest accuracy is for iPlasma tool with value of 0.978. Our approach came in the second place with accuracy equal to 0.91, while the JDeodorant tool has the lowest accuracy value equal to 0.246.

For a decisive comparison of different proposals of code smell and anti-pattern detection, a comprehensive data set is needed but to the best of our knowledge, such a set has not been documented.

Our proposed code smell detection methods have not been implemented yet as software tools. After their implementation, more complex software projects can be examined and further comparison with other code smell detection methods can be done. In their current states, our methods can identify some code smell instances that are not identified by other tools as seen in Sections 4.1 and 4.2.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Code smell detection is an extremely challenging task, because the available detection tools do not deal with all types of code smells. That makes the task of assuring high quality software more difficult for the developers. In this thesis we proposed a metric based detection technique of Refused Bequest and Feature Envy smells. Only a few studies had been made to detect Refused Bequest type of code smell. Also, we built a standard threshold values for the detection of Feature Envy code smell by analyzing six open source Java projects.

We tried to diagnose Refused Bequest sub-classes by investigating more details about the symptoms that we can convert to an appropriate metrics. We focused in our study on the similarity measure between the overriding methods with the original ones and finding the Average similarity for the sub-class under inspection. This helps us to detect this type of code smells precisely. In other hand, we used our threshold values to detect Feature Envy methods. These values are calculated precisely and when we compared them with actual needs they are acceptable values. The validation of our approach is based on a Java project of 367 classes.

As a plan for our future work, we will work to detect more code smells and build a fully automated tool for the analysis and detection processes. After such an implementation, we will use more complex software for validation. Also, we plan to use other techniques to find an accurate threshold values and compare the results with the current work.

For the detection of code smells we will investigate the ability to use machine learning algorithms to detect types of code smell and compare the results with the current work.

Our goal is to cooperate with all researchers in this field all over the world to put standard criteria in the detection of each code smell and build a standard tool able to analyze, detect and refactor all code smells types in all object oriented programming languages. Since the first definition of code smell types by Fowler in 1979 until now there are no standard rules or unified tool to do that. We hope to integrate all these offers and produce one global tool. That will help the developers to develop and maintain their software projects in short time and with high quality. This is an important mission in order to close the folder of such problems.

REFERENCES

[1] McCall, J. A., Richards, P. K., and Walters, G. F. (1977). Factors in Software Quality: Concept and Definitions of Software Quality. Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, New York, 1(3).

[2] International Organization for Standardization, and International Electro Technical Commission. (2001). Software Engineering-Product Quality: Quality model. ISO/IEC 9126.1.

[3] Henry, S., Humphrey, M., and Lewis, J. (1990)."Evaluation of the Maintainability of Object-Oriented Software". In Computer and Communication Systems, 1990 IEEE Region 10 Conference on: pp. 404-409.

[4] Zou, Y. (2005). "Quality Driven Software Migration Of Procedural Code to Object-Oriented Design. In Software Maintenance". ICSM'05. Proceedings of the 21st IEEE International Conference: 709-713.

[5] Shalloway, A., Trott, J. (2004), Design Patterns Explained: A New Perspective on Object-Oriented Design, 2nd edition, Addison-Wesley.

[6] Lanza, M. and Marinescu, R. (2007). Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Springer.

[7] Abreu, F. B., and Melo, W. (1996). "Evaluating The Impact of Object-Oriented Design on Software Quality". In Software Metrics Symposium, Proceedings of the 3rd International: 90-99.

[8] Kumar, P., and Singh, S. K. (2016). "A Comprehensive Evaluation of Aspect-Oriented Software Quality (AOSQ) Model Using Analytic Hierarchy Process (AHP) Technique". In Advances in Computing, Communication, and Automation (ICACCA)(Fall), International Conference: 1-7.

[9] Boehm, B. W., Brown, J. R., and Lipow, M. (1976)."Quantitative Evaluation of Software Quality". In Proceedings of the 2nd international conference on Software Engineering: 592-605.

[10] ISO25000, http://iso25000.com, access date: 24-07-2017 .

[11] Rodríguez, M., and Piattini, M. (2014). "Software Product Quality Evaluation Using ISO/IEC 25000". ERCIM NEWS: 39-40.

[12] Shaik, A., Manda, B., Prakashini, C., Deepthi, K., and Reddy, C. R. K. (2010). "Metrics for Object Oriented Design Software Systems: A Survey". Journal of Emerging Trends in Engineering and Applied Sciences, 1(2): 190-198.

[13] Yadav, V., and Singh, R. (2012). "Validating Object Oriented Design Quality Using Software Metrics". In at International Conference on Advances in Electronics, Electrical and Computer Science Engineering-EEC.

[14] R.S.Pressman, (1997).Software Engineering-A Practioners Approach. Fourth Edition, McGraw Hill International Edition.

[15] Sharma, A., and Dubey, S. K. (2012)."Comparison of Software Quality Metrics for Object-Oriented Oriented System". IJCSMS International Journal of Computer Science and Management Studies, 12: 2231 –5268.

[16] Albrecht, A. J., and Gaffney, J. E. (1983). "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation". IEEE transactions on software engineering, (6): 639-648.

[17] DeMarco, T. (1978).Structured Analysis and System Specification, New York: Yourdon.

[18] Myers, G. J. (1977). "An Extension to the Cyclomatic Measure of Program Complexity". ACM Sigplan Notices, 12(10): 61-64.

[19] Henry, S., and Kafura, D. (1981). "Software Structure Metrics Based on Information Flow". IEEE transactions on Software Engineering, (5): 510-518.

[20] Chidamber, S. R., and Kemerer, C. F. (1994). "A Metrics Suite for Object Oriented Design". IEEE Transactions on software engineering, 20(6): 476-493.

[21] Fowler, M., and Beck, K. (1999). Refactoring: Improving the Design of Existing Code, Addison-Wesley Professional.

[22] N. Moha, Y-G. Gueheneuc, L. Duchien, and A-F. Le Meur, (2010). "DECOR: A Method for the Specification and Detection of Code and Design Smells", IEEE Trans. Software Eng., 36(1): 20-36.

[23] Mens, T., and Tourwé, T. (2004). "A Survey of Software Refactoring", IEEE Transaction on Software Engineering, 30(2): 126-139.

[24] Suryanarayana, G., Samarthyam, G., and Sharma, T. (2014). Refactoring for Software Design Smells: Managing technical debt, Morgan Kaufmann.

[25] Mantyla, M., Vanhanen, J., & Lassenius, C. (2003)." A Taxonomy and an Initial Empirical Study of Bad Smells in Code". In Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference: 381-384.

[26] Vlissides, J., Helm, R., Johnson, R., and Gamma, E. (1995). Design patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 49(120):11.

[27] Palomba, F., De Lucia, A., Bavota, G., and Oliveto, R. (2015)."Anti-Pattern Detection: Methods, Challenges, and Open Issues". Advances in Computers, 95:201-238.

[28] Slinger, S. (2005)."Code Smell Detection in Eclipse". Delft University of Technology. Netherlands.

[29] Marinescu, R. (2004)." Detection Strategies: Metrics-Based Rules for Detecting Design Flaws". Software Maintenance Proceedings. 20th IEEE International Conference: 350-359.

[30] Mihancea, P. F. (2006). "Towards A Client Driven Characterization of Class Hierarchies". 14th IEEE International Conference: 285-294.

[31] Fard, A. M., and Mesbah, A. (2013)." JSNOSE: Detecting JavaScript Code Smells". In Source Code Analysis and Manipulation (SCAM), IEEE 13th International Working Conference: 116-125.

[32] Tourwé, T., and Mens, T. (2003)." Identifying Refactoring Opportunities Using Logic Meta Programming". In Software Maintenance and Reengineering. Proceedings. Seventh European Conference: 91-100.

[33] Ligu, E., Chatzigeorgiou, A., Chaikalis, T., and Ygeionomakis, N. (2013). "Identification of Refused Bequest Code Smells". In Software Maintenance (ICSM), 29th IEEE International Conference: 392-395.

[34] Tsantalis, N., and Chatzigeorgiou, A. (2009). "Identification of Move Method Refactoring Opportunities". IEEE Transactions on Software Engineering, 35(3): 347-367.

[35] Sales, V., Terra, R., Miranda, L. F., and Valente, M. T. (2013). "Recommending Move Method Refactorings Using Dependency Sets". In Reverse Engineering (WCRE), 20th Working Conference: 232-241.

[36] Oliveto, R., Gethers, M., Bavota, G., Poshyvanyk, D., and De Lucia, A. (2011). "Identifying Method Friendships to Remove the Feature Envy Bad Smell (NIER track)". In Proceedings of the 33rd International Conference on Software Engineering: 820-823.

[37] Dexun, J., Peijun, M., Xiaohong, S., and Tiantian, W. (2012). "Detecting Bad Smells with Weight Based Distance Metrics Theory". In Instrumentation, Measurement, Computer, Communication and Control (IMCCC), Second International Conference: 299-304.

[38] Qualitas Corpus Index: Release (20130901):http://qualitascorpus.com/docs/catalogue/20130901/corpus-catalogue-evolution.html, access date: 01-04-2017.

[39] LOOSE Research Group website, http://www.loose.upt.ro/reengineering, access date: 07-05-2017.

[40] Kan, S. H. (2002).Metrics and Models in Software Quality Engineering, Addison-Wesley Longman Publishing.

[41] Bonja, C., and Kidanmariam, E. (2006). "Metrics for Class Cohesion and Similarity Between Methods". In Proceedings of the 44th annual Southeast regional conference: 91-95.

[42] Riel, A. J. (1996). Object-Oriented Design Heuristics, Addison-Wesley.

[43] Sweet Home 3D 4.0 releases. http://www.sweethome3d.com/, access date: 12-08-2017.

[44] Sauer, F. (2013). Eclipse metrics plugin 1.3.6, http://metrics.sourceforge.net/. Access date: 20-08-2017.

[45] Marinescu, C., Marinescu, R., Mihancea, P. F., and Wettel, R. (2005). "iPlasma: An Integrated Platform for Quality Assessment of Object-Oriented Design". In ICSM (Industrial and Tool Volume).

[46] Ptidej: Pattern Trace Identification, Detection, and Enhancement in Java, http://www.ptidej.net/tools/. Access date: 28-10-2017.

[47] Concordia University, https://users.encs.concordia.ca/~nikolaos/jdeodorant/. Access date: 07-10-2017.

APPENDIX-A

FULL PATH TABLES

The full path Tables of all methods mentioned in chapter 4 are listed in this appendix.

Table -1- Full path of our approach methods results

Method Name	Full Path
addAreaSidesGeometry	com.eteks.sweethome3d.j3d.Ground3D.addAreaSidesGeometry()
getAreaOnFloor	com.eteks.sweethome3d.j3d.ModelManager.getAreaOnFloor()
updateView	com.eteks.sweethome3d.swing.HomeComponent3D.updateView()
updateViewPlatformTransform	com.eteks.sweethome3d.swing.HomeComponent3D.updateViewPlatformTransform()
updateWall	com.eteks.sweethome3d.swing.HomeComponent3D.updateWall(l)
updateShininessRadioButtons	com.eteks.sweethome3d.swing.HomeFurniturePanel.updateShininessRadioButtons()
propertyChange	HomePane.FocusOwnerChangeListener.propertyChange()
paintRoomNameOffsetIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintRoomNameOffsetIndicator()
paintRoomAreaOffsetIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintRoomAreaOffsetIndicator()
paintWallsOutline	com.eteks.sweethome3d.swing.PlanComponent.paintWallsOutline()
paintWallResizeIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintWallResizeIndicator()
paintPieceOFFurnitureIndicators	com.eteks.sweethome3d.swing.PlanComponent.paintPieceOFFurnitureIndicators()
paintDimensionLineResizeIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintDimensionLineResizeIndicator()
paintLabels	com.eteks.sweethome3d.swing.PlanComponent.paintLabels()
paintWallAlignmentFeedback	com.eteks.sweethome3d.swing.PlanComponent.paintWallAlignmentFeedback()
equalsWallPoint	com.eteks.sweethome3d.swing.PlanComponent.equalsWallPoint()
paintDimensionLineAlignmentFeedback	com.eteks.sweethome3d.swing.PlanComponent.paintDimensionLineAlignmentFeedback()
equalsDimensionLinePoint	com.eteks.sweethome3d.swing.PlanComponent.equalsDimensionLinePoint()
getPageFormat	com.eteks.sweethome3d.swing.HomePrinTableComponent.getPageFormat()
getValueAt	LevelPanel.LevelsTableModel.getValueAt()
createComponents	com.eteks.sweethome3d.swing.PhotoSizeAndQualityPanel.createComponents()
createComponents	com.eteks.sweethome3d.swing.RoomPanel.createComponents()
compareCameraLocation	com.eteks.sweethome3d.swing.VideoPanel.compareCameraLocation()
getApplicationOrLibraryUpdateMessage	com.eteks.sweethome3d.viewcontroller.HomeController.getApplicationOrLibraryUpdateMessage()
storeCamera	com.eteks.sweethome3d.viewcontroller.HomeController3D.storeCamera()
alignPieceOfFurnitureAlongSides	com.eteks.sweethome3d.viewcontroller.FurnitureController.alignPieceOfFurnitureAlongSides()

Table -1- (cont'd)

doReverseWallsDirection	com.eteks.sweethome3d.viewcontroller.PlanController.doReverseWallsDirection()
splitSelectedWall	com.eteks.sweethome3d.viewcontroller.PlanController.splitSelectedWall()
getReferenceWall	com.eteks.sweethome3d.viewcontroller.PlanController.getReferenceWall()
adjustPieceOfFurnitureSideBySideAt	com.eteks.sweethome3d.viewcontroller.PlanController.adjustPieceOfFurnitureSideBySideAt()
isIntersectionEmpty	com.eteks.sweethome3d.viewcontroller.PlanController.isIntersectionEmpty()
isIntersectionEmpty	com.eteks.sweethome3d.viewcontroller.PlanController.isIntersectionEmpty()
joinNewWallEndToWall	com.eteks.sweethome3d.viewcontroller.PlanController.joinNewWallEndToWall()
getPieceOfFurnitureRotatedNameAt	com.eteks.sweethome3d.viewcontroller.PlanController.getPieceOfFurnitureRotatedNameAt()
moveItems	com.eteks.sweethome3d.viewcontroller.PlanController.moveItems()
moveWallStartPoint	com.eteks.sweethome3d.viewcontroller.PlanController.moveWallStartPoint()
moveWallEndPoint	com.eteks.sweethome3d.viewcontroller.PlanController.moveWallEndPoint()
reverseDimensionLine	com.eteks.sweethome3d.viewcontroller.PlanController.reverseDimensionLine()
moveWallPoints	com.eteks.sweethome3d.viewcontroller.WallController.moveWallPoints()
pressMouse	PlanController.SelectionState.pressMouse()
setMode	PlanController.PanningState.setMode()
setMode	PlanController.WallDrawingState.setMode()
setMode	PlanController.DimensionLineDrawingState.setMode()
enter	PlanController.DimensionLineDrawingState.enter()
pressMouse	PlanController.DimensionLineDrawingState.pressMouse()
setMode	PlanController.RoomDrawingState.setMode()
equalsWallPoint	PlanController.WallPointWithAngleMagnetism.equalsWallPoint()
setMode	PlanController.AbstractModeChangeState.setMode()
getWallAngleInDegrees	PlanController.AbstractWallState.getWallAngleInDegrees()
showWallAngleFeedback	PlanController.AbstractWallState.showWallAngleFeedback()

Table -2- Full path of iPlasma methods results

Method Name	Full Path
addAreaSidesGeometry	com.eteks.sweethome3d.j3d.Ground3D.addAreaSidesGeometry()
getAreaOnFloor	com.eteks.sweethome3d.j3d.ModelManager.getAreaOnFloor()
computeRoomBorderGeometry	com.eteks.sweethome3d.j3d.Room3D.computeRoomBorderGeometry()
updateView	com.eteks.sweethome3d.swing.HomeComponent3D.updateView()
updateViewPlatformTransform	com.eteks.sweethome3d.swing.HomeComponent3D.updateViewPlatformTransform()
updateWall	com.eteks.sweethome3d.swing.HomeComponent3D.updateWall()
addShadowOnFloor	com.eteks.sweethome3d.swing.HomeComponent3D.addShadowOnFloor()
createActions	com.eteks.sweethome3d.swing.HomePane.createActions()
createTransferHandlers	com.eteks.sweethome3d.swing.HomePane.createTransferHandlers()
createImportModifyBackgroundImageAction	com.eteks.sweethome3d.swing.HomePane.createImportModifyBackgroundImageAction()
createHideShowBackgroundImageAction	com.eteks.sweethome3d.swing.HomePane.createHideShowBackgroundImageAction()
propertyChange	FurnitureCatalogListPanel.PreferencesChangeListener.propertyChange()
getToolTipText	com.eteks.sweethome3d.swing.FurnitureCatalogTree.getToolTipText()
moveCamera	HomeComponent3D.CameraInterpolator.moveCamera()
computeTransform	HomeComponent3D.CameraInterpolator.computeTransform()
updateShininessRadioButtons	com.eteks.sweethome3d.swing.HomeFurniturePanel.updateShininessRadioButtons()
propertyChange	HomePane.FocusOwnerChangeListener.propertyChange()
print	com.eteks.sweethome3d.swing.PlanComponent.print()
paintRoomsNameAndArea	com.eteks.sweethome3d.swing.PlanComponent.paintRoomsNameAndArea()
paintRoomNameOffsetIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintRoomNameOffsetIndicator()
paintRoomAreaOffsetIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintRoomAreaOffsetIndicator()
paintWallsOutline	com.eteks.sweethome3d.swing.PlanComponent.paintWallsOutline()
paintWallResizeIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintWallResizeIndicator()
getWallAreas	com.eteks.sweethome3d.swing.PlanComponent.getWallAreas()
getDoorOrWindowShapeAtWallIntersection	com.eteks.sweethome3d.swing.PlanComponent.getDoorOrWindowShapeAtWallIntersection()
getDoorOrWindowSashShape	com.eteks.sweethome3d.swing.PlanComponent.getDoorOrWindowSashShape()
paintFurnitureName	com.eteks.sweethome3d.swing.PlanComponent.paintFurnitureName()
paintPieceOfFurnitureIcon	com.eteks.sweethome3d.swing.PlanComponent.paintPieceOfFurnitureIcon()
paintPieceOFFurnitureIndicators	com.eteks.sweethome3d.swing.PlanComponent.paintPieceOFFurnitureIndicators()
paintDimensionLines	com.eteks.sweethome3d.swing.PlanComponent.paintDimensionLines()
paintDimensionLineResizeIndicator	com.eteks.sweethome3d.swing.PlanComponent.paintDimensionLineResizeIndicator()
paintLabels	com.eteks.sweethome3d.swing.PlanComponent.paintLabels()
paintCompass	com.eteks.sweethome3d.swing.PlanComponent.paintCompass()
paintCompassOutline	com.eteks.sweethome3d.swing.PlanComponent.paintCompassOutline()
paintWallAlignmentFeedback	com.eteks.sweethome3d.swing.PlanComponent.paintWallAlignmentFeedback()
equalsWallPoint	com.eteks.sweethome3d.swing.PlanComponent.equalsWallPoint()
paintDimensionLineAlignmentFeedback	com.eteks.sweethome3d.swing.PlanComponent.paintDimensionLineAlignmentFeedback()
equalsDimensionLinePoint	com.eteks.sweethome3d.swing.PlanComponent.equalsDimensionLinePoint()
paintCamera	com.eteks.sweethome3d.swing.PlanComponent.paintCamera()
getPageFormat	com.eteks.sweethome3d.swing.HomePrinTableComponent.getPageFormat()
getValueAt	LevelPanel.LevelsTableModel.getValueAt()
setMaterialAt	ModelMaterialsPanel.MaterialsListModel.setMaterialAt()
createComponents	com.eteks.sweethome3d.swing.PhotoSizeAndQualityPanel.createComponents()
showWallAngleFeedback	PlanController.AbstractWallState.showWallAngleFeedback()

Table-2- (cont'd)

createComponents	com.eteks.sweethome3d.swing.RoomPanel.createComponents()
collectionChanged	TexturePanel.TexturesCatalogListener.collectionChanged()
createComponents	com.eteks.sweethome3d.swing.UserPreferencesPanel.createComponents()
compareCameraLocation	com.eteks.sweethome3d.swing.VideoPanel.compareCameraLocation()
getVideoFramesPath	com.eteks.sweethome3d.swing.VideoPanel.getVideoFramesPath()
enableDefaultActions	com.eteks.sweethome3d.viewcontroller.HomeController.enableDefaultActions()
addNotUndoableModificationListeners	com.eteks.sweethome3d.viewcontroller.HomeController.addNotUndoableModificationListeners()
getApplicationOrLibraryUpdateMessage	com.eteks.sweethome3d.viewcontroller.HomeController.getApplicationOrLibraryUpdateMessage()
updateProperties	com.eteks.sweethome3d.viewcontroller.UserPreferencesController.updateProperties()
storeCamera	com.eteks.sweethome3d.viewcontroller.HomeController3D.storeCamera(java.lang.String)
updateProperties	com.eteks.sweethome3d.viewcontroller.CompassController.updateProperties()
alignFurnitureSideBySide	com.eteks.sweethome3d.viewcontroller.FurnitureController.alignFurnitureSideBySide()
alignPieceOfFurnitureAlongSides	com.eteks.sweethome3d.viewcontroller.FurnitureController.alignPieceOfFurnitureAlongSides()
alignPieceOfFurnitureAlongLeftOrRightSides	com.eteks.sweethome3d.viewcontroller.FurnitureController.alignPieceOfFurnitureAlongLeftOrRightSides()
getPieceBoundingRectangleWidth	com.eteks.sweethome3d.viewcontroller.FurnitureController.getPieceBoundingRectangleWidth()
getPieceBoundingRectangleHeight	com.eteks.sweethome3d.viewcontroller.FurnitureController.getPieceBoundingRectangleHeight()
undoAlignFurniture	com.eteks.sweethome3d.viewcontroller.FurnitureController.undoAlignFurniture()
updateProperties	com.eteks.sweethome3d.viewcontroller.Home3DAttributesController.updateProperties()
doReverseWallsDirection	com.eteks.sweethome3d.viewcontroller.PlanController.doReverseWallsDirection()
splitSelectedWall	com.eteks.sweethome3d.viewcontroller.PlanController.splitSelectedWall()
getReferenceWall	com.eteks.sweethome3d.viewcontroller.PlanController.getReferenceWall()
getDimensionLinesAlongWall	com.eteks.sweethome3d.viewcontroller.PlanController.getDimensionLinesAlongWall()
adjustPieceOfFurnitureElevation	com.eteks.sweethome3d.viewcontroller.PlanController.adjustPieceOfFurnitureElevation()
adjustPieceOfFurnitureSideBySideAt	com.eteks.sweethome3d.viewcontroller.PlanController.adjustPieceOfFurnitureSideBySideAt()
isIntersectionEmpty	com.eteks.sweethome3d.viewcontroller.PlanController.isIntersectionEmpty()
isIntersectionEmpty	com.eteks.sweethome3d.viewcontroller.PlanController.isIntersectionEmpty()
joinNewWallEndToWall	com.eteks.sweethome3d.viewcontroller.PlanController.joinNewWallEndToWall()
getRoomRotatedNameAt	com.eteks.sweethome3d.viewcontroller.PlanController.getRoomRotatedNameAt()
getPieceOfFurnitureRotatedNameAt	com.eteks.sweethome3d.viewcontroller.PlanController.getPieceOfFurnitureRotatedNameAt()
getRotatedLabelAt	com.eteks.sweethome3d.viewcontroller.PlanController.getRotatedLabelAt()
moveItems	com.eteks.sweethome3d.viewcontroller.PlanController.moveItems()
moveWallStartPoint	com.eteks.sweethome3d.viewcontroller.PlanController.moveWallStartPoint()
moveWallEndPoint	com.eteks.sweethome3d.viewcontroller.PlanController.moveWallEndPoint()
reverseDimensionLine	com.eteks.sweethome3d.viewcontroller.PlanController.reverseDimensionLine()
doAddWalls	com.eteks.sweethome3d.viewcontroller.PlanController.doAddWalls()
postPieceOfFurnitureMove	com.eteks.sweethome3d.viewcontroller.PlanController.postPieceOfFurnitureMove()
postWallResize	com.eteks.sweethome3d.viewcontroller.PlanController.postWallResize()
postPieceOfFurnitureHeightResize	com.eteks.sweethome3d.viewcontroller.PlanController.postPieceOfFurnitureHeightResize()
postPieceOfFurnitureWidthAndDepthResize	com.eteks.sweethome3d.viewcontroller.PlanController.postPieceOfFurnitureWidthAndDepthResize()
postPieceOfFurnitureResize	com.eteks.sweethome3d.viewcontroller.PlanController.postPieceOfFurnitureResize()
postDimensionLineResize	com.eteks.sweethome3d.viewcontroller.PlanController.postDimensionLineResize()
updateProperties	com.eteks.sweethome3d.viewcontroller.LevelController.updateProperties()
updateProperties	com.eteks.sweethome3d.viewcontroller.AbstractPhotoController.updateProperties()
moveWallPoints	com.eteks.sweethome3d.viewcontroller.WallController.moveWallPoints()

