Software Qual J (2018) 26:585–605
DOI 10.1007/s11219-017-9362-x

Stability prediction of the software requirements specification

José del Sagrado[1] · **Isabel M. del Águila**[1]

Published online: 8 April 2017
© Springer Science+Business Media New York 2017

Abstract Complex decision-making is a prominent aspect of Requirements Engineering. This work presents the Bayesian network *Requisites* that predicts whether the requirements specification documents have to be revised. We test *Requisites'* suitability by means of metrics obtained from a large complex software project. Furthermore, this Bayesian network has been integrated into a software tool by defining a communication interface inside a multilayered architecture. In this way, we add a new decision-making functionality that provides requirements engineers with a feature to explore software requirement specification by combining requirement metrics and the probability values estimated by the Bayesian network.

Keywords Requirements engineering · Software requirements specification · CASE tools · Bayesian network

1 Introduction

Since the appearance of the first intelligent editors 20 years ago, the challenge has been to support software development using artificial intelligence (AI) techniques. In spite of the success of some occasional results and certain progress made over these years, the intelligent environment for software development is still very much under construction. This might be because software engineers are typically focused on prosaic, practical engineering concerns rather than on building smart algorithms (Harman 2012).

✉ Isabel M. del Águila
imaguila@ual.es

José del Sagrado
jsagrado@ual.es

[1] Department of Informatics, University of Almería, Ctra. Sacramento s/n, 04120, Almería, Spain

Expert knowledge is fundamental to every software development project since developers must make expertise-based decisions throughout all the development stages, from requirements to maintenance (Meziane and Vadera 2010). Consequently, software engineering (SE) can be considered a knowledge-intensive process and can therefore be framed within the AI domain (Harman 2012). In fact, the SE community has used many algorithms, methods, and techniques that have emerged from AI (Shirabad and Menzies 2005; Zhang et al. 2012). Furthermore, if a portion of expert knowledge could be modeled and then incorporated into the SE life-cycle (as well as into the tools that support it), then any development process would be greatly benefited.

Our work presents a BN model called *Requisites* (del Sagrado and del Águila 2010), which allows assessment of the software requirements specification (SRS) and demonstrates how this BN model has been embedded into a previously constructed software tool. We also provide some evidence of *Requisites* validity by testing the suitability of BN using data measured according to a dataset of a specific software development project. The model was designed to be used as a predictor to tell us whether the software requirements specification (SRS) possesses sufficient quality to be considered as a baseline. Once several measurements are computed or obtained, their values will be introduced as evidence. Then, belief propagation will be used to determine if the process should stop because the requirements specification has sufficient stability. In addition, we present a successful solution that instantiates an architecture for the seamless integration of a computer-aided requirements engineering (CARE) tool so one is able to manage requirements with certain AI techniques (del Sagrado et al. 2011). In particular, we present Bayesian network integration in an academic CARE tool, InSCo-Requisite (Orellana et al. 2008). This tool has been extended with a new feature that adapts the requirements metrics to the variables used in the Bayesian network, allowing the application of this AI technique in the requirements specification stage.

The rest of the paper is structured as follows: after describing the basic requirements workflow, Section 2 describes the related works dealing with the benefits of using AI in software development. Section 3 includes a description of the Bayesian network *Requisites* and how it has been tested using a real world large-scale dataset. Section 4 is devoted to defining the process followed to integrate BN *Requisites* inside the InSCo-Requisite tool; this is done to define a software project baseline. The section also gives some examples of its use on a specific software development project. The threats that could affect the validity of this research are discussed in the Section 5. Finally, Section 6 includes the conclusions.

1.1 Background: requirements workflow

Requirements express the needs and constraints established for a software product that contribute to the solution of some real world problem (Kotonya and Sommerville 1998). Requirements development is considered a good domain for applying AI techniques because requirements are imprecise, incomplete, and ambiguous by nature (Meziane and Vadera 2010; Harman 2012). This area of SE is quite different from others because requirements belong to problem space whereas other artefacts, which are also managed in a software project, reside in the solution space (Cheng and Atlee 2007). If requirement-related tasks are poorly executed, usually the software product obtained becomes unsatisfactory from a software factory point of view (Sommerville 2011; Standish Group 2008).

Requirements are critical to the success of a software project as they collect the needs or conditions that have to be met by the product; that is to say, they are the basis for the rest of the development process, even more so when agile methods are applied. Changes in requirements are always welcome in these methods. Their highest priority is to satisfy

the customer through early and continuous delivery of valuable software. Therefore, any improvements in the requirements development stage will favorably affect the whole production life-cycle.

Work related to requirements is articulated by the execution of several activities and has been divided into processes with small variations by various authors (Kotonya and Sommerville 1998; Abran et al. 2004; Wiegers and Beatty 2013). The simplest set of activities for creating requirements (Alexander and Beus-Dukic 2009) comprises discovery, documentation, and validation. The discovering-documenting-validating cycle (DDV) is carried out over several iterations in order to complete the requirements specification and then it moves toward the next development task. When an agile method is applied, these activities have to be taken into account at the beginning of each iteration or sprint.

Discovering requirements is the task which determines, through communication with customers and users, what their requirements actually are. Requirements are elicited (or gathered) by interviews and other techniques such as stakeholder workshops or inspections. This is where the problem has to be understood, and which software is then going to solve it. It is usually a complex task because it requires efficient communication between software users and software engineers. Requirements have to be conceived without ambiguities so as to define what the system is expected to do.

Documenting requirements is about capturing software requirements. These requirements are captured in a document or its electronic equivalent, known as a software requirements specification (SRS). Software requirement documents play a crucial role in SE (Nicolás and Toval 2009). Early approaches developed to perform this activity worked with word processors but such a method for supporting SRS was error prone and tedious. CARE tools offered a solution to these problems, providing environments that made use of databases, allowing effective requirements management of any software project over its entire life-cycle. These tools also allow the use of modeling languages (e.g., use cases, UML) or informal languages (storyboards) for describing requirements.

Validating requirements checks if requirements present any inconsistencies, ambiguities, or errors. It is concerned with the process of analyzing requirements to detect or resolve conflicts, and to properly define the limits of the software system.

2 Related works

Existing works have already demonstrated that there is considerable potential for software engineers to take advantage of AI. The algorithms, methods, and techniques that emerged from the AI community then merged with the SE community. They can be arranged into three main areas: 'search-based software engineering'(SBSE), 'classification, learning, and prediction for software engineering,' and 'probabilistic software engineering' (Harman 2012). SBSE reformulates SE problems as optimization problems; this has proven a widely applicable and successful approach from the requirements to test stages (Harman et al. 2012). In classification, learning and prediction, some authors propose models for the prediction of risky issues in SE (Menzies and Shepperd 2012), either related to the study of defects (Kastro and Bener 2008) or the process of predicting the effort required to develop a software system (Wen et al. 2012). A probabilistic AI technique that is highly applicable in SE is Bayesian probabilistic reasoning. This models different software topics (Misirli and

Bener 2014) such as quality management (Tosun et al. 2015) or defect prediction (Mısırlı et al. 2011). The boundary is blurred between the three main areas so some of these works can be included in more than one.

When focusing on the requirements stage, Requirements Engineering (RE) is the least covered by the AI approaches. SBSE focuses on requirements in only 3% of the works, (Harman et al. 2012; de Freitas and de Souza 2011), and few papers deal with how to apply Bayesian networks to requirements (del Águila and del Sagrado 2015). AI can provide a new dimension to the requirements development stage by defining methods and tools that will assist the engineer in enhancing the execution of the entire software development project.

