2012 Eighth International Conference on the Quality of Information and Communications Technology

A Framework to Diminish the Gap between the Business Specialist and the Software Designer

Fernando Wanderley
Programa de Pós-Graduação em Engenharia da Computação
Universidade de Pernambuco - UPE
Pernambuco, Brasil
fjaw@ecomp.poli.br

Denis Silva da Silveria
Programa de Pós-Graduação em Administração - PROPAD
Departamento de Ciências Administrativas - DCA
Universidade Federal de Pernambuco, UFPE
dsilveira@ufpe.br

Abstract - **Requirements Engineering establishes the process for defining requirements as one in which elicitation, modeling and analysis are tasks which must be carried out. This process should involve different stakeholders and their different viewpoints. Among these stakeholders, there is the software designer, responsible for creating models based on the information gathered by business specialists. However, this communication channel may create some "noise" that leads to information being lost. This loss produces a semantic gap between what is desired and what will be developed. The semantic gap is characterized by inconsistencies in the requirements represented by scenarios – user stories in a behavior-driven context – and by the conceptual model. This paper presents an interactive approach to the agile requirements modeling, thus fostering greater consistency between the artifacts of the scenarios and the conceptual model. This consistency is ensured by using a mind model specification which will serve as a basis for transforming the definitions of the scenario and generating a conceptual model represented by a UML class diagram. The mind model represents the main role of this approach, and functions as a bond that represents the business entities, thus enabling the requirements to be more consistent with the reality of the business.**

Keywords-component; Agile Modeling Requirements, Behaviour Driven Development, UML, Mind Map Modeling, Domain Model

I. INTRODUCTION

According to a study conducted by the Standish Group [1], about 66% of software programs did not meet users' expectations with regard to their functionalities and behaviors. In 1987, Brooks [2] pointed out that the most difficult part in building a software program is to understand, comprehend and decide precisely what will be built. However, no other part of the work is as hard as establishing communication that is aligned between the business specialist and the software designer that can consolidate the technical details of the business in one and the same model of requirements. No activity of the software development process generates so much loss as the Requirements Specification if this is conducted incorrectly [3, 4]. According to Evans [18], business experts talk to analysts, who should understand, model and pass the specification to the programmers. However, this process fails, because there is a lack of feedback on the common understanding among these stakeholders.

The responsibility of designers is to create models based on information conveyed to them by business specialists. But this channel of communication generates noises, dispensing – in a large majority of cases – with details on the business. In other words, in the process of communication between the domain expert and the software designer, information normally gets lost. This loss produces a semantic gap between what is wanted and what will actually be produced. The semantic gap is characterized by inconsistencies between the artifacts of requirements, here represented by scenarios (User Stories), within a context of a behavior-driven process, and the conceptual model of information.

Given the foregoing, this paper puts forward an interactive approach to the agile requirements modeling that fosters greater consistency among the artifacts of scenarios, scripts by business specialists, and the conceptual model of information. This article sets out to increase this consistency by transforming the mind models of the domain into a conceptual model of information, represented by the UML (Unified Modeling Language) class diagram, and of the generation of a vocabulary extracted from this conceptual model to support the business specialist in defining the scenarios. In other words, the mind model will play a central role in this approach, as it acts as a unifier in representing the business entities, thus enabling a transformation that is able to recover inferences and predictions that are compatible with what was really modeled by the business specialist.

This article is structured into the following sections: Section II below, addresses the theoretical framework needed to understand the approach proposed, Section III presents the details of the framework proposed; in Section IV a case study is presented that gives a practical illustration of the benefits expected from such an approach, and finally Section V lists conclusions and suggests further related studies that could usefully be undertaken.

II. THEORETICAL FOUNDATION

A. *Mind Map Modeling*

A Mind Map is a diagram used to connect words, ideas and concepts to a central idea or concept, which is used to view, classify and structure concepts and generate new ideas [19]. It is similar to a semantic network, or a cognitive map. However, there are no restrictions on the types of connections used (Figure 1).

In a mind map, the elements are ranked intuitively, in accordance with the importance of the concepts related to a domain, which are then organized into groupings, branches or areas. In other words, a mind map is a radial diagram that by using vocabulary (keywords), distributed in a certain order and

978-0-7695-4777-0/12 $26.00 © 2012 IEEE
DOI 10.1109/QUATIC.2012.9

correlated can cognitively represent and model a concept or a specific domain.

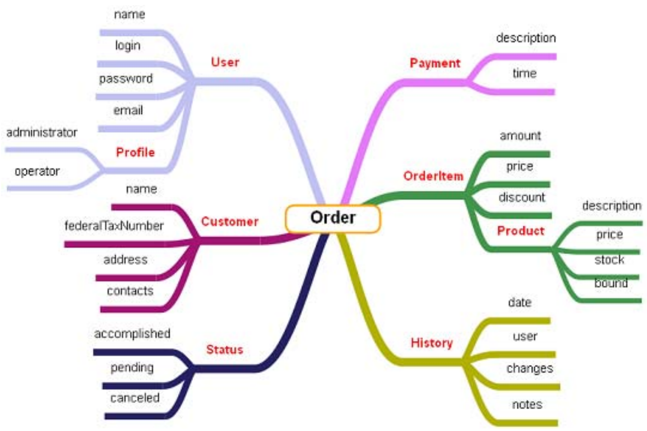


Figure1. Mind Map Modeling.

According to Buzan [19], the main benefits of using mind maps are: organization, use of keywords, association, grouping ideas, visual memory, creativity and innovation with simplicity

1) Mind Maps forExtracting a Domain

A mind map, in the context of this article, represents ubiquitous language, the purpose of which is to represent a cognitive map. Its objective is to extract, structure and organize knowledge or the domain in a spatial form. Some studies [20, 21] emphasize that this kind of representation makes it easier for the human mind to process information better, thus reducing the cognitive load so as to absorb knowledge or the domain.

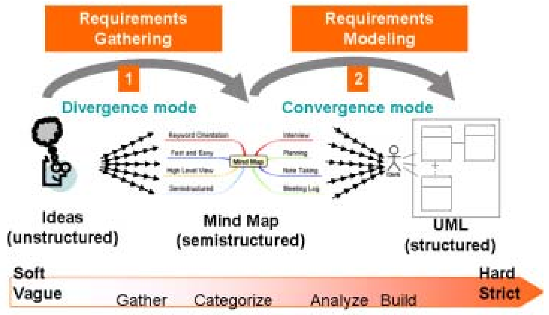


Figure 2. Semistructured Mind Mapping.

In this context, a mind map was used in this study with a view to facilitating communication, thus diminishing, according to some authors [22, 24], the semantic gap between domain specialists and designers. Figure 2 illustrates the role of the model which enables unstructured and vague ideas, coming from specialists, to be captured, so as to transform them - in a semi-automatic way - into a structured model, represented by the designers by using a conceptual diagram of information (class diagram). This transformation will result in a conceptual model of information that is more in adherence with the business.

Within an agile context, some authors [23, 24] introduce the idea of using mind maps as one of the agile practices to obtain and represent the requirements.

B. *Model and Ubiquitous Language*

Domain experts have a limited understanding of the technical jargon used in software development because they only use the jargon of their own dominion area. On the other hand, software designers can understand and discuss the system in descriptive and functional terms, shorn of the meaning transmitted by the expert´s language.

In the midst of this linguistic division, the experts of a dominion give a vague description of what they want. And, for their part, designers struggle to understand a new domain. Some members of both teams (experts and designers) manage to become bilingual, but they become bottlenecks of the information flow and their translations are inaccurate, thus generating semantic gaps.

In a project without a common language, without aligned knowledge generated by using an effective domain model, designers and developers have to translate for the domain experts and vice versa. This translation can generate imprecise models, which, typically, require to be shunted back and forth between them, thus increasing the cost of the project (Figure 3).