Table-2- (cont'd)

moveCamera	HomeController3D.ObserverCameraState.moveCamera()
updateAerialViewBounds	HomeController3D.TopCameraState.updateAerialViewBounds()
enter	PlanController.SelectionState.enter()
pressMouse	PlanController.SelectionState.pressMouse()
toggleMagnetism	PlanController.SelectionMoveState.toggleMagnetism()
enter	PlanController.RectangleSelectionState.enter()
setMode	PlanController.PanningState.setMode()
enter	PlanController.DragAndDropState.enter()
toggleMagnetism	PlanController.WallCreationState.toggleMagnetism()
setMode	PlanController.WallDrawingState.setMode()
toggleMagnetism	PlanController.WallDrawingState.toggleMagnetism()
toggleMagnetism	PlanController.WallResizeState.toggleMagnetism()
enter	PlanController.PieceOfFurnitureRotationState.enter()
toggleMagnetism	PlanController.PieceOfFurnitureRotationState.toggleMagnetism()
enter	PlanController.PieceOfFurnitureElevationState.enter()
moveMouse	PlanController.PieceOfFurnitureElevationState.moveMouse()
toggleMagnetism	PlanController.PieceOfFurnitureElevationState.toggleMagnetism()
enter	PlanController.PieceOfFurnitureHeightState.enter()
toggleMagnetism	PlanController.PieceOfFurnitureHeightState.toggleMagnetism()
enter	PlanController.PieceOfFurnitureResizeState.enter()
toggleMagnetism	PlanController.PieceOfFurnitureResizeState.toggleMagnetism()
enter	PlanController.LightPowerModificationState.enter()
moveMouse	PlanController.PieceOfFurnitureNameRotationState.moveMouse()
toggleMagnetism	PlanController.PieceOfFurnitureNameRotationState.toggleMagnetism()
enter	PlanController.CameraPitchRotationState.enter()
moveMouse	PlanController.CameraPitchRotationState.moveMouse()
enter	PlanController.CameraElevationState.enter()
toggleMagnetism	PlanController.DimensionLineCreationState.toggleMagnetism()
setMode	PlanController.DimensionLineDrawingState.setMode(e)
enter	PlanController.DimensionLineDrawingState.enter()
pressMouse	PlanController.DimensionLineDrawingState.pressMouse()
toggleMagnetism	PlanController.DimensionLineDrawingState.toggleMagnetism()
enter	PlanController.DimensionLineResizeState.enter()
toggleMagnetism	PlanController.DimensionLineResizeState.toggleMagnetism()
moveMouse	PlanController.DimensionLineOffsetState.moveMouse()
toggleMagnetism	PlanController.RoomCreationState.toggleMagnetism()
setMode	PlanController.RoomDrawingState.setMode()
toggleMagnetism	PlanController.RoomDrawingState.toggleMagnetism()
toggleMagnetism	PlanController.RoomResizeState.toggleMagnetism()
toggleMagnetism	PlanController.RoomAreaRotationState.toggleMagnetism()
toggleMagnetism	PlanController.RoomNameRotationState.toggleMagnetism()
toggleMagnetism	PlanController.LabelRotationState.toggleMagnetism()
enter	PlanController.CompassRotationState.enter()
enter	PlanController.CompassResizeState.enter()
equalsWallPoint	PlanController.WallPointWithAngleMagnetism.equalsWallPoint()

Table-2- (cont'd)

setMode	PlanController.AbstractModeChangeState.setMode()
deleteSelection	PlanController.AbstractModeChangeState.deleteSelection()
getWallAngleInDegrees	PlanController.AbstractWallState.getWallAngleInDegrees()

Table -3- Full path of JDeodorant tool methods results

Method Name	Full Path
reverseDimensionLine()	com.eteks.sweethome3d.viewcontroller.PlanController::reverseDimensionLine()
joinNewWallEndToWall()	com.eteks.sweethome3d.viewcontroller.PlanController::joinNewWallEndToWall()
getDoorOrWindowShapeAtWallIntersection()	com.eteks.sweethome3d.swing.PlanComponent::getDoorOrWindowShapeAtWallIntersection()
compareCameraLocation()	com.eteks.sweethome3d.swing.VideoPanel::compareCameraLocation()
getDoorOrWindowSashShape()	com.eteks.sweethome3d.swing.PlanComponent::getDoorOrWindowSashShape()
addSelectObjectMenuItems()	com.eteks.sweethome3d.swing.HomePane::addSelectObjectMenuItems()
equalsWallPoint()	com.eteks.sweethome3d.swing.PlanComponent::equalsWallPoint()
equalsDimensionLinePoint()	com.eteks.sweethome3d.swing.PlanComponent::equalsDimensionLinePoint()
moveDimensionLinePoint()	com.eteks.sweethome3d.viewcontroller.PlanController::moveDimensionLinePoint()
equalsWallPoint()	com.eteks.sweethome3d.viewcontroller.PlanController.WallPointWithAngleMagnetism::equalsWallPoint()
getTextureCoordinates()	com.eteks.sweethome3d.j3d.HomePieceOfFurniture3D::getTextureCoordinates()
getSunDirection()	com.eteks.sweethome3d.j3d.PhotoRenderer::getSunDirection()
getPieceBoundingRectangleWidth()	com.eteks.sweethome3d.viewcontroller.FurnitureController::getPieceBoundingRectangleWidth()
getPieceBoundingRectangleHeight()	com.eteks.sweethome3d.viewcontroller.FurnitureController::getPieceBoundingRectangleHeight()
addAreaSidesGeometry()	com.eteks.sweethome3d.j3d.Ground3D::addAreaSidesGeometry()
computeRoomBorderGeometry()	com.eteks.sweethome3d.j3d.Room3D::computeRoomBorderGeometry()
getFurnitureComparator()	com.eteks.sweethome3d.swing.FurnitureTable.FurnitureTreeTableModel::getFurnitureComparator()
toggleCameraSelection()	com.eteks.sweethome3d.swing.PhotosPanel::toggleCameraSelection()
isPieceOfFurniturePartOfBasePlan()	com.eteks.sweethome3d.viewcontroller.FurnitureController::isPieceOfFurniturePartOfBasePlan()
sortFurniture()	com.eteks.sweethome3d.viewcontroller.FurnitureController::sortFurniture()
addComponent3DRenderingErrorObserver()	com.eteks.sweethome3d.applet.ViewerHelper::addComponent3DRenderingErrorObserver()
getOptionalLocalizedString()	com.eteks.sweethome3d.io.DefaultUserPreferences::getOptionalLocalizedString()
createNavigationPanel()	com.eteks.sweethome3d.swing.HomeComponent3D::createNavigationPanel()
setPlanRulersVisible()	com.eteks.sweethome3d.swing.HomePane::setPlanRulersVisible()
cloneHomeInEventDispatchThread()	com.eteks.sweethome3d.swing.HomePane::cloneHomeInEventDispatchThread()
getOptionalString()	com.eteks.sweethome3d.swing.ResourceAction::getOptionalString()
getTextures()	com.eteks.sweethome3d.swing.TextureChoiceComponent.TexturePanel::getTextures()
getMinX()	com.eteks.sweethome3d.viewcontroller.FurnitureController::getMinX()
getMaxX()	com.eteks.sweethome3d.viewcontroller.FurnitureController::getMaxX()
getMinY()	com.eteks.sweethome3d.viewcontroller.FurnitureController::getMinY()
getMaxY()	com.eteks.sweethome3d.viewcontroller.FurnitureController::getMaxY()
getObserverCameraMinimumElevation()	com.eteks.sweethome3d.viewcontroller.HomeController3D::getObserverCameraMinimumElevation()
getRoomSideLength()	com.eteks.sweethome3d.viewcontroller.PlanController.AbstractRoomState::getRoomSideLength()
getRoomSideAngle()	com.eteks.sweethome3d.viewcontroller.PlanController.AbstractRoomState::getRoomSideAngle()
getPaintedItems()	com.eteks.sweethome3d.swing.PlanComponent::getPaintedItems()
isIntersectionEmpty()	com.eteks.sweethome3d.viewcontroller.PlanController::isIntersectionEmpty()
moveHomeItemsToLevel()	com.eteks.sweethome3d.viewcontroller.PlanController::moveHomeItemsToLevel()
getDimensionLineAngle()	com.eteks.sweethome3d.viewcontroller.PlanController.DimensionLineResizeState::getDimensionLineAngle()
sortFurniture()	com.eteks.sweethome3d.viewcontroller.FurnitureController::sortFurniture()
alignPieceOfFurnitureAlongLeftOrRightSides()	com.eteks.sweethome3d.viewcontroller.FurnitureController::alignPieceOfFurnitureAlongLeftOrRightSides()
updateOpenRecentHomeMenu()	com.eteks.sweethome3d.swing.HomePane::updateOpenRecentHomeMenu()