The discovering requirements task can be assisted by machine learning techniques to organize stakeholder collaboration. These use clustering techniques to manage discussion forums on requirements (Castro-Herrera et al. 2009) or automatic clustering of product features for a given domain (Dumitru et al. 2011). Requirements are the bricks that build the different stages in software project development. Consequently, if we have a risky requirements process, we will probably have a risky project. In order to mitigate the risks, we need to identify and assess what the requirements risks are. Bayesian network classifiers can also assist the process of predicting the risk level of a given requirement (del Águila and del Sagrado 2011). Resource constraints usually appear in earlier development stages and prevent the development of all the defined requirements, forcing developers to negotiate requirements. Therefore, a basic action is to select the set of requirements to be included in the subsequent steps of the development project. This problem, known as the next release problem (Bagnall et al. 2001), has come to the attention of AI researchers and is considered an optimization problem (Bagnall et al. 2001; Karlsson and Ryan 1997; Greer and Ruhe 2004; del Sagrado et al. 2015).

Probabilistic approaches have been used less in RE, maybe because RE decision making is not sufficiently mature, and decision problems in RE are an unclosed set that hinder them being solved using probabilistic models. Furthermore, there are several major challenges about how to apply BN to RE, such as how to deal with network validation, or how to embed the models obtained in computer-aided software engineering tools (del Águila and del Sagrado 2015). This is the reason why, in this paper, we include not only the probabilistic model and some evidence of its validity but also the integration of the model in an academic CARE tool, InSCo-Requisite (Orellana et al. 2008), expanding it with new probabilistic functionality.

3 A probabilistic requirements engineering solution

Bayesian networks (Jensen 2007; Kjaerulff and Madsen 2007) allow us to graphically and concisely represent knowledge regarding an uncertain domain. A Bayesian network has:

- a qualitative component, $G = (U, A)$, which is a directed acyclic graph (DAG), where the set of nodes, $U = \{V_1, V_2, \cdots, V_n\}$, represents the system variables, and the set of directed edges, A, represents the existence of dependences between variables.
- a quantitative component, P, which is a joint probability distribution over U that can be factorized according to:

$$P(V_1, V_2, \cdots, V_n) = \Pi_{i=1}^{n} P(V_i | Pa(V_i)) \tag{1}$$

where $P(V_i|Pa(V_i))$ is the conditional probability for each variable V_i in U given its set of parents $Pa(V_i)$ in the DAG.

The structure of the associated DAG determines the dependent and independent relationships among the variables. In addition, the local conditional probability distributions measure the strength of the direct connections between variables.

A BN can be used as a predictor simply by considering one variable as the class and the others as features that describe the object that has to be classified. The posterior probability of the class is computed given the features observed. The value assigned to the class is that which reaches the highest posterior probability value. A predictor based on a BN model provides more benefits than traditional predictors for decision support because it can perform powerful what-if problem analysis.

3.1 Bayesian network *Requisites*

BNs are very useful in SE, since the representation of causal relationships among variables is meaningful to software practitioners (Harman 2012; Misirli and Bener 2014). A specific case of this AI technique in requirements workflow is the Bayesian network *Requisites* (del Sagrado and del Águila 2010). This was built through interactions with experts and using several information sources. Its aim is to provide developers with assistance, in the form of probabilistic advice, to help them make decisions about the stability of a particular requirements specification. *Requisites* provides an estimation of the degree of revision for a given requirements specification (i.e., SRS). Thus, it helps when identifying whether a requirements specification is sufficiently stable and needs no further revision (i.e. whether it is necessary to perform a new DDV cycle or not).

In order to build *Requisites*, the general BN construction process (Korb and Nicholson 2010) was applied manually, assisted by two software engineers in an interview-evaluation cycle (for more details, see (del Sagrado and del Águila 2010)). The steps followed were:

1. Structure elicitation. The network topology has to capture the relationships between variables (i.e., two variables should be connected by an arc if one affects the other). In general, any independence suggested by a lack of an arc should correspond to real independence in the knowledge domain. Table 1 shows the variables identified by the experts and the structure of *Requisites* (see Fig. 1) reflects the dependencies identified by experts in order to assess the goodness of the SRS.
2. Parameter elicitation. Once the DAG structure had been set up, experts defined the strength of the relationships between variables by specifying the conditional probabilities in the network.
3. Validation and testing. This step checks if the model meets the criteria for use (see Section 3.2).

Once *Requisites* was built, it could be used to determine whether the requirements specification should, or should not, be revised. The values of the known variables are considered as evidence and are taken as the BN inputs in order to carry out the inference process. Specifically, variable elimination (Jensen 2007), a simple and general exact inference algorithm, is used for inferring the maximum a posteriori state of the unknown variables subset. As a result of this process, the marginal probability distribution of the variable's 'degree of revision' is also obtained and, consequently, the value assigned to the 'degree of revision' is that which achieves the maximum probability.

It is worth noting some of the relationships between variables. If the 'degree of commitment' (i.e., the number of requirements that have to be agreed) increases, then the level of 'specificity' will drop. If stakeholders have little 'experience' in RE processes, one is

Table 1 Variables in *Requisites* (del Sagrado and del Águila 2010)

Variable	Description and value meanings
Stakeholders expertise	Represents the degree of familiarity that stakeholders have in respect to RE processes. (High: stakeholders have already collaborated in several projects using RE techniques. Medium: stakeholders have collaborated in few projects using RE techniques. Low: stakeholders are not familiar with RE approaches)
Domain expertise	Expresses the level of knowledge that the development team has about the project domain. (High: Developers use the same concepts as stakeholders. Medium: Developers use similar concepts as stakeholders. Low: Developers do not share concepts with stakeholders)
Reused requirement	Checks if there are reused requirements. The reuse is an attempt to reduce the development cost by enhancing the productivity of the development team. Thus, if the number of requirements that comes from reusable libraries is high, the requirements specification does not generally need new iterations. (Many, Few, None)
Unexpected dependencies	In some cases, unexpected dependencies or relationships can appear between requirements or groups of them. This usually involves a new revision of the requirements specification. (Yes, No)
Specificity	Represents the number of requirements that have the same meaning for all stakeholders. (High: Most of the requirements are similarly interpreted. Medium: Several requirements present different interpretations. Low: Most of the requirements need to be clarified because their interpretation is confused)
Unclear cost/benefit	Represents stakeholders or developers including requirements that do not have direct quantifiable benefits for the business, or the organization, in which the software to be developed will operate. (High: Cost/benefit for most of the requirements has been rated. Medium: Several requirements have no clear benefit. Low: Most of the requirements do not have a cost/benefit rate)
Degree of commitment	Represents the number of requirements that need a negotiation to be accepted. The requirements for a project are a complex combination of requirements from different stakeholders, and some of these can generate conflicts that unbalance the specification. (High, Medium, Low)
Homogeneity of the description	A good SRS must be described at the same level of detail. If some requirements have been described in a detailed way, all the requirements should be described at the same level of detail (Yes, No). If there is not homogeneity, the SRS will need to be revised.
Requirement completeness	Indicates if all significant requirements have been elicited and/or specified. (High, Medium, Low)
Requirement variability	Represents the number of requirements that were changed. If the specification of a requirement changes, it is quite possible that this modification will affect the whole SRS, and an additional revision is likely to be needed. (High, Medium, Low)
Degree of revision	Is the value predicted by *Requisites* and this indicates that a SRS is sufficiently accurate as to not require further revisions. (Yes, No)

more likely to get 'unclear' requirements in terms of cost/benefits. The 'requirement completeness' and the 'homogeneity of the description' are influenced by the 'experience of the software engineers' in the project domain and by the 'stakeholders' expertise' in the RE processes or tasks. If experience is high, the specification will be 'complete and homogeneous' because developers have been able to describe the requirements with the same level of detail and all requirements have been discovered. Finally, 'requirement variability' represents the number of requirements that have been changed. A change will likely occur if 'unexpected

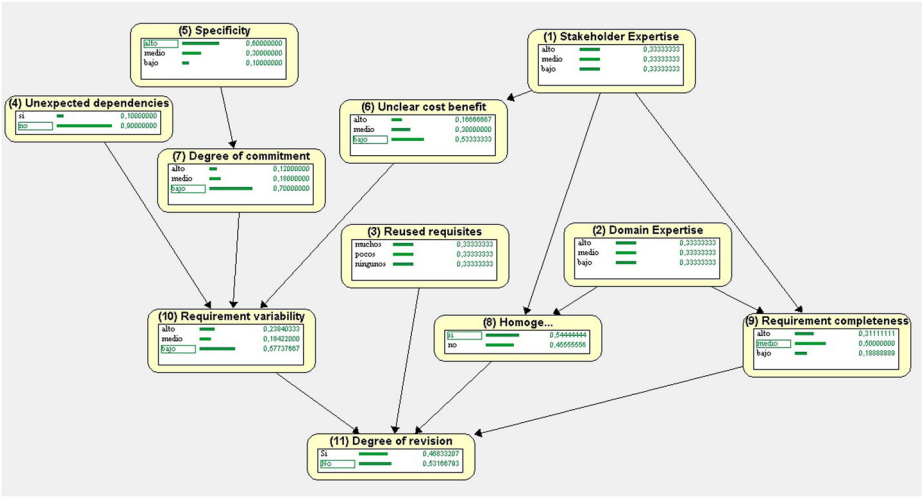

Fig. 1 Bayesian network *Requisites*

dependencies' are discovered or if there are requirements that do not add any value to the software; or if there are missing requirements; or if requirements have to be negotiated.

