IDENTIFYING FUNCTIONAL REQUIREMENTS INCONSISTENCIES IN MULTI-TEAM PROJECTS FRAMED INTO A MODEL-BASED METHODOLOGY

J.A. GARCÍA-GARCÍA [1], M. URBIETA [2,3], M.J. ESCALONA [1], G. ROSSI [2,3], J.G. ENRÍQUEZ [1]

[1] *Web Engineering and Early Testing (IWT2) Research Group. University of Seville,*

[2]*LIFIA, Facultad de Informática, Universidad Nacional de La Plata, Argentina*

[3]*CONICET, Argentina*

{julian.garcia, jose.gonzalez}@iwt2.org, mjescalona@us.es, {murbieta, gustavo}@lifia.info.unlp.edu.ar

REP (Requirements Engineering Process) is one of the most essential processes within the software project life cycle because it allows describing software product requirements. This specification should be as consistent as possible to enable estimating in a suitable manner the effort required to obtain the final product. REP is complex in itself, but this complexity is greatly increased in big, distributed and heterogeneous projects with multiple analyst teams and high integration among functional modules. This paper presents an approach for the systematic conciliation of functional requirements in big projects dealing with a model-based approach. It also explains how this approach may be implemented in the context of NDT (Navigational Development Techniques) methodology and finally, it describes a preliminary evaluation of our proposal in CALIPSOneo project by analyzing the improvements obtained with our approach.

Key words: Functional Requirements, NDT, Consistency, ambiguity, requirement gathering, distributed teams

1. Introduction

Requirements Engineering Process (REP) is the process of eliciting, understanding, specifying and validating customers' and users' requirements. Eliciting and defining these requirements are some of the most critical tasks in requirements engineering [1] since they are part of a complex, iterative and co-operative process that demands to analyze and identify the degree of functionality that the system has to fulfil in order to satisfy users' and customers' needs.

It is possible that ambiguities or inconsistencies appear in projects where teams of analysts carry out the application's requirements elicitation, due to different points of view of the same business concept [2]. There are some classical studies [3] showing the high impact of defining incomplete or ambiguous requirements. For example, Boehm suggested that *«requirements, specification and design errors are the most numerous in a system, averaging 64% compared to 36% for coding errors»*.

Ambiguities and/or inconsistencies do not only cause errors in systems, but also variations in the project scheduling due to requirement conciliation tasks. These problems may be exacerbated in projects where: (i) high integration between its modules is required; and (ii) big teams of analysts are simultaneously working, but in different modules. Consequently, it is necessary to carry out a validation and conciliation process consisting in analysis and consistency-checking tasks among requirements in

order to eliminate requirements ambiguity and contradictions as soon as possible, especially in early stages of software life cycle.

Traditionally, conciliation tasks have been performed through meeting-based techniques and tools [5]. However, a large number of requirement inconsistencies are not usually discovered on time (this is one of the most severe reasons of projects' overruns [6]).

In this context, the effort to correct the faults is a bit strong than correcting requirements at early stages [7]. This labor may not be too relevant in small projects, but it may be critical in big projects with complex modules where interaction and integration among modules is essential and different and multidisciplinary teams work together. In this situation, several viewpoints of the same requirement can raise. Some of the most visible problems regarding large teams working collaboratively on a project are listed below.

- In most cases, software analysts are not experts in the application domain, thus it can lead to misunderstandings when different analysts are defining parts or views of the same requirement. This situation can occur in a project with high degree of integration among the modules.
- There is often inadequate communication among analysts (and/or potential end users), consequently analysts and users may not have a common understanding of the utilized terms.
- Natural languages are often used to describe requirements, what entails inherent ambiguities and can lead to misinterpretations.

All these problems have been corroborated in a large and real R&D project called CALIPSOneo (advanCed Aeronautical soLutIons using Plm proceSses & tOols), which has been carried out by Airbus Military in liaison with our team IWT2 (Ingeniería Web y Testing Temprano) both to investigate how to improve the current industrialization design processes [41] and to facilitate a collaborative working environment for the multidisciplinary design teams. Considering the collaborative nature of CALIPSOneo, we uncover the necessity of proposing a formal and methodological approach that makes possible early recognition of ambiguities and inconsistencies (as well as conciliation of different viewpoints of the same requirement) when models of functional requirements are defined.

It must be explained that a functional requirement defines a function of a system or its component. This function is described as a set of inputs, behavior and outputs. At present, there are many techniques to define the behavior of functional requirements, but the two most commonly used are: scenarios (a textual and no-formal representation) and UML (Unified Modeling Language) activity diagrams.

In this context, this paper presents a theoretical and methodological proposal to identify inconsistencies and ambiguities in functional requirements when different teams of analysts work collaboratively. Our proposal also provides guidelines to cope with these inconsistencies in the early stages of the software life cycle. For this purpose, well-known techniques of text analysis and graph theory are combined to distinguish inconsistencies on functional requirements that are defined by means of scenarios or UML activity diagrams.

Once our theoretical proposal has been presented, the second objective of this paper is to integrate it into the practical environment of NDT (Navigational Development Techniques) [11] in order to have a real context where it can be practiced, validated and measured to improve costs. This paper also presents a real scenario (based on experiences of CALIPSOneo project) where the proposal has been validated and measured.

NDT is a Web methodology based on MDE (Model-driven Engineering) paradigm [38] and mainly focused on requirements. One of the main advantages of NDT is its prior and current usage in business

and academia environments[1] [8, 42, 43]. The great number of previous experiences has offered us a very relevant database for experimentation. For this purpose, we have selected NDT for our paper, although it is important to note that our proposal is general and it can be used on other approaches that use text for requirement's definition and activity diagrams such as UWE does.

The rest of this paper is structured as follows. Section 2 presents some related works in requirements validation and our background. Section 3 presents the problem that has been our catalyst to carry out this research (indeed, we have relied on a real project, i.e., CALIPSOneo project). Section 4 describes our model-based approach for identifying inconsistencies in functional requirements and how to deal with them. Section 5 presents the results of evaluating the approach on CALIPSOneo project but first a brief global vision of NDT offered. Next, Section 6 introduces the economic impact of the proposal on the cost of the project. And, finally, Section 7 states some conclusions and future lines of work.

2. Related Work & Background

Before developing our proposal to discover functional requirements inconsistencies in multi-teams projects, a Systematic Literature Review (SLR) has been carried out in order to understand the state-of-the-art of this issue as well as to take into account the existing proposals before facing the indicated problem up. This SLR is based on the protocol defined in [44]. This section will list several works related to requirements validation.

Requirements Engineering (RE) represents a coordinated effort to allow clients, users and software engineers to jointly formulate assumptions, constraints and goals to find out a software solution. However, one of the most challenging aspects of RE deals with identifying inconsistencies among requirements in the requirements phase. Thus, this phase is considered the most critical tasks in RE [1]. A global view presented in [10] divides this phase in three main tasks: requirements capture, requirements definition and requirements validation, being the last the task where conflicts are normally distinguished.

There are many works related to requirements validation, such as: [13], which analyzes how relevant a natural language in a systematic way is so as to improve the communication with the user; [14], which introduces the concept of Viewpoint as a standing or mental position used by an individual when examining a universe of discourse; or [15], where the authors propose the use of Petri Nets as a specific technique to validate requirements consistency defined as use cases. Nonetheless, these works do not specifically focus on techniques to ensure requirements consistency.

There have been several proposals in the last decade that have basically focused on identifying conflicts. The next paragraphs will describe some of them.

In [16], the authors recommend identifying the concerned divergences by means of a Multiple-Criteria Decision-Making method that can support aspectual conflicts management in aspect-oriented requirements. The results are limited since they point out the treatment regarding aspect-oriented requirements and they only deal with concerned conflicts.

From a UML-based perspective, the conflict-detection process in other phases of software life cycle has been deeply studied. In [17], the author proposes to find out conflicts in a twofold process: analyzing syntactic differences by raising candidate conflicts and understanding these differences from a semantic view. In [18], the authors suggest an approach based on logic descriptor, i.e., UML models are

[1]/ References to papers related to NDT experiences can be found at www.iwt2.org. Last accessed March 2017.

transformed into logic descriptor documents that are later processed by a first-order logic engine in charge of reasoning.

In [19], the authors present a tool for identifying conflicts in aspect-oriented requirements called EA-Analized, that processes Requirement Definition Language (RDL) specifications. It is possible to uncover conflict dependencies with high accuracy by classifying texts following Naive Bayes learning method.

Other authors recommend that inconsistencies among requirements can be identified using knowledge-based techniques. [20] presents a knowledge-based requirements engineering tool, named REInDetector. This tool supports automatic identification of inconsistencies studying the semantic of requirements after capturing each requirement by means of its descriptive logic language.