Table -3- (cont'd)

doAddFurniture()	com.eteks.sweethome3d.viewcontroller.FurnitureController::doAddFurniture()
doToggleBackgroundImageVisibility()	com.eteks.sweethome3d.viewcontroller.HomeController::doToggleBackgroundImageVisibility()
storeCamera(java.lang.String)	com.eteks.sweethome3d.viewcontroller.HomeController3D::storeCamera(java.lang.String)
isPieceOfFurnitureVisibleAtSelectedLevel()	com.eteks.sweethome3d.viewcontroller.PlanController::isPieceOfFurnitureVisibleAtSelectedLevel()
getDetecTableRoomsAtSelectedLevel()	com.eteks.sweethome3d.viewcontroller.PlanController::getDetecTableRoomsAtSelectedLevel()
getDetecTableWallsAtSelectedLevel()	com.eteks.sweethome3d.viewcontroller.PlanController::getDetecTableWallsAtSelectedLevel()
postPieceOfFurnitureWidthAndDepthResize()	com.eteks.sweethome3d.viewcontroller.PlanController::postPieceOfFurnitureWidthAndDepthResize()
selectLevelFromSelectedItems()	com.eteks.sweethome3d.viewcontroller.PlanController::selectLevelFromSelectedItems()
computeRoomPartGeometry()	com.eteks.sweethome3d.j3d.Room3D::computeRoomPartGeometry()
getHeaderRenderer()	com.eteks.sweethome3d.swing.FurnitureTable.FurnitureTableColumnModel::getHeaderRenderer()
createLockUnlockBasePlanButton()	com.eteks.sweethome3d.swing.HomePane::createLockUnlockBasePlanButton()
addColorListener()	com.eteks.sweethome3d.swing.ImportedFurnitureWizardStepsPanel.AbstractModelPreviewComponent::addColorListener()
addIconYawListener()	com.eteks.sweethome3d.swing.ImportedFurnitureWizardStepsPanel.AbstractModelPreviewComponent::addIconYawListener()
savePhoto()	com.eteks.sweethome3d.swing.PhotosPanel::savePhoto()
deleteLastRecordedCameraLocation()	com.eteks.sweethome3d.swing.VideoPanel::deleteLastRecordedCameraLocation()
doDeleteFurniture()	com.eteks.sweethome3d.viewcontroller.FurnitureController::doDeleteFurniture()
toggleFurnitureSort()	com.eteks.sweethome3d.viewcontroller.FurnitureController::toggleFurnitureSort()
toggleFurnitureVisibleProperty()	com.eteks.sweethome3d.viewcontroller.FurnitureController::toggleFurnitureVisibleProperty()
writePreferences()	com.eteks.sweethome3d.viewcontroller.HomeController.UserPreferencesChangeListener::writePreferences()
deleteCameras()	com.eteks.sweethome3d.viewcontroller.HomeController3D::deleteCameras()
doAddWalls()	com.eteks.sweethome3d.viewcontroller.PlanController::doAddWalls()
doDeleteWalls()	com.eteks.sweethome3d.viewcontroller.PlanController::doDeleteWalls()
doAddRooms()	com.eteks.sweethome3d.viewcontroller.PlanController::doAddRooms()
doDeleteRooms()	com.eteks.sweethome3d.viewcontroller.PlanController::doDeleteRooms()
doAddDimensionLines()	com.eteks.sweethome3d.viewcontroller.PlanController::doAddDimensionLines()
doDeleteDimensionLines()	com.eteks.sweethome3d.viewcontroller.PlanController::doDeleteDimensionLines()
doAddLabels()	com.eteks.sweethome3d.viewcontroller.PlanController::doAddLabels()
doDeleteLabels()	com.eteks.sweethome3d.viewcontroller.PlanController::doDeleteLabels()
postPieceOfFurnitureHeightResize()	com.eteks.sweethome3d.viewcontroller.PlanController::postPieceOfFurnitureHeightResize()
paintRoomAreaOffsetIndicator()	com.eteks.sweethome3d.swing.PlanComponent::paintRoomAreaOffsetIndicator()
paintRoomNameOffsetIndicator()	com.eteks.sweethome3d.swing.PlanComponent::paintRoomNameOffsetIndicator()
getAreaOnFloor()	com.eteks.sweethome3d.j3d.ModelManager::getAreaOnFloor()
getVisibleItemsAtSelectedLevel()	com.eteks.sweethome3d.viewcontroller.PlanController::getVisibleItemsAtSelectedLevel()
doDeleteItems()	com.eteks.sweethome3d.viewcontroller.PlanController::doDeleteItems()
addSizeListeners()	com.eteks.sweethome3d.swing.ImportedFurnitureWizardStepsPanel.AbstractModelPreviewComponent::addSizeListeners()
updateViewPlatformTransform()	com.eteks.sweethome3d.swing.HomeComponent3D::updateViewPlatformTransform()
goToCamera()	com.eteks.sweethome3d.viewcontroller.HomeController3D::goToCamera()
updateView()	com.eteks.sweethome3d.swing.HomeComponent3D::updateView()
alignFurnitureSideBySide()	com.eteks.sweethome3d.viewcontroller.FurnitureController::alignFurnitureSideBySide()
alignPieceOfFurnitureAlongSides()	com.eteks.sweethome3d.viewcontroller.FurnitureController::alignPieceOfFurnitureAlongSides()
addRooms()	com.eteks.sweethome3d.viewcontroller.PlanController::addRooms()
createWall()	com.eteks.sweethome3d.viewcontroller.PlanController::createWall()
selectItems()	com.eteks.sweethome3d.viewcontroller.PlanController::selectItems()