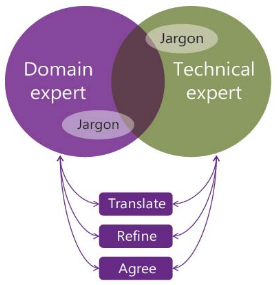


Figure 3. Translation Costs.

An indirect communication conceals the formation of divergences. This leads to a software program not being very reliable. The attempt at translation prevents the mutual exchange of knowledge and ideas that lead to consistent and uniform views of the model. The cost of any translation, besides the risk of misunderstandings, is normally high. According to Evans [18], a project needs a common language, which transmits the knowledge linked to the area, which may have a minimum of common understanding. The ideal is to use a ubiquitous language for the work of both teams. This language should include terms to discuss rules that will be made explicit in the model, and be complemented with terms from the domain (Figure 4).

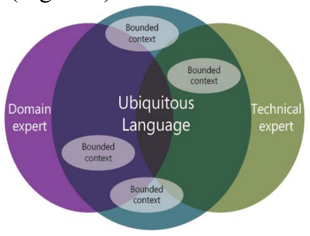


Figure 4. Ubiquitous Language.

The concept of a ubiquitous language is at the heart of Behaviour Driven Development (BDD) [5]. A ubiquitous language is built from the structure of a domain model. Using the domain model, vocabulary (keywords) is derived that will be used, like a dictionary, to define the behavior of the system [18]. The definition of a ubiquitous language needs to involve all the stakeholders.

1) Behaviour Driven Development

Having been defined by Dan North in [5], with the focus being on behavior, Behaviour Driven Development (BDD) emerged as the evolution of the concepts and practices of TDD (Test Driven Development) [6] and ATDD (Acceptance Test Driven Development [8]). According to North [5], TDD and ATDD are adopted by the industry so as to provide quality and high rates of productivity in software [9, 10]. Nevertheless, developers still raise some questions (i.e. what to test and what not to test; what to write in their tests; and how to understand why a test fails). Also according to North, both approaches (TDD and ATDD) are focused on verifying only the inputs and outputs of the functionalities. In other words, there is no concern about the description of the behavior that transforms the input elements into output. In these approaches, the description of behavior is made with the aid of natural language, which, typically, results in test cases that are difficult to understand.

Therefore, the main objective is to define the BDD specifications of the system that can be carried out, in which the tests can be written and easily understood by using a specific and ubiquitous language which should make it easier for the business specialists to define the expected behaviors for their respective functionalities.

Currently, there are several toolkits [13, 14, 15] used by the industry that support BDD. Some authors see BDD as a specification technique that automatically ensures that all functional requirements are dealt with appropriately by the source code, by connecting the textual description of these requirements in the automated tests, based on a common language between business specialists and software developers [16, 17].

BDD provides a pre-defined ubiquitous language for the review process that is independent of the domain, the scenarios and user stories of which it is proposed to structure (Figure 5).

[StoryTitle] (One line describing the story) As a [Role] I want a [Feature] So that I can get [Benefit]	Scenario 1: [Scenario Title] **Given** [Context] And [Some more contexts]…. **When** [Event] **Then** [Outcome] And [Some more outcomes]…. Scenario2: [Scenario Title] ….

Figure 5. Template of User Stories and Scenarios.

The next section presents the artifacts of the framework and gives evidence – using a case study – of the transformations involved

III. FRAMEWORK

As illustrated in Figure 6, the infrastructure of the framework proposed consists of three components: a *MindMappingModeler*; a *DomainModelExtractor* and a *DomainModelTool* which produce two artifacts: a *DomainMindModel* and a *DomainModel*. The infrastructure was developed on a Web platform, using Java, HTM5 and Javascript technologies.

The idea was to provide an environment in which all the stakeholders (business specialists and software designers) could share their views and abstractions on a domain in an environment that is considered simple and accessible to them all.

As per the diagram, the first section of interactivity of the environment occurs on using the *MindMappingModeler* component during which the business specialists and the software designers undertake the mind modeling of the domain collaboratively and produce the *DomainMindModel* artifact, as illustrated in Figure 1.

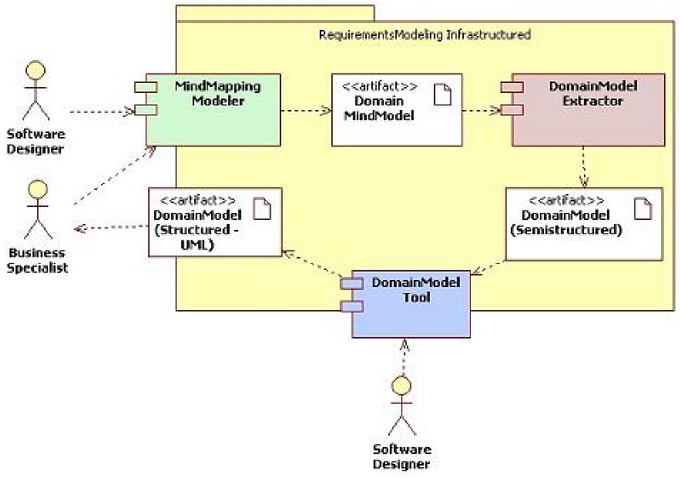

RequirementsModeling Infrastructured
MindMapping Modeler
<<artifact>> Domain MindModel
DomainModel Extractor
Software Designer
Business Specialist
<<artifact>> DomainModel (Structured - UML)
<<artifact>> DomainModel (Semistructured)
DomainModel Tool
Software Designer

Figure 6. Infrastructure Details.

The mind model artifact produced is exported by the component in a data structure represented in a Javascript - JSON file [25]. From this file, the *DomainModelExtractor* component performs processing (binding), with API Jackson support [26], and produces the *DomainModel* artifact, a semi-structured conceptual model, containing only the entities defined in the mind model and its attributes, as shown in Figure 8.

This (semi-structured) conceptual model of information is then developed by the software designer who attributes relationships between the entities (simple association, aggregation and composition) and describes its attributes with data types and access modifiers, with a view to modeling this domain represented by the mind model in a more formal perspective, in the format of the class diagram of the UML, than that of representing the conceptual model of information.

With the aim of avoiding losing information in this transformation process, the business specialist takes on the role of Reviewer of this conceptual model of information, by questioning the software designer on whether the relations designed in the class diagram are semantically consistent with the mind model. Should the specialist feels the need to make changes to this conceptual model, the process returns to the activity of mentally modeling the domain, so that the specialist may make his changes. This short cycle is repeated iteratively until the business specialist approves the class diagram, thus confirming that it represents the domain of his problem.

In the next section, the detailed process for using this infrastructure is demonstrated by means of a case study of a Subsystem for Managing Orders.

IV. CASE STUDY: *SUBSYSTEM FOR MANAGING ORDERS*

The case study deals with a system for managing product orders within a context of Business to Business (B2B) e-commerce. If we take this from the point of view of the suppliers, benefits arise such as that of there being a constant cycle of orders, thus creating a continuous flow of capital and avoiding situations of ad hoc requests at indeterminate times. However, new challenges loom up: The development of a logistics operation that serves the client within a satisfactory period of time is a *sine qua non* for the success of the process.

Against this context, the business rules for controlling orders should be expressly guaranteed and drawn up in a sufficient space of time.

According to the domain expert for the subsystem in question, in conceptual terms, an *Order*, must have: (1) an *Order Item* that is directly related to the (2) *Product*, the description and price of which is displayed on an invoice, and emphasizing moreover that an *Order Item* may only have one *Product* associated with it, resulting in there being a field for the quantity of the Item. The system should be able to manage the information of the (3) *User* who generated the order, and who can access the system with his login and password, with the (4) *Operator Profiles* or those of the *Administrator*. Due to the large flow of orders, the expert suggests that the system should manage the states (*Status*) of the orders and that these may be tagged as *accomplished*, *pending* or *cancelled*. As a functionality of a historical basis, the system must manage the information as to (6) *Payment* by holding a brief description and the time at which it was made, as well as a *History* (7) that reports the changes registered by the user.