Moreover, UML has been widely studied for providing extensions and tools that allow modeling and developing high-quality applications. [30] analyzes empirically the relation between the level of detail of UML models and the resultant application's defect density. The outcome shows that the more models are detailed the less defects they report. The same authors do a throughout empirical research on consequences of imperfect models. They point out, that although there are defects that are easily uncovered by developers, they hardly identify most of them, such as Classes duplication, and therefore they rarely propagate the solution. Duplicated element definitions in models, such as Classes or Business Processes, are named Clones. Different techniques have been addressed for identifying clones in UML models [32; 33] or models repositories [31].

[33] offers a characterization of model clones causes and techniques for detecting duplication. Additionally, it proposes a tool for automatic detection of model clones, named MQlone, consisting in a plug-in for MagicDraw UML CASE tool.

In [31], the authors provide two approximate clone detection methods and evaluate their effectiveness in Business Processes repositories finding out clones that are refactored in shared processes.

Finally, regarding our background, we have conducted different research projects to look for inconsistencies and reconcile different kinds of requirements. In [23], we extend NDT-Suite with a new tool (NDT-Merge) that aims to help analysts in this task to save time. The process, using NDT methodology for the systematic identification of requirements inconsistencies, extends it to the conflicts resolution that already exists in some methodologies like WebSpec. The supporting tool is capable of solving conflicts for any types of requirements of NDT and their models. In that work, we mainly focus on the model of interaction requirements that organizes functional requirements through the construction of the future interface prototype, because these mechanisms bring into play many specific aspects of NDT and include generic processes used for the whole merging step. In [12], we present the application of a general model-driven approach for the systematic recognition of requirements inconsistencies. The approach is assessed by analyzing requirements for a project called Mosaico and the results show a reduction in effort-managing conflicts resolution, whenever we use our approach against an ad-hoc one.

3. Motivating Scenario: CALIPSOneo

In last ten years, NDT has been used in a large number of real projects developed by different companies, either public or private, which have provided us with an important feedback. One of them is CALIPSOneo project [8, 36, 37], which has been developed by Airbus Military together with other diverse and heterogeneous teams that have worked collaboratively in its different development phases. From the experience with this project, we have realized that requirements are difficult to conciliate in

projects involving multiple teams (in this case, it was carried out by Airbus Military together with the University of Seville in Spain, the Polytechnic University of Madrid in Spain, research foundations, such as FIDETIA, and other private software companies). After developing this project, an important feedback has been obtained concerning problems of conciliation when defining functional requirements. These problems are the source of this paper, which proposes an approach to solve these problems along requirements' analysis phase.

CALIPSOneo is an ambitious project based on PLM (Product Life cycle Management) [29] that aims to design a new PLM methodology to conform to a PLM collaborative design and the required development of the software that satisfies that concept. This methodology allows defining, simulating, optimizing and validating aeronautical assembly processes in a 3D virtual environment before these processes are implemented in a real shop floor.

Consequently, CALIPSOneo addresses three main areas: creation of the assembly process, creation of work instructions (WIs) and deployment of WIs. In this sense, it is organized into three subprojects: PROTEUS, MARS and ELARA. Figure 1 shows the interaction among those subprojects.

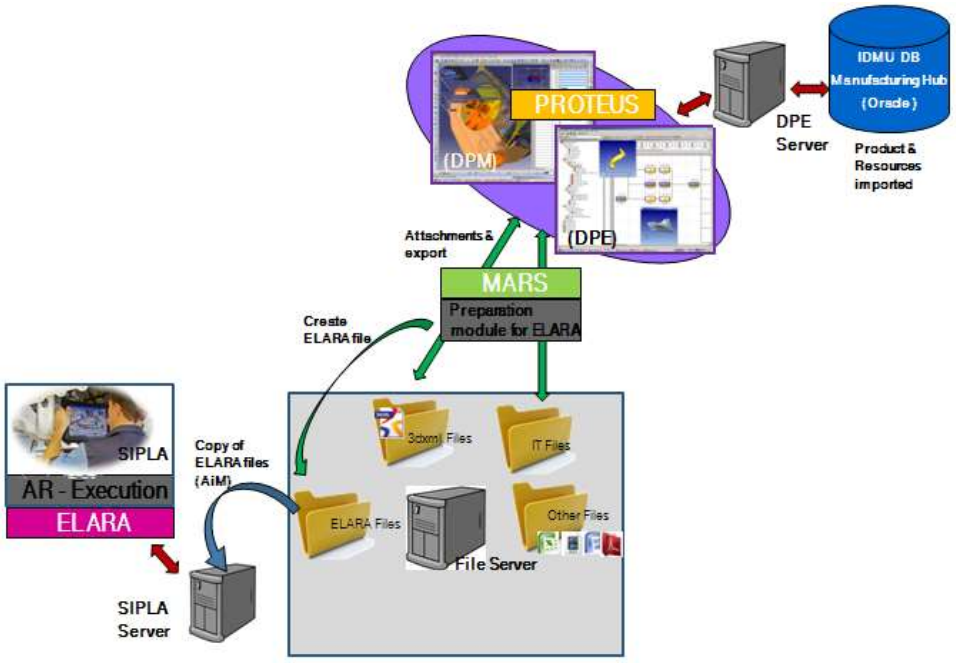

Figure 1. General scheme of the CALIPSOneo project.

PROTEUS focuses on the creation and maintenance of the assembly process that conforms iDMU (industrialization Digital Mock-Up) [27]. It includes the definition of process structures and resources, the allocation of product components and resources corresponding to each process node, the validation of process nodes and the definition of needed 3D simulations. The main issue addressed in PROTEUS is to define the data structure and the functions needed to create iDMU.

MARS is based on the creation and maintenance of WIs needed for the execution of the process defined in PROTEUS subproject. The definition of WIs must be as much automatic as possible, must

support multi-language and must comply with the requirements for the WIs deployment in the shop floor. WIs document the lowest level of process nodes. Since the execution of the assembly, processes can be carried out in different industrial plants located in several countries and the language of the WIs must be adapted to the language used in each location, without having to define the WIs again.

Finally, ELARA focuses on providing the information contained in WIs to the shop floor personnel. It includes the creation of augmented reality (AR) solution to show the right information to the worker when executing an assembly operation. This subproject uses the information generated in the subprojects PROTEUS and MARS, but adapted to its exploitation by means of AR. In prior projects, iDMU information was extracted in the form of 3DXML files and such files were used to create other files following the format used by AR solution. The positioning and tracking system was based on fiducial markers. ELARA provides two main contributions: (i) the use of information from an iDMU database previously prepared ad-hoc by MARS; and (ii) the solution to use aircraft components as natural markers for positioning and tracking.

These subprojects are independent and teams involved in each of them also differ. However, subprojects must be coordinated and they must be correctly integrated because they have common actors who demand regular functionality.

NDT and NDT-Suite were adapted to work in this project where a collaborative and distributed environment was necessary taking into account quality assurance along the definition and development of the project. Specifically, NDT-Profile was adapted to provide a collaborative environment according to NDT; NDT-Quality was used to measure and ensure quality and traceability among project results; and NDT-Driver was used both to automate the systematic generation of analysis and to test models from the requirements phase. This methodological working environment is described in detail in [45].

4. Approach for Identifying Conflicts in Functional Requirements

As mentioned above, this paper focuses on highlighting and resolving conflicts and inconsistencies of functional requirements when having distributed analysts teams whereas our previous proposals were focused on defining foundations [12, 23] that were composed in this approach. This section will describe our new approach in detail.

Due to large project complexity, our approach proposes a four-step process based on dividing and conquering by promoting different analyst teams who look at a specific subset of requirements. Figure 2 shows an overall schema of this process. As mentioned before, the proposal presented in this paper consists in an integral and comprehensive approach to uncover inconsistencies of different kinds of requirements. Figure 2 refers to this point together with the analysis of storage requirements, functional requirements and interaction requirements (user interfaces).

Basically, this paper focuses on the analysis of functional requirements taking into account their particular and specific features; storage and interaction requirements are studied in [23] and [12], respectively.

The process shown in this figure is iteratively utilized, which generates a new set of requirements each time it is used. The new incoming set of requirements is checked with each of the already consolidated requirements of the system space. Below, each step of this process will be further explained.

