International Journal of Computer Applications (0975 – 8887)
Volume 67– No.10, April 2013

Detecting and Scheduling Badsmells using Java Agent Development (JADE)

S.Ayshwarya Lakshmi
Assistant professor
University college of Engineering
Panruti.

S.Shanmuga Vadivu
M.E (Computer Science)
University college of Engineering, Trichirapalli.

A.Ramachandran
Assistant professor
University college of Engineering
Panruti.

ABSTRACT

For any software developer it is a fact that as source code is developed, it becomes large and complex .As the code becomes large and complex it moves from its original design and reduces the quality of code. These which reduce the code quality are termed as Bad Smells by experts [1].The methods of removing these Bad Smells are called as Refactoring. These Bad smells have so far been detected and removed manually or semiautomatically.In this paper we say about the automatic detection and resolution of Bad smell with the help of JADE (Java Agent Development Environment).JADE is a middleware developed in Java.

General Terms

Software Refactoring, Agent Based Software Engineering.

Keywords

Bad Smells, Refactoring, Detection and Scheduling

1. INTRODUCTION

1.1 What is Refactoring

Refactoring is used to give a good internal structure to the object oriented Software. The way of reducing the complexity in the software without affecting its behavior .This cleans up all the complexity in the source code and makes it easier to understand[1].Though there are many tools to develop software they just provide to add new enhancement to the source code which in turn increases the code size and complexity. This then reduces the design of the software. Normally Developers do not concentrate on design in the early stage of development but during quality analysis and maintenance this becomes a problem. Hence the technique developed to reduce this problem of complexity in the source code of an object oriented system is called as *refactoring*.

1.2 Sequencing Bad Smells

Bad Smells are terms coined by experts such as Fowler et al [3]. Let us consider an example for the term *badSmells*.Long Method is a bad smell in Object Oriented Systems. These method make the code to have too many responsibility .This makes it difficult to understand and maintain. Hence the way to Refractor this type of Code Smell is to Split the method that is to extract the method in to smaller methods.

The way to Refactor these bad smells has been done in two ways [5] [6] XP-Style and Batch Model. XP-style in which bad smells are found only in few files. Batch model in which Smells in large system are refactored in one attempt kind. There are in fact two schemes to detect bad Smells. One is *Kind Level Scheme* and the other is *Instance Level Scheme*. In Instance level scheme only one kind of Bad Smell is detected. Where as in kind Level scheme different kind of bad smells are detected. Our paper focuses to sort different kind of bad smells and refactor them based on the refactoring rules. These bad smells are detected and scheduled using agent which analyses the bad smells in the source code and uses the algorithm provided to it to sequence the bad smell and then refactors the source code.

When one type of Bad smell is found the other type of bad smell could then be detected and resolved. (i.e) if large class is detected first long method in it could be solved. Similarly long parameters could then be resolved. Hence we need to provide the proper sequence for finding the Bad Smell and then removing it.

1.3 TYPES OF BAD SMELLS

1.3.1 Long Method:

These are code smells where a source code contains large amount of lines (i.e.) if for rule we say loc>20, then we say it long method. The logic is also complex and difficult to understand.

1.3.2 Large class:

In this the class is so large (i.e.) the work done is more and the line of code is also very high.

1.3.3 Long Parameter List:

The number of parameter for an method might be so large. Such type of smells are called long parameter list.

1.3.4 Feature Envy:

A method or fragment of method interested in the features of another class.

1.3.5 Primitive Obsession:

Some may not be interested in using objects and use a number of primitives. The way of using object in case of primitives.

1.3.6 Useless field, Class Method:

Methods, fields and classes which are defined and not used.

Here we perform a pair wise Analysis of the bad smell. (i.e) if a useless class is removed useless methods in it could be removed simultaneously.

Analysis of Bad smells using pair wise analysis is shown in Figure 2. The vertexes represent the type of bad smells the resolution sequences are given in as the adjacent vertices of the edges [5]. This graph in Figure 2 is so complex and redundant hence there occurs a need to sequence these bad smells by removing the redundancies.