There are free software packages available to build and use Bayesian networks. In our work, we used Elvira (Elvira Consortium 2002), a package that allows the implementation of a BN model using Java classes to support the model itself and the inference processes. That is to say, through Elvira's application programming interface (API), we gain access to model specification facilities and to the variable elimination inference algorithm. However, from a practical point of view, the integration of a built network within a specific software application is not trivial, because it is necessary to define the communication paths between both components by matching variables and results.

3.2 Testing *Requisites* validity

Requisites makes a prediction indicating whether a requirements specification can be considered as the next baseline or whether it needs further revision. The inference process uses the evidence established by the metrics, worked out from the current SRS version, which is usually stored in the CARE tool database; this calculates the marginal probability distribution of the 'degree of revision' variable. The aim of this subsection is to test the suitability of the BN model for doing the job it is designed for and also to understand how the network can be used in a real project to perform inference processes.

A real-world large-scale dataset was adopted to evaluate the *Requisites* approach. RALIC is the acronym for replacement access, library and ID cards. This was a large-scale software project to replace the existing access control system at University College, London and to consolidate the new system with library access and borrowing (Lim and Finkelstein 2012). The RALIC objectives include replacing existing access card readers, printing reliable access cards, managing cardholders' information, providing access control, and automating the provision and suspension of access and library borrowing rights. The stakeholders involved in the project had different and sometimes conflicting requirements. The project duration was two and a half years, and the system has already been deployed at

University College, London. By measuring this dataset, we obtain the evidence values and the metrics are inserted into the BN to analyze inference through the probabilistic model.

RALIC requirements were organized into three hierarchical levels: project objectives, features, and specific requirements. A feature that contributed toward a project objective was placed under the project objective, and a specific requirement that contributed toward the feature was placed under features. This requirements organization indicates the level of detail at which requirements are described, i.e., the 'homogeneity of the description.' The level reached by project objectives can be studied in order to set the homogeneity of the description value.

We study the proportion of the requirements linked to a project objective that have been described in terms of specific requirements. Figure 2 shows the percentage of detail distribution applied to describe project objectives in a box plot. 75% of project objectives were described in terms of specific requirements in a percentage above 50.96. This indicates that the branches in the hierarchical structure of the project have a similar depth, which translates as a homogeneous description. Thus, the 'homogeneity of the description' value is set to 'yes' in *Requisites*.

Stakeholders are asked to rate requirements rather than rank them all, because previous work has shown that large projects can have hundreds of requirements, and stakeholders experienced difficulty providing a rank order of requirements when so many existed (Lim and Finkelstein 2012). Each stakeholder assigns ratings to the identified requirements. A rating is a number on an ordinal scale ($0-5$, see Fig. 3), reflecting the importance of the requirement to the stakeholder (e.g., 0 means that the requirement is not considered important to the stakeholder; 1 means that the requirement is not very important to the stakeholder; and 5 means that the requirement is very important).

'Specificity' deals with the meaning of the requirements for the stakeholders. Thus, the higher the number of stakeholders who agree, the lower the degree of revision needed. In order to measure the specificity value, a consensus measure has been used, which is the average of the ratings assigned to a given project objective; that is to say, the higher the average rating of a project objective, the higher the specificity. Figure 4.a shows the distribution of the average rating of each project objective, while Fig. 4.b shows the distribution of the specificity of each project objective. The specificity value for each project objective has been computed directly from its average rating by adapting the range from $\{0, 1, 2, 3, 4, 5\}$ to $\{1(low), 2(medium), 3(high)\}$. As a result of this process, the specificity value is set to 'high' in *Requisites* because 90% of the project objectives map to a 'high' specificity value.

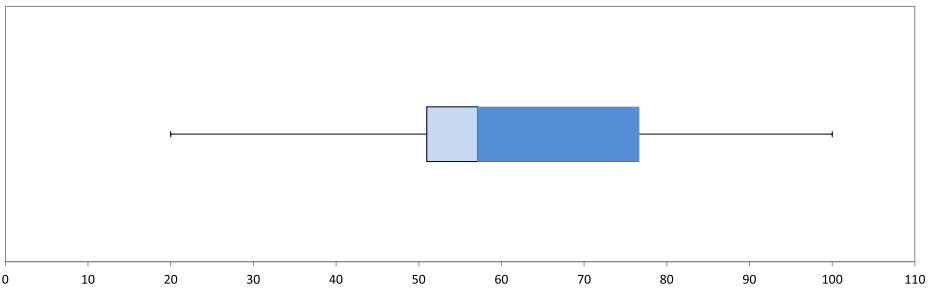

Fig. 2 Percentage of detail distribution applied to describe project objectives

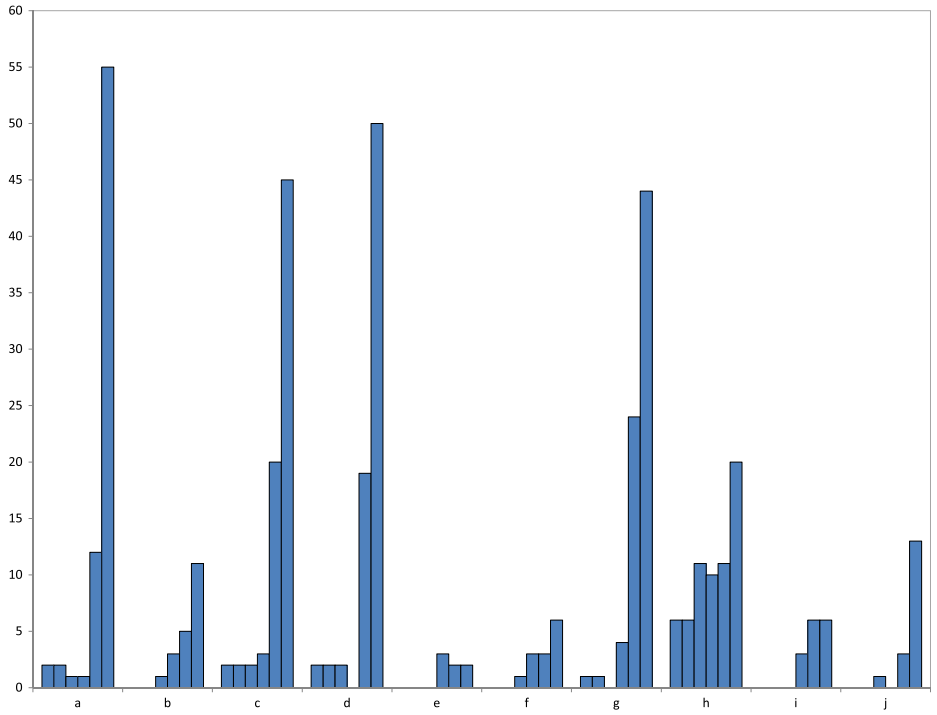

Fig. 3 Ratings assigned by stakeholders to project objectives

The stakeholders' priority data for RALIC was also collected (Lim and Finkelstein 2012). The stakeholders were asked to recommend people whom they think should be stakeholders in the project. Their recommendations were then used to build a social network, where the stakeholders were nodes and their recommendations were links. The output was a prioritized list of stakeholders and their requirements preferences.