Armed with the description of the system, the software designer will help the business specialist with the modelling of this domain by raising questions such as:

- What is the term and/or principal element of the domain of the problem?
- What are the terms and/or elements that are directly associated with this central term, that are relevant to the system?
- What information, essentially, must be managed, that is related to the terms defined above?

These questions help to extract from the business specialist responses relevant to the mind model that will be built. In this paper a metamodel is defined as a structure of the mind map of the domain, based on the patterns or building blocks defined by Evans [18], as shown in Figure 7.

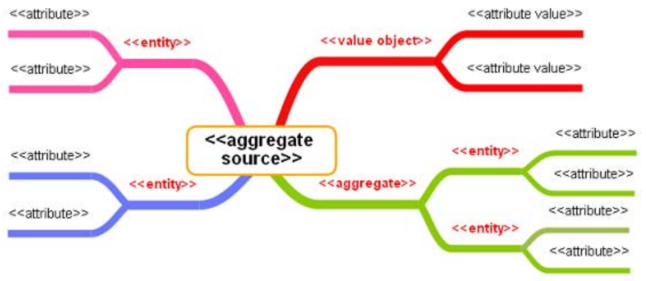


Figura 7. Metamodel of the Mind Map.

The central term is represented in the metamodel as an aggregate source. Evans, in [18], asserts that in a domain, there will always be a central object that will aggregate other secondary, correlated entities, in which the services requested by the client, to manipulate data from the domain, must pass through the source. What can be related to the source are entities - classes of objects that need an identity, which are normally elements of the domain that have a life cycle within the application, e.g. Customer; or value objects - classes of objects that only carry values**.**

Note that in the metamodel (Figure 10), there can be various terms and/or aggregate elements¸ that refer to when an entity aggregates another entity. However, there is always only one aggregate source.

Based on questions put forward by the designer, the business specialist produces the artifact of a mind model of the domain for the sub-system of managing orders, as shown in Figure 8.

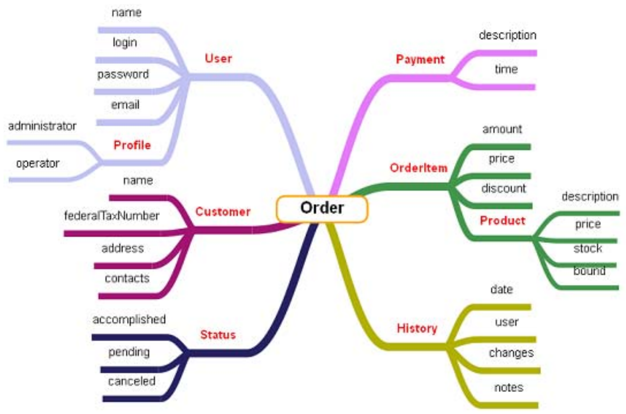


Figure 8. Mind Map Order Subsystem Domain.

Once designed and reviewed, the mind model is then imported into the extraction component of entities (*DomainModelExtractor*) in order to transform it into a (semi-structured) diagram of entities, as shown in figure 9.

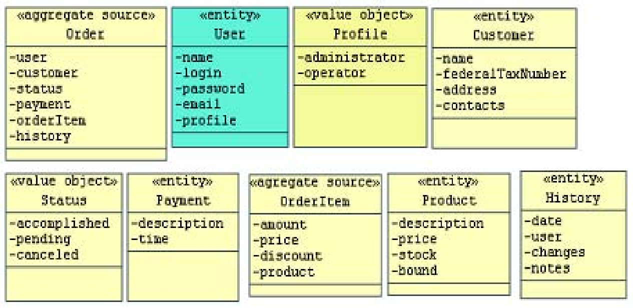


Figure 9. Conceptual Model (Semistructured).