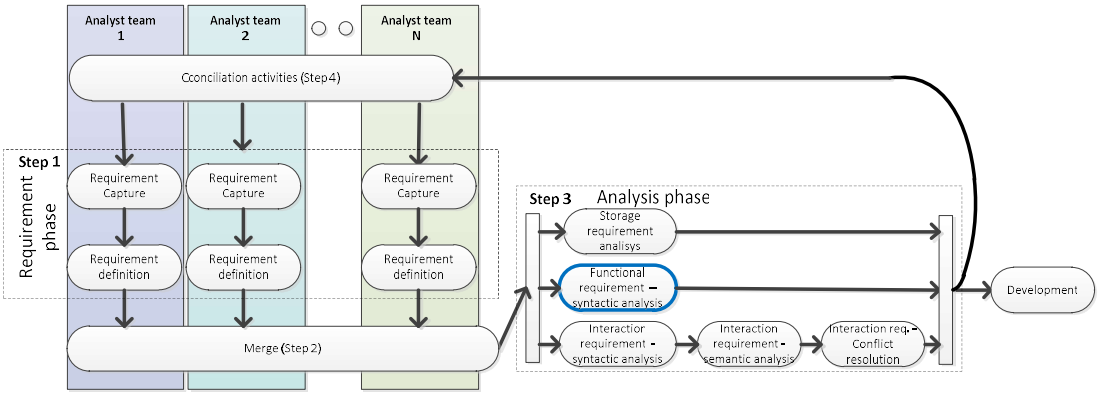

Figure 2. Distributed requirement-gathering process.

Step 1. Requirement Phase: Requirement Capture and Requirement Definition

This proposal combines classical capture requirements techniques, such as interviews or brainstormings (see [11]) for gathering functional requirements. Each team of analysts, normally isolated from others, performs this task.

There are different techniques to document functional requirements, for instance, use cases technique, which is a powerful and flexible method offered by UML for requirement elicitation. However, it is possible to define use cases in different and complementary ways as well as with several levels of detail. In this sense, the UML use case metaclass only defines the name of the use case and its relations with actors and other use cases, but it does not define its behavior.

To worsen this handicap, this paper proposes two ways to define the behavior of functional requirements (based on use cases of UML) depending on the desired level of detail. The first one is based on templates to express the behavior using textual scenarios, which are complete sentences with subject and predicate written in a non-formal language (commonly natural language). This mechanism is very suitable for simple functional requirements, but it can become too tricky for complex requirements with several routes or alternative ways. For this reason, this paper proposes a second way that is based on UML activity diagrams.

Step 2. Merging Requirements among different teams

It is necessary to merge the works of analyst teams when they specify different functional modules with a high degree of interaction and integration among them. This commitment is audited by a cross-domain analyst who watches over consistency of software requirement specification.

Step 3. Analysis Phase: Identifying Inconsistency in Functional Requirements

When dealing with model-driven methodologies, requirements are formalized using specific and formal languages (based on metamodels and their models) that help to describe the behavior of a system. These models face the same ambiguity issues as traditional techniques of requirement gathering, but the former

has an advantage: business concepts (such as entities, processes or tasks, among others) have already been pre-processed by the analyst obtaining a simplified and clear problem to solve.

Thus, once the previous step (merging) has been completed, a conciliation task starts (within the analysis phase of the project) in order to distinguish semantic, syntactic and structural inconsistencies in the specification of functional requirements. After that, if an issue is identified, it is reported to those analysts' owners of these functional requirements in order to solve the inconsistency. The detection of inconsistencies can entail holding meetings among team leaders of each subproject of a collaborative project.

As mentioned above, the two most commonly used techniques to define the behavior of functional requirements are: scenarios (a textual and no-formal representation) and UML activity diagrams. Therefore, we propose to combine different techniques to uncover inconsistencies in these types of behavior and improve the quality of its specification, which also affects the quality of the developed product.

1. The textual information of a functional requirement (description, scenarios and constraints, among others) is analyzed to search and find out syntactic conflicts. In light of this, we carry out the text analysis by means of varying the technique described in [28] known as *"vector space model technique"*.
2. The diagrammatic representation (based on UML activity diagrams) of a functional requirement is analyzed to discover structural inconsistencies. For this purpose, we use techniques of well-known graph theory, such as graph similarity or graph isomorphism. This analysis allows finding out structural conflicts. Nevertheless, we also propose to combine this technique with the aforementioned to identify inconsistencies in the textual information of elements included in an activity diagram (for example, description and name of activities, conditions in decision nodes or name of stakeholders.)

Regarding the identification of inconsistencies in the textual information of functional requirements, we use the *"vector space model technique"*, which has been successfully applied to another different context concerning this paper [23] and will be described briefly in the next paragraphs.

Identifying objects' duplication depends on the analysis of objects' description. Our study uses a variation of *"vector space model technique"* based on *"term frequency-inverse document frequency technique"*. This technique associates a mathematical equivalence to any text, i.e., n-dimensional vector, where *n* is the numbers of terms of the text. Each component stores the weight of each term, which is calculated by the scalar multiplication of two parameters: *tf* and *idf*. The former refers to the frequency of the word in the text, i.e., the number of occurrences of the term in the text divided by the total number of terms in the text. The latter, *idf*, refers to the inverse document frequency, and it evaluates the importance of the considered term in the whole set of descriptions. This definition allows giving a greater weight to the less frequent terms, which are considered as the most characteristic words. It is calculated by taking the logarithm of the quotient obtained by dividing the number of descriptions by the number of descriptions that contains the term. Equation 1 shows the mathematical expression of *idf*, where *D* is the corpus or set of descriptions analyzed, $|D|$ is the number of descriptions in the corpus and the denominator of the division represents the number of descriptions where the term *t* appears (this expression avoids a division-by-zero in cases where the term would be absent).

$$idf(t, D) = log \frac{|D|}{1 + |\{d \in D : t \in d\}|}$$

Equation 1. Importance of a term in the whole set of descriptions

Finally, Equation 2 shows the mathematical expression of the scalar multiplication of *tf* and *idf*. The similarity of the descriptions is evaluated taking into account that descriptions are vectors of words.

Since we consider them as vectors, we have to apply a single order of words. All the words in the whole set of descriptions have to be assessed and each new world implies a new dimension in the vector. The description's original order is not relevant, as having all the words is only necessary.

$$tf * idf(t, d, D) = tf\ (t, d) * idf(t, D)$$

Equation 2. Calculation of each component of the vector that represents textual definitions

Above, we have described techniques for building word vectors, but now we need a method to compare them in order to determine similarity of words meaning. After formulating two vectors (one for each description or text of two functional requirements), we can find out the similar measure between the two descriptions. For this purpose, we apply the cosine to calculate the angle between two vectors (Equation 3 where V_1 and V_2 are vectors for texts of words). The cosine with value 1 implies that the angle between vectors is 0, which implies that the texts are similar.

$$cos(\alpha) = \frac{V1.V2}{||V1||.||V2||}, (0 \leq cosine \leq 1)$$

Equation 3. Calculation of similarity between two textual definitions

Note that the dot product in the numerator is calculating numerical overlap between two vectors of words. Dividing by the respective lengths provides a length normalization that leads to the cosine of the angle between vectors.

Normalization is key since we would discard two word vectors to score highly for similarity simply because those words were frequent in the corpus (leading to high-term frequency counts as vector coefficients, and hence, as high numerical overlap). In this sense, first of all, words are stemmed to their roots to apply the technique described, i.e., plurals, verbal forms or other forms are not considered. We neither consider pronouns, articles nor other connexion terms. Then, the cosine similarity is applied, thus the algorithm calculates cosines between two vectors. Therefore, we understand that all the relevant words of the corpus have to be represented in the vectors. We have implemented the stemming algorithm using the string processing language called Snowball[2]. It has allowed us to use specific stemmers for languages such as English, Spanish and French.

The algorithm returns a number for similarity ratio between 0 and 1 when comparing two texts. Zero stands for completely different texts and numbers near to one mean similar texts. When comparing texts in Spanish, by means of the Spanish stemmer, the algorithm returns lower values for unrelated texts and higher numbers for similar texts.

Regarding the identification of structural inconsistencies in functional requirements, this proposal manages each functional requirement (defined as UML activity diagram) as a graph or state machine. Taking into account this issue, the methodology applies techniques of well-known graph theory (graph similarity or graph isomorphism) to find out inconsistencies. We profit from developments performed in UML field by [32; 33], in order to avoid model duplications and inconsistencies. Thus, our proposal takes each functional requirement (modeled using an activity diagram of UML) and builds an equivalent and optimized graph.

[2] Snowball is available at http://snowball.tartarus.org

Figure 3 shows two simple activity diagrams, their corresponding graph and how their differences are identified. On the one hand, the lower activity diagram has a start state, a simple state and a final state. Its graph representation shows five nodes. On the other hand, the upper activity diagram is a bit more complex and has a start state, two linked states and a final state. Its graph representation has seven nodes, among which two additional nodes are highlighted in grey. These last two nodes represent the difference between two activity diagrams.

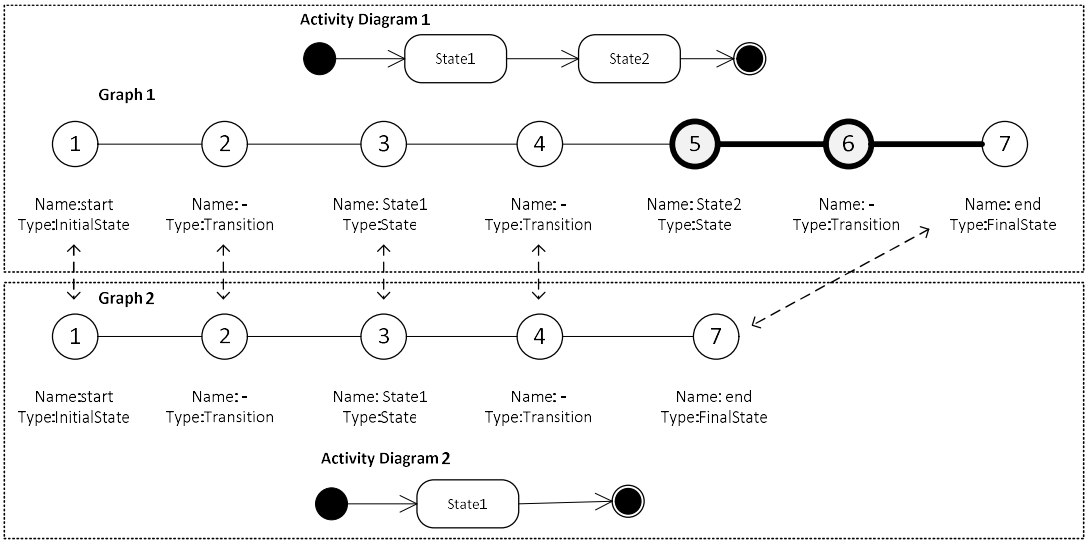

Figure 3. Graph representation of activity diagrams

Once activity diagrams have been derived to a navigable graph, our proposal takes two graphs and compares them looking for equality and inclusion in graph definition (same vertexes and edges). By means of identifying equality, we can improve budget estimation, since a given requirement does not compute twice. By detecting inclusion, it allows defining reusable concepts that simplify development and maintenance tasks.

We use well-known graph algorithms for isomorphism and equivalence analysis, in order to discover differences among models. Our tool is built on top of JGraphT [34] library, which provides a framework for graph computation. At this stage of this research, we identify a couple of common problems when modeling that can be identified using graph manipulation. For sake of space, next we will list two supported inconsistencies identification with a simplified graph operation: similarity and duplication among elements.

The first one is structural similarity on the definition of functional requirements. A similarity ratio is defined based on the amount of different element over total graph elements. This similarity is defined by means of comparing elements in model specification. $graph_{ad1}$ is the graph representation of the first activity diagram (ad_1) and $graph_{ad2}$ is a graph representation of the second activity diagram (ad_2). Equation 4 shows the mathematical formula to calculate the similarity between two requirements. Ratio values range from 0 to 1, where 0 stands for totally different models and values closer to 1 mean similar models.

$$ratio(ad1, ad2) = \frac{(graph_{ad1} \cup graph_{ad2}) - (\Delta(graph_{ad1}, graph_{ad2}))}{\#(graph_{ad1} \cup graph_{ad2})}$$

$$0 \leq ratio\ (ad1,ad2) \leq 1$$

Equation 4. Calculation of structural similarity between two functional requirements

Moreover, the second activity diagram refers to duplication of elements in the definition of functional requirements, i.e., occurrences of model duplications within other elements. This analysis is quite straightforward because, after removing redundant elements such as initial and final state, it can be checked whether a model is included or not within others. Equation 5 shows the mathematical expression to calculate duplications. If the intersection result is not empty, both diagrams share a few elements definition. It is noteworthy that the comparison among elements uses the technique for text analysis described before.

$$dupl\ (ad1, ad2) = 1 < graph_{ad1} \cap (graph_{ad2} - \{initial\ state, final\ state\})$$

Equation 5. Calculation of duplications between two functional requirements

Step 4. Conciliation Process

So far, we have shown how to identify conflicts that must be resolved in order to keep the requirements document sound and complete. Next, we will introduce a set of heuristics that helps to resolve structural conflicts of activity diagrams that have been implemented as suggested refactoring.

When facing structural conflicts up, there are functional requirements that may differ in their type or configuration. For example, the scenario of a functional requirement contains a step that is essential for the team of analysts of a system, whereas it is not taken into account by a different team of analysts.

In cases where a given element is absent in an activity diagram, but present in the other, we can be optimistic understanding that the best solution is to include the construction as an improvement when it is not present. This idea comes from the fact that new requirements may improve other requirement´s functionality; therefore, the new functional requirement may enrich an existing interaction.

Moreover, there may be ambiguity when two different teams analyze the same functional requirement from different points of view, i.e., when two stakeholders select different activity diagrams for the same requirement. This situation is naturally resolved by enriching the scenario in such a way, that the conflict can be solved by increasing the scenario detail.

As we have previously mentioned, different stakeholders may provide slightly different specification for the same application goal. Nonetheless, there are scenarios that are prone to address inconsistencies such as the presence of business objects hierarchies. At the requirement elicitation stage, business objects hierarchies may not be clearly uncovered and defined, and as a consequence, several structurally different business objects are referenced with the same name.

Conciliation cycles are strictly related to the introduction of a new requirement in the system. As Figure 2 shows, every time a new requirement is identified, it is modeled, and later syntactically and semantically analyzed. When an inconsistency is found, it must be solved, if possible, either by following the previously proposed conciliation rules or by means of meetings with stakeholders for disambiguating the situation. Once conflicts disappear, the requirement analysis process can start over the analysis cycle

for new requirements. Model checking process only finishes when there are no conflicts, with the aim of avoiding any inconsistencies. In addition, when our proposal does not provide clear solutions against conflicts in a specific case, our approach suggests solving them by means of meetings with stakeholders.

4.1. Extending existing MDWE approaches

Our approach relies on the use of textual definition and activity diagrams as the basic foundations but other tools has been developed as extensions for less standard approaches such us WebSpec and WebRE for OOHDM. In order to describe how to adopt this approach, we will present a set of suggestions that enables the approach in widely adopted Model-Driven Web Engineering approaches: WebML, OOHDM, and UWE. We do not consider NDT in the analysis as it is described in this work. These approaches have been studied deeply in [10] for understanding how they manage requirement gathering phases. Next, we present the Table 1 indicating whether an approach is compliant or not, and some guidelines to adapt the approach when it does not. In the analysis, following aspects are considered:

- **Natural Language**. It is an ambiguous technique to define requirements. Requirements are described in natural language without any kind of rules. Although this procedure is often criticized, it is quite often used in practice.
- **Glossary and Ontology** are used to define the terminology that should be used in every software project where stakeholders with different background work together.
- **Templates**. They are used to describe the objectives and requirements using natural language, but in a structured way. A template is a table whose fields have a predefined structure and are filled in by the development team using the user's terminology. Templates – also known as patterns – are less ambiguous than descriptions in natural language due to their structure. However, if templates are too structured they could be difficult to fill and maintain.
- **Scenarios** consist of the description of the characteristic of the application by means of a sequence of steps [46]. Scenarios can be represented in different ways: as texts or in a graphical form, e.g. by use cases [47].
- **Use Case Modeling** has been widely accepted as a technique to define requirements although it is also used in requirements eliciting as described in the previous section.
- **Formal description** is another important group of techniques that proposes in contrast to natural descriptions the use of formal languages to specify requirements. Algebraic specifications for example, have been applied in software engineering for some years. However, they are difficult to be used and understood by customers. Its main disadvantage is that they do not facilitate the communication between customer and analyst. Conversely, it is the least ambiguous requirements representation allowing for automatic verification techniques.
- **Prototypes** are a valuable tool for providing a context within which users are able to better understand the system they want to be built. There is a wide variety of prototypes that range from mock-ups of screen designs to test versions of software products. There is a strong overlap with the use of prototypes for validation.

	OOHDM	UWE	WebML

Natural Language			√
Glossaries		√	
Templates/Patterns			√
Scenarios	√ (based onWebSpec)	√	
Use Cases Analysis	√	√	√
Formal Language			
Prototyping	MockDD		
Other techniques	UIDs		
Compliant	⊗	⊗	⊗
Required adaptations	OOHDM uses WebSpec for specifying navigation requirements that has been enhanced with conflict detection features in [1]. For non-formal requirement definition either Software Requirement Specification (SRS) or User Stories (US) can be used. The *vector space model technique* is used for identifying conflicts.	UWE is fully compliant with the approach because it is rooted in UML standard. Activity diagrams are documented as part of any requirement. For non-formal requirements, SRS or US can be used which are suitable for the *vector space model technique* analysis.	WebML uses natural language for non-formal requirement analysis which can be processed with our approach for detecting inconsistences. Activity diagrams must be derived from Use Cases in order to check its validity.