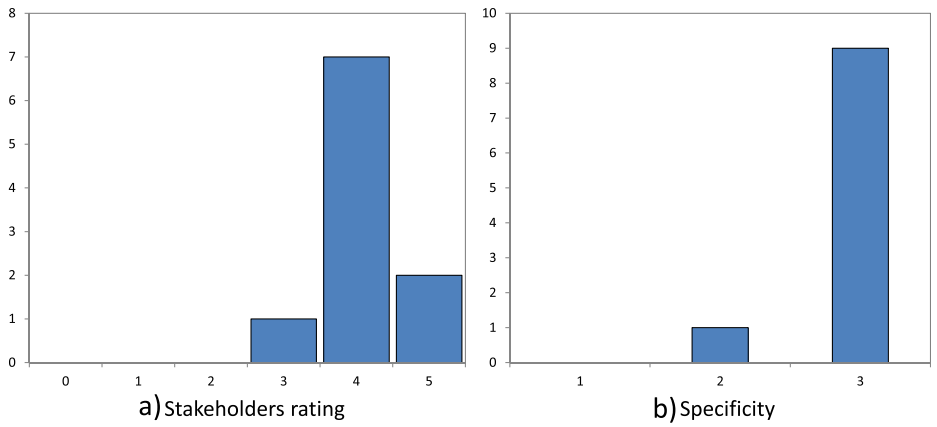

Fig. 4 Distribution of the project objectives rating and the specificity

Stakeholders were requested, through the OpenR questionnaire, to make recommendations on the salience (i.e., level of influence on the system) of other stakeholders. The salience of a stakeholder is assigned as an ordinal variable whose domain is the set {1, 2, 3, 4, 5, 6, 7, 8}. Salience and expertise have a straight relationship as the influence on a system that each stakeholder has. Thus, it can be considered a measure of the level of expertise. In order to get an overall estimation of the stakeholders' expertise, the recommendations received by each stakeholder are first summarized as the average (Fig. 5a shows the stakeholders' salience distribution computed in this way). Then, the expertise of each stakeholder is obtained mapping her/his salience to the set {1(low), 2(medium), 3(high).}. Figure 5b shows the stakeholders' expertise distribution computed thus. One can observe that 92% of stakeholders receive a 'low' expertise recommendation, thus the value of stakeholders' expertise is set to 'low' in *Requisites*.

Subsequently, we need to study the behavior of the BN when the data extracted from the RALIC dataset are incorporated into the model (see the gray highlighted nodes shown in Fig. 6). The a priori probabilities are shown in Fig. 1. If the value of the variable 'homogeneity of the description' is set to 'yes,' the probability of the value 'no' to the degree of revision will rise to 0.54; that is, the more homogeneous the overall description of the requirements, the less revision is needed. This trend is reinforced when the evidence value for 'specificity' was included, the 'degree of revision' becomes 0.45, 0.55 for 'yes' and 'no,' respectively. Nevertheless, because the 'expertise for stakeholders' had a 'low' value, the final prediction shown in Fig. 6 is to review the SRS. The final values become 0.52 for 'yes' and 0.48 for 'no.'

4 Integrating *Requisites* into a CARE tool

One of the biggest breakthroughs in requirements management workflow was produced when we stopped focusing on documents and started focusing on information. Therefore, developers had to resort to databases to handle this information; specifically to documentary databases that have evolved as part of the current CARE tools. Nowadays, there are many CARE tools available (de Gea et al. 2012). Among these, the most well known are IRqA (Visure Solutions 2012), Telelogic DOORS (IBM 2012) and Borland Caliber (Borland Software Corporation 2016). InSCo-Requisite is an academic web CARE tool that was developed by the DKSE (Data Knowledge and Software Engineering) group at the University of Almería. It partially supports the requirements development stage (Orellana et al. 2008) and also provides a basic functionality where groups of stakeholders collaboratively

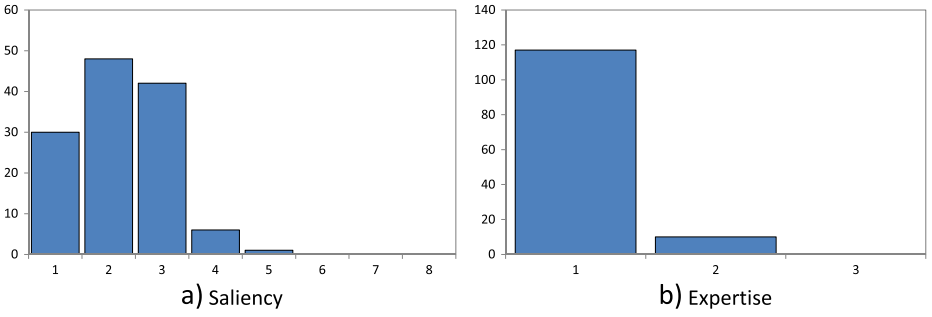

Fig. 5 Distribution of stakeholders' salience and expertise

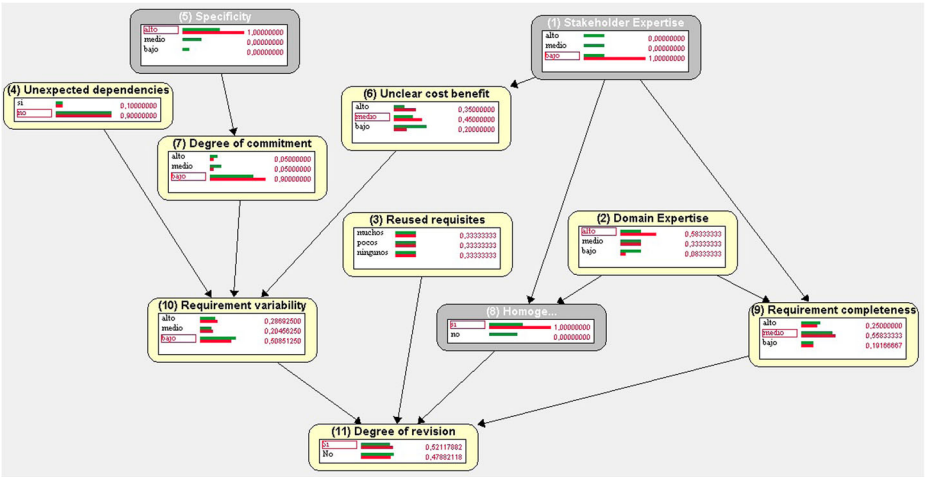

Fig. 6 *Requisites* state after including the evidence (*gray* highlighted nodes) obtained from RALIC

work via the Internet to define the SRS. Because of the ease in making changes to this tool, we have an exceptional opportunity to try to integrate AI techniques into this CARE tool. Furthermore, this solution successfully instantiates an architecture for the seamless integration of a CARE tool with certain AI techniques (del Sagrado et al. 2011); it also offers a solution to the problem of embedding the BN into CARE tools (del Águila and del Sagrado 2015).

4.1 InSCo-Requisite

Commercial CARE tools offer powerful solutions for capturing the requirements of a software development project and also include methods for analyzing the requirements, or to monitor the changes on each requirement. The purpose of InSCo-Requisite is not to compete against these commercial tools. It has been developed within an academic setting to deal with the problem of software project management, which includes components that are based, or not based, on knowledge (del Águila et al. 2010; Cañadas et al. 2009). Its main goal is to offer an intuitive and easy-to-use tool that manages requirements in a distributed environment.