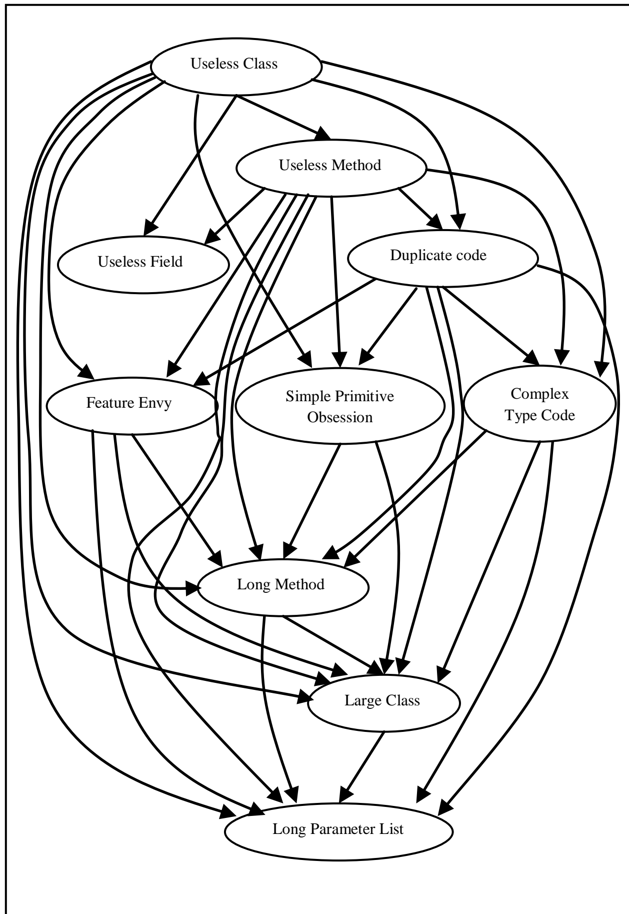

Figure 1. Pair wise resolution sequences of bad smells

These redundancies are removed using topological sorting algorithm. The graph of the algorithm shown in Figure 4. The algorithm is as such if there are two paths from v_1 to v_2 one of the redundant path from v_1 to v_2 could be removed (i.e) if there is a parallel edge of $e(v_1,v_2)$. Then, e (v_1, v2) could be removed.

```
1   /* Input:     Directed graph with redundancies
2      Output:  Directed graph without redundancies */
3   for each vertex v in the graph
4     {
5       for every edge e(v,d) in vertex v
6         {
7           if present some other  path p1(v,d) besides e1(v,d)
8            {
9              // the length of p1(v,d) must be longer than
10             // that of e1(v,d) because there are no
11             // parallel edges in the graph
12
13             remove e1(v,d)
14
15            //Topological orders are as such without any
16  effect on removal of e1(v,d)
17           }
18        }
19    }
```

Figure 2. Algorithm for removing redundancies

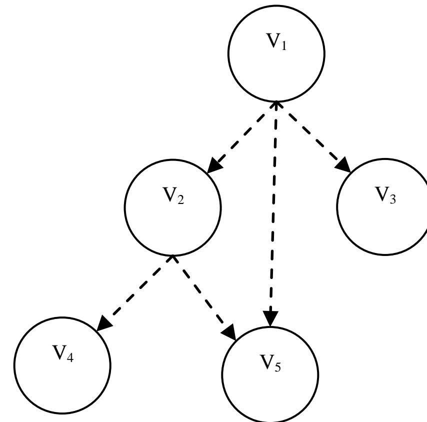

Figure 3. Graph of Redundancy removing Algorithm

In this paper these Bad Smells are detected automatically and then resolved. This is done with the help of JADE, an Agent development environment. The rest of the paper is depicted as follows: Section 2 says about the way the bad smells are removed.Section 3 presents a short view of related work. Section 4 talks about agents and JADE. Section 5 about the Analysis and Section 6 about conclusion.

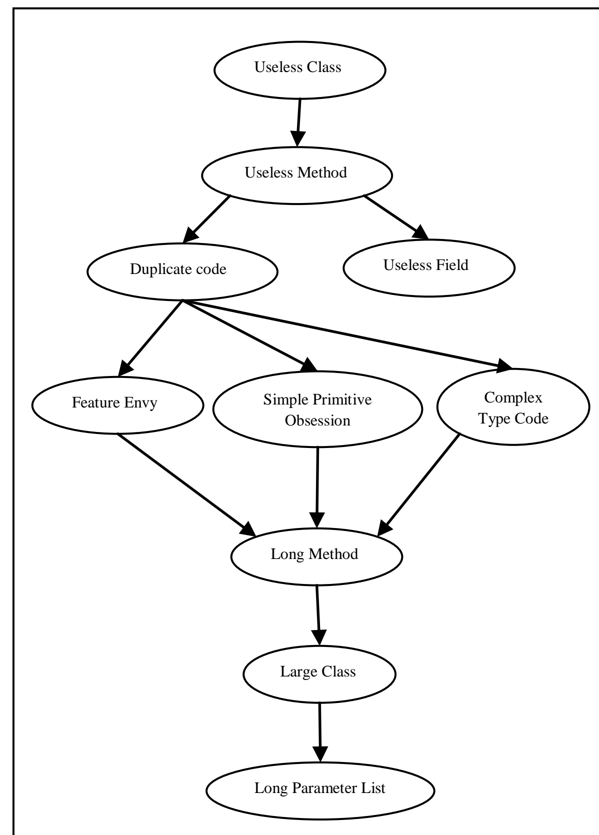

Figure 4. Resulting sequence of bad smells

2. REFACTORING TECHNIQUES

Once a Bad smell has been detected we then can apply refactoring techniques to remove these bad smells. Some of the bad smells and refactoring techniques are given below.

Table 1. Refactoring Techniques for Bad Smells

S. No.	Bad Smell	Refactoring Technique
1.	Long Method	• Extract Method • Replace Temp with Query • Preserve Whole Object • Replace Method with Method Object
2.	Large Class	• Extract Class • Extract Subclass
3.	Long Parameter List	• Replace Parameter Method • Preserve Whole Object • Introduce Parameter Object
4.	Duplicate Code	• Extract Method • Pull Up Field • Extract Class
5.	Primitive Obsession	• Replace Type code with class • Replace Data value with object • Replace type code with subclass
6.	Feature Envy	• Move Method • Extract Method

2.1 Extract Method

This is a technique used in long method where there are too many logic available in the method. The number of lines of code is also increased. Hence certain lines of code are removed and replaced as separate method [2].

Example:

Bad Code

```
void printOwing1(double amount)
 {
   printBan ();
   //print details
   System.out.println ("name:" + _name1);
   System.out.println ("amount" + amount1);
  }
```

Refactored Code

```
void printOwing1 (double amount)
{
   printBan();
   printDetails(amount);
}
 void printDetails (double amount)
 {
   System.out.println ("name:" + _name1);
   System.out.println ("amount" + amount1);
 }
```

Rules to be followed

- Identify the function of the method and name the method
- Take the code from the source method and copy them to the destination method.
- Go through the extracted method to capture the references to variables that are local in scope of the source method. Treat these as local variables and parameters to the method.
- Look for any temporary variables that are used only within the extracted code. If any, retain them as temporary variables in the target method.
- Check if any modifications have been made to the local variables that need to be returned. If the number of such variables is limited to one, return it.If it's two or more, split the method again or make them final.
- Make a call to the target method in place of the extracted code.
- Compile and test. (Fowler et al., 2000)

2.2 Replace Method with Method object

In case when there are large numbers of local variables in long Method we could use Replace method with Method Object instead of Extract method. To do this make the local variables become fields on that object by making the method into its own object. You can then decompose the method into other methods on the same object.

Example:

Bad Code

```
class Account
{.....
  int gamma (int iVal, int quantity, int yearTodDate)
  {
    int IValue1= (iVal * quant) + delta ();
    int IValue2= (iVal * yearToDate) + 100;
    if ((yearToDate - IValue1) > 100)
      IValue2 -= 20;
    int IValue3=IValue2 * 7;
    // and so on..
    return important Value3-2 * importantValue1;
  }
}
```

Refactored Code

```
class Gamma
{...
   private final Account_account;
   private int iVal;
   private int quant;
   private int yToD;
   private int IValue1;
   private int IValue2;
   private int IValue3;
   Gamma (Account source, int iValArg, int quantArg, int
yToDArg)
   {
      _account = source;
      iVal = iValArg;
      quant = quantArg;
      yToD = yToDArg;
   }
   int calculate()
   {
      IValue1= (iVal * quant) +_account.delta();
      IValue2= (iVal*yToD) +100;
      IThing();
      int IValue3=IValue2 * 7;
      // and so on...
      return IValue3 - 2 * IValue1;
    }
    void IThing()
    {
     if ((yToD - IValue1) > 100)
       IValue2 -=20;
    }
    Class Account {...
      int gamma(int iVal,int quant, int YToD){
```

```
    return new Gamma(this, iVal, quant, YToD).compute();
  }
}
```

Rules to Refactor

- Create a new class with the name of the method.
- Create a final field in the new class of the source object. Also, create a field for each temporary variable and parameter in the method.
- Create a constructor of the new class that takes the source object and each parameter.
- Create a method named compute in the new class.
- Copy the body of the original method into calculate. Use the object field of the source for any method invocation on the original object.
- Compile
- Replace the old method call with the new one and call compute (Fowler et al., 2000)

2.3 Replace temp with Query

The long method smell is because of temporary variables declaration which assigns value only once. Hence we assign the methods to the variables which make it easy to apply extract method. This type of refactor is called Replace Temp with Query.

Example:

Bad Code

```
double getPrice()
{
 int basePrice = _quant*_itemPrice;
 double discountFactr;
 if(basePrice>1000)
   discountFactor =0.95;
 else
   discountFactr=0.98;
 return Price*discountFactr;
}
```

Refactored Code

```
double getPrice()
{
 return basePrice()*discountFactr();
}
private double discountFactor()
{
 if(BPrice()>1000)
   return 0.95;
 else
   return 0.98;
}
private int BPrice()
{
 return_quant*_itemPrice;
}
```

Rules to Refactor

- Search for an assigned temporary variable.
- Let it be final.
- Make the right hand assignment as body of the new method.
- Compile and test (Fowler et al., 2000).

3. RELATED WORK

There are many agent applications being developed in today's research and Organization [18]. Some the Agent Programming Languages, IDÉ, Platforms and Frameworks have been discussed here. There are many Agent Oriented Programming Languages divisible as imperative, declarative and Hybrid Approaches. Many Developed Languages have Integrated Development Environment which helps programmers to ease their work. CLAIM (**C**omputational **L**anguage for **A**utonomous **I**ntelligent and **M**obile Agents)[20] is Declarative .It is Declarative because it is formal and is Grounded to Logic.

An example of an imperative Language is **JACK A**gent **L**anguage(JAL)[22].A purely imperative Language is very less common due to abstraction related to Agent Oriented design. Some Languages combine both imperative and declarative languages (Hybrid approach).3APL programming Language (**A**n **A**bstract **A**gent **P**rogramming **L**anguage triple a-p-l) [21].This implements cognitive Agents. **JASON [19]** is also a Language that uses a Hybrid Approach. All the languages and their features have been Listed Below.

Table 2. Multi Agent System Programming languages

	CLAIM	JAL	3APL	JASON
Language	Declarativ e	Imperative	Hybrid	Hybrid
Supported /Developed	SyMPA std	Agent Oriented Software	University of trecht,Nethe rland	Open Source
Semantics	Formal	NotFormal	Dual	Operation al
Applicatio ns	Load Balancing, Resource sharing	Commercial	Application requiring mental attitudes	Testing

There are also many IDE that tend to provide functionalities that can be classified into five categories: Project Management, Creating and Editing source files, Refactoring and Testing. 3APL, JASON, Agent Factory offer IDE Support.

Table 3. Platforms and Framework

TuCSoN	JADE	DESIRE
MAS coordination and communication	Distributed MAS coordination and communication	Compositional development of MAS
Open source	Open source	Treur et al. Vrije University Amsterdam.
Object Oriented	Object Oriented	Component based
Distributed workflow-learning applications	e-learning, Tutorial research	Load balancing of electricity
Open Framework	Open Framework	Closed Framework

Other than these IDE there are many Agent Platforms and frameworks such as TuCSoN, JADE, and DESIRE. TuCSoN (Tuple Centre Spread over Network) Provide General purpose programming service. Its Model is based on Tuple Center.

DESIRE (DEsign and Specification of Interacting REasoning Components) [23] is compositional development method for multi agent systems. A table view of all these frameworks have been given above.

Among all the Frameworks available the most popular solution is JADE. Because JADE is flexible, interoperable and this strictly adheres to FIPA Specifications. JADE due to its Decentralized messaging architecture, the cooperating agents continue to work even in the presence of AMS failure. Since JADE is efficient, many researchers are using JADE and it's been upgraded for each software release. The recent release of JADE is 4.3.0, released on 29th march 2013.This available on official website of JADE [4]. A graph representing its downloads after each software release has been given below

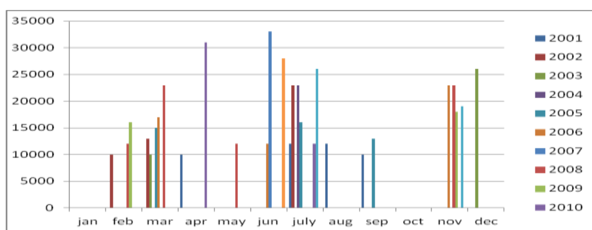

Figure 5. JADE Downloads after each software release

4. USING JADE

4.1 Agent

The term agent is very broad and has different meanings to different people. The Methodology defines agent as

Agents reside on a platform that, consistent with the presented vision, provides the agents with a proper mechanism to communicate by names, regardless of the complexity and nature of the underlying environment [12].

4.2 Agent Properties

The Agent has some properties [15] and they are specified below:

- Autonomy: This states that an agent could operate directly without any help from Human or others. There is a sought of control to their actions and their internal state.
- Social ability: Agent use ACL (Agent Communication Language) to communicate within them.
- Reactivity: Agents respond to environment in a timely manner based on what they perceive in their environment.
- Pro-activeness: they not only respond to the environment, they also take the initiative by exhibit their goal oriented behavior to the environment.

4.3Agent Architecture

Agents in the multi-agent system are organized in a hierarchy structure, called Agent Hierarchy. Low level agents are coordinated by high level agents. These agents are working together to achieve the goals in the system.

Multi-agent system is derived by using the agent identification policies. These policies [8] are affected by the following factors:

- *Modularity:* split the agents which are working to solve a sub-problem.
- *Reusability:* split the agents in the system which are all can be reused to solve a new problem.
- *Location:* split the agents for a distributed environment.
- *Load-balancing:* split the agents with balanced work load.
- Organizational role: make a group of agents assigned for particular role in an organization.

4.4 Artificial Agent

An agent is one that acts on the environment through effectors by observing it with the help of sensor [13].

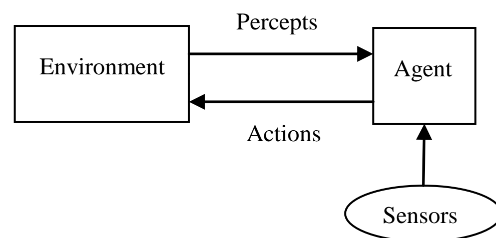

Figure 6. Agents interact with environments through sensors and detectors

The job of AI is to design the agent program: a function that implements the agent mapping from percepts to actions. The architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the effectors as they are generated. Thus architecture and a program sum up to form an agent:

Agent = architecture + program

4.5 Agent Development Using JADE

There are many Multi agent Languages and Platforms which are being used in many active projects. Few Languages and their features have been discussed in the above section. But the most important features of JADE in comparision to the others are

1. JADE is completely based on FIPA specifications.

2. JADE provides a proper set of functionalities which helps in the development of mulit-agent systems.Users could use and write the Java code without to know anything new.

3. JADE could be deployed on many places which use JEE, JSE JME devices.

4.5.1JADE

JADE is a software framework to facilitate the development of interoperable intelligent multi agent system that is used by a heterogeneous community of users as a tool for both supporting research activities and building real applications [16].

JADE was developed by Telecom Italia Lab in compliance FIPA (Foundation for Intelligent Physical Agent) specification. This FIPA is a non-profit organization which develops standards for the interoperation of heterogeneous

Agents [14]. We here develop Agents using the following steps.

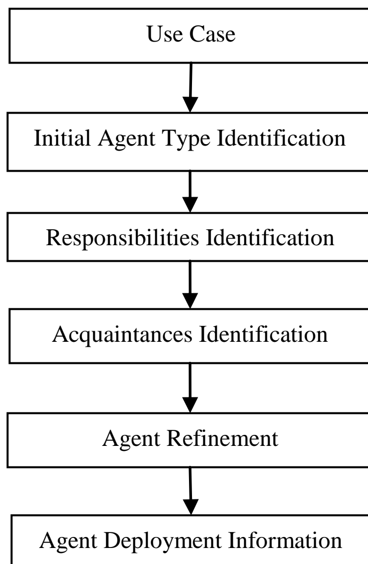

Figure 7. Analysis Phase of Methodology

JADE is an open source distributed by TIL Lab (www.TiLab.com) under LGPL license [20 g direct].

JADE is middleware written using JAVA. In JADE we make agents to be interoperable. These agents are supposed to follow the same standard. JADE uses a set of APIs. This could be deployed both in many environments such as JEE, JSE and JME. This is easy to use because the complexity of the middleware is not seen behind the APIs. It is also not necessary to use all features of the middleware [17].

JADE includes libraries for developing application agents and the runtime environments.

These runtime environments need to be active even before agents are executed. JADE container contains all the agents. The containers form the platform. These help to mask the complexity of the H/W, OS, JVM and type of network from the agents.

For every platform a Directory Facilitator (DF) is provided which gives the timely list of agents. These DF gives the most updated information of the agents in its directory on a non-discriminates basis to all authorized agents. Hence is said to provide a yellow pages directory services to agents. It is required to have a minimum of one DF in a platform. But an AP(Agent Platform) could support any number of DFs. Every agent wishing to publish its service should find a DF and request for its registration. An agent searches the DF for the request information. Thus a DF has to perform

- register
- deregister
- modify
- search

JADE has an AMS for each platform. all the agents activities like creation, deletion, registration with DF are managed by the AMS. The AMS in addition to performing register, deregister modify, search and agent-description it also say the AP to suspend, terminate, execute, invoke agents. It also instructs AP to does many other operations.

All the Agents communicate with each other using an ACL (Agent communication Language) as defined by FIPA. The format for the messages comprise of number of fields such as sender of message, receiver (list) of the messages, performative (what the sender intends) it could be REQEST message, PROPOSE message, ACCEPT_PROPOSAL etc. It also provides the content included in the message the general format to express the content, the words and synonyms used in the content (ontology). It is also responsible for controlling the several concurrent conversations. This is implementing from the jade.lang.acl.ACLMessage class.

Example:

```
ACLMessage m = new ACLMessage(ACLMessage .REQUEST);

m.addReceiver(new AID (“magesh”,AID.ISLOCALNAME));

m.setLanguage (“English”);

m.setOntology (“Large Class”);

m.setContent (“is a large class”);

send (m);
```

4.5.2Behaviours:

JADE Agent Concurrency Model:

As said Agents are autonomous (i.e.) they require each agent to be an active object with at least a thread, to positively start a new communication make plans and pursue goals. The social ability allows agents to have concurrent conversations they are pro active by specifying their goal oriented behaviour and allowing a way of exchanging information between two agents and as its autonomous both the sender and receiver have equal rights.

Agent duties are carried out based on the behaviours being scheduled and executed. When a Jade behaviour runs until it returns from the main cannot be preempted by other behaviours. Here is how java method needs to be converted to JADE behaviours: [7]

1. Turn the method body into an object whose class inherits from Behaviour.
2. Turn method local variables into behaviour instance variables.
3. Add the behaviour object to agent behaviour list during agent start up.

Using Behaviours to build complex Agents

For implementing an Agent such as file Agent, etc. we need to extend the Agent class. User Agents are inherited from the super class Agent. We have two methods addBehaviour (Behaviour) and removeBehaviour(Behaviour), which manage the behaviour list of the agent.

Behaviour is an abstract class, which helps to perform some tasks. The methods of the Behaviour class are

1. action () : task to be performed by the particular behaviour class.

2. done () : used by agent scheduler, returns true if behaviour has been executed successfully, false if action() is not executed successfully and must be executed again.
3. restart (): start the behaviour again.

JADE also allows application developer to build their own behaviours [11]. There are Behaviours such as complex Behaviour and simple Behaviour, where complex behaviour has some sub-behaviour with two methods such as addSubBehaviour (Behaviour) and removeSub-Behaviour (Behaviour). This allows agent writers to develop a tree with behaviours of different kinds. A simple Behaviour helps to implement small slips of the agent work. This simple Behaviour is executed by JADE scheduler. To send and receive messages are done using sender Behaviour and Receiver Behaviour. This way we implement more work.

5. ANALYSIS

From the methodology given in the Figure 7, an idea of what agents are required for detecting and scheduling bad smells have been developed. Along with it the responsibilities of each agent are also defined.

5.1 Agent Identification

The major agents required for scheduling and refactoring bad smells have been given below.

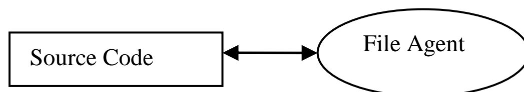

Figure 8. File Agent

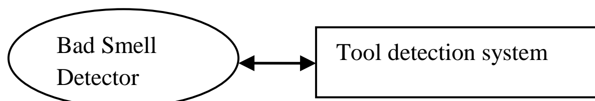

Figure 9. Smell detector Agent

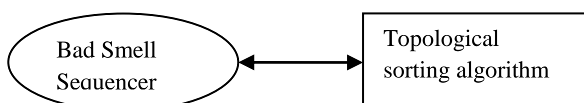

Figure 10. Sequencing Agent

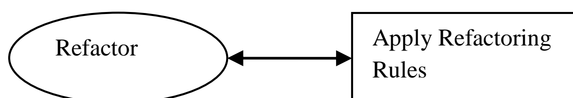

Figure 11. Refactoring Agent.