Table 1. MDWE approaches summary and required adaptations

In this section we presented how the approach can be combined with MDWE approaches. The analysis only considers few of existing approaches but the analysis' rationale can be applied to other approaches as well.

5. Applying the approach on CALIPSOneo

5.1. Introduction to NDT

NDT (Navigational Development Techniques) [11] is a model-driven methodology that was initially defined to deal with requirements in Web development. Nowadays, NDT offers a complete MDE-based support for each phase of the software development life cycle (feasibility study, requirements, analysis, design, construction, as well as maintenance and testing phases) and it provides support to classical iterative and agile life cycles.

NDT defines formally a set of metamodels for each phase of its life cycle and uses OCL [4] in order to define semantic constraints, which ensure the definition of well-defined models. In addition, NDT

defines transformation rules (using QVT [39]) which make possible generate models from others systematically. This implies lower cost and quality improvement for software development.

Regarding the requirements phase, NDT offers a set of techniques to capture, define and validate different kinds of requirements: *storage requirements*, which define the information that is stored in the system; *actor*, which defines the user roles that interact with the system; *functional requirements*, which describe the functionality offered by the system; *interaction requirements*, which define user interfaces, the interaction of user role with these interfaces and how she/he can navigate through them; and *non-functional requirements*, which are used to catalog any other needs of the system which cannot be classified according to the above requirements. These kinds of requirement are formally defined by a metamodel and they can be traced to the remaining artifacts of the life cycle by managing them in a suitable manner.

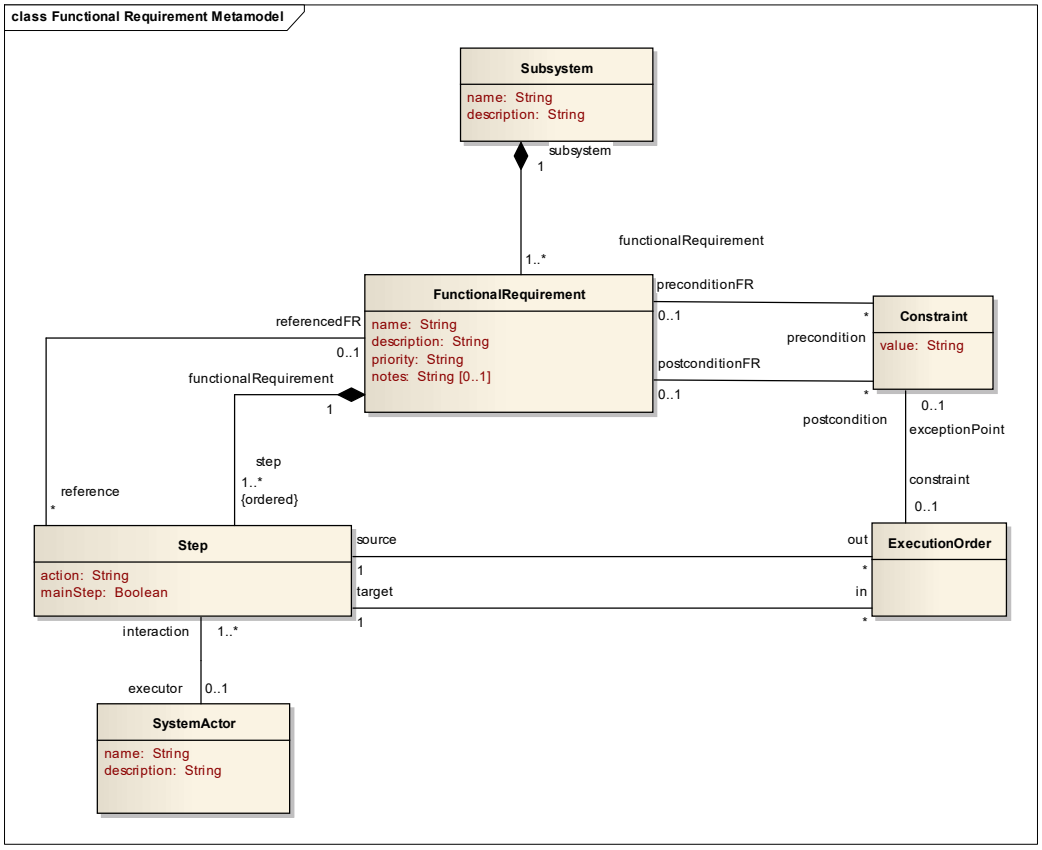

Figure 4. Functional requirements metamodel of NDT.

Regarding functional requirements, Figure 4 shows a simplified view of the functional requirements metamodel of NDT. In this metamodel, the *FunctionalRequirement* metaclass is the central concept that is composed of a set of ordered steps (represented by the *Step* metaclass). Such steps are executed by an actor (represented by the *SystemActor* metaclass) in a particular order of execution.

As stated above, NDT also defines transformation rules to generate models of a specific phase from other models of previous phases. These transformations use a set of heuristics to ensure consistency among models [11]. For example, NDT uses information previously captured, defined and validated in the requirements phase as the basis for the analysis phase (among others) [21].

The application of MDE-based methodologies (such as NDT) and, particularly, the application of transformations among models, may become monotonous and very expensive if there are no supporting tools. These tools automate the process in order to get all the potential of MDE, what provides a practical and useful environment to the enterprise environment. This aspect constitutes one of the virtues of NDT by which this methodology has been successfully applied to many real projects. In this sense, NDT defines a set of supporting tools grouped into NDT-Suite [22]. The main tools in NDT-Suite are: (i) NDT-Profile, which defines UML profiles in Enterprise Architect (EA) [9] for each NDT metamodel; (ii) NDT-Quality [24], which allows both, measuring automatically the quality of use of NDT at each of the software life cycle phases and checking the correct track of the transformation rules defined in NDT; and (iii) NDT-Driver [21], which allows running each QVT transformation among models in an automatic manner. Another interesting tool is NDT-Merge [23], which uses the comparison among models to identify syntactic and semantic inconsistencies among storage requirements in NDT.

With this toolkit, NDT has been and is being successfully applied to a large number of real projects. This experience has been important to choose this methodology for CALIPSOneo project [8].In addition, NDT has been an effective methodology in CALIPSOneo because it offers a formal process-based definition (named NDTQ-Framework [40]) that helps to develop software under quality framework, security and management processes. This environment is based on different reference models like CMMi (Capability Maturity Model Integration) [25] and ITIL (Information Technology Infrastructure Library) [26], and it is certified to be applied to real projects according to different standards like ISO 27001, ISO 9001:2008, UNE EN 16602 and ISO 14000. This paper does not aim to present NDTQ-Framework in detail, therefore further information can be downloaded from IWT2 website.

5.2. Identifying inconsistencies among functional requirements with NDT

The approach described in this paper has been integrated into NDT methodology as a real environment where the approach can be tested. For this purpose, each step shown in Figure 3 has been adapted to NDT features.

Step 1. Requirement phase: Requirement Capture and Requirement Definition

As mentioned in Section 5.1, NDT defines a theoretical metamodel to model functional requirements. This metamodel supports two ways to define the behavior of functional requirements, depending on the desired degree of detail: textual scenarios (for simple requirements) and UML activity diagrams (for complex requirements).

NDT also provides a supporting tool to instance this metamodel friendly. This tool is NDT-Profile and each analyst team should use it in order to specify structurally the catalog of requirements that each module of the end system contains. It is important to remember that this proposal is geared towards multi-team projects with a high degree of interaction and integration among modules; each partial functional view is a perspective of the application meanwhile requirement specification is a stable view of the whole solution.

When analysts have completed each requirement catalog represented in NDT-Profile, they should execute the next steps in order to identify requirements inconsistencies. Each analyst team carries out this task usually isolated from other teams.

Step 2. Merging Requirements among different teams

Once analyst teams define each requirement catalog, a cross-domain analyst, who watches over the consistency of requirements, needs to merge the work. According to NDT, this role is responsible for keeping consistency in requirement specification as well as for checking the correct use of NDT guidelines.

Step 3. Analysis Phase: Inconsistency identification in Functional Requirements

The process for uncover inconsistencies in functional requirements described in Section 4 has been implemented within changes in NDT. As previously commented, NDT allows defining functional requirements using scenarios (a textual and no-formal representation) or UML activity diagrams (a diagrammatic representation). In this context, Equation 1, Equation 2 and Equation 3 (*"vector space model technique"*) are used to define functional requirements using scenarios, whereas Equation 4 and Equation 5 (techniques based on graph similarity or graph isomorphism) are used to define functional requirements through activity diagrams. Both groups use NDT-Profile and EA as data sources because NDT-Profile contains the structured definition of each requirement catalog.

In order to avoid model mistakes, it has been necessary to extend the method by which the equivalent and optimized graph is built when a functional requirement is defined using an activity diagram. The reason is that EA includes extra information that is not present directly in the functional requirement. For instance, activities of an activity diagram (modeled in EA) are not directly related to the stakeholder (i.e., *swimlanes* in the terminology of EA). Elements are placed over *swimlanes* as overlaps, although a relationship among them is not defined but perceivable by a designer. The translation is straightforward: objects are Vertexs and relationships are Edges. An example is described below.

Figure 5 shows a simple activity diagram associated with a functional requirement of CALIPSOneo (this image has been blurred for confidentiality reasons). This requirement specifies functionality for reading a 3DXML file to be processed in ELARA in order to deploy augmented reality from industrial design.

Moreover, Figure 6 shows the activity diagram of another functional requirement. Now, this requirement describes how a reference of a 3DXML file is used to deploy augmented reality (this image has also been blurred for confidentiality reasons). This diagram is similar to the diagram shown in Figure 5, but expecting a reference instead of a XML file. These functional requirements are modeled by two different analyst teams and neither of them has noticed that the same requirement has been already defined.

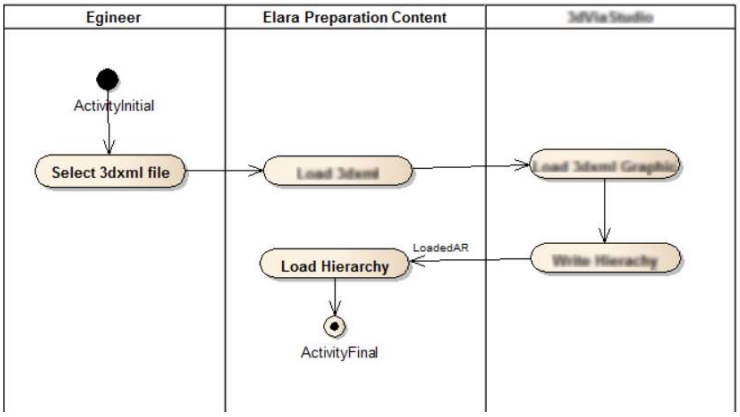

Figure 5. Functional requirement to process a 3DXML file

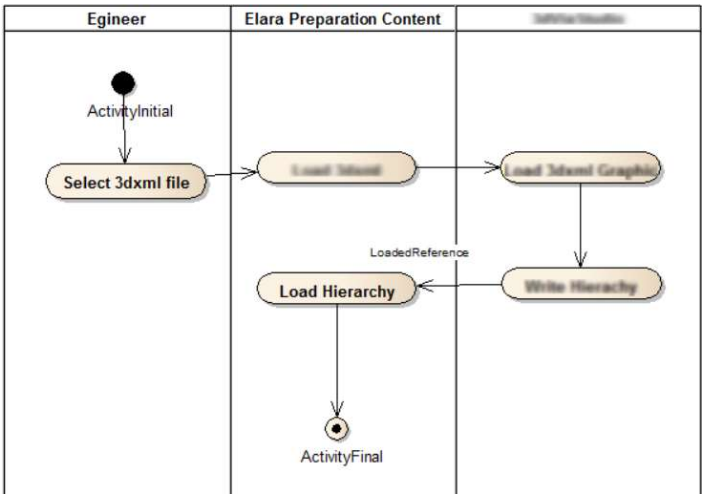

Figure 6. Functional requirement to process a 3DXML file by reference.

Firstly, a pair-based syntactic study is carried out for each activity of each activity diagram. Thus, each pair of functional requirements is compared among themselves to determine if there are similar activities in both requirements. Next, we use the aforementioned Equation 4 for detecting similarity. Equation 6 shows this result.

$$\boldsymbol{ratio} = \frac{\mathbf{7} - \#(\emptyset)}{(\{\boldsymbol{ActivityInitial}, \boldsymbol{Select\ 3dxml\ file}, \\ \dots, \\ \dots, \dots, \\ \boldsymbol{Load\ Hierarchy}, \boldsymbol{ActivityFinal}\})} \qquad \boldsymbol{ratio} = \frac{\mathbf{7} - \mathbf{0}}{\mathbf{7}} = \mathbf{1}$$

Equation 6. Example of calculation of the similarity between two functional requirements

After evaluating the phrases in the equation, we can observe that they represent the same requirement because the ratio is 1. Consequently, a conflict between requirements of Figure 5 and 6 is identified.

```
Check results with a high degree of similarity
RF-03.Create 3dxml files for ELARA it's included in RF-001.Create 3dxml files for ELARA
RF-114 Show/hide Graphic Tree and RF-012.Show/Hide Graphic Tree promedio 1.0
RF-122 Frames Management and RF-117 Annotation Management promedio 1.0
RF-014.Show/Hide Piece and RF-115 Show/hide Piece promedio 1.0
RF-102 Load 3dxml for AR and RF-103 Load 3dxml for Reference promedio 0.973018549747049
RF-013.Select Piece and RF-116 Select Piece promedio 0.7087628865979382
RF-001.Create 3dxml files for ELARA and RF-03.Create 3dxml files for ELARA promedio 0.9192610331850837
RF-103 Load 3dxml for Reference and RF-102 Load 3dxml for AR promedio 0.973018549747049
RF-110 Rotate and RF-109 Zoom promedio 0.9369257511795377
RF-116 Select Piece and RF-013.Select Piece promedio 0.8289655172413795
```

Figure 7. Tool report showing duplication requirements

Although such techniques can execute tasks for finding inconsistencies easier, it is important to provide supporting tools to automate all these systematic calculations. For this purpose, an initial software prototype has been developed and included in NDT-Suite. Once this software prototype is running in Calipso models (defined on NDT-Profile), our tool identifies both similar objects and duplicated definitions in different functional requirements. After comparing every requirement against all the other requirements, the report lists the similarity degree for each pair of aforementioned computed-based heuristics. Figure 7 shows a report dealing with the identified inconsistencies.

Step 4. Conciliation Process

This step of the process for detecting inconsistencies in functional requirements described in Section 4 has been implemented within changes on NDT.

6. Validation on a Real Application Case

This section presents the validation of our approach and evaluates its suitability in the real environment of CALIPSOneo. At present, CALIPSOneo is an implanted system that is being used by technicians in some parts of the aircraft manufacturing process. Although CALIPSOneo is already finished, we use time spent in the requirements gathering phase during the project execution to compare such values against our approach performance. This information was used for evaluating time and effort that our approach saves.

The aim of this section is not to conduct a full experiment because it is out of this article's scope; indeed, it aim at introducing insights, issues and ideas for an empirical evaluation with statistical significance. Our purpose is to present an application case to validate our proposal and have initial data to validate our proposal. This evaluation does not pretend to be exhaustive; in fact, it aim at providing an initial proof for approach's benefits. Nonetheless, we consider future work to perform a formal empirical experiment to validate our proposal.

6.1 Context of the validation: teams and scenario

Our approach tries to reduce the effort required by analysis tasks, that is 367 hours of dedication per each team of analysts (three teams in total: MARS, PROTEUS and ELARA). The approach presented in Section 5 systematizes the identification of conflicts in functional requirements. For that purpose, we

will analyze how this approach could improve the execution of a project such as CALIPSOneo as well as its quality and cost.

We evaluated our proposal with a small team of analysts and extrapolated the result to the context of the whole project. This approach aims at providing evidence of an improvement that will be deeply empirically studied in an experiment as part of a further work. Reproducing the whole requirement gathering and analysis phases was not feasible as it would require several and hardly available stakeholders, analysts and project managers' time. In this sense, our evaluation proposed three teams (for MARS, PROTEUS and ELARA, respectively). Each team was composed of a junior analyst plus a senior analyst, who were experts in their own project. For this experiment, we followed the next steps presented in Table 2.

1.	We provided each team with the simplified catalog of functional requirements of MARS, PROTEUS and ELARA. The first step was not the key point of our validation process since assessing the rest of steps of our proposal was more important. Thus, each team started with the requirements modeled in NDT-Profile by each team of analysts. Each subproject had three functional requirements of different complexity (low, medium and high).
2.	Starting with these requirements, junior analysts had to analyze inconsistencies in their functional requirements and, later, they checked each found difference together with senior analysts. That step of the experiment was performed using the guidelines we had followed along the project, i.e. each comparison task and reconciliation through meetings were handmade. This application will be explained in Section 6.2.
3.	We presented in detail the approach this paper proposes to each team.
4.	The step 2 of this experiment was performed again, but that time, each team used our proposal and our software prototype in order to apply them in an automatic way.