Requirements are more than a list of 'the system shalls.' In a broad sense, requirements are a network of interrelated elements or artefacts. These artefacts (e.g., objectives, constraints or priorities) must be modified and managed during the discovering requirements task. Our tool guides requirements management by using templates, which can be classified into two groups—*functional requirements*: *objectives*, which specify business-related information and *user requirements*, which are related to customer needs. At the lowest level of refinement, templates are fulfilled using natural language and scenarios. *Non-functional requirements* have their own template to collect the quality attributes and the constraints. Figure 7 shows the InSCo-Requisite artefact model. Users can be assigned to various projects, which are organized into folders that permit a hierarchical structuring of the requirements. A user can participate in a project by proposing new requirements, changes in requirements or comments on them.

The tool can represent relationships between requirements, as those defined between objectives and users' requirements. It maintains a hierarchical structure of templates, which offers a global perspective of the project content and displays parent/child relationships in an explorer window.

InSCo-Requisite offers some facilities for informal collaboration through rich contextual discussions about requirements. Users can start discussions about any of the existing requirements in order to propose changes, improvements, or discuss the template contents. The tool provides access to a change log for each requirement, stakeholder or project. There are unlimited events, changes, or stakeholder comments associated to any requirement.

Java EE (Java Enterprise Edition) is the platform chosen for developing and deploying InSCo-Requisite. It has a multi-layered architecture based on Struts, an open-source multiplatform framework created by the Apache Software Foundation. The architecture adopted in the current version of the tool separates presentation logic from business logic and persistent storage. The interface layer represents the client side; that is to say, web browsers that send requests to the application server at the service layer, using the hypertext transfer protocol (HTTP). This layer is also in charge of representing the data received from the service layer, basically under the form of HyperText Markup Language (HTML), cascading style sheet (CSS), and Javascript code. The service layer is composed by an Apache Tomcat server that gives support to the Java EE Platform for the InSCo-Requisite application. The server processes the incoming requests, executes the appropriated Java servlets, and exchanges data with the data layer by means of a Java database connection (JDBC). Finally, the data layer is in charge of maintaining data persistence; this is composed by an Oracle database server.

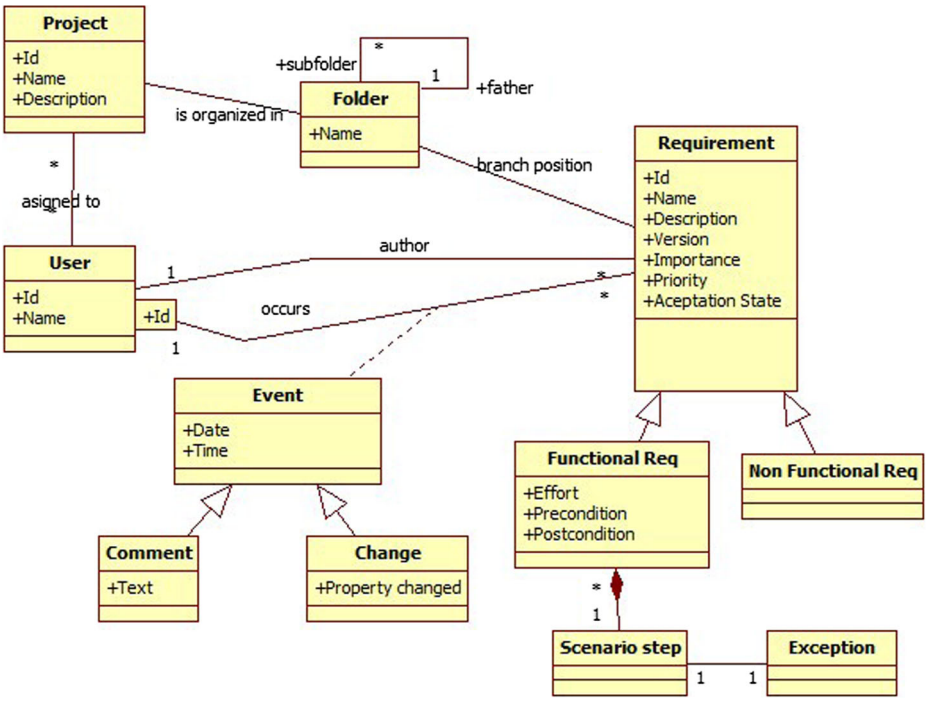

Fig. 7 Conceptual model of InSCo-Requisite

4.2 Connection between *Requisites* and InSCo-Requisite

It would be of considerable help to any development team to have AI techniques available inside a CARE tool (del Sagrado et al. 2012). Since the requirements management task is performed by means of a CARE tool (e.g., InSCo-Requisite), this tool should provide the information required by the BN *Requisite* (evidence) in order to obtain the value for the variable (e.g., degree of revision) that the development team wants to predict.

However, AI techniques and CARE tools must evolve independently of each other. Therefore, it is necessary to define a communication interface between them, preserving the independence of both areas and achieving a synergetic profit (del Sagrado et al. 2011). The CARE tool is in charge of all the information management related to the development project (requirements, customers, etc.) which is stored in a database. Next, the communication interface connects the CARE tool and the Bayesian network, interchanging the required information needed to execute the appropriated processes and adapt the languages used by each tool (see Fig. 8). Thus, the validating requirements task receives metrics from the SRS and returns a degree of revision estimation for the SRS; that is say, the knowledge-based tool helps the requirements validation task based on the DDV cycle.

The BN *Requisites* makes a prediction for the degree of SRS revision. The evidence (i.e., the observed variables) is provided by the InSCo-Requisite CARE tool, which is in charge of obtaining the variable values from: the data regarding projects, the requirements, the users' activity, and so on. Table 2 shows the data gathered by InSCo-Requisite to obtain this evidence. Once these data have been collected, the evidence is computed in the following way:

– Stakeholders' expertise. Firstly, for each stakeholder in the current project, the proportion of projects in which she/he was previously involved is computed (i.e., the value obtained when we divide the number of previous projects in which the stakeholder was involved by the total number of previous projects managed in InSCo-Requisite). Moreover, we analyze how the stakeholder behaves within the current project as a way of giving weight to the previous proportion; that is to say, the number of requirements

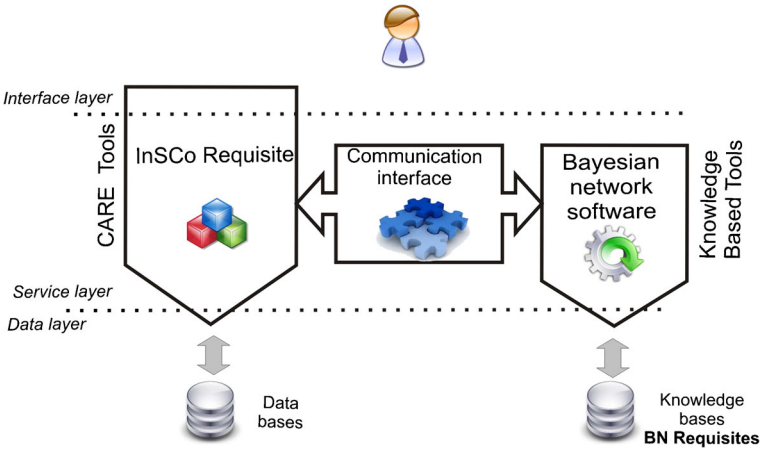

Fig. 8 Generic Architecture

Table 2 Connections between measures in InSCo-Requisite and *Requisites*

Requisite variables	Measures in InSCo-Requisite	
Stakeholders'	For every stakeholder in the project counts:	High
expertise	• Projects to which stakeholders have been assigned	Medium
	• Requirements with stakeholders' participation	Low
Domain expertise	Manual assignment.	High Medium Low
Reused requirement	InSCo-Requisite does not support it	Many Few None
Unexpected dependencies	InSCo-Requisite does not support it	Yes, No
Specificity	Number of ACCEPTED requirements in which:	High,
	Several stakeholders have participated	Medium Low
Unclear cost/benefit	Number of requirements whose:	High
	Status has changed (ACCEPTED-REJECTED)	Medium
	Comments have been sent by several stakeholders	Low
Degree of	Number of requirements in which:	High
commitment	• Several stakeholders have sent comments	Medium
	• Several stakeholders have performed changes	Low
Homogeneity of the description	All the branches in the hierarchical structure of the project have a similar depth	Yes, No
Requirement completeness	Level of fulfilment of template fields of all the requirements	High, Medium Low
Requirement variability	Number of changes registered on requirements	High Medium Low

where each stakeholder is involved is computed (i.e., we divide the number of requirements in which the stakeholder was involved by the total number of requirements in the current project). Finally, as a measure of the stakeholder's expertise, we give the average of these values. If the weighted mean is below $1/3$, the stakeholder's expertise is set to 'Low.' If the weighted mean is below $2/3$, the stakeholder's expertise is set to 'Medium.' Otherwise, the stakeholder's expertise is set to 'High.' Then, the global value of the variable *Stakeholders' expertise* is computed as the weighted mean of the individual stakeholder's expertise values, using as weights their assigned importance inside the project that is under study.