5.2 Responsibility Identification:

Responsibilities of all agents are identified. This says what an agent should perform during the time it is activated. We have already said there are four agents required for the Detection and sequencing such as File Agent., Bad Smell Detector Agent, Bad Smell Sequencing Agent and Refactoring agent. The Responsibilities of each and every Agent are listed below.

File Agent

- Receives the input(source code) From the user
- Validates the input. (i.e.)Checks whether it's a proper code to be refactored.
- Valid input is sent as a notification message to Detector agent.
- Display an error message for invalid file.

Bad Smell Detector Agent:

- Receives the notification message from the File agent.
- Reads the input and parses it for Bad smell Detection.
- Checks for the tool availability.
- Detects the bad smell.
- Initiates the Bad smell Sequencer to Sequence the Bad Smell.

Bad Smell Sequencer:

- Group's similar kind of Bad Smells.
- Performs a Pair Wise Analysis
- Removes Redundancies using Topological sorting
- Outputs the Sequenced Bad Smells to Refactor Agent.

Refactoring Agent:

- Receives the Sequenced Bad Smell.
- Checks for the Refactoring Rules for it
- Output the Refactored Source Code.

5.3 Acquaintances Identification:

This Acquaintance identifies the environment of how agents interact with each other along with its own responsibilities are given below.

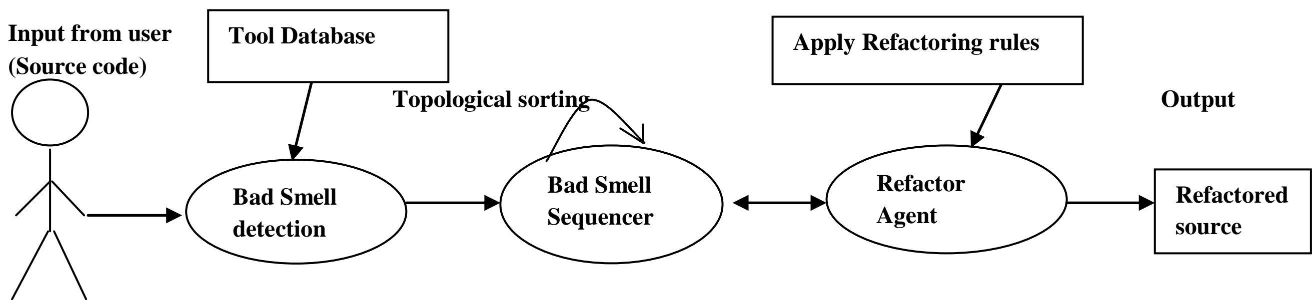

Figure 12. An Agent Diagram for Bad Smell Detection and Sequencing.

Here according to our identification of Agents, the File Agent needs to get the proper input from the user. It needs to check whether the input given by the user is available in the system he specified. If it is available it needs to check whether the input is valid input for Refactoring (i.e.) is it an source code. Once this has been done it then sees to check whether the Bad Smell Detector agent is free if so sends the input to it for processing.

As Soon a it gets the input the detector has to check its data base for the tool availability. It then needs to check which tool could be available for which Bad Smell. Once this has been done this interacts with the Bad smell sequencer for analysis and sequencing the Bad smells. It has to apply the Topological sorting Algorithm as said in Figure 3 for ordering the Bad Smells.

These ordered Bad Smells are then sent to the Refactoring Agent for Refactoring. It checks the Bad Smells and thus sees the Refactoring Rules applied for each Bad smell and Refactors them as said.

5.4 Agent Refinement and Deployment:

In this the agents that have been identified are then refined by analyzing the supporting information it requires for accomplishing its work. This also discovers by their acquaintance relationship. And a system to take care of what is happening before starting and stopping agents are performed. According to our work the supporting information required for the Bad smell detector is a Tool Database .It has to be maintained for the set of available tools. Then proper Refactoring rules have to be given to the Refactoring agent when at the right place it is required.

Deployment says the way we are going to deploy this detection and sequence process. A Deployment diagram is given in Figure 12.

This paper says that an Agent could find and sequence the bad smells. Many Tools such as PMDmetric, JDeodarant and more are available.PMD is a source analyzer .It finds unused variables empty catch block, unnecessary object creation etc. JDeodarant executes in Batch Processing mode. Likewise many Tools are there which could be embedded. Few of the Refactoring rules have been said for each of the Bad smells. The order the Agent takes it into account depends on the nature of the Bad smells in source code.