Table -3- (cont'd)

addWalls()	com.eteks.sweethome3d.viewcontroller.PlanController::addWalls()
addDimensionLines()	com.eteks.sweethome3d.viewcontroller.PlanController::addDimensionLines()
addLabels()	com.eteks.sweethome3d.viewcontroller.PlanController::addLabels()

APPENDIX-B

CODE LISTING

In this part of the thesis we are listing the actual code of the methods under the test. These methods are indicated as Feature Envy instances by our approach and explained in chapter 4 section 2.

```
private void addAreaSidesGeometry(Shape3D groundShape,
                                  HomeTexture groundTexture,
                                  float [][] areaPoints,
                                  float elevation,
                                  float sideHeight) {
Point3f [] geometryCoords = new Point3f [areaPoints.length * 4];
int [] stripCounts = new int [areaPoints.length];
int [] contourCounts = new int [stripCounts.length];
TexCoord2f [] geometryTextureCoords = groundTexture != null
? new TexCoord2f [geometryCoords.length]
: null;
Arrays.fill(stripCounts, 4);
Arrays.fill(contourCounts, 1);
for (int i = 0, j = 0; i < areaPoints.length; i++) {
float [] point = areaPoints [i];
float [] nextPoint = areaPoints [i < areaPoints.length - 1 ? i +
1 : 0];
geometryCoords [j++] = new Point3f(point [0], elevation, point
[1]);
geometryCoords [j++] = new Point3f(point [0], elevation +
sideHeight, point [1]);
```

```
geometryCoords [j++] = new Point3f(nextPoint [0], elevation +
sideHeight, nextPoint [1]);

geometryCoords [j++] = new Point3f(nextPoint [0], elevation,
nextPoint [1]);

if (groundTexture != null) {

float distance = (float)Point2D.distance(point [0], point [1],
nextPoint [0], nextPoint [1]);

geometryTextureCoords [j - 4] = new TexCoord2f(point [0] /
groundTexture.getWidth(),              elevation              /
groundTexture.getHeight());

geometryTextureCoords [j - 3] = new TexCoord2f(point [0] /
groundTexture.getWidth(),    (elevation    +    sideHeight)    /
groundTexture.getHeight());

geometryTextureCoords [j - 2] = new TexCoord2f((point [0] -
distance) / groundTexture.getWidth(), (elevation + sideHeight) /
groundTexture.getHeight());

geometryTextureCoords [j - 1] = new TexCoord2f((point [0] -
distance)    /    groundTexture.getWidth(),    elevation    /
groundTexture.getHeight());

}

}

private     void     updateViewPlatformTransform(TransformGroup
viewPlatformTransform,Camera  camera,boolean  updateWithAnimation
) {

if (updateWithAnimation) {

// Get the camera interpolator

CameraInterpolator             cameraInterpolator             =
(CameraInterpolator)viewPlatformTransform.getChild(viewPlatformT
ransform.numChildren() - 1);

cameraInterpolator.moveCamera(camera);

} else {

Transform3D transform = new Transform3D();

updateViewPlatformTransform(transform,            camera.getX(),
camera.getY(),

camera.getZ(), camera.getYaw(), camera.getPitch());

viewPlatformTransform.setTransform(transform);

}

clearPrintedImageCache();

}
```

```
private HomePieceOfFurniture adjustPieceOfFurnitureSideBySideAt(HomePieceOfFurniture piece,
boolean forceOrientation, Wall magnetWall) {
float [][] piecePoints = piece.getPoints();
Area pieceArea = new Area(getPath(piecePoints));
boolean doorOrWindowBoundToWall = piece instanceof HomeDoorOrWindow && ((HomeDoorOrWindow)piece).isBoundToWall();
float pieceElevation = piece.getGroundElevation();
float margin = 2 * PIXEL_MARGIN / getScale();
BasicStroke stroke = new BasicStroke(margin);
HomePieceOfFurniture referencePiece = null;
Area intersectionWithReferencePieceArea = null;
float intersectionWithReferencePieceSurface = 0;
float [][] referencePiecePoints = null;
for (HomePieceOfFurniture homePiece : this.home.getFurniture()) {
float homePieceElevation = homePiece.getGroundElevation();
if (homePiece != piece && isPieceOfFurnitureVisibleAtSelectedLevel(homePiece)
&& pieceElevation < homePieceElevation + homePiece.getHeight()
&& pieceElevation + piece.getHeight() > homePieceElevation
&& (!doorOrWindowBoundToWall
// Ignore other furniture for doors and windows bound to a wall
|| homePiece.isDoorOrWindow())) {
float [][] points = homePiece.getPoints();
GeneralPath path = getPath(points);
Area marginArea;
if (doorOrWindowBoundToWall && homePiece.isDoorOrWindow()) {
marginArea = new Area(stroke.createStrokedShape(new Line2D.Float( points [1][0], points [1][1], points [2][0], points [2][1])));
marginArea.add(new Area(stroke.createStrokedShape(new Line2D.Float(points [3][0], points [3][1], points [0][0], points [0][1]))));
} else { marginArea = this.furnitureSidesCache.get(homePiece);
if (marginArea == null) {
```

```
marginArea = new Area(stroke.createStrokedShape(path));
this.furnitureSidesCache.put(homePiece, marginArea);
}      }
Area intersection = new Area(marginArea);
intersection.intersect(pieceArea);
if (!intersection.isEmpty()) {
Area exclusiveOr = new Area(pieceArea);
exclusiveOr.exclusiveOr(intersection);
if (exclusiveOr.isSingular()) {
Area insideArea = new Area(path);
insideArea.subtract(marginArea);
insideArea.intersect(pieceArea);
if (insideArea.isEmpty()) {
float surface = getArea(intersection);
if (surface > intersectionWithReferencePieceSurface) {