– Specificity. Within a InSCo-Requisite project, this is measured as the proportion of accepted requirements in which several stakeholders were involved (i.e., the number of accepted requirements in which several stakeholders were involved divided by the number of accepted requirements). Once again (as in the *stakeholders' expertise* situation), the values 'Low,' 'Medium,' and 'High' are used to set the value of specificity, taking $1/3$ and $2/3$ as thresholds, respectively.

- Unclear cost/benefit. In InSCo-Requisite, this variable is measured as the proportion of requirements that have been commented on and that have also changed status from accepted to rejected over the total number of requirements inside a project. Once more, the values of the variable are set to 'Low,' 'Medium,' or 'High' using $1/3$ and $2/3$ as thresholds, respectively.
- Degree of commitment. InSCo-Requisite measures the degree of commitment as the proportion of requirements that have been commented on or modified over the total number of requirements in a project, when these changes were made by different stakeholders. The values 'Low,' 'Medium,' and 'High' are assigned to the variable using $1/3$ and $2/3$ as thresholds, respectively.
- Homogeneity of the description. In InSCo-Requisite, a description is considered homogeneous if the standard deviation of the depth of the branches in the hierarchical structure is less than or equal to 1.
- Requirement completeness. This is measured as the average level of fulfilment of the template fields of all requirements in the project. Here, the template's level of fulfillment for a requirement is defined as the number of fields that are filled in the template divided by the total number of template fields. The values of requirement completeness are set to 'Low,' 'Medium,' and 'High' using $1/3$ and $2/3$ as thresholds, respectively.
- Requirement variability. InSCo-Requisite measures this as the proportion of requirements that have been modified over the total number of requirements in the project. Thus, a proportion less than $1/3$ corresponds to a 'Low' requirement variability. If the proportion is less than $2/3$, then the requirement variability is set to 'Medium.' Otherwise, the requirement variability is set to 'High.'

Note that some variables cannot be obtained from the InSCo-Requisite tool, and must be estimated by users, such as 'domain expertise' or 'reused requirement.' But our architecture allows these metrics to be included in *Requisites* when InSCo-Requisite would be able to measure them (see Table 2).

4.3 A use case

Requirements engineers execute the DDV cycle several times until they get a complete requirements specification. The complete process is shown in Fig. 9, and the enhancement, obtained by the integration of *Requisites* in InSCo-Requisite, is specifically concerned with the task of validating requirements. The BN can be employed using two approaches or modes: *analytic* or *exploratory*. The *analytic mode* (see Fig. 10) is used when the user wants to obtain the posterior probability distribution of any target variable, given some evidence (i.e., an assignment of the values for some of the other variables). In *exploratory mode* (see Fig. 11), the user can choose a target variable and the system returns the variables list thus isolating it from the rest of the variables in the network (i.e., the set of nodes composed by the variable's parents, its children, and the other parents of its children) together with the measurements (see Table 2) obtained by InSCo-Requisite. From this, the posterior probability of the target variable is obtained, given the fixed evidence for the relevant variables. Both modes compute the posterior probability distribution of the 'degree of revision' which indicates if the current SRS needs further revision.

The analytic mode is suitable for situations where the requirements engineer wants to check whether the current SRS can be considered as a baseline for a software project. In this mode, the list of variables in *Requisites* is displayed and InSCo-Requisite can extract and get the variable values by applying the process to obtain evidence (see Section 4.2 and

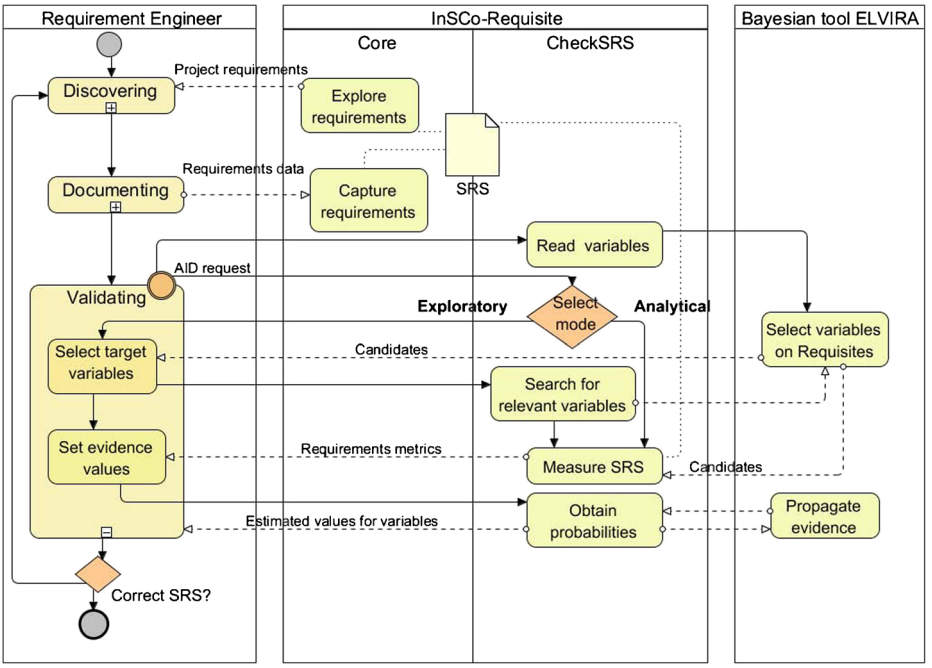

Fig. 9 A processes model for requirements engineering

Table 2). The engineer can consider these values as evidence and can also set them manually in the column *Evidence* (see Fig. 10). The posterior probability of any target variable, given the assignment of evidence, can be obtained by clicking on *Propagate to the network* button. In the case depicted in Fig. 10, the probability value of 0.775, associated to the 'degree of revision' variable, indicates that the current SRS has to be reviewed; and the value 0.609 given to the 'domain expertise' variable tells the engineer of low the knowledge level that the development team possesses about the project domain.

The exploratory mode is indicated when requirements engineers are interested in guessing which topics can be enhanced to improve the SRS. Consider a situation where the engineer wants to investigate 'requirements completeness.' In this mode, the engineer chooses this variable as the target. Automatically, the list of variables that shield it from the influence of the rest of the variables in the network is displayed in the column *Relevant variables* (see Fig. 11). The evidence values for each of these can be selected by means of a drop-down list of values. Finally, the a posteriori 'degree of revision' probability, the target variable and all the variables in the column *Relevant variables* that have not received evidence are obtained by clicking on the *Propagate to the network* option. For the case depicted in Fig. 11, the probability value of 0.661, associated to the 'degree of revision' variable, indicates that the current SRS has not to be reviewed; the value 0.458 for the 'requirement completeness' variable tells the engineer that a large number of the significant requirements have been elicited and shows that the 'domain of expertise' is high (reaching a probability value of 0.583).

These two modes complement each other. The analytic mode allows one to find out the aspects of the project which the developer should focus on to improve the requirements specification and to afford the next phases of software development a greater chance of success.