This conceptual model, represented in a UML diagram, was generated from the existing correlation between the metamodel, of the mind model described in Figure 7 and the metamodel of the conceptual model presented in Figure 10.

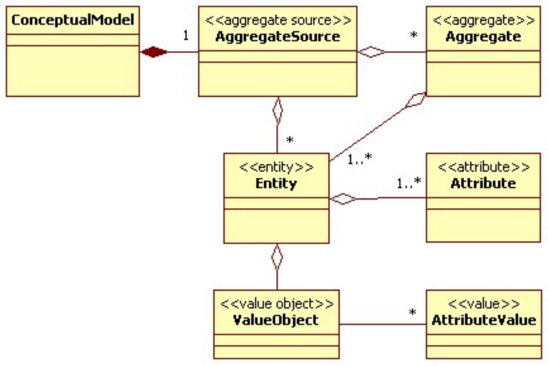
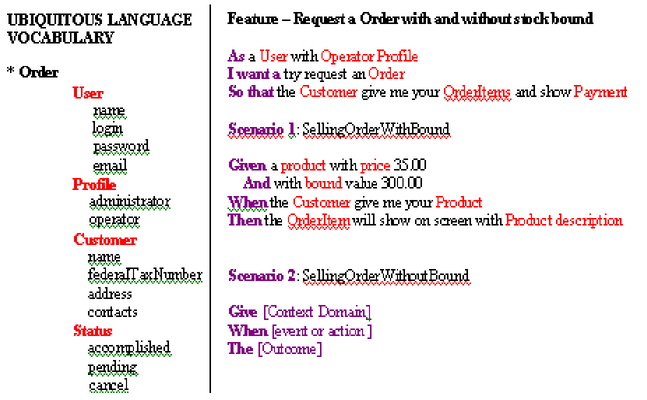


Figura 10. Conceptual Model Metamodel

In Figure 7 it can be noted that some colors were used in the metamodel to serve as a support, both as a cognitive one for the business specialist, and so that the transformation process may have a well-structured basis for reading and generating the conceptual model. This semi-structured model will be displayed in the component of modeling (*DomainModelTool*) so that the software designer, whose basis is the domain represented in the mind map, defined



UBIQUITOUS LANGUAGE VOCABULARY

*** Order**

User
name
login
password
email
Profile
administrator
operator
Customer
name
federalTaxNumber
address
contacts
Status
accomplished
pending
cancel

Feature – Request a Order with and without stock bound

As a User with Operator Profile
I want a try request an Order
So that the Customer give me your OrderItems and show Payment

Scenario 1: SellingOrderWithBound

Given a product with price 35.00
And with bound value 300.00
When the Customer give me your Product
Then the OrderItem will show on screen with Product description

Scenario 2: SellingOrderWithoutBound

Give [Context Domain]
When [event or action]
The [Outcome]

.Figure 14. Vocabulary Scenario Generated.

V. RELATED WORKS

Some works may be mentioned for being focused analyzing and specify a domain offering problem solving domain as this above, can design the correct associations between the entities, by relating them with the name of the relationship and the type of association. Shortly thereafter, the designer will define the data types and access modifiers to the attributes. This will evolve to the conceptual model of information represented in Figure 11.

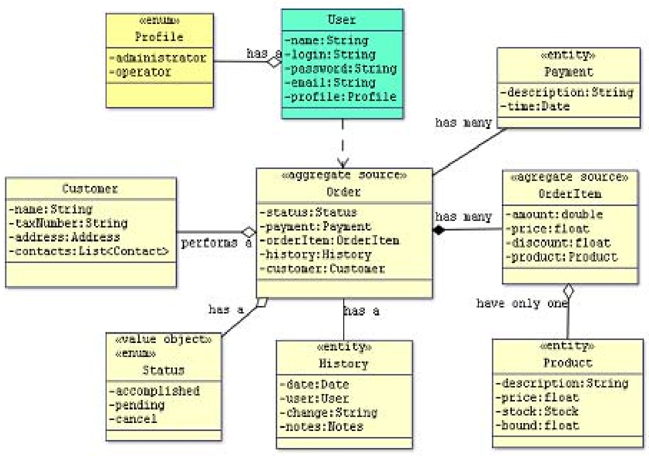
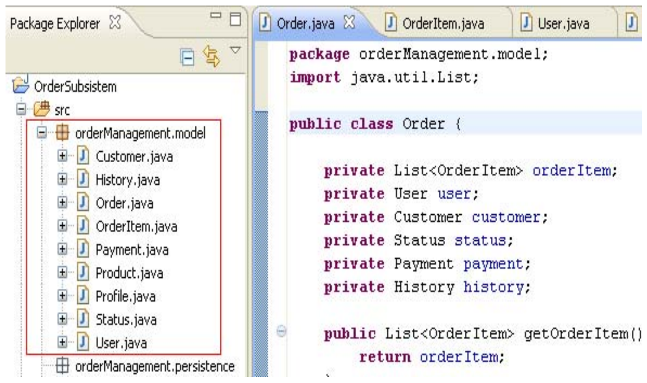


Figure 11. Conceptual Model of Information.

When the artifact of the conceptual model of information is finalized, the software designer asks the business specialist to review the model, with the focus on finding semantic inconsistencies with regard to the domain.

This proposed collaborative activity between the business specialist and the software designer makes it possible to generate more consistent models and proposes an exchange of knowledge using feedback obtained from the main stakeholders involved in the design at this specification phase.

After having been validated by the business specialist, the conceptual model of information will serve as the basis for extracting the vocabulary from the scenarios (Figure 12) and generating Java classes (Figure 13).



```
package orderManagement.model;
import java.util.List;

public class Order {

    private List<OrderItem> orderItem;
    private User user;
    private Customer customer;
    private Status status;
    private Payment payment;
    private History history;

    public List<OrderItem> getOrderItem()
        return orderItem;
```

Figure 13. Java Files Generated.

previously mentioned semantic gap. An important related work is the construction of domain-specific languages (DSLs) that define appropriate structures and notations for the specification of a particular domain of application. Thus, gains in expressiveness and usability are expected for domain experts. According to [29] between some of the main advantages of DSL (s) are: i) domain-specific abstractions predefined concepts to represent the application domain, and ii) concrete syntax through the support offered by a natural and intuitive notation for a domain and avoids confusion syntactic. Another important work related to the same context refers to the work of [30], Problem Frames, a requirements analysis approach that separates the problem description of system description to be developed, and proposes problems decomposition of into sub problems until classes already known and documented problems are found. These descriptions are based on areas (physical world parts, identifiable), which are composed of phenomena (such as entities, events and states), and their interactions, in order to facilitate the specification of the area separating the real world concepts and machines. However, compared to the approach proposed in this paper, in both techniques may be cited as a negative point - the cost of learning a new language or diagramming and modeling (Problem Frames) as well non-technical domain experts can find it hard to write or modify DSL programs by themselves [31].

VI. Conclusions and Future Studies

The main contribution of this study is to improve communication between Business Specialists and Software Designers with a view to reducing the semantic gaps and thus to produce more accurate and consistent requirements aligned to the real needs and expectations of the domain experts.

Other contributions obtained from the modeling of a domain by using a simple and cognitive notation, such as a mind map were evidenced by means of the support generated to the Behaviour Driven Development process (BDD).

Generating more consistent Domain Models related to users scenarios and stories, based on the semi-automatic processing of the Mind Model provided a systematized and flexible guide for constructing domain models for Software Designers.

Another important contribution was the support provided to Behaviour Driven Development, the analysis phase proposed, with the generation of a supporting vocabulary to define the scenarios (user stories).

With the aim of consolidating this infrastructure, some studies are already in progress: (1) formalizing the metamodel of the mind map of the domain, (2) using a transformation language that manipulates the elements of the model that takes into consideration the metamodels as a basis for transformation (i.e. Atlas Transformation Language – ATL [27]) for the transformation between (mind and domain) models and (3) conducting further experiments to validate the infrastructure.

References

[1] Available at: http://blog.standishgroup.com/ [Accessed November 20, 2011].
[2] F. P. Brooks, "No Silver Bullet- Essence and Accidents of Software Engineering" Computer Magazine; April - 1987 - University or North Carolina at Chapel Hill.
[3] Pressman, R.S., Software Engineering, McGraw-Hill, 6ª edição, 2006.
[4] Sommerville, I., Software Engineering, 8ª Edição. Pearson – Addison Wesley, 2007.
[5] D. North, Introducing BDD, 2006. Available at: http://dannorth.net/introducing-bdd [Accessed December 13, 2010].
[6] Beck, K. Test-Driven Development by Example, Addison Wesley, 2003
[7] D. Janzen, D.H. Saiedian, Test-driven development: concepts, taxonomy and future directions, Computer, vol.38, no. 9, pp. 43-50, Sept, 2005
[8] L Koskela.Test Driven: TDD and Acceptance TDD for Java Developers, Manning Publications, 2007.
[9] A.Gupta and P. Jalote. An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. In Proc. of Empirical Software Engineering and Measurement, 2007, pp.285-294
[10] D. Janzen and H. Saiedian. Does Test-Driven Development Really Improve Software Design Quality? IEEE Software. vol. 25, no. 2, 2008.
[11] D. Chelimsky, D. Astels, Z. Dennis, A. Hellesoy, D. North. The RSpec book: Behaviour Driven Development with RSpec, cucumber and friends, Pragmatic Bookshelf, 2010.
[12] D. Astels, A new look at test driven development, http://techblog.daveastels.com/files/BDD_Intro.pdf
[13] JBehave, http://jbehave.org/ [Accessed March 2012]
[14] Cucumber, http://cukes.info/ [Accessed March 2012]
[15] RSpec, http://rspec.info/ [Accessed March 2012]
[16] R. Carvalho, R. Soares Manhães, and F.L. de Carvalho, Filling the Gap between Business Process Modeling and Behaviour Driven Development, CoRR, 2008.
[17] R. Carvalho, F.L. de Carvalho, and R. Soares, Mapping Business Process Modeling constructs to Behavior Driven Development Ubiquitous Language, CoRR, 2010.
[18] E. Evans. Domain -Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003
[19] T. Buzan, "The Mind Map Book", 2003 BBC Active.
[20] R. G. Downs and D. Stea "Image & Environment: Cognitive Mapping and Spatial Behavior", 1973.
[21] R. M. Kitchin (1994). Maps Cognitive: Que são eles e porque estudo eles? *Jornal do psychology ambiental*, 14: 1-19.
[22] S. Robertson, J. Robertson, "Mastering the Requirements Process", 1999 Addison-Wesley
[23] C. Larman, "Agile and Iterative Development", 2003 Prentice Hall
[24] S. W. Ambler "Agile Modeling: Effective Practices for Extreme Programming and the Unified Process", 2002.
[25] C. Douglas, "Introduction JSON", 2008 – json.org [Accessed December 2010]
[26] Jackson JSON Processor http://jackson.codehaus.org/ [Accessed March 2012]
[27] ATL Language Transformation http://www.eclipse.org/atl/ [Accessed April 2012]
[28] Mernik, M.; Heering, J.; Sloane, A. M. When and how to develop domain-specific languages. ACM Computing Surveys, New York: ACM, 2005, v. 37, n. 4, p.316-344
[29] Czarnecki, K. Overview of Generative Software Development In: BANÂTRE, J.- P. et al. (Eds.). Unconventional Programming Paradigms. Berlin: Springer, 2005.p.326-341.
[30] Jackson, M. (2001). Problem Frames. Addison-Wesley. pp. 9,10.
[31] Freudenthal, M. "Domain Specific Languages in a Customs Information System.