Table 2. Validation scenario

We measured the time spent in steps 2 and 4 of this scenario to obtain comparable results with the original conciliation of CALIPSOneo, this way we could compare the effectiveness of our proposal. The next section will further describe the second step of the validation scenario.

6.2 Execution of the validation scenario

The first phase of the validation was to measure how much time the analysts spent to find out inconsistencies in their functional requirements in a manual way according to CALIPSOneo's guidelines. In each CALIPSOneo's subproject (MARS, PROTEUS and ELARA), meetings were held every week. Along the meetings, work teams discussed the possible integration problems when they were analyzing the catalog of functional requirements in each subproject.

Inconsistencies in low-complexity requirements were easily detected by junior analysts. However, the intervention of senior analysts was necessary to assess inconsistencies of requirements. That situation posed a greater expenditure of time during analysis sessions. In these reviewing sessions, two main problems were pointed out:

1. Inconsistences were "discovered", without any special mechanism or technique and their detection depended on the team's experience, especially when the requirement was more complex.
2. A found inconsistency was solved by means of discussions among teams and depending on the nature of such inconsistency, for instance, some discussions were held in local teams (MARS, PROTEUS or ELARA) or even, if the inconsistency affected several subprojects, it entailed global meetings involving several teams.

Besides the human factors that can compromise the detection of inconsistences such as skills, experience and focus, among others, the execution of the second point affects directly the project's budget. To make it even worse, if they affected the three teams, the conciliation process would become too expensive as members of three teams together with project leaders of the affected subprojects and functional users would have to participate and discuss about different solutions.

Basically, our approach was used when each team analyzed its three functional requirements manually. At that moment, inconsistencies were detected and the time spent in meetings was reduced.

6.2 Measure the quality: impact in budget

The evaluation was focused on measuring efficiency and affectivity of our approach against an ad-hoc way to identify requirement conflicts.

Regarding efficiency, Table 3 presents the hours used by each Senior Analyst (SA; the cost is around 35€/hour) and each Junior Analyst (JA; the cost is around 20€/hour) for step 2 (first detection and resolution of inconsistencies in a manual way) and step 4 (second detection and resolution of inconsistencies using our proposal and our software tool) of the validation scenario described in Table 2.

Validation Scenario	**MARS**		**PROTEUS**		**ELARA**		**Averages**	
	JA	**SA**	**JA**	**SA**	**JA**	**SA**	**JA**	**SA**
First detection (step 2)	32.5	19	31	17	32	17	31.8	17.6
Second detection (step 4)	16	8	14	7	15.5	8.5	15.2	7.8

Table 3. Detailed cost (hours) by each step of the validation scenario

After carrying out the experiment, the time the approach took was reduced 52.20% for JA and 55.60% for SA (taking into account average values). The amount invested in the first and the second phase of the experiment were 1,252 € and 577 € (average cost of SA and JA), respectively. The difference was 675 € (reduction of 53% of the budget). Obviously, these measures are only a simulation, but they offer very attractive results to continue incorporating this approach in NDT.

Regarding efficiency, the detection of inconsistencies was around 75%, that is, our approach did not find the same inconsistencies as the analysis did. However, with regard to semantic inconsistencies, the approach only detected differences but not inconsistencies.

The performance difference can be argued by the fact of having a method that ruled the experience. Originally, inconsistencies were analyzed informally, what led analysts to perform such task together with stakeholders. With this approach, most of the work was organized and, although it was tedious to evaluate, as it will be later discussed, the whole process was controlled and effective.

Additionally, each team performed manually the process using a spread sheet that was derived from NDT-Profile. This method was chosen to evaluate the proposal without conditioning it by using a specific tool. This document stored different data structures obtained from requirement analysis. Using the embedded spread sheet query engine, the most important set of operations (described in Section 5) needed to automate analysis tasks were implemented. This simple resource ease, together with the method already described, made effort and time spent originally decrease.

7. Conclusions and Future Work

One of the most relevant phases in the life cycle of a software project is the requirements phase, which conditions the development of all the aspects of a project, mainly those regarding costs. Either the diversity of data that the system has to manage or the diversity of users shows the complexity that analysts must face up. With the increase of complexity in applications within big, distributed and heterogeneous projects, this phase acquires a more relevant role because these systems are often specified by multiple analyst teams and in this context, it is necessary to perform an effective conciliation of requirements.

In cases where there are different sets of requirements, they have been merged to obtain conciliated requirements with the aim to initiate the system development. However, this task frequently depends on the analyst's experience or it is done manually, without a particular normalized support to develop it. Thus, it is necessary to establish formal mechanisms to combine different requirement specifications and detect conflicts among these requirements.

This paper is an extension of a previous paper that presented the use of a general model-driven approach for the systematic identification of requirements inconsistencies and how that approach was adapted and extended to improve NDT methodology.

With this new study, we have the intention to define a proposal focused on identifying inconsistencies among functional requirements in NDT. For this purpose, we have looked at how NDT defines functional requirements, that is, using textual templates (with non-formal language) to express the behavior, using steps or by means of UML activity diagrams.

Consequently, our proposal is based on techniques for detecting similarities between graphs and techniques for the detection of syntactic conflicts in a textual manner. In addition, this paper illustrates the use of our proposal in a project, through a real example that measures the improvement that a project can offer, with an empirical example named CALIPSOneo that originally was conciliated by hand without the use of any mechanism to supervise it.

We plan to extend the current characterization of common problems present in use case artifacts and provide a suitable solution.

Identified problems in functional requirements, use cases and user interfaces are analyzed in isolation without taking into account other models. Thus, we propose to study the relation of inconsistency with other models in order to provide trace tools. We will apply new developments over the catalog of applications that our group have modeled with NDT, to measure the benefits that our approach can offer.

Obviously, the advantages regarding cost reduction described in Table 4 are quite encouraging. Nevertheless, when applying our approach, the analysts discovered some relevant problems using the current guidelines of CALIPSOneo (i.e., guidelines based on manual tasks and meetings) to detect inconsistencies among functional requirements.

On the one hand, the manual application of these guidelines is completely unrealistic. In fact, the analyst team has to check manually every description and activity diagram for each functional requirement. This is a laborious task that involves an investment that must be considered. Furthermore, as it is a manual task performed by technicians, it may pose some errors. The teams of every subproject hold meetings weekly to review the evolution of the requirements and they arrange global meetings every month. Besides, the team responsible for quality in the project participates in every meeting. The cost of these meetings is too high and it could be reduced through approaches like the one we propose in this paper.

On the other hand, although this paper presents a preliminary evaluation in order to assess the approach feasibility. During the simple evaluation, analysts have obtained very valuable results to detect inconsistencies. This advantage allows reducing the number of errors and failures in the communication among systems (MARS, PROTEUS or ELARA) as well as improves the quality of the developed product. The outcome of this preliminary evaluation is going to be used to design and run an experiment that aim at reporting empirical evidence of this approach.

Finally, we plan to introduce this approach in most projects currently running in the research laboratory in order to gather relevant statistical evidence of its contributions and improvements in the development process.

Acknowledgements

This research has been supported by the POLOLAS project (TIN2016-76956-C3-2-R), by the MEGUS project (TIN2013-46928-C3-3-R), by the SoftPLM Network (TIN2015-71938-REDT) of the Spanish Ministry of Economy and Competitiveness and by the VPPI of the University of Seville.

References

1. Robles, E., Garrigós, I., Grigera, J., Winckler, M.: Capture and Evolution of Web Re-quirements Using WebSpec. ICWE 2010:173-188 (2010).
2. Kotonya, G.; Sommerville, I.: Software Engineering Journal, vol. 11, no. 1, pp. 5-18 (1996).
3. Boehm, B.: Industrial software metrics top 10 list. IEEE software, 4, (5), pp. 84-85. 1987.
4. ISO/IEC. ISO/IEC 19507:2012 Information Technology - Object Management Group Object Constraint Language (OCL). International Organization for Standardization, formal/2012-05-09, 2012.
5. De Lucía, A., Qusef, A.: Requirements Engineering in Agile Software Development. In Journal of Emerging Technologies in Web Intelligence, Vol. 2, No. 3 (2010), pp. 212-220 (2010).
6. Yang, D., Wang, Q., Li, M., Yang, Y., Ye, K., Du, J.: A survey on software cost estimation in the Chinese software industry. ESEM 2008:253-262 (2008).
7. Leffingwell, D.: Calculating the Return on Investment From More Effective Require-ments Management. AMERICAN PROGRAMMER, 1997, vol. 10; No. 4, pp. 13-16 (1997).
8. Escalona, M.J., García-García, J.A., Mas, F., Oliva, M., Del Valle, C.: Applying model-driven paradigm: CALIPSOneo experience. Proceedings of the Industrial Track of the Conference on Advanced Information Systems Engineering 2013 (CAiSE'13), vol. 1017, pp. 25-32. Valencia, Spain, June 21, 2013.