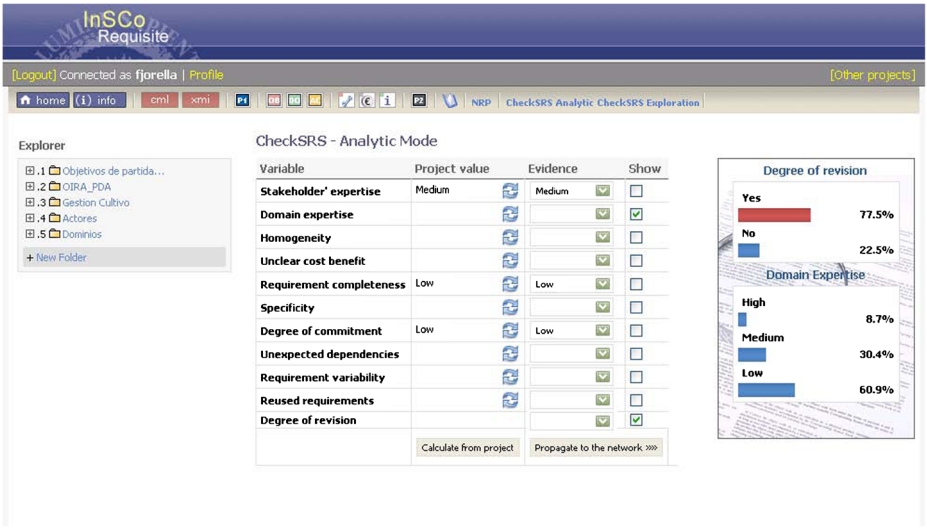

Fig. 10 Analytic mode of InSCo-Requisite

Consequently, after setting the 'degree of revision,' the developer can compare the values of the variables obtained by BN propagation, with the values obtained by direct measurement in the CARE tool. Based on this comparison, the developer will take corrective actions. Moreover, the exploratory mode allows the engineer to focus on a single variable. In this way, the improvements to be made in project management can be focused on actions that directly affect the set of measurements that influence this variable in order to keep it at a

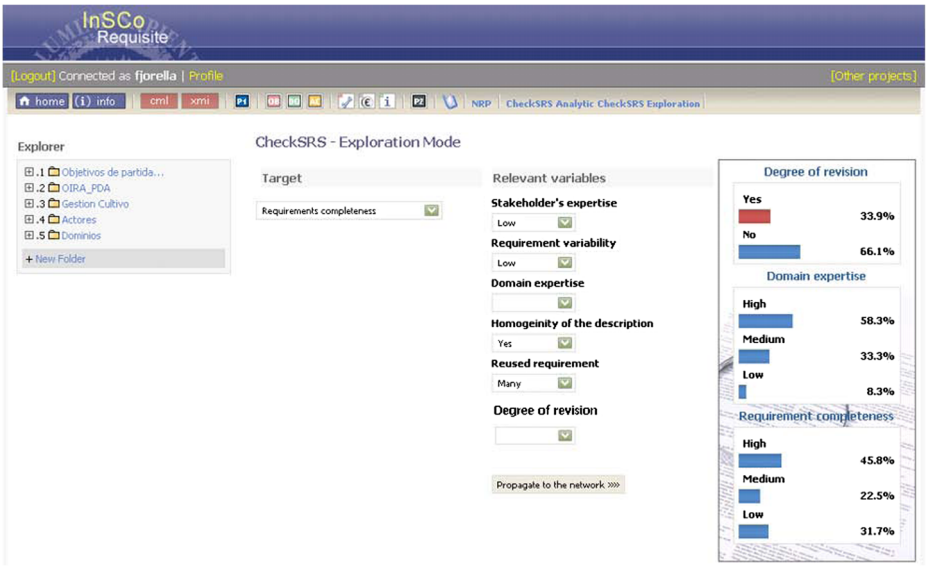

Fig. 11 Exploratory mode of InSCo-Requisite

specific quality level. At the end of a DDV cycle, engineers can use the *analytic mode* to assess whether the SRS is sufficiently accurate to not need further revision. If a *Requisites* suggests the need for revision, the team should apply the *exploratory mode* to guess which SRS concerns (represented by the variables in the BN) might be candidates for enhancement in order to improve SRS stability. This enhancement usually involves carrying out another DDV cycle (see Fig. 9).

5 Threats to validity

This section discusses the threats that could affect the validity of our study in order to provide a complete understanding of limitations and extent that the work has. Construct validity concerns the relation between the theory and the observations. Internal validity concerns possible bias with the results obtained by the proposal. External validity is related to the generalization of observed results outside the sample instances used.

The most important threat to construct validity to be discussed is derived from the fact that the BN model presented in this work has been built following a completely manual process, not based on any empirical data, based on experts' knowledge. In other words, the process does not ensure neither all relevant variables are included in the model nor all the included variables are relevant for the assessment of the SRS, beyond the knowledge of the experts involved. That is why the Section 3.2 deals with giving some evidence of *Requisites* validity by testing the suitability of the model using data measured according to a dataset that is totally independent of this research work.

Threats to internal validity concern mapping processes between model variables and metrics gathered from projects. These mapping processes were managed in Section 3.2 and when the connection between *Requisites* and InSCo-Requisite was defined in Section 4.2. In the first case, to prevent any bias when the data extracted from the RALIC dataset that will be incorporated into the BN model, a detailed study of the statistical distribution of raw data has been applied. For instance, in order to measure the *specificity* value, a consensus measure has been used. The value has been computed directly from its average rating, getting a 'high' value because 90% of the project objectives map to a 'high' specificity value (see Fig. 4).

The second threat in the mapping process takes place when an adaptation of the requirements metrics from InSCo-Requisite to the variables used in the *Requisites* is needed. The BN model receives the evidences that the CARE-tool obtains by measuring the SRS and from usage data. Then, after making an exact inference process (i.e., variable elimination) based on the evidence gathered, it is computed an estimation of the degree of revision for the SRS. The communication interface has been defined based on specific methods and algorithms that manage the numerical representation of each element of the conceptual model in InSCo-Requisite. The assumption made in the communication interface is that variables are random and follow an uniform distribution, due to the manual nature of the model and the lack of data.

Threats to external validity are those related to the generalization of the results. In order to test the suitability of the BN model for doing the job it is designed for, only one dataset has been used because it is extremely difficult to obtain reliable data with which we could generalize our work. This is the reason why Requisite must be embedded in a CARE tool, in our case by extending InSCo-Requisite. The easy availability of the BN model will promote the widespread use of it that will also facilitate the task for checking whether the results are general enough.

6 Conclusions

Complex decision-making is a prominent aspect of Requirements Engineering since it is mainly a human activity with the least technical load of the entire software project. In this work, we have defined how to enhance requirements specification development through the integration of a Bayesian network in a requirements management tool. The approach we have taken does not consist of solving a particular problem. Instead, we have instantiated an architecture that can be adapted to other reasoning models and other software tools. This has been achieved by establishing a communication interface, between the academic CARE tool, InSCo-Requisite, and the Bayesian network *Requisites* within a multilayered architecture. Furthermore, we have provided evidence of the validity of *Requisites*, not only by its integration within InSCo-Requisite but also by using the evidence extracted from a large-scale project, RALIC. The metrics calculated using this project data allow us to successfully predict the need for SRS revision. The integration between *Requisites* and InSCo-Requisite provides the decision-maker with a way of exploring the state of the requirements work carried out using both analytic and exploratory modes to improve the overall results of this stage.

In the near future, we plan to carry out an empirical evaluation of the enhanced InSCo-Requisite tool. We also want to refine the Bayesian Network Requisites using the data recorded by this tool, and to perform a study of its applicability to a subset of requirements instead of to the whole SRS; this will involve a change in the way the requirements metrics are computed. In this setting, the exploration of alternative mappings in the communication interface also deserves special attention.

Acknowledgements This research has been financed by the Spanish Ministry of Economy and Competitiveness under projects TIN2013-46638-C3-1-P, TIN2015-71841-REDT and partially supported by the Data, Knowledge and Software Engineering (DKSE) research group (TIC-181) of the University of Almería.

References

Abran, A., Moore, J., Bourque, P., Dupuis, R., & Tripp, L. (2004). *Guide to the Software Engineering Body of Knowledge*. Los Alamitos: IEEE Computer Society.

del Águila, I.M., & del Sagrado, J. (2011). Requirement risk level forecast using Bayesian networks classifiers. *International Journal of Software Engineering and Knowledge Engineering*, *21*(2), 167–190.

del Águila, I.M., & del Sagrado, J. (2015). Bayesian networks for enhancement of requirements engineering, a literature review. *Requirements Engineering*, 1–20.

del Águila, I.M., del Sagrado, J., Túnez, S., & Orellana, F.J. (2010). Seamless software development for systems based on Bayesian networks - an agricultural pest control system example. In *5th International Conference on Software and Data Technologies, (ICSOFT)*, (Vol. 2 pp. 456–461). Athens.

Alexander, I., & Beus-Dukic, L. (2009). How to specify products and services. *Discovering requirements*. New York: Wiley.

Bagnall, A.J., Rayward-Smith, V.J., & Whittley, I. (2001). The next release problem. *Information & Software Technology*, *43*(14), 883–890.

Borland Software Corporation (2016). Caliber. Manage Agile requirements through visualization and collaboration. http://www.borland.com/en-GB/Products/Requirements-Management/Caliber/. Accessed 1 Mars 2016.

Cañadas, J., Orellana, F.J., del Águila, I., Palma, J., & Túnez, S. (2009). A tool suite for hybrid intelligence information systems. In *Proceedings of Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA'09)* (pp. 9–13). Sevilla.

Castro-Herrera, C., Cleland-Huang, J., & Mobasher, B. (2009). Enhancing stakeholder profiles to improve recommendations in online requirements elicitation. atlanta, georgia, usa. In *17th IEEE International Requirements Engineering Conference, (RE '09)* (pp. 37–46). Atlanta.

Cheng, B.H.C., & Atlee, J.M. (2007). Research directions in requirements engineering. In *Future of Software engineering, (FOSE)* (pp. 285–303). Minneapolis.

Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., & Mirakhorli, M. (2011). On-demand feature recommendations derived from mining public product descriptions. In *33rd International Conference on Software Engineering (ICSE)* (pp. 181–190). Waikiki, Honolulu.

Elvira Consortium (2002). Elvira, An environment for probabilistic graphical models. In *1st International Workshop on Probabilistic Graphical Models (PGM02)* (pp. 222–230). Cuenca.

de Freitas, F.G., & de Souza, J.T. (2011). Ten years of search based software engineering, A bibliometric analysis. *Search Based Software Engineering Lecture Notes in Computer Science*, *6956*, 18–32.

de Gea, J.M.C., Nicolás, J., Alemán, J.L.F., Toval, A., Ebert, C., & Vizcaíno, A. (2012). Requirements engineering tools, Capabilities, survey and assessment. *Information and Software Technology*, *54*(10), 1142–1157.

Greer, D., & Ruhe, G. (2004). Software release planning, an evolutionary and iterative approach. *Information & Software Technology*, *46*(4), 243–253.

Harman, M. (2012). The role of artificial intelligence in software engineering. In *Proceedings of the 1st International Workshop on Realizing AI Synergies in Software Engineering* (pp. 1–6): IEEE.

Harman, M., Mansouri, S.A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, *45*(1), 11.

IBM (2012). Rational DOORS. http://www-03.ibm.com/software/products/es/ratidoor. Accessed 1 mars 2016.

Jensen, F.V. (2007). Information Science and Statistics. *Bayesian Networks and Decision Graphs*: Springer. corrected edition.

Karlsson, J., & Ryan, K. (1997). A cost-value approach for prioritizing requirements. *IEEE Software*, *14*(5), 67–74.

Kastro, Y., & Bener, A.B. (2008). A defect prediction method for software versioning. *Software Quality Journal*, *16*(4), 543–562.

Kjaerulff, U.B., & Madsen, A.L. (2007). *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*, 1st edn: Springer.

Korb, K.B., & Nicholson, A.E. (2010). *Bayesian Artificial Intelligence*: CRC Press.

Kotonya, G., & Sommerville, I. (1998). *Requirements engineering: Processes and techniques*. New York: Wiley.

Lim, S.L., & Finkelstein, A. (2012). Stakerare: using social networks and collaborative filtering for large-scale requirements elicitation. *IEEE Transactions on Software Engineering*, *38*(3), 707–735.

Menzies, T., & Shepperd, M. (2012). Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, *17*(1), 1–17.

Meziane, F., & Vadera, S. (2010). *Artificial intelligence applications for improved software engineering development: New prospects*. New York: IGI Global.

Misirli, A.T., & Bener, A.B. (2014). Bayesian networks for evidence-based decision-making in software engineering. *IEEE Transactions on Software Engineering*, *40*(6), 533–554.

Mısırlı, A.T., Bener, A.B., & Turhan, B. (2011). An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*, *19*(3), 515–536.

Nicolás, J., & Toval, A. (2009). On the generation of requirements specifications from software engineering models: A systematic literature review. *Information and Software Technology*, *51*(9), 1291–1307.

Orellana, F.J., Cañadas, J., del Águila, I.M., & Túnez, S. (2008). InSCo requisite - a web-based RM-tool to support hybrid software development. In *10th International Conference on Enterprise Information Systems (ICEIS) (3-1)* (pp. 326–329). Barcelona.

del Sagrado, J., & del Águila, I.M. (2010). Artificial intelligence applications for improved software engineering development, new prospects. In Meziane, F., & Vadera, S. (Eds.) *A Bayesian network for predicting the need for a requirements review*, (pp. 106–128). New York: IGI Global.

del Sagrado, J., del Águila, I.M., & Orellana, F.J. (2011). Architecture for the use of synergies between knowledge engineering and requirements engineering. *Lecture Notes in Computer Science*, *7023*, 213–222.

del Sagrado, J., del Águila, I.M., & Orellana, F.J. (2012). Metaheuristic aided software features assembly. In *20th European Conference on Artificial Intelligence (ECAI 2012), Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track* (pp. 1009–1010). Montpellier.

del Sagrado, J., del Águila, I.M., & Orellana, F.J. (2015). Multi-objective ant colony optimization for requirements selection. *Empirical Software Engineering*, *20*(3), 577–610.

Shirabad, J.S., & Menzies, T. (2005). Predictor models in software engineering (promise). In *27th international conference on Software engineering, (ICSE '05)* (pp. 692–692). New York: ACM.

Sommerville, I. (2011). *Software engineering*, 9 edn. Boston: Pearson Education.
Standish Group (2008). Chaos report. (2002).
Tosun, A., Bener, A., & Akbarinasaji, S. (2015). A systematic literature review on the applications of bayesian networks to predict software quality. *Software Quality Journal*, 1–33. cited By 0; Article in Press.
Visure Solutions (2012). Visure Requirements. Software for Requirements Engineering. http://www.visuresolutions.com/visure-requirements-software. Accessed 1 mars 2016.
Wen, J., Li, S., Lin, Z., Hu, Y., & Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, *54*(1), 41–59.
Wiegers, K., & Beatty, J. (2013). *Software requirements*: Pearson Education.
Zhang, Y., Harman, M., & Mansouri, A. (2012). The SBSE repository: A repository and analysis of authors and research articles on search based software engineering. crestweb. cs. ucl. ac. uk/resources/sbse repository.

José del Sagrado is currently a Professor of Computer Science in the Department of Informatics at the University of Almería. He obtained his PhD in Computer Science from the University of Granada in 2000. He is a member of the Spanish Association for Artificial Intelligence (AEPIA). His current research interests are mainly focused on probabilistic graphical models, specially combination of models, and probabilistic decision graphs, and also include the application of metaheuristic optimization in software engineering problems.

Isabel M. del Águila is currently a Professor of Computer Science at Almería University. She received her Ph.D. degree in Computer Science from the University of Almería in 2010. She is specialized in Software Engineering, UML, information systems, and search based Software Engineering. Her research interests include Software engineering and Knowledge engineering methods, particularly the unification of these engineering approaches.