intersectionWithReferencePieceSurface = surface;
referencePiece = homePiece;
referencePiecePoints = points;
intersectionWithReferencePieceArea = intersection;
}
}
}
}
}
}
if (referencePiece != null) {
boolean alignedOnReferencePieceFrontOrBackSide;
if (doorOrWindowBoundToWall && referencePiece.isDoorOrWindow())
{
alignedOnReferencePieceFrontOrBackSide = false;
 }
 else
{
GeneralPath           referencePieceLargerBoundingBox           =
getRotatedRectangle(referenePiece.getX()                        -
referencePiece.getWidth(),
```

```
referencePiece.getY() - referencePiece.getDepth(),
referencePiece.getWidth() * 2, referencePiece.getDepth() *
2,referencePiece.getAngle());

float [][] pathPoints =
getPathPoints(referencePieceLargerBoundingBox, false);

alignedOnReferencePieceFrontOrBackSide =
isAreaLargerOnFrontOrBackSide(intersectionWithReferencePieceArea
, pathPoints);

}

if (forceOrientation)

{

piece.setAngle(referencePiece.getAngle());

}

Shape pieceBoundingBox = getRotatedRectangle(0, 0,
piece.getWidth(), piece.getDepth(), piece.getAngle() -
referencePiece.getAngle());

float deltaX = 0; float deltaY = 0;

if (!alignedOnReferencePieceFrontOrBackSide) {

Line2D centerLine = new Line2D.Float(referencePiece.getX(),
referencePiece.getY(), (referencePiecePoints [0][0] +
referencePiecePoints [1][0]) / 2, (referencePiecePoints [0][1] +
referencePiecePoints [1][1]) / 2);

double rotatedBoundingBoxWidth =
pieceBoundingBox.getBounds2D().getWidth();

double distance = centerLine.relativeCCW(piece.getX(),
piece.getY())* (-referencePiece.getWidth() / 2 +
centerLine.ptLineDist(piece.getX(), piece.getY()) -
rotatedBoundingBoxWidth / 2);

deltaX = (float)(distance *
Math.cos(referencePiece.getAngle()));

deltaY = (float)(distance *
Math.sin(referencePiece.getAngle()));

}

else

{

Line2D centerLine = new Line2D.Float(referencePiece.getX(),
referencePiece.getY(),

(referencePiecePoints [2][0] + referencePiecePoints [1][0]) / 2,
(referencePiecePoints [2][1] + referencePiecePoints [1][1]) /
2);
```

```
double rotatedBoundingBoxDepth = pieceBoundingBox.getBounds2D().getHeight();
double distance = centerLine.relativeCCW(piece.getX(), piece.getY())* (-referencePiece.getDepth() / 2 + centerLine.ptLineDist(piece.getX(), piece.getY()) - rotatedBoundingBoxDepth / 2);
deltaX = (float)(-distance * Math.sin(referencePiece.getAngle()));
deltaY = (float)(distance * Math.cos(referencePiece.getAngle()));
if (!isIntersectionEmpty(piece, magnetWall, deltaX, deltaY)) {
deltaX = deltaY = 0;
}
}
if (!isIntersectionEmpty(piece, referencePiece, deltaX, deltaY)) {
piece.move(deltaX, deltaY);
return referencePiece;
} else
{
if (forceOrientation)
{  piecePoints = piece.getPoints();  }
boolean alignedOnPieceFrontOrBackSide = isAreaLargerOnFrontOrBackSide(intersectionWithReferencePieceArea, piecePoints);
Shape referencePieceBoundingBox = getRotatedRectangle(0, 0, referencePiece.getWidth(), referencePiece.getDepth(), referencePiece.getAngle() - piece.getAngle());
if (!alignedOnPieceFrontOrBackSide) {
// Search the distance required to align piece on its left or right side
Line2D centerLine = new Line2D.Float(piece.getX(), piece.getY(),
(piecePoints [0][0] + piecePoints [1][0]) / 2, (piecePoints [0][1] + piecePoints [1][1]) / 2);
double rotatedBoundingBoxWidth = referencePieceBoundingBox.getBounds2D().getWidth();
double distance = centerLine.relativeCCW(referencePiece.getX(), referencePiece.getY())* (-piece.getWidth() / 2 + centerLine.ptLineDist(referencePiece.getX(), referencePiece.getY()) - rotatedBoundingBoxWidth / 2);
```

```
deltaX = -(float)(distance * Math.cos(piece.getAngle()));
deltaY = -(float)(distance * Math.sin(piece.getAngle()));
} else
{
Line2D centerLine = new Line2D.Float(piece.getX(), piece.getY(),
(piecePoints [2][0] + piecePoints [1][0]) / 2, (piecePoints
[2][1] + piecePoints [1][1]) / 2);
double rotatedBoundingBoxDepth =
referencePieceBoundingBox.getBounds2D().getHeight();
double distance = centerLine.relativeCCW(referencePiece.getX(),
referencePiece.getY())* (-piece.getDepth() / 2 +
centerLine.ptLineDist(referencePiece.getX(),
referencePiece.getY()) - rotatedBoundingBoxDepth / 2);
deltaX = -(float)(-distance * Math.sin(piece.getAngle()));
deltaY = -(float)(distance * Math.cos(piece.getAngle()));
if (!isIntersectionEmpty(piece, magnetWall, deltaX, deltaY)) {
deltaX = deltaY = 0; } }
if (!isIntersectionEmpty(piece, referencePiece, deltaX, deltaY))
{
piece.move(deltaX, deltaY); return referencePiece; } }
return referencePiece; }
return null; }
```

CURRICULUM VITAE

PERSONAL INFORMATION

Name Surname : Baydaa MERZAH

Date of birth and place : 4th of July 1982- Baghdad

Foreign Languages : English , Turkish

E-mail : baidaamuhammed@gmail.com

EDUCATION

Degree	Department	University	Date of Graduation
Undergraduate	Computer science	University of Baghdad	2004
High School	Scientific department	Al-Faroq High School for girls	2000

WORK EXPERIENCE

Year	Corporation/Institute	Enrollment
2008- continue	Al-Nahrain University	2008

PUBLISHMENTS

Conference Papers

1. Merzah.B, Selcuk Y. "Metric Based Detection of Refused Bequest Code Smell". (2017). 9th International Conference on Computational Intelligence and Communication Networks (CICN 2017), IEEE.