9. SparxSystems, Enterprise Architect. Available in http://www.sparxsystems.com. Accessed in May 2016.
10. Escalona, M.J., Koch, N.: Requirements Engineering for Web Applications: A Survey. Journal of Web Engineering. Vol. II. No. 2. pp. 193-212 (2004).
11. Escalona, M.J., Aragón, G. 2008. NDT: A Model-Driven Approach for Web requirements, IEEE Transactions on Software Engineering. vol. 34. no. 3. pp. 370-390.
12. Escalona M. J., Urbieta M., Rossi G., García-García J. A., Luna E. R.: Detecting Web requirements conflicts and inconsistencies under a model-based perspective. Journal of Systems and Software, 86(12), 3024-3038. 2013.
13. Leite, J.C.S.P.: Eliciting Requirements Using a Natural Language Based Approach: The Case of the Meeting Scheduler Problem. Monografías em Ciência da ComputaÇao. No. 13. (1993).
14. Leite, J.C.S.P.: Requirements Validation through Viewpoint Resolution. IEEE Transaction on Software Engineering. Vol. 17. No.12. pp. 1253-1269. 1991.
15. Silva, J.R., dos Santos, E.A.: Applying Petri Nets to requirements validation. ABCM Symposium. Series in Mechatronics. Vol 1. pp. 508-517. (2004).
16. Brito, I. S., Vieira, F., Moreira, A., Ribeiro, R. A.: Handling conflicts in aspectual re-quirements compositions. In Transactions on aspect-oriented software development III, LNCS, Vol. 4620. Springer-Verlag, Berlin, Heidelberg 144-166 (2007).
17. Altmanninger, K.: Models in Conflict - Towards a Semantically Enhanced Version Control System for Models. MoDELS Workshops 2007:293-304 (2007).
18. Van Der Straeten, R., Mens, T., Simmonds, J., Jonckers, V.: Using Description Logic to Maintain Consistency between UML Models. UML 2003:326-340 (2003).
19. Sardinha, A., Chitchyan, R., Weston, N., Greenwood, P., Awais, R.: EA-Analyzer: Automating Conflict Detection in Aspect-Oriented Requirements. ASE 2009: 530-534, (2009).
20. T.H. Nguyen, B.Q Vo, M. Lumpe, J. Grundy. 2012. REInDetector: a framework for knowledge-based requirements engineering. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012). ACM, New York, NY, USA, 386-389. DOI=10.1145/2351676.2351754.
21. García-García, J.A., Cutilla, C.R., Escalona, M.J., Alba, M., Torres, J. 2011. NDT-Driver, a java tool to support QVT transformations for NDT. The 20th International Conference on Information Systems Development (ISD 2011). ISBN: 978-1-4614-4950-8.
22. García-García, J. A., Alba, M., García-Borgoñon, L., Escalona, M. J.: NDT-Suite: A Model-Based Suite for the Application of NDT, LNCS 7387, pp. 469–472, (2012a)
23. García-García, J. A., Escalona, M. J., Ravel, E., Rossi, G., Urbieta, M.: NDT-merge: a future tool for conciliating software requirements in MDE environments. iiWAS 2012:177-186 (2012). ISBN/ISSN: 978-1-4503-1306-3. Bali, Indonesia. 2012b.
24. J.A. García-García, J, Victorio, L. García-Borgoñón, M.A. Barcelona, F.J. Domínguez-Mayo, M.J. Escalona. "A Formal Demonstration of NDT-Quality: A Tool for Measuring the Quality using NDT Methodology". The 21st Annual Software Quality Management (SQM) conference. 2013. ISBN 978-0-9563140-8-6.
25. Chrissie, M.B., Konrad, M., Shrum S.: "CMMI® for Development: Guidelines for Process Integration and Product Improvement". Editorial Pearson Education. 2011.
26. Jong A., Kolthof A.: "Fundamentos de ITIL, Volumen 3," Van Haren Publishing, ISBN 9087530609, 2008.
27. Mas, F., Gómez. A., Menéndez, J.L., Ríos, J.: Proposal for the conceptual design of aeronautical final assembly lines based on the iDMU concept. 10th International Conference on Product Lifecycle Management. 2013.
28. Salton, G., Buckley C. 1988. Term-Weighting approaches in automatic text retrieval. Department of Computer Science, Cornell University, Ithaca, NY 14853, USA.

29. Stark, J.: Product Lifecycle Management: 21st Century Paradigm for Product Realisation. Springer Science, ISBN: 978-0-85729-545-3, 2011.
30. A. Nugroho, B. Flaton, M. R. Chaudron. Empirical Analysis of the Relation between Level of Detail in UML Models and Defect Density. In Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS '08).
31. C. C. Ekanayake, M. Dumas, L. García-Bañuelos, M. La Rosa, and A. H. M. ter Hofstede. 2012. Approximate clone detection in repositories of business process models. In Proceedings of the 10th International Conference on Business Process Management (BPM'12)
32. Störrle, H. 2010. Towards clone detection in UML domain models. In Proceedings of the 4th European Conference on Software Architecture: Companion Volume (ECSA '10), Carlos E. Cuesta (Ed.). ACM, New York, NY, USA, 285-293
33. Kelter, U., Wehren, J. & Niere, J. (2005). A Generic Difference Algorithm for UML Models. In P. Liggesmeyer, K. Pohl & M. Goedicke (eds.), Software Engineering (pp. 105-116)
34. JGraphT, http://jgrapht.org/, last accessed March 5th 2013.
35. Li, C., Ling, T. W.: OWL-Based Semantic Conflicts Detection and Resolution for Data Interoperability. ER (Workshops) 2004:266-277 (2004).
36. Mas, F., Menéndez, J.L., Oliva, M., Ríos, J.: Collaborative Engineering: an Airbus case study. Procedia Engineering, 2013, 63 (2013), pp.336-345.
37. Menéndez, J.L., Mas, F., Serván, J., Arista, R., Ríos, J.: Implementation of the iDMU for an aerostructure industrialization in AIRBUS. Procedia Engineering, 63 (2013), pp.327-335.
38. Schmidt, D. C. 2006. Model-Driven Engineering. Published by the IEEE Computer Society vol 39.
39. OMG. Documents Associated with Meta Object Facility (MOF) 2.0 Query, View, Transformation. Object Management Group. URL: http://www.omg.org/spec/QVT/1.0/. 2008.
40. Ponce, J., García-Borgoñon, L., García-García, J.A., Escalona, M.J., Domínguez-Mayo, F.J., Alba, M., Aragón, G.: A Model-Driven Approach for Business Process Management. Covenant Journal of Engineering & Technology (CJICT) Vol. 1, No. 2, pp. 32-52. 2013.
41. Mas, F., Ríos, J., Menéndez, J.L., Gómez. A.: A process-oriented approach to modeling the conceptual design of aircraft assembly lines, International Journal of Advanced Manufacturing Technology 2012, Vol. 62. 2012.
42. Escalona, M.J., Gutiérrez, J.J., Rodríguez-Catalán, L., Guevara, A.: Model-Driven in reverse. The practical experience of the AQUA Project. Euro American Conference on Telematics and Information Systems. pp. 90-95. Czech Republic. (2009).
43. Escalona, M.J., Aragón, G., Molina, A., Martínez-Force, E.: A Model-Driven tool framework for the improvement in the use of NDT. 8th International Conference on Software Quality Management. The British Computer Society. pp. 147-157. UK. (2010).
44. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Version 2.3. Department of Computer Science, University of Durham, Durham, UK. EBSE-2007-01. 2007.
45. Mas F., Oliva M., Ríos J., Gómez A., Olmos V, García-García J.A.: PLM Based Approach to the Industrialization of Aeronautical Assemblies. Procedia Engineering, MESIC Manufacturing Engineering Society International Conference 2015, Volume 132, pp. 1045–1052, Doi:10.1016/j.proeng.2015.12.594. 2015.
46. Liu, L., & Yu, E.: From requirements to architectural design-using goals and scenarios. In ICSE-2001 Workshop: From Software Requirements to Architectures (STRAW 2001) 2001
47. Weidenhaupt, K., Pohl, K., Jarke, M., & Haumer, P. : Scenarios in system development: current practice. IEEE software, 15(2), 34-45. 1998.