6. Conclusion

Software is about to be developed day by day, this development of the software makes it difficult and too big. There are ways for removing these bad smells .But here we have focused of scheduling and detecting bad smells by developing Agents. This paper here focuses only on the idea of saying that Bad smells could be Detected and Scheduled using a JADE platform and the agents have been identified for its purpose.

These bad smells which have been detected could be then refactored using agents. There are many bad smells available only few of the bad smells have been used here where as other type of bad smells like middle man, data class etc. could be tested for future work. Here the recommended resolution though is good but it is only for reference, when other kind of bad smells involved sequence might be changed. Moreover we have specified tools for detecting bad smells, the tools support for detecting bad smells are under investigation.

7. References

[1] T.Mens and T.Touwe" A Survey of Software Refactoring", IEEE Trans. Software Eng., vol. 30, no. 2 pp. 126-139, Feb. 2004.

[2] M.Fowler, K.Beck, J.Brant, W.Opdyke, and D.Roberts, Refactoring: Improving the Design of Existing Code. Additional Wesley Professional, 1999.

[3] W.C.Wake, Refactoring Workbook. Addison Wesley, Aug. 2003.

[4] JADE Website available from <http://jade.tillab.com/>

[5] H.Liu, L.Yang, Z.Niu, Z.Ma, and Shao, "Facilitating software refactoring with Appropriate Resolution Order of Bad Smells," Proc. Seventh Joint Meeting of the European Software Eng. Conf and the ACM SIGSOFT Symp. the Foundation of Software Eng., pp.265-268,2009.

[6] H.Liu, L.Yang, Z.Niu, Z.Ma, and W.Shao, "Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort," ", IEEE Trans. Software Eng., vol. 38, no. 1, Feb. 2012.

[7] F. Bellifemine, A. Poggi and G. Rimassa, "Developing Multi-agent Systems with JADE".

[8] Z. Shen, C. Miao, R. Gay and D. Li "Goal-Oriented Methodology for Agent System Development", IEICE Trans. INF. & SYST., Vol. E89-D, No.4 April 2006.

[9] M. Wooldridge and N.R.Jennings, "Intellignent agents: theory and practice," The Knowledge Engineering Review, vol. 10(2), pp.115-152, 1995.

[10] M.Nikraz, G.Caire and Parisa A.Bahri "A Methodology for the Analysis and Design of Multi-Agent System using JADE"

[11] G.Caire (TILAB, formerly CSELT) "JADE PROGRAMMING FOR BEGINNERS"

[12] M.R.Genesereth and S.p and Ketchpel,"Software Agents", "Communication of the Acm, Vol.37 (7), 1994.

[13] A Modern Approach by Stuart Russell and Peter Norvig, Prentice-Hall, 1995.

[14] Foundation for Intelligent Physical Agents (FIPA), see http://www.fipa.org/.

[15] M.Woolridge "An Introduction to MultiAgent Systems" JohnWiley & sons 2002.

[16] Antonio Barella, Soledad Valero and Carlos Carrus Cosa "JGOMAS: A New Approach to AI Teaching" IEEE Trans on Education, Volume 52, No: 2, May 2009.

[17] Fabia Bellifemine, G.Caire, AgostinoPoggi, G.Rimassa "JADE: A Software Framework for developing MultiAgent Application". Lessons Learned. Information and Software Technology. 50(2008) 10-21 Elsevier.www.ScienceDirect.com

[18] Rafael H.Bordini et al A Survey of Programming Languages and Platforms for Multi-Agent Systems "Informatica 30 (2006) 33-44.

[19] R.H.Bordini, J.F.Hubner and R.Viera. ”Jason and the Golden Fleece of agent oriented programming”. In Bordini et al[5], chapter 1, pages 3-37.

[20] L.Cardelli and A.D.Gordan. ”Mobile Ambients”. In M.Nivat, editor, Foundation of Software science and computational structures, volume 1378 of LNCS,pages 140-155.springer,1998.

[21] M. Dastani, M. B. Van, Riemsdijk and J.J.C.Meyer. ”Programming multi-agent systems in 3APl”. In Bordini et al[5], chapter 2, pages 39-67.

[22] R.Evertsz, M.Fletcher, R.Jones, J.Jarvis, J.Brusey and S.Dance.”Implementing industrial multi-agent system using JACK”. In Programming multi-agent systems, First International workshop (ProMAS’03), volume 3067 of LNAI, pages 18-48.SpringerVerlag, 2004.

[23] F. Brazier, C. Jonker and J. Treur. ”Principles of compositional multi-agent system development.”. In Proceedings of conference on Information technology and Knowledge systems, pages 347-360.Austrian Computer Society, 1998.