DIM

- A systematic and lightweight method for identifying dependencies between requirements

Arturo Gómez and Gema Rueda
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 40 person-weeks of full time studies.

Contact Information:
Author(s):
Arturo Gómez, Gema Rueda
E-mail: a.gomez.del.castillo@gmail.com, gema.rueda@gmail.com

University advisor(s):
Darja Šmite
School of Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25
1. **Introduction**

Requirements are related to and affect each other in complex manners [1][6][26]. In fact, Carlshamre et al. [6] reported that only about 20% of all requirements are relatively singular, making dependencies a highly relevant factor in requirement analysis. Therefore, efficient ways of managing dependencies are needed [4][16].

The management of requirements dependencies is central [7]. Nowadays the requirements dependencies are fairly unexplored [4] and further research is requested [7][17][25]. As a consequence of this fact, practitioners lack an effective, affordable and reliable method for identifying dependencies.

Requirements dependencies are not problematic per-se, but they influence a number of development activities and decisions made during the software engineering process, e.g. in release planning [6][17], change management [19][30], requirements design and implementation [27], testing [8], and requirements reuse [31]. Therefore the impact of dependencies on a software project can be tremendous [4].

Requirements dependencies constraint the sequence in which the requirements are implemented [5], increase the complexity of requirements selection for a certain release [4][20] and hinder the planning [5][6]. As the number of unhandled dependencies increases, the average revenue decreases as well as the possibility of delay increases [4][5]. Likewise, if not explicitly managed, it could drive to customer dissatisfaction and important system failures [12][27].

It is accepted that dependencies between requirements makes it necessary to have some implemented before others [4][6][15][21]. If the order of requirements implementation does not take into account these dependencies it may have a large number of preventable refactoring, increasing the total cost of the project needlessly [4]. In addition, if one requirement changes, other dependent requirements could be affected too. Consequently identifying beforehand the dependencies between requirements increases the ability to effectively deal with changes [4]. Therefore, the dependencies could be related to cost, value, changes, people, competence, technical precedence, etc [8][28].

This thesis proposes and evaluates a lightweight and systematic method for identifying dependencies between requirements, further referred to as DIM (Dependencies Identification Method). DIM provides support to decisions made during software development. It enables the effective handling of dependencies in the software requirements specification. DIM establishes a set of well-defined rules for identifying dependencies between requirements (see Part 1). These rules can be easily automated through tool support (see Part 2, Section IV). DIM acknowledges that practitioners are reluctant to introduce more analysis activities in the already tight project agenda [6]. Therefore, it does not add any additional load to the project since it is based on draft versions of architectural models frequently developed in most of the projects. Draft versions are encouraged in order not to force the development team to design complete and correct versions of the architectural models. This pretends to allow iterative and incremental life-cycles. DIM considers two perspectives:

1. **Data perspective**: Generally represented by the intersection between the Entity-Relationship diagram and the SRS.
2. **Service perspective**: Usually represented by the intersection between component or class diagram and the SRS.
Applying DIM also has lateral benefits such as the reinforcement of the architectural value, improvement of traceability and increase of response to change among others (See Part 1).

The structure for this thesis is as follows: The first part presents a research in which DIM is developed and explained in detail. This part has already been published at the 11th International Conference on Agile Software Development at Trondheim, Norway. The second part presents an experiment for evaluating DIM in comparison with ad-hoc methods based on pair-wise comparison. The paper from the second part will be submitted to 19th IEEE International Requirements Engineering Conference.
2. **Scope**

As can be seen in Figure 1 many project areas are impacted by dependencies between requirements. Due to resource constraints, this thesis will not cover all these related areas. Instead, it will be focused on the requirement dependencies identification area by defining and testing a dependencies identification method. Other lines for deeper analysis concerning the rest of the areas will be indicated as further works.

![Figure 1: Project areas influenced by dependencies between requirements.](image-url)
3. AIM AND OBJECTIVES

The major aim for this thesis was to develop a reliable, affordable and systematic method for identifying dependencies between requirements. Reliable, meaning it should detect most of the existing dependencies between requirements. Affordable, in the sense that it should not imply additional cost to the project and must have lateral benefits for the development. Finally, systematic, meaning that it can be applied following a well defined set of rules and therefore obtaining similar results regardless of the practitioner applying the method.

Based on the major aim detailed above, two independent sub-aims have been defined, one for each of the parts for this thesis.

Sub-aim 1: Contribute with the requirement dependencies area by designing a method that enables its identification in a reliable, affordable and systematic manner.

Sub-aim 2: Compare the number of dependencies and practitioners’ profile influence on the results for the Dependencies Identification Method (DIM) and methods based on pair-wise comparison (PWC).

The objectives to reach each sub-aim are described at Table 1.

<table>
<thead>
<tr>
<th>Sub-aim Id</th>
<th>Objective id</th>
<th>Objective description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-aim 1</td>
<td>O1.1</td>
<td>Analyze the main techniques used to identify and manage the dependencies among requirements.</td>
</tr>
<tr>
<td>Sub-aim 1</td>
<td>O1.2</td>
<td>Design a method to identify and manage dependencies among requirements based on the challenges, weaknesses and strengths from the existing techniques.</td>
</tr>
<tr>
<td>Sub-aim 1</td>
<td>O2.1</td>
<td>Select an open source project / SRS for the experiment.</td>
</tr>
<tr>
<td>Sub-aim 1</td>
<td>O2.2</td>
<td>Apply DIM and PWC in order to identify dependencies between the requirements.</td>
</tr>
<tr>
<td>Sub-aim 1</td>
<td>O2.3</td>
<td>Summarize results by subject’s profile.</td>
</tr>
<tr>
<td>Sub-aim 1</td>
<td>O2.4</td>
<td>Compare the average dependencies found by DIM and PWC.</td>
</tr>
<tr>
<td>Sub-aim 1</td>
<td>O2.5</td>
<td>Analyze if the profile has an influence on the results.</td>
</tr>
</tbody>
</table>

Table 1: Relation between thesis sub-aims and objectives.
4. **Research Questions and Hypothesis**

The research questions presented in this section were defined for measuring the degree of fulfillment for the objectives set for this thesis. The questions that are intended to be answered by this research are the following:

**RQ1.** Do existing techniques identify dependencies among requirements? How? What are their strengths and weaknesses?

**RQ2.** What are the characteristics that a requirement dependencies identification method should have?

**RQ3.** Is it possible to identify dependencies between requirements based on analyzing a draft version of the software architecture?

**RQ4.** Does DIM identify more dependencies than methods based on pair-wise comparison? How many more dependencies?

**RQ5.** To what extent does the practitioner’s profile affect the dependencies identified by DIM?

**RQ6.** Do methods based on PWC aide in the identification of dependencies between requirements related to different concepts?

Three hypotheses are defined for RQ4, RQ5 and RQ6. They are presented on Table 2.

<table>
<thead>
<tr>
<th>Sub-aim id</th>
<th>Research Question</th>
<th>Hypothesis id</th>
<th>Hypothesis description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-aim 1</td>
<td>RQ1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RQ2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RQ3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-aim 2</td>
<td>RQ4</td>
<td>H1</td>
<td>DIM identifies more dependencies than methods based on pair-wise comparison.</td>
</tr>
<tr>
<td></td>
<td>RQ5</td>
<td>H2</td>
<td>The subject’s profile has more influence on the dependencies identified when applying methods based on pair-wise comparison than when applying DIM.</td>
</tr>
<tr>
<td></td>
<td>RQ6</td>
<td>H3</td>
<td>Dependencies found by methods based on pair-wise comparison tend to identify dependencies among requirements related to the same concept.</td>
</tr>
</tbody>
</table>

Table 2: Relationships between sub-aims, research questions and hypothesis.
5. **Thesis Outcomes**

The outcomes obtained from this thesis are listed below:

**OUT1**: A systematic and lightweight method for identifying dependencies between requirements.

**OUT2**: Empirical evidence showing that DIM identifies more dependencies than methods based on pair-wise comparison.

**OUT3**: Empirical evidence showing that a practitioner’s profile does not affect the dependencies identified by DIM as much as the dependencies identified by a method based on pair-wise comparison.

The Table 3 below shows the relationships among the objectives, research questions and outcomes.

<table>
<thead>
<tr>
<th>Sub-aim id</th>
<th>Research Questions</th>
<th>Objectives</th>
<th>Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub-aim 1</td>
<td>RQ1</td>
<td>O1.1</td>
<td>OUT1</td>
</tr>
<tr>
<td></td>
<td>RQ2</td>
<td>O1.2</td>
<td>OUT1</td>
</tr>
<tr>
<td></td>
<td>RQ3</td>
<td>O2.2</td>
<td>OUT1</td>
</tr>
<tr>
<td>Sub-aim 2</td>
<td>RQ4</td>
<td>O2.1, O2.2, O2.3 O2.4</td>
<td>OUT2</td>
</tr>
<tr>
<td></td>
<td>RQ5</td>
<td>O2.1, O2.2, O2.3 O2.5</td>
<td>OUT3</td>
</tr>
<tr>
<td></td>
<td>RQ6</td>
<td>O2.1, O2.2, O2.3</td>
<td>OUT2</td>
</tr>
</tbody>
</table>

Table 3: Relationship among objectives, research questions and expected outcomes

After conducting the experiment, it has been found that both, OUT2 and OUT3 back up the hypotheses for this thesis. This provides evidence for considering DIM as a feasible alternative to methods based on PWC for identifying dependencies between requirements.
6. METHODOLOGY

Due to the difference in nature of each sub-aim defined, two different approaches will be followed in order to address them individually.

During the initial literature review, it was difficult to find peer reviewed material addressing the identification of dependencies between requirements. Therefore, for Sub-aim 1, the approach selected was exploratory research. A literature review was conducted and later the method for identifying dependencies was developed.

For Sub-aim 2, the approach followed was empirical research. The experiment designed aimed at comparing DIM with an ad-hoc method based on pair-wise comparison. It was intended to test DIM under conditions close to the ones found in real projects. Therefore, the dependencies were identified using a real Software Requirements Specification (SRS – see appendix A) extracted by reverse engineering from a medium-size open source project. Both methods identified dependencies from the same SRS. After that, the results were collected and analyzed as described in Part 2, Section IV. Figure 2 shows the detailed process followed for the experiment. This process and more details of the experiment are described in Part 2.

![Figure 2: Overview for the process followed during the empirical research.](image)

Figure 2 shows the life-cycle followed for this thesis in terms of activities, their inputs and outputs. Also, it presents how the research questions were answered by the outcomes obtained during each activity.

![Figure 3: Thesis process life-cycle.](image)
7. THREATS TO VALIDITY

7.1 Internal validity

Researcher bias is an important threat to internal validity. This was avoided by defining strict rules for summarizing the results and automating this process using Excel™ spreadsheets. Statistical methods to test the hypotheses were also used to reduce the impact of this threat.

Another threat identified for this experiment is the influence on the results of the assistant tools used to guide the subjects during the experiment. In order to mitigate this threat usability testing was performed by 3 experienced beta-testers (at least one usability course at Master’s level). Their feedback was taken into account and the applications were modified accordingly.

A threat to validity related to the experiment design is that DIM and PWC were not applied by the same number of subjects. More precisely, DIM was applied by 2 more subjects. The reason behind this threat was lack of resources and no palliative actions could be taken to avoid it.

7.2 Conclusion validity

The main threat to validity was that the set of real dependencies was not known for the SRS used in conducting this experiment. The identification of this reference set was not found feasible since the SRS used was drawn from a real project in order to test DIM under a context as close as possible to a real one. One alternative considered to deal with this threat was to extract the set of real dependencies by inspecting the code, but this was not reliable since not all the dependencies have their equivalence in code and the accuracy of the reference set would depend on the accuracy of the code analysis. Another alternative was based on applying a reliable, tested and systematic method to identify dependencies (different from the methods being tested) but none were found in the literature. Finally, an appraisal was done by analyzing (one by one) all the dependencies identified by both methods and determining whether each dependency was false or not based on the following procedure: The requirements associated by each dependency were read. Afterward, each member of the research team individually determined whether the dependency was false or not based on:

1. Personal knowledge on the open source project.
2. Personal experience developing CASE tools.

After each decision, if no consensus was reached among the research team, a brief discussion was held to determine if it was a false dependency.

Notice that this process for identifying false dependencies is time consuming and overwhelming and it is not recommended for further replications of this experiment.

7.3 External validity

The number of subjects applying each method is a threat for generalizing conclusions since the results would be more representative if a higher number of subjects would apply each identification method. Due to resource constraints, it was impossible to have more people applying each method.
8. GENERAL CONCLUSIONS AND FURTHER WORK

This thesis highlights the importance of the dependencies between requirements as a factor affecting many project areas. In addition to the business factors, these dependencies should be considered to establish the sequence in which requirements are implemented. The implementation cost is non commutative due to the existence of dependencies between requirements. If the non commutative cost is obviated, it could generate overrun in the development of a product. This overrun comes from unnecessary refactoring that could have been avoided with a different implementation order. In addition, it has been found that requirement dependencies should be taken into account in many other project areas: requirements prioritization, change management, coordination among distributed teams, development and testing.

Two definitions of dependencies have been provided: dependency on key and dependency on service. The use of high-level architectural diagrams has been found useful for supporting the identification of these two types of dependencies. Moreover, a lightweight and systematic method to identify dependencies between requirements has been proposed (DIM). If this method is applied at the beginning of the project, it helps to create a global perspective of the whole system for the entire team and it also encourages the development of initial architectural models for the software product. It is important to note that DIM is not based on pair-wise comparison. This is one of the main inconveniences for the existing methods to identify dependencies.

Based on the results obtained from the experiment, it can be concluded that DIM is a method for identifying dependencies that is ready to be tested in real software projects. It has been found to find more dependencies than traditional methods based on pair-wise comparison (PWC). Furthermore, it has been found that DIM requires fewer resources than PWC. Resources in terms of time and practitioner’s profile needed to identify the set of dependencies among the requirements. The difference in time increases as the number of requirements grows, since the complexity of DIM is not exponential like PWC methods. Also, the data evidence that applying DIM does not require practitioners with high expertise, since it has been found that the results are not affected by the subject’s profile.

From the experiment, it can be also concluded that DIM is a method easy to understand and follow. This has been concluded according to the subject’s survey in which 100% of the subjects applying DIM mentioned that. They describe DIM as an interesting and quick method. In addition, 50% of the subjects mentioned explicitly that they found useful having an architectural diagram for understanding the system. This understanding has been found useful for identifying dependencies between requirements.

Unexpectedly, it was found that DIM also helps to appraise the quality of the software architecture, since it provides a ground for the analysis and discussion of better solutions.

The requirements’ abstraction level must be equivalent to the architectural models’ abstraction level otherwise it was found that one requirement could impact most of the system generating useless information. More precisely, if very high level requirements are intersected with a detailed class diagram, probably many requirements could impact most of the classes, generating useless information. Therefore, high level abstraction requirements must be intersected with high abstraction level architectural models.
Additionally, it has been found that DIM requires mature Requirements Engineering processes since; at least, the requirements must be documented and maintained. Nevertheless, these terms do not refer to the traditional understanding of “documented” and “maintained” since agile approaches can be applied.

Once DIM is defined and compared with existing methods for identifying dependencies, several lines for further work are opened. It would be desirable to replicate the experiment conducted in this thesis in order to obtain stronger evidence, increase the statistical significance and be able to generalize the conclusions we have obtained. Important aspects like efficiency and effectiveness of DIM should be empirically tested. Comparing DIM with other requirement dependencies identification methods would be also desirable.

Other potential research line can also investigate the application of DIM to reduce the coordination efforts among distributed teams. This thesis defines a set of descriptive metrics for a requirement and a set of requirements. The applicability of these metrics on requirements prioritization and release planning can also be researched. In addition, it would be interesting to research how the response to change is improved when using the dependency graph or matrix obtained after applying DIM. In a more technical perspective, the development of a tool that fully supports DIM integration into the development process is encouraged.
REFERENCES

3. Åsa G. Dahlstedt; Anne Persson: Requirements Interdependencies-Moulding the State of Research into a Research Agenda.
25. Par Carlshamre.: Release planning in market-driven software product development Provoking an understanding
PART I:

DIM - A METHOD FOR IDENTIFYING DEPENDENCIES BETWEEN REQUIREMENTS
A Systematic and Lightweight Method to Identify Dependencies between User Stories

Arturo Gomez¹, Gema Rueda¹, and Pedro P. Alarcón²

¹ School of Computing
Blekinge Institute of Technology (BTH), Sweden
{argo09,geru09}@student.bth.se

² E.U. Informática
Technical University of Madrid (UPM), Madrid, Spain
pedrop.alarcon@eui.upm.es

Abstract. The order in which user stories are implemented can have a significant influence on the overall development cost. The total cost of developing a system is non-commutative because of dependencies between user stories. This paper presents a systematic and lightweight method to identify dependencies between user stories, aiding in the reduction of their impact on the overall project cost. Initial architecture models of the software product are suggested to identify dependencies. Using the method proposed does not add extra load to the project and reinforces the value of the architecture, facilitates the planning and improves the response to changes.

Keywords: User Stories Dependencies, Agile Development, Dependencies Identification Method, Non Commutative Implementation Cost.

1 Introduction

The elements that comprise the system under construction interact with each other, establishing dependencies among them [1]. In Figure 1, element A requires element B, generating a dependency between them. Such dependencies are naturally inherited by the user stories (US_i cannot be implemented until US_j is implemented). Therefore, the natural dependencies between User Stories (US from now on) should be accepted as inevitable. In fact, only a fifth of the requirements can be considered with no dependencies [2]. The existence of dependencies between USs makes necessary to have some implemented before others [2] [3] [1] [4]. If the order of user stories implementation does not take into account these dependencies it may have a large number of preventable refactoring, increasing the total cost of the project needlessly. Identifying beforehand the dependencies increases the ability to effectively deal with changes. Therefore light systematic mechanisms, as shown in this paper, are needed to help identify dependencies between USs.

The rest of the paper is structured as follows. The second section describes the problem of dependencies. The third section defines the concept of dependency...
between user stories. The fourth section describes the method to identify dependencies. The fifth section presents an example applying the method proposed. The sixth section presents related work. Finally, the conclusions are listed.

2 Problem Description

The existence of dependencies between USs hampers planning [5] [4]. Not considering them increases the chances of not complying with the release plans [6]. Therefore, the sequencing of USs is seen as a challenge [7]. Depending on the established implementation order of USs the number of refactoring may increase. For example, suppose that at time t, once the user story US$_i$ has been implemented, there is a database (DB) in production with the entity T$_1$ and primary key k$_1$. At time t+1, after implementing US$_j$, the data model shown in Figure 2 is obtained, in which the primary key of the entity T$_2$ is k$_2$. Given the cardinality, the primary key attributes from T$_2$ become part of the table generated for entity T$_1$. This will require a refactoring of the DB and all components that access T$_1$ and an update of all rows of table T$_1$. If US$_j$ had been implemented before US$_i$ there would be no need to refactor, so the refactoring cost would be zero. Hence, due to the existing dependencies, the total cost of developing a system depends on the order in which the USs are implemented. Therefore, the total cost of developing a system is non commutative. Generalizing, if US$_j$ depends on US$_i$ and being C the cost function of implementing a user story in a given time t, considering RC as the cost of carrying out a determined refactoring j, then: C(US$_j$)$_t$ + C(US$_i$)$_{t+1}$ = C(US$_i$)$_t$ + C(US$_j$)$_{t+1}$ + RC$_j$. Note that refactoring can become a complex process with a very high cost [8], which is directly proportional to the number of implemented user stories [9].
3 User Stories Dependency Concept

This section defines the concepts: Dependency on key (Definition 1) and dependency on service (Definition 2).

Definition 1. Considering an agile project P, and E as the data model of P. Given that US$_i$ and US$_j$ are user stories from P that respectively require data represented in the entities E$_i$ and E$_j$ belonging to E. If after E is transformed into the target model (usually relational model) the data structure generated for the entity E$_i$ adds the primary key attributes of the entity E$_j$, then US$_i$ has a dependency on key with US$_j$, and it is expressed as: US$_j$ → US$_i$. In Figure 3, the following dependencies on key are found: K={US$_2$ → US$_1$}.

![Fig. 3. Example of simplified conceptual data and component diagram](image)

Definition 2. Considering an agile project P which has been represented by a component diagram C. Given that US$_i$ and US$_j$ are user stories from P, which are implemented respectively in the components C$_i$ and C$_j$ included in C. The user history US$_i$ has a dependency on service with respect to US$_j$, if and only if C$_j$ implements at least one service used by US$_i$ in C$_i$, expressed as: US$_j$ → US$_i$. In Figure 3, the following dependencies on service are found: S={US$_2$ → US$_1$}.

Based on the above definitions, the complete set of dependencies is defined as: D={K ∪ S}. Note that D can vary because of changes in user stories.

4 User Stories Dependencies Identification Method

The dependencies cannot be clearly inferred from the definition of USs. Building an initial architecture (data and component models) helps to identify them. Both models are transversal to the USs, see architectural models boxes at Figure 4. The evaluation of the interaction of each user story with both models allows the identification of possible dependencies. The proposed method identifies USs dependencies. Its duration depends on the size of the project and the presence of the whole team is recommended during its application to gain a project overview. It is lightweight in the sense that it does not add load to the project, since the activities or products needed are carried out in initial stages. If the USs or models change, the identification method should be executed before starting the next iteration (see Figure 4).

To identify dependencies between USs: First, a quick study of user stories defined so far is suggested, generating a simplified data model (without attributes). The use of the entity-relationship model is recommended since it helps to generate an overall view of the system. It is usually generated in software projects and
therefore it does not add additional load. Notice that this diagram is not an objective in itself. Its purpose is to identify the elements from the data model that each user story requires to be implemented, writing its identifier next to the data element required. For example, brackets can be used as shown in Figure 3. Second, establish the set of dependencies on key from the diagram, according to Section 3. To do so, for example, the transformation rules from an entity-relationship model to relational model can be used. Thus, given two elements A and B of a model M, if element A migrates the primary key attributes to element B, then the user stories related to B will have dependency on key of the user stories related to A.

To identify dependencies on service it is proposed: First, use a simplified component model which will represent the list of user stories identified so far. This diagram will include the components identified as well as the service relationship between them. It has a high level of abstraction that allows to easily identify the dependencies on service. Its creation provides a global perspective of the system to the team, which is important for understanding the dependencies. As in the previous case, this diagram is not a goal in itself. It can be replaced by any other that allows identification of such dependencies. The USs involved in the implementation of each component should be written within brackets (see Figure 3). Second, identify the set of dependencies on service from the diagram, according to Section 3. Thus, given two elements A and B of a model M, if the element A implements a service required by B, then the user stories related to B will depend on the user stories related to A.

The mechanism to register dependencies is to record them using a directed graph like the one shown in Figure 5. Initially, all the USs are represented as disconnected vertices. As soon as US$_j$ → US$_i$ is identified, an edge pointing US$_i$ is drawn between vertices US$_j$ and US$_i$. This representation informs quickly about the dependencies among USs. Additionally, it helps to quickly identify dependency chains between USs. The graph generated can be used as basis to support planning or as an input for well known algorithms [1] [3] to generate an implementation sequence that reduces the impact of dependencies.
When interpreting the results, a vertex without incoming edges means that this user story has no dependencies. If a vertex (US$_i$) has incoming edges but these edges come from vertices representing USs already developed, it is also considered that US$_i$ has no dependencies. From the technical perspective, a user story without dependencies can be implemented at any time or assigned at any release and business value would be the main factor when prioritizing and planning it. When planning, the development team must be aware that if a user story (US$_i$) is developed and it depends on other USs not developed yet, there could be additional costs associated with refactoring and other technical risks. The customer should be warned with this information before prioritizing the user story. When a user story changes or a new one is introduced, the directed graph must be checked to identify the USs that depend on the changed or new user story. The architectural elements associated to these dependent user stories are more likely to be impacted by this change. Therefore, the set of architectural elements that are likely to change is reduced, facilitating the response to change.

5 Example of Use

This section focuses on a subset of USs extracted from a real project in which the authors of this paper participated. This project included the development of a software tool called Agile Management Tool (AMT). The subset of user stories selected from AMT project is: US$_1$ (Create User Stories); US$_2$ (Create Iterations); US$_3$ (Create Projects). Due to the paper’s size restrictions this section focuses only on the data model (see Figure 5). Following the identification method proposed, references to USs related to each model element have been included. Notice that when the simplified data model is transformed into relational tables, the primary key attributes from the entity Project (related to US$_3$) will migrate into the entity User Story (related to US$_1$), which implies that US$_1$ depends on US$_3$, therefore an edge from US$_3$ vertex pointing to US$_1$ vertex must be drawn. This way the team will continue identifying dependencies, generating at the end a graph like the one showed on Figure 5. Based on it, the dependency set is: \( D=\{US_3 \rightarrow US_1; \ US_3 \rightarrow US_2; \ US_2 \rightarrow US_1\} \). Then, from the technical point of view, since every user story depends on US$_3$, the recommendation to the customer would be implementing US$_3$ first. Otherwise, the cost of refactoring should be added to the cost of developing US$_1$, US$_2$ and US$_3$.

Fig. 5. Original scanned data model from selected US
6 Related Work

Some well known methods consider dependencies such as IFM [1] and Evolve [4][3]. Nevertheless none of them provide a systematic mechanism for identifying dependencies between user stories as the method proposed in this paper. In [2] is proposed a method to identify dependencies but it relies on pairwise assessment among the requirements. This is applicable for a small number of requirements but requires too much effort facing a large number of requirements. Mike Cohn states that if two user stories are dependent they must merge [5]. However, in practice it has been seen that large user stories that cannot be completed in one iteration, hinder the feeling of progress and therefore team motivation [7].

7 Conclusions

The implementation cost is non commutative due to the existence of dependencies between user stories. If this fact is obviated, it could generate overrun in the development of a product. This overrun comes from unnecessary refactoring that could have been avoided with a different implementation order. Two definitions of dependencies have been provided: dependency on key and dependency on service. This paper contributes with a very lightweight method that identifies dependencies between user stories, helping the planning and reducing the technical risks of the project, while reinforcing the architectural value as a lateral effect. Furthermore, if this method is applied at the beginning of the project, it helps to create a common perspective of the system. This method has been designed to fit in an agile environment, following the agile values and principles.

References

PART II:

EMPIRICAL EVALUATION OF DIM
Empirical Research Comparing DIM with Ad-hoc Methods

Arturo Gómez del Castillo¹, Gema Rueda Montenegro¹, and Darja Smite¹

¹Blekinge Institute of Technology, Ronenby, Sweden

Dependencies between requirements are a crucial factor for any software development since they impact many project areas. Nevertheless, their identification remains a challenge. Some methods have been proposed but none of them are really applicable to real projects due to their high cost or low accuracy. DIM is a lightweight method for identifying dependencies proposed on a previous paper. This paper presents an experiment comparing the sets of dependencies found by DIM and a method based on pair-wise comparison. The experiment was executed using a requirement specification for an open source project. These requirements were extracted by reverse engineering. Our results have provided evidence confirming that DIM finds more dependencies and its results (the dependencies identified) do not depend on the profile of the practitioner applying it. Another important result is that DIM requires fewer resources when applied, since it does not rely on pair-wise comparisons and it can be easily automated.

Index Terms—Requirement Interdependencies, Dependencies Identification Method, Reverse Engineering, Empirical Research.

I. INTRODUCTION

Requirements are related to and affect each other in complex manners [1][6][26]. Individual requirements are usually not independent, in relation with each other, since they describe a same system [9][13][32][33]. In fact, Carlshamre et al. [6] reported that only 20% of all requirements are relatively singular, making dependencies a highly relevant factor in requirement analysis. Dependencies between requirements should be accepted as inevitable [14].

The management of requirements dependencies is central [7]. Requirements dependencies are not problematic per se, but they influence a number of development activities and decisions made during the software engineering process, e.g. release planning [6][17], change management [19][30], requirements design and implementation [27], testing [8], and requirements reuse [31]. Therefore the impact of dependencies can be tremendous [4]. If they were not explicitly managed it could drive to customer dissatisfaction and important system failures [12][27].

Requirements dependencies are fairly unexplored [9] and further research is requested [7][17][25]. One of the main challenges is to develop approaches that enable to identify, describe and effectively deal with them in the software development process [4]. Therefore, efficient ways of identifying and managing dependencies are needed [4][16].

In order to face the challenge mentioned above, the authors have developed an approach for identifying dependencies between requirements, further referred to as DIM (Dependencies Identification Method) [14]. This paper presents an experiment for validating DIM with respect to ad-hoc methods based on pair-wise comparison.

The rest of the paper is structured as follows. The second section presents other works related to this research. The third section includes information about the applicability of DIM. On section four, the experiment for comparing DIM with a method based on pair-wise comparison is reported; section five summarizes the data collected from the experiment.

II. RELATED WORK

Requirement dependencies identification is a relatively unexplored field. In fact, it is considered as one of the major research issues in the requirements engineering field [3]. Practitioners still lack an effective and affordable method for the identification of dependencies between requirements [17]. Therefore, not many companies identify and document requirements dependencies explicitly [3][6]. In the cases where they are identified, it is based on expert opinion from knowledgeable personnel [3]. Two main method types are found in the literature: the methods based on similarity scanning and the methods based on pair-wise comparison together with expert opinion.

In the methods based on scanning for similarity, two requirements having similar slogans means that there could be a functional relationship and therefore a dependency between them. This type of methods are neither accurate nor reliable enough [6]. [23] presents how to use language tools to identify similarities between requirements. In spite of the low accuracy and reliability of this kind of methods, low cost can justify its use [6].

Other type of methods found in the literature are the ones based on pair-wise comparison [1][2][6][11][18][24]. These methods compare all the possible pairs of requirements and determine whether two requirements depend or not based on expert opinion. Pair-wise analysis also works as requirements inspection, identifying other problems within the SRS [3]. However, the cost associated to this kind of methods makes them unaffordable for most projects. The number of comparisons they require are n(n-1)/2, being n the number of requirements. Due to its exponential complexity, this type of methods is non-scalable for large amounts of requirements.

[6] presents alternatives for decreasing the time required to
apply this type of methods, but always relying on expert opinion.

In [6] a requirements dependencies identification method that relies on pair-wise comparison is proposed. It looks for similarities in order to determine whether a dependency between two requirements exists. This method has the inconvenience of methods based on similarity scanning and the high cost implied by methods based on pair-wise comparison.

In summary, none of the methods presented previously are really applicable for real projects, since scanning for similarity has proven to not be accurate enough [6] and pair-wise comparisons combined with expert opinion methods imply a cost that is not affordable by most of the projects and offer a solution based on subjective opinions.

Identifying dependencies is not always an easy task since requirements sometimes describe parts of the system that are independent. Nevertheless, as described in [14], it is possible to systematically identify the interactions between requirements apparently independent based on the analysis of the intersection between architectural models and SRS. This is the core idea behind the Dependencies Identification Method (DIM) [14].

The purpose of DIM is to systematically deal with requirements dependencies to improve decisions made during software development. It enables the identification and effective handling of dependencies in the software requirements specification. DIM establishes a set of well-defined rules for identifying dependencies between requirements [14]. These rules can be easily automated through tool support (see section IV.G). DIM is not based on pair-wise comparison and it has a linear complexity. It acknowledges that practitioners are reluctant to introduce more analysis activities in the already tight project agenda [6]. Therefore, it does not add any additional load to the project since it is based on architectural models frequently developed in most of the projects. Applying DIM has also some lateral benefits such as the reinforcement of the architectural value, improvement of traceability and increase of response to change among others [14]. In this paper the authors validate DIM in comparison with pair-wise comparison method for identifying the dependencies among requirements.

III. APPLICABILITY OF DIM

This section presents how the results obtained after applying DIM can be interpreted at different project areas.

As introduced in [14], two mechanisms can be used to record dependencies: A directed graph or a matrix.

If the directed graph was used, a vertex without incoming edges means that this user story has no dependencies. If a vertex Ri has incoming edges but these edges come from vertices representing requirements already developed, it is also considered that Ri has no dependencies. On the other hand, if the matrix was used, a row Ri containing zero in all its cells means that this requirement has no dependencies. If there is any value different than zero in the row Ri, but the corresponding column represents a requirement already developed, then it is also considered that Ri has no dependencies.

A. Related metrics

We have defined a set of metrics that can be used to describe a requirement and to assist the decision making.

1. Total Number of Providers (TNP)

The TNP of a requirement Ri is equal to the total number of requirements that provide some service or data to Ri. If a directed graph was used to record the dependencies, the TNP of a requirement Ri can be calculated as the number of incoming edges to the node representing Ri. Otherwise, if a matrix was used to record the dependencies, the TNP of a requirement Ri can be calculated applying the following equation:

\[ TNP(R_i) = \sum_{j=1}^{n} M(R_i, R_j) \]  

Where:

- \( n \): Total number of requirements.
- \( M \): Matrix used to record the set of dependencies.

The value of TNP for Ri will indicate the degree of dependency that Ri has with respect to the rest of requirements. The higher the value of TNP, the higher the complexity of its development.

2. Total Number of Consumers (TNC)

The TNC of a requirement Ri is equal to the total number of requirements that require some service or data provided by Ri. If a directed graph was used to record the dependencies, the TP of a requirement Ri can be calculated as the number of outgoing edges from the node representing Ri. Otherwise, if a matrix was used to record the dependencies, TNC of a requirement Ri can be calculated by applying the following equation:

\[ TNC(R_i) = \sum_{j=1}^{n} M(R_j, R_i) \]  

Where:

- \( n \): Total number of requirements.
- \( M \): Matrix used to record the set of dependencies.

The value of TNC for Ri will indicate the degree of impact that Ri has with respect to the rest of requirements. The higher is the value of TNC, the higher the impact on the rest of the system for not meeting this requirement. Therefore this metric can be taken into account when risk analysis is done. I.e. Considering the requirement P2.R3.4.1 from the SRS (Appendix B) used at the experiment described in section Y. TNC(P2.R3.4.1) = 28, that means that P2.R3.4 provides some service or data to 28 requirements. Therefore if the team fails developing these requirements, it would have some impact on
other requirements. The impacted requirements by P2.R3.4.1 are easily identified at the matrix generated when applying DIM.

TNC and TNP are descriptive metrics and can be used to analyze a requirement in terms of data and services consumed and provided.

### 3. Project Coupling Index (PCI)

PCI is defined as the total number of dependencies between requirements; its formal definition can be seen at Eq. 3 and Eq. 4. PCI can be considered as a quality attribute and as a metric for the complexity of the project. This complexity could be used to evaluate the project from a technical point of view.

\[
.PCI(P) = \sum_{i=1}^{n} TNP(R_i)
\]

or

\[
.PCI(P) = \sum_{i=1}^{n} TNC(R_i)
\]

Where:

- \(n\) : Total number of requirements.
- \(M\) : Matrix used to record the set of dependencies.

The number of dependencies affects the way in which a project is coordinated [38]. When a project has a high PCI value (high complexity), it requires higher costs related to coordination.

In the context of distributed projects, a project with high PCI could have costs associated to the coordination that can make it not profitable for distribution. Nevertheless, this is just a hypothesis and further research is needed to prove it.

### B. Prioritization and Release planning

Using business perspective as the only criteria for requirements prioritization might cause major problems [10]. For instance, not taking into account dependencies between requirements when planning increases the chances of not complying with the established release plans [35]. Considering both perspectives, business and technical, is not an easy task since occasionally they can be contradictory and their management complicated [15].

In the development of a project, many factors should be considered when planning, like business priorities and technical factors among others. The existence of dependencies is one of the technical factors that must be taken into account, since requirements dependencies influence requirements selection and release planning [29]. They constrain the sequence in which the requirements are implemented [5][10] [4] [15][21], increase the complexity of requirements selection for a certain release. [4][20] and hinder the planning [6][5].

There are some well known methods for release planning such as IFM [10] and EVOLVE [15]. They consider the existence of dependencies for generating release plans but do not provide a systematic mechanism to identify them.

### C. Traceability and Response to change

DIM is based on identifying dependencies from the intersection between SRS and architectural models. This reinforces enormously the traceability and therefore the response to change.

When a requirement changes or a new one is introduced, the matrix or directed graph can be checked to identify the requirements that depend on the changed or new requirement. The architectural elements associated to these dependent requirements are more likely to be impacted by this change. Therefore, the set of architectural elements that are likely to change is reduced, facilitating the response to change.

### IV. EXPERIMENTAL DESIGN

The template used for reporting this experiment has been obtained from “Reporting Guidelines for Controlled Experiments in Software Engineering” [37].

The experiment described in this section aims to compare the sets of dependencies found by DIM and by another method based on pair-wise comparison (PWC). Even though efficiency is one of its main disadvantages, the experiment is not aiding to measure it. Comparison will be focused on the number of dependencies found by both methods and the intersection between these sets (dependencies found by both methods). Therefore, the reason for choosing PWC is its high accuracy. On the other hand, Scanning for Similarity Method was not chosen because of its low accuracy.

In order to reach the aim, the following goals were defined:

- G1 Select an open source project / SRS for the experiment.
- G2 Apply DIM and PWC in order to identify dependencies between the requirements.
- G3 Summarize results by subject’s profile.
- G4 Compare the average dependencies found by DIM and PWC.
- G5 Analyze if the profile has an influence on the results.

The experiment presented involved participants from different countries (Spain, Poland, Dominican Republic and Sweden) during the summer of 2010.

### A. Hypotheses, Response Variables and Metrics

Considering the aim for this study, the following hypotheses were defined:

- H1 DIM finds more dependencies than methods based on pair-wise comparison.
- H2 The subject’s profile has more influence on the dependencies identified when applying methods based on pair-wise comparison than when applying DIM.
- H3 Dependencies found by methods based on pair-wise comparison tend to identify dependencies among requirements related to the same concept.
The response variable for H1 is the number of dependencies identified by each method. Therefore, the metrics collected for this response variable are:

H1.M1 Number of dependencies found.
H1.M2 Percentage of false dependencies.

The response variable for H2 is the Identification Coefficient (IC) by profile. This variable will be measured applying the formula presented at Eq. 24.

\[
IC(P_i, P_j) = \frac{\text{Dependencies identified by } P_i}{\text{Dependencies identified by } P_j}
\]

Where \( P_i \) and \( P_j \) are different profiles. Therefore, the metrics collected by this response variable are:

H2.M1 Number of dependencies found by profile.

The response variable for H3 is the percentage of dependencies that exists between requirements related to the same concept. Therefore, the metrics collected for this response variable are:

Being M the method applied for identifying the set of requirements dependencies,

H3.M1 Number of dependencies between two requirements related to the same concept by M.
H3.M2 Number of dependencies between two requirements related to different concepts by M.

Additionally, qualitatively expressed opinions/feedback provided by the subjects are also analyzed.

B. Factors and Levels

Consider a factor as any characteristic whose effect on the response variable is studied. According to this definition, two main factors have been identified: the subject’s profile and the method used for identifying dependencies, since both have an impact on the set of dependencies identified that is intended to ascertain. On the other hand, a level refers to possible values of the factors. These factors and levels are described in detail further below.

Factor 1 - Identification method. This factor refers to the set of steps or tasks defined for identifying dependencies between requirements. This experiment considers two possible methods:
- Dependencies Identification Method. This method identifies dependencies based on the analysis of between architecture and requirements specification. For more information see [14].
- Method based on Pair-Wise Comparison. This ad hoc method prescribes comparing all the possible pairs of requirements and determining if a dependency between two requirements exists based on expert opinion.

Factor 2 - Subject’s profile. This factor has been considered since one of the levels for the first factor is based on expert opinion. Therefore, the experience, domain knowledge and educational background have considerable effect on the results. The attributes considered for this factor are: Educational Background, Job position and Experience.

The information presented above is summarized on Table 1.

<table>
<thead>
<tr>
<th>Table 1: Factors and Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor</td>
</tr>
<tr>
<td>Method</td>
</tr>
<tr>
<td>Educational</td>
</tr>
<tr>
<td>Background</td>
</tr>
<tr>
<td>Profile</td>
</tr>
<tr>
<td>Job position</td>
</tr>
<tr>
<td>Experience (years)</td>
</tr>
</tbody>
</table>

C. Parameters

The variables that remain unchanged for all the subjects are:

Time restriction: There is no time restriction.
SRS: This parameter refers to the Software Requirements Specification used to apply both identification methods (see Factor 1). This SRS was extracted by reverse engineering from an open source project (see Section 3.6.1).

D. Subjects

The total number of subjects for this experiment was 14. The profiles of the subjects were very diverse. This variation in the subject’s profiles could affect the results. There were two options for avoiding this threat: Randomization or profile analysis. Randomization was discarded since the number of subjects was low. Thus, profile analysis was performed to block its influence on the results.

Four different subject profiles were identified. As mentioned earlier, a profile is defined by the subject’s educational background; the area of experience and the number of years working on this area (see Table 2). Since we had two methods for identifying dependencies, the subjects from each profile were divided in two equal groups. Later, each identification method was randomly assigned to a group.

None of the subjects were trained for any of the methods applied. Nevertheless, the subjects were provided with assistant tools (see section 4.5.2) that guided them in the tasks they were asked to perform. Therefore, the prior knowledge needed concerning each method was lower or null.
Table 2: Profiles identified from the experimental subjects.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Educational Background</th>
<th>Job area</th>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>Bachelor on Economics &amp; MPhil on Financial Economics</td>
<td>Economist</td>
<td>More than 3 years</td>
</tr>
<tr>
<td>P1</td>
<td>Master degree with major in Software Engineering</td>
<td>Developer at an IT company</td>
<td>Between 2 and 5 years</td>
</tr>
<tr>
<td>P2</td>
<td>PhD related to Software Engineering</td>
<td>Researcher in a University</td>
<td>More than 15 years of experience</td>
</tr>
<tr>
<td>P3</td>
<td>Bachelor on Software Engineering</td>
<td>Project Manager at an IT Company</td>
<td>More than 15 years</td>
</tr>
</tbody>
</table>

The number of subjects for each profile and method applied during the experiment can be seen at Table 3. Notice that DIM has two more subjects than PWC, this is because subject participation was voluntary and not all the subjects that were asked to participate in the experiment returned the results.

Table 3: Number of subjects by profile and method applied.

<table>
<thead>
<tr>
<th>Profile</th>
<th>DIM</th>
<th>PWC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

E. Procedure

All subjects were contacted by email. In this mail the experiment was introduced and the subjects were asked to confirm if they were willing to participate followed by instructions (see Appendix B) and an assistant tool for the method to be applied (see Section 4.5.2) was sent in case of approval. Notice that all subjects were asked to identify dependencies from the same software requirements specification. The deadline for the work was set to a 10-day period, during which the subjects were asked to apply the assigned method and return the results. Since the subject’s participation was voluntary, their work was assisted and guided whenever required. This was one of the reasons motivating the development of the methods assistant tools (see Section 4.5.2). Figure 1 illustrates the process described above and the outcomes of each step.

F. Data Collection Process

An identifier was assigned to the results when they were received from the subjects. This helped guarantee that they were treated anonymously. The dependencies found by each subject were assigned to their corresponding profile and then, the dependencies were automatically aggregated by Excel Workbooks (see Figure 2).

G. Instrumentation

The assistant tool developed helped in guiding the subjects in the tasks they were asked to perform. These applications also helped to collect dependencies identified, time metrics, subject’s profile and feedback.

These tools had to be as intuitive as possible; therefore usability testing was performed by 3 experienced beta-testers (at least one usability course at Master’s level). Their feedback was taken into account and the applications were modified accordingly.

1. Pair-Wise Assistant tool

This application supports the identification of dependencies by pair-wise comparison. It presents all possible pairs of requirements and the user determines, based on her/his opinion whether the requirements are dependent or not. The possible pairs do not consider the order in which the requirements are presented, meaning that the pair [Req A, Req B] is the same as [Req B, Req A]. This consideration reduces
the number of comparison to \( \frac{n(n-1)}{2} \), being \( n \) the number of requirements. In the case of this study the number of comparisons was 2016, since the SRS has 64 requirements.

The tool works as follows: At the beginning, the instructions window is presented (see Appendix B). After the user has read the instructions and click on “Start” button, the tool displays the main user interface window. After the work is done, the application requests the feedback from the subject.

2. **DIM Assistant tool**

This tool assists the application of DIM. It presents the requirements one by one. The subjects are asked to identify the concepts associated with each requirement. The tool also displays an Entity-Relationship diagram that can be used whenever the subjects need it.

The tool works as follows: At the beginning, the instructions window is presented (see Appendix B). After the user has read the instructions and click on “Start” button, the tool displays a window with 3 main areas: The first one presenting the requirement under analysis. The second one presents an Entity-Relationship diagram. The last one contains a list of all the possible concepts that might be related to the requirement.

3. **DIM Engine**

This application takes as input the architectural models codified in XML format; the Software Requirements Specification and the Intersection generated by DIM Assistant tool (see 4.5.2.2). It applies the identification rules defined in [14] for identifying dependencies between requirements and, finally, it produces an Excel file containing the identified dependencies. This application does not have any user interface and it was used only by the researchers conducting this study. It is intended to be integrated with DIM Assistant to develop a tool that provides full support to DIM.

6. **Open Source Project**

The open source project described in this section was used for extracting the requirements specification from code. This was performed by applying reverse engineering and considering different information sources as it is shown on Figure 3.

1. **Reverse Engineering artifacts description**

Reverse engineering was performed on the bases of the following artifacts:

**Code**: The open source project code was downloaded from the official code repository: http://sourceforge.net.

**SRS**: The Software Requirements Specification was developed following the processes described at 4.6.2. The SRS was recorded and managed using Excel™ and it was structured in four abstraction levels.

---

**Figure 3**: Diagram for the reverse engineering process.

2. **Reverse Engineering sub-processes description**

The reverse engineering process was composed by the next set of sub-processes:

**Code Inspection**: This sub-process consists of inspecting the code interfaces to extract the functionality supported by the system. It also includes the UML model analysis. To generate those models from the code, an eclipse plug-in called “GREEN” was used.

**Project Documentation**: This sub-process includes both JavaDoc documentation and manuals or descriptions found on the official project web site.

**User Interface Analysis**: This analysis was performed based on available screenshots and running the system. This sub-process is useful for identifying general functionality supported by the system.

**Domain Analysis**: This consisted on analyzing the system domain, exploring which functionality should have been provided by the tool and then checking if the analyzed project actually supports this main or general functionality. An Entity-Relationship diagram was generated to support this sub-process.

3. **Project Selection Criteria**

Different repositories have been consulted for selecting the proper open source project matching the goals for this experiment. A selection criteria was set to perform this task systematically and it was specified as follows [34]:

**Project Size**: Medium Projects.

Rationale: medium projects have realistic constraints and constitute a good ground for testing DIM.

**Programming Language**: Java.

Rationale: It is the programming language the team is most proficient with. This fact will help for reverse engineering. Furthermore, it is a popular programming language and therefore it is possible to find a wide range of open source projects.

**Application Domain**: CASE tools.

Rationale: In order to facilitate the process and increase the result accuracy, it is preferable to select applications within domains known by the team.

4. **Project Description**

The selected tool for the experiment was an open source project named Kunagi. It is a CASE tool and therefore, provides support to project management. It is specially
adapted for agile projects based on Scrum but also considers best practices from other methodologies.

The main functionality supported by this tool is: Requirements (User Stories) management, bugs and acceptance criteria management. As any other CASE tool, it supports sprint backlogs, releasing, risk and impediments management. Kunagi provides dashboards for controlling the state of the project at each moment. The following metrics illustrate the size of the project:

Number of classes: 1040
Lines of code: 61369
Method calls: 20179
Max. Cyclomatic complexity: 44
Avg. Cyclomatic complexity: 1.58

V. RESULTS

In this section the hypotheses defined at 4.A are tested based on the data collected during the experiment. An experimental package has been created for further replications of this experiment. This package contains the experiment instructions, SRS, assistant tools, excel templates used for aggregating the results and other data collected during this experiment. The package is accessible at [36].

The real set of dependencies was not a controlled variable, since it was not known beforehand. The SRS was extracted from a real project and therefore, the researchers were not able to introduce controlled dependencies. After the results were aggregated by profile, they were summarized by each method. The criterion for aggregating the results was to consider a dependency identified if it was identified by at least the 50% of the subjects. Once the results were obtained, an analysis looking for false dependencies identified by each method was performed.

A. Results Analysis

Concerning the number of dependencies identified by the method based on Pair-Wise Comparison (PWC), it has been found that it is much lower than the number of dependencies identified by DIM. More precisely, DIM found 899 dependencies and PWC found 118 dependencies. These sets of dependencies are distributed as follows: 842 dependencies were found only by DIM, 61 dependencies were found only by PWC and 57 dependencies were found by both methods.

The number of false dependencies identified by each method was similar. More precisely, 15.02% (135 dependencies) of the dependencies found using DIM were false positives and 15.25% (18 dependencies) of the dependencies found applying the method based on PWC were false positives. Based on this data, it can be said that DIM identifies more real dependencies than methods based on PWC.

The percentage of false dependencies was obtained by appraising (one by one) all the dependencies identified by both methods and determining whether each dependency was false or not based on the following procedure: The requirements involved in each dependency were read. Afterward, each member of the research team individually determined whether the dependency was false or not based on:

- Knowledge from the open source project.
- Experience developing CASE tools.

After each decision, if no consensus was reached among the research team a brief discussion was held to determine if it was a false dependency.

Additionally, even if it is not the purpose of this experiment it has been found that, according to our data, the average time needed to apply a method based on PWC was 89.66 minutes. Meanwhile, the average time needed to apply DIM was only 14.14 minutes. The comparison between both times can be seen at Figure 7.

![Figure 4: Comparison between the time needed to apply DIM and PWC.](image)

At this point, it is important to note that assistant tools (see section IV.G) automatically collect the time spent in each comparison or identification. These time metrics have been analyzed and outliers have been obviated since the assistant tools do not allow pausing during the experiment and it has been detected that subjects took breaks during the experiment (mainly the ones that applied the PWC method since it requires more time). Following the definition from outlier provided in [22], any time measure \( T \) was considered as outlier if it did not belong to the following interval:

\[
Q_1 - 1.5(Q_3 - Q_1) \leq T \leq Q_3 + 1.5(Q_3 - Q_1)
\]

Being:

\[ Q_1 \] \quad The first quartile.
\[ Q_3 \] \quad The third quartile.
\[ T \] \quad The time measure under analysis.

With respect to variability based on the subject’s profile, our results show that for all the variables analyzed, the method based on PWC has a higher CV (Coefficient of Variation) than DIM. In particular, the following observations are made:

- The number of dependencies identified by each profile applying the method based on PWC has a CV equal to 44.77% (see Figure 5) while DIM has a CV equal to 18.07% (see Figure 6).
It can be said that the time needed by each profile to apply either DIM or PWC remains constant since the CVs were 13.46% and 13.64% respectively.

Finally, our data indicates that 65.54% of the dependencies found applying PWC are dependencies whose requirements are related to the same concept therefore they could be considered as obvious dependencies. On the other hand, the dependencies found by DIM in which both requirements were related to the same concept represent 36.18% of the total. Note that 88.79% of the dependencies found by “Profile 1” applying PWC were dependencies whose requirements were related to the same concept.

VI. Discussion

This sub-section discusses the results presented on the previous section in relation to the hypotheses from Section 4.

A. Comparison between the set of dependencies identified by both methods.

After analyzing the data from the two populations, it can be assumed that both populations: subjects applying DIM (P1) and subjects applying PWC (P2), follows a normal distribution since the standardized Kurtosis coefficients were -0.839486 and 0.703613 respectively (both values belongs to the interval [-2,2]). In addition, F-Test was run in order to test if equal variances can be assumed. The hypotheses set for the F-Test were:

Null Hypothesis: $\sigma_{P1} = \sigma_{P2}$

Alternative Hypothesis: $\sigma_{P1} \neq \sigma_{P2}$

The p-value obtained after running the F-Test was 0.0540323. This means that Null hypothesis can be rejected and therefore we cannot assume equal variances at 96% of statistical confidence interval.

Regarding the comparison between the number of dependencies identified by DIM and by PWC, the T-Test was run considering the data provided above. In order to run the T-Test, hypothesis H1 was decomposed as follows:

Null hypothesis: H1.1 – DIM identifies less or equal number of dependencies than PWC.

Alternative hypothesis: H1.2 – DIM identifies a greater number of dependencies than PWC.

The p-value obtained was equal to 0.00000189445, therefore H1.1 can be rejected at a 99.99% of statistical confidence interval. In summary and based on the results obtained from the experiment, it can be said that DIM identifies more dependencies than PWC. More precisely, in this experiment DIM identified 4.5 times more dependencies than the method based on PWC.

The set of dependencies found by PWC and not by DIM was analyzed finding that 49 out of 61 (80.33%) were transitive dependencies. Considering that Req A has a transitive dependency with respect to Req B, if there is a Req C such that Req B -> Req C and Req C -> Req A. The reason why these dependencies are identified by PWC is because it compares all the possible pairs of requirements and therefore there is a higher probability of identifying them. Nevertheless, these dependencies are easily inferred from the graph or matrix generated after applying DIM.

In addition, the percentage of false dependencies identified by both methods is very similar, 15.02% when DIM was applied and 15.25% for PWC.

The reason behind the percentage of false positives found by DIM is because dependency identification is based on analyzing the intersection between draft versions of architectural models and requirements. In this experiment, this intersection was between an Entity-Relationship diagram and requirements. Following DIM [14], the ER diagram was a draft version (not normalized) in order not to force the development team to design complete and correct versions of the architectural models. This pretends to allow iterative and incremental life-cycles. Therefore, since the ER diagram was no normalized, there were relationships that would not exist in further versions of that diagram. These relationships imply false dependencies between some requirements (15.02% of the total number of dependencies). Nevertheless, DIM considers two perspectives: Data Perspective (usually the intersection between ER diagram and SRS) and Service Perspective (usually the intersection between component or class diagram and SRS). Because of resource constraints, only the Data Perspective was considered for this experiment.
However, when considering both Data and Service Perspectives, some of these false dependencies would be highlighted as not confirmed by both perspectives and the development team could quickly analyze and obviate them. In addition, these false dependencies have been found useful for evaluating and improving the software architecture, since they are originated from not normalized relationships between architectural elements. Therefore, they can be considered as a lateral benefit of DIM.

The cost of identifying false dependencies varies depending on whether DIM is applied considering both perspectives. If it is the case, this identification is done automatically and the cost could be considered residual. If only one perspective is applied, an appraisal of the set of dependencies identified by DIM should be done implying higher costs. For this experiment, it took 8 hours/person to appraise 899 dependencies. Notice that this appraisal could be parallelized or performed by teams. Nevertheless, running DIM considering Data and Service perspectives is recommended in order to increase identification accuracy and to reinforce the synergy between requirements engineering and architecture.

B. Results variation based on the subject’s profile

When the results were grouped by the profiles applying PWC, the standardized kurtosis coefficient did not belong to the interval [-2, 2], therefore it cannot be considered following a normal distribution. Considering this fact and the small sample size, statistical tests would not have the strength enough to reject or accept any hypothesis. Nevertheless, from the variance analysis performed in the previous section, it is highlighted that variances are not equal at a 96% of confidence. Moreover, the variance from the population applying PWC is greater than the variance from the population applying DIM. Meaning that when applying PWC the results tends to vary more depending on the subject than when DIM is applied. More precisely, we have found that Profile 2 (senior researchers) identified 3 times more dependencies than Profile 1 (developers). On the other hand, the coefficient of variation for the number of dependencies identified applying DIM was 18.07%, considering that this value includes results from Profile 0 (professionals not related to IT), meaning that, when DIM was applied the results remained stable, independent from the profile. These data is aligned with sub-hypotheses H2, nevertheless further replications of this experiment is needed in order to aggregate the results and increase their statistical significance. However, the impact of the subject’s profile on the results applying PWC was expected since it relies on subjective opinions based on experience. The results from DIM were also expected because it relies on the intersection between SRS and architectural models and the set of rules for identifying dependencies are fixed and can be automated.

C. Dependencies between requirements related to the same concept.

The results show that when PWC was applied, 77 dependencies (65.25%) were between requirements related to the same concept. Therefore, it provides evidence supporting H3. This figure can be explained because sometimes concepts are related in non-obvious manners and PWC methods do not explicitly use diagrams for supporting this identification.

D. Subject’s feedback

Subject’s feedback has been also taken into account for this analysis. For subjects applying DIM, 100% found it as a method easy to apply. They describe DIM as an interesting and quick method. 50% of the subjects mentioned explicitly that they found useful having an architectural diagram for understanding the system. The subjects that applied PWC described it as a long and tiring process (100% of subjects), around 20% said that it is a stressing process with too many comparisons. On the other hand, 20% of the subjects admitted that it helps inspect the requirements.

E. Additional findings

The requirements’ abstraction level must be equivalent to the abstraction level of the architectural models. Otherwise, it was found that one requirement could impact most of the system and could generate useless information. For example, if very high level requirements are intersected with a detailed class diagram, probably many requirements would impact most of the classes, generating useless information. Therefore, high level abstraction requirements must be intersected with high abstraction level architectural models, etc. A mapping can be done between the abstraction levels for requirements defined at RAM [39] and the models suggested following MDA [40] approach which guides the development from high abstraction level models to detailed ones. Nevertheless, this would require further studies and validations.

The use of DIM evolves together with the project. At the beginning, when very high level requirements are gathered and draft versions for the architecture are available, DIM can be used for supporting decisions such as team architecture/structure, project geographic distribution and other decisions made early in the projects. When more detailed requirements and high level architecture are available, results from DIM can be used as input for release planning and teams work coordination. Finally, when detailed requirements and architectural models are available, results from DIM aid in increasing traceability and response to change.

Additionally, it has been found that DIM requires mature Requirements Engineering processes since, at least, the requirements must be documented and maintained. Nevertheless, these terms do not refer to the traditional understanding of “documented” and “maintained” since agile approaches can be applied.

For this experiment, it was found that DIM required (in average) 15.8% of the total time required by the method based on PWC. It is important to highlight that any method based on pair-wise comparison has an exponential complexity, therefore the time needed increases exponentially when the number of requirements increases. Nevertheless, experiments testing this finding are needed.
VII. CONCLUSIONS AND FURTHER WORK

This paper presents an experiment that aimed at testing the set of dependencies identified by DIM (Dependency Identification Method) with the set of dependencies identified by PWC. DIM is a systematic and lightweight method for identifying dependencies between requirements. Detailed information about DIM can be found at [14]. The experiment presented in this paper is justified by the need of validating DIM.

The experiment has provided evidence supporting that DIM finds more dependencies than a method based on pair-wise comparison. In addition, it has been found that the results obtained applying DIM are not affected by the subject’s profile as much as methods based on pair-wise comparisons. Also, we have identified that methods based on pair-wise comparison tend to identify dependencies between requirements closely related or requirements with an impact on the same concept. The reason behind this fact could be that some relationships between concepts are difficult to identify and these kinds of methods do not rely on any model that enable its identification, for example entity-relationship diagrams.

The further work aims at developing an algorithm for release planning that uses the dependencies found by DIM as an input. In addition, we aim at developing a tool that fully supports DIM integration into the development process. Important aspects like efficiency and effectiveness of DIM should be empirically tested. Comparing DIM with other requirement dependencies identification methods would be also desirable. A mapping between RAM abstraction levels and architectural models’ abstraction levels would be desirable to select the right architectural abstraction level based on the available set of requirements. A potential research perspective can also investigate the application of DIM to reduce the coordination efforts among distributed teams. The authors also encourage replication of the reported experiment in order to obtain stronger evidence, increase the statistical significance and be able to generalize the conclusions we have obtained. This paper defines a set of descriptive metrics for a requirement and for a set of requirements. The applicability of these metrics on requirements prioritization and release planning can also be researched. In addition, it would be interesting to research how the response to change is improved when using the dependency graph or matrix obtained after applying DIM.

REFERENCES


APPENDIX A

This appendix presents the Software Requirements Specification for the open source project used in the experiment (Kunagi case tool). It is composed by 64 requirements and they are arranged into four different abstraction levels (see Table 1). Nevertheless, the difference in abstraction level was considered before starting the experiment and the abstraction level for each requirement was homogenized.

Requirements belonging to the same abstraction level are shaded in the same color.
<table>
<thead>
<tr>
<th>Id</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2.R1</td>
<td>The tool should be able to support Project Management (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R1.1</td>
<td>The tool should support Impediments Management (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R1.1.1</td>
<td>The users shall be able to create/edit/delete impediments to their work</td>
<td>An impediment is composed by: Label, Date, Description, Solution</td>
</tr>
<tr>
<td>P2.R1.1.2</td>
<td>The user shall be able to see the change history from an impediment</td>
<td></td>
</tr>
<tr>
<td>P2.R1.1.3</td>
<td>The users shall be able to export in PDF format the list of impediments</td>
<td></td>
</tr>
<tr>
<td>P2.R1.1.4</td>
<td>The users shall be able to post comments on impediments</td>
<td></td>
</tr>
<tr>
<td>P2.R1.2</td>
<td>The tool should support Risk Management (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R1.2.1</td>
<td>The users shall be able to create/edit/delete risks to their work</td>
<td>The risks are composed by: Label, Description, Impact (Extreme, Substantial, Medium, Minor, Negligible), Probability (Very Likely, Likely, Possible, Unlikely, Very Unlikely) and a calculated impact</td>
</tr>
<tr>
<td>P2.R1.2.2</td>
<td>The user shall be able to see the change history from a risk</td>
<td></td>
</tr>
<tr>
<td>P2.R1.2.3</td>
<td>The users shall be able to export in PDF format the list of risks</td>
<td></td>
</tr>
<tr>
<td>P2.R1.2.4</td>
<td>The users shall be able to post comments on risks</td>
<td></td>
</tr>
<tr>
<td>P2.R1.3</td>
<td>The tool should record a Project activity log (Journal)</td>
<td>This log shows what has been done by every user. It will contain information on which user stories have been developed by each user, the comments that each user post, the entries post in their blogs, ...</td>
</tr>
<tr>
<td>P2.R1.4</td>
<td>The user should be able to link/associate user stories to the next spring</td>
<td></td>
</tr>
<tr>
<td>P2.R1.5</td>
<td>The system shall be able to show information and user stories from previous springs</td>
<td></td>
</tr>
<tr>
<td>P2.R1.6</td>
<td>The system shall be able to support multiple projects at the same time.</td>
<td>Each project must have their own information and they should get mixed. Single sing on. The users only sings on one time per session</td>
</tr>
<tr>
<td>P2.R1.7</td>
<td>The user shall be able to change the working project without log out from the application.</td>
<td></td>
</tr>
<tr>
<td>P2.R2</td>
<td>The tool should be able to support Sprint Management (+)</td>
<td>Sprints are composed by user stories</td>
</tr>
<tr>
<td>P2.R2.1</td>
<td>The tool shall support task tracking system (+)</td>
<td>The task could be in one of the following states: Free Tasks, Claimed Tasks, Compeleted Tasks</td>
</tr>
<tr>
<td>P2.R2.1.1</td>
<td>The user shall be able to change the state of any user story</td>
<td></td>
</tr>
<tr>
<td>P2.R2.2</td>
<td>The user shall be able to see the stories at the current sprint and other general information for the current sprint.</td>
<td></td>
</tr>
<tr>
<td>P2.R2.3</td>
<td>Export Spring Backlog as PDF</td>
<td></td>
</tr>
<tr>
<td>P2.R3</td>
<td>The tool should be able to support Product Management (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R3.1</td>
<td>Product Backlog (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R3.1.1</td>
<td>The user shall be able to create/edit/delete user stories</td>
<td></td>
</tr>
<tr>
<td>P2.R3.1.2</td>
<td>The user shall be able to estimate the SPRINT VELOCITY in terms of story</td>
<td></td>
</tr>
<tr>
<td>P2.R3.1.3</td>
<td>The system should suggest a Sprint planning based on user story priority, sprint velocity</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>P2.R3.1.4</td>
<td>The user shall be able to request planning poker game to estimate a user story</td>
<td></td>
</tr>
<tr>
<td>P2.R3.1.5</td>
<td>The user shall be able to select a poker card with the story points for the user story under estimation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Once the poker card is selected the system updates the estimation for the user story</td>
<td></td>
</tr>
<tr>
<td>P2.R3.2</td>
<td>Acceptance Criteria (Quality) (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R3.2.1</td>
<td>The users shall be able to create/edit/delete acceptance criteria</td>
<td></td>
</tr>
<tr>
<td>P2.R3.2.2</td>
<td>The system shall be able to list all the acceptance criteria introduced by the users</td>
<td></td>
</tr>
<tr>
<td>P2.R3.2.3</td>
<td>The users shall be able to post comments on each acceptance criteria</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The acceptance criteria is composed by a label, a description and a test (it is plain text)</td>
<td></td>
</tr>
<tr>
<td>P2.R3.3</td>
<td>Bug Management (Issues) (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R3.3.1</td>
<td>The users shall be able to create/edit/delete bugs</td>
<td></td>
</tr>
<tr>
<td>P2.R3.3.2</td>
<td>The system shall be able to list all the bugs introduced by the users</td>
<td></td>
</tr>
<tr>
<td>P2.R3.3.3</td>
<td>The users shall be able to post comments on each bug</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The bug is composed by a Label, Severity (Critical, Severe, Normal, Minor), Description, Issuer, Statement, Affected Releases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The bugs will be arranged by the bugs state (Closed, Open)</td>
<td></td>
</tr>
<tr>
<td>P2.R3.4</td>
<td>Release Management (Releases) (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R3.4.1</td>
<td>The users shall be able to create/edit/delete releases</td>
<td></td>
</tr>
<tr>
<td>P2.R3.4.2</td>
<td>The system shall be able to list all the releases introduced by the users</td>
<td></td>
</tr>
<tr>
<td>P2.R3.4.3</td>
<td>The users shall be able to post comments on each release</td>
<td></td>
</tr>
<tr>
<td>P2.R3.4.4</td>
<td>The users shall be able to see the change history for each release</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The releases is composed by a Label, Release date, Description, Release notes, Set of sprints, Affected bugs, fixed bugs</td>
<td></td>
</tr>
<tr>
<td>P2.R4</td>
<td>Each project should have its own WIKI (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R4.1</td>
<td>Any user should be able to create the project wiki page</td>
<td></td>
</tr>
<tr>
<td>P2.R4.2</td>
<td>Any user should be able to edit the wiki page</td>
<td></td>
</tr>
<tr>
<td>P2.R4.3</td>
<td>Any user should be able to post comments to the wiki page</td>
<td></td>
</tr>
<tr>
<td>P2.R4.4</td>
<td>It should be possible to upload files to the wiki</td>
<td></td>
</tr>
<tr>
<td>P2.R4.5</td>
<td>The wiki page editor should be able to support text formatting</td>
<td></td>
</tr>
<tr>
<td>P2.R4.6</td>
<td>The user must be able to insert Hyperlinks when editing the wiki page</td>
<td></td>
</tr>
<tr>
<td>P2.R4.7</td>
<td>The user must be able to format data in Tables when editing the wiki page</td>
<td></td>
</tr>
<tr>
<td>P2.R4.8</td>
<td>The user must be able to insert images when editing the wiki page</td>
<td></td>
</tr>
<tr>
<td>P2.R4.9</td>
<td>The user must be able to check the change history for the wiki page</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This comments should be visible at the same interface from the wiki project</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Including: Bold, Italic, Bullets, Numbered list, text like code</td>
<td></td>
</tr>
<tr>
<td>P2.R5</td>
<td>Each project should have its own FORUM (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R5.1</td>
<td>Users could read published items in the forum</td>
<td></td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>P2.R5.2</td>
<td>Users could post comments on any item in the forum</td>
<td></td>
</tr>
<tr>
<td>P2.R5.3</td>
<td>Any user could create/edit/delete items in the forum</td>
<td></td>
</tr>
<tr>
<td>P2.R6</td>
<td>The tool should provide a CALENDAR (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R6.1</td>
<td>Any user should be able to add/edit/delete events to the calendar</td>
<td></td>
</tr>
<tr>
<td>P2.R6.2</td>
<td>The tool should list all the events added to the calendar</td>
<td></td>
</tr>
<tr>
<td>P2.R6.3</td>
<td>The users shall be able to post comments on events</td>
<td></td>
</tr>
<tr>
<td>P2.R7</td>
<td>A FILE REPOSITORY is needed to manage files from each project (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R7.1</td>
<td>The users will be able to upload files to a common repository</td>
<td></td>
</tr>
<tr>
<td>P2.R8</td>
<td>The users should have their own BLOGS (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R8.1</td>
<td>The user should be able to create/edit/delete entries to her/his blog</td>
<td></td>
</tr>
<tr>
<td>P2.R8.2</td>
<td>The user can publish and unpublish entries from his/her blog</td>
<td></td>
</tr>
<tr>
<td>P2.R8.3</td>
<td>Users could post comments his/her entries</td>
<td></td>
</tr>
<tr>
<td>P2.R9</td>
<td>Each user should be able to set him/her personal PREFERENCES (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R9.1</td>
<td>Define the color scheme for the user interface for the current Project</td>
<td></td>
</tr>
<tr>
<td>P2.R9.2</td>
<td>Define the default color scheme for the user interface to be applied at any project</td>
<td></td>
</tr>
<tr>
<td>P2.R9.3</td>
<td>Change user name, password and/or email address</td>
<td></td>
</tr>
<tr>
<td>P2.R10</td>
<td>DASHBOARD or Project Indicators (+)</td>
<td></td>
</tr>
<tr>
<td>P2.R10.1</td>
<td>The tool should present a Sprint Burndown chart</td>
<td></td>
</tr>
<tr>
<td>P2.R10.2</td>
<td>The dashboard should present the team's work</td>
<td></td>
</tr>
<tr>
<td>P2.R10.3</td>
<td>The dashboard should present the open Impediments</td>
<td></td>
</tr>
<tr>
<td>P2.R10.4</td>
<td>The dashboard should present the Risks with highest priority</td>
<td></td>
</tr>
<tr>
<td>P2.R10.5</td>
<td>The dashboard should present the latest Events</td>
<td></td>
</tr>
</tbody>
</table>

The initial state for new entries is unpublished. Published entries would appear automatically as an item in the Forum (see req P2.R5.1).
APPENDIX B

This appendix contains the documents and screenshots from the assistant tools used for conducting the experiment. All this documentation is available at [36].
B.1 E-Mail sent to the subjects applying PWC-EO method.

Hi [Subject name],

We are conducting an experiment for our Master thesis. The goal is to compare the effectiveness of our method for identifying dependencies against the ad hoc approaches and we need your help. All you would have to do is to identify the dependencies in a set of requirements. In order to assist you during this task we have developed a tool that presents all the possible pairs of requirements (one by one) and you just have to determine whether they depend or not.

Realize that your contribution is very important since we are researching on a field quite unexplored (requirement dependencies identification). And this research aims at contributing to this field and therefore to the overall Software Engineering.

Instructions for participating:
1) Unzip the attachment (PairWiseComparisonAssistant.zip)
2) Run the file /runMe.jar (located inside the zip file)
3) Read the instructions
4) Identify dependencies
5) Zip the folder generated in the first step
6) Send the zip file back to us

The time estimated to complete the experiment is around 90 minutes. If we can count you in please send an email confirming your participation as soon as possible and then, when you have identified the dependencies, send the results before Monday, August 9th.

If you are not able to participate in this experiment, we will appreciate if you could tell us as soon as possible.

We hope to get news from you soon.

Thanks in advanced and best regards,

Gema & Arturo.
Hi [Subject name],

We are conducting an experiment for our Master thesis. The goal is to compare the effectiveness of our method for identifying dependencies against the ad hoc approaches and we need your help. All you would have to do is to identify the dependencies in a set of requirements. In order to assist you during this task we have developed a tool that supports the intersection between architectural models and requirements. You just have to determine which requirements are related to which architectural elements.

Realize that your contribution is very important since we are researching on a field quite unexplored (requirement dependencies identification). And this research aims at contributing to this field and therefore to the overall Software Engineering.

Instructions for participating:

1) Unzip the attachment (DIMAssistant.zip)
2) Run the file /runMe.jar (located inside the zip file)
3) Read the instructions
4) Identify dependencies
5) Zip the folder generated in the first step
6) Send the zip file back to us

The time estimated to complete the experiment is around 15 minutes. If we can count you in please send an email confirming your participation as soon as possible and then, when you have identified the dependencies, send the results before Monday, August 9th.

If you are not able to participate in this experiment, we will appreciate if you could tell us as soon as possible.

We hope to get news from you soon.

Thanks in advanced and best regards,

Gema & Arturo.
B.3 Instructions sent to the subjects.

B.3.1 Instructions for applying PWC-EO method.

First of all thanks for participating in this study. Your contribution is unvaluable for this research.

BEFORE STARTING:
You are going to identify dependencies between requirements from a tool that provides support for project management (Requirement specification, iteration management, bugs management, etc).

IMPORTANT: Please, notice once you start identifying you can not stop until finishing, otherwise the information will be lost.

This application will present pairs of requirements. Your job is to determine (based on your opinion) if there is a dependency between each pair of requirements.

TYPES OF DEPENDENCIES
There are many types of dependencies. But, for this study we consider that "requirement A" depends on "requirement B", if "requirement A" requests service or data provided by "requirement B".

Therefore, for each pair of requirements (Requirement A and Requirement B), select one of the following four options:

- B depends on A, "Requirement A" provides services or data to "Requirement B".
- A depends on B, "Requirement B" provides services or data to "Requirement A".
- A depends on B AND B depends on A, "Requirement A" provides services or data to "Requirement B" and viceversa.
- NONE, There is no dependency between "Requirement A" and "Requirement B".

AFTER FINISHING:
The application will automatically save the results. After identifying dependencies, please zip the application folder and send it back to us.

Thanks again for your time, your collaboration is very important for this research.
B.3.2 Instructions for applying DIM.

First of all thanks for participating in this study. Your contribution is invaluable for this research.

BEFORE STARTING:
You are one of the first people in the world to test DIM (Dependencies Identification Method). You are going to work with requirements from a CASE tool. It provides support for project management (Requirement specification, iteration management, bugs management, etc).

IMPORTANT:
Please, notice once you start you can not stop until finishing (it will take between 10 or 15 mins), otherwise the information will be lost. If you want to take a rest, just leave this application running.

YOUR JOB:
Your job is pretty simple. Requirements will be presented to you (one by one) and you just have to identify the entities related or impacted by each requirement. A list of all the possible entities (concepts) will appear below the requirement. The Entity Relationship diagram will appear at the right side of the requirement, just in case you want to use it.

REMEMBER:
- The results are treated anonymously. The results from all the subjects will be automatically summarized.
- You are participating in a research. We expect to have enough data to draw conclusions, if you want we could send the final results with the conclusions to you.

AFTER FINISHING:
The application will automatically save the results. After finishing, please zip the application folder and send it back to us.

Thanks again for your time, your collaboration is very important for this research.
B.4 Screenshots for the assistant tools used to guide the subjects.

This section presents screenshots for the assistant tools used to guide the subjects. Figure 1 corresponds to the main interface for the Pair-Wise assistant tool. Figure 2 corresponds to the main interface for DIM assistant tool.

Figure 1: User interface for Pair-Wise assistant tool.

Figure 2: User interface for DIM assistant tool.