Modelling Risks in Open Source Software Component Selection

Alberto Siena, Mirko Morandini, and Angelo Susi

Fondazione Bruno Kessler
I-38123, Trento – Italy
{siena,morandini,susi}@fbk.eu

Abstract. Adopting Open Source Software (OSS) components is a decision that offers many potential advantages – such as cost effectiveness and reputation – but even introduces a potentially high number of risks, which span from the inability of the OSS community to continue the development over time, to a poor quality of code. Differently from commercial off-the-shelf components, to assess risk in OSS component adoption, we can rely on the public availability of measurable information about the component code and the developing communities. In the present paper, we present a risk evaluation technique that uses conceptual modelling to assess OSS component adoption risks. We root it in the existing literature on OSS risk assessment and validate it by means of our industrial partners.

Keywords: Risk assessment, Open Source Software, Automated reasoning.

1 Introduction

Developing complex software systems requires to make a high number of critical decisions, which could contribute to the success of the development project or ratify its failure. Choosing the right software components is one among the most critical decisions, as it concerns the evaluation of both, the technical aspects of the components, and their possible impact on higher level strategic objectives.

More and more, companies are interested in adopting Open Source Software (OSS) components, which promise to ease the achievement of internal objectives, such as a reduction of cost and time to market, quality improvements or independence from producers. However, OSS software components also introduce various risks that are not visible at the time of the adoption, but can manifest in later development and maintenance phases, causing unexpected failures.

Inadequate risk management was identified among the top mistakes when implementing OSS-based solutions. Understanding, managing and mitigating OSS adoption risks is therefore crucial to avoid a potential adverse impact on the business, in terms of time to market, customer satisfaction, revenue, and reputation. In OSS, the available data and meta-data, in code repositories, bug trackers, and discussion forums, can be an important source to discover correlations with prospective failure, thus indicating the presence of risks. The need to properly analyse such data calls for a (semi-)automated risk analysis.

E. Yu et al. (Eds.): ER 2014, LNCS 8824, pp. 335–348, 2014.

In this paper we present a framework for risk modelling and risk evaluation, which is tailored to assess OSS adoption risks. The framework is comprised by a conceptual modelling language and a quantitative reasoning algorithm that analyses models. It relies on the capability to exploit OSS measures as possible indicators of risk, and to relate them to higher level organisational elements. Several model-based approaches to risk analysis have been presented in literature, e.g. [1,3]. In contrast to defensive risk analysis approaches such as [8], which are appropriate e.g. for security risks, our approach considers also an offensive analysis, including the assessment of alternatives.

The paper is structured as follows. In Section 2 we define the problem and introduce the concepts used in the modelling language. In Section 3, the modelling language and its semantics are defined, while Section 4 explains how risk analysis is applied upon these models. Section 5 shows an application of a tool that implements the semantics, to a small scenario. Relevant related work is given in Section 6, while Section 7 concludes the work.

2 Context and Problem

Adopting OSS components exposes the adopter to various risks, some are general, but some also specific to OSS. Although OSS components have often technical qualities comparable to those of Commercial Off-The-Shelf (COTS) components, OSS is typically provided without legal contracts and certifications that guarantee the adopter over time about the component functionalities and qualities. Part of the risks is intrinsic to the openness and distribution of the OSS communities developing the component. Such factors are often not fully perceivable at the time of the choice, but could appear in the future, causing unpredicted costs. For example, *an OSS project in which people don't participate actively, contributing with frequent code commits and online discussions, may progressively by abandoned by contributors, thus letting the component development to die and the adopters without necessary new releases*. These uncertainties represent the risks for the adopter's business objectives.

The present work starts from the intuition that the possible future evolution of the OSS project (and of the OSS component thereof) is estimable from the analysis of some characteristics of the project itself. In particular, and differently from COTS, OSS projects make freely available their data, from the source code and its metadata in repositories, to bug trackers, forums and other communication channels. If we are able to get measures of such data, we can use them to estimate not only the quality of the software but also the dynamics in the community, and consequently the various risks to which an adopter is exposed when integrating a particular OSS component. The challenge is to identify a conceptual formalism, capable to provide at the same time (i) a good representation of the OSS domain under analysis, (ii) a representation of business goals of the adopter, and (iii) the available knowledge on how measures can propagate risk quantification on business goals.

We may think to the problem of inferring knowledge from OSS component measures as a Business Intelligence (BI) problem. In our case, we need to relate something, which is *external* to a company (the OSS component), to something *internal* (the business objectives), modulo a certain amount of *uncertainty* about the relation itself (the risk).

In BI, ***indicators*** (or *risk indicators*) provide a means to measure the state of things [2,7]. For example, "Number of commits per month" is a possible measure about an OSS, captured by an indicator. States of affairs can be represented by means of ***situations*** [2,4]. A situation is expressed by means of a proposition that describes the facts that are true or false in that situation. Consequently, it can have associated a certain evidence to be true. For example, "There have been no commits since the last 30 days" and "There are bugs that need to be fixed" are propositions expressing situations. Risk indicators combine available measures (in our context, OSS code and community measures), to contribute evidence for being in a certain situation. This evidence can be obtained e.g. through an empirical data analysis, or from experts' experience.

Situations are inherently linked to the concept of ***event***. An event is a happening at a given place and time, and changes the state-of-affairs, thus it can modify a situation. Events describe a state change, also expressed through propositions. For example, "A new version is committed" and "A members leaves the community" are propositions describing an event, while "There is a lack of support in the community" describes a situation.

Such concepts help in providing primitives to model risk. The concept of ***risk*** is defined in several ways in literature. The ISO 31000:2009 standard defined risk as *an effect of uncertainty on objectives*, using the concepts of ***likelihood*** of occurrence of risky events and ***severity*** of their ***impact*** on business objectives. In this paper, for reasons of clarity, we limit to adverse, i.e. negative impacts, in line with other definitions such as NIST SP 800-30. Based on this definition of *risk* as a combined concept [11], the severity of the impact, together with the likelihood of the event, are the parameters that define ***risk exposure***, which can be expressed in terms of a discrete (e.g. *low-risk*, *high-risk*) or a continuous value. We call an event for which a risk analysis is performed ***risk event***. Certainly, not necessarily a consequence is severe enough and each event non-deterministic enough for the stakeholders to deserve a detailed risk analysis.

Situations may describe ***possible causes*** or preconditions for risk events. Examples in the security domain are the presence of a vulnerability or of an attacker. The ***consequences*** of an event can have a negative effect to the stakeholders' ***goals***, which express a desire to maintain some asset or to achieve some situation.

3 Modelling OSS Component Risks with RiskML

In this section we propose a modelling language for risk modelling, called *RiskML*, defining its concepts, the underlying metamodel, its syntax and the ascribed semantics for performing risk assessment.

3.1 Concepts and Relations

Indicator. An Indicator is an abstract representation of a measure about a certain property of the OSS [2]. The *value* of an indicator by definition ranges between 0 and 1. An indicator determines the evidence of being in a certain situation. It is ideally retrieved with tool support.

Situation. We use the concept of situation to model the state of affairs under which a certain risk is possible. A situation is *satisfied* if the state of affairs (i.e. a partial state of the world) that it represents, holds [2,14]. If ϕ is a proposition describing a situation, $sat(\phi)$ defines the satisfaction of the situation. However, we do not aim at a complete, formal description of the world. Rather, situations capture circumstances relevant for modelling risk.

Event. We use the concept of event to model a change in the state of affairs, with a potential negative impact on goals. Events may have a certain quantitative *significance* and their occurrence may happen with a certain *likelihood*. If ϕ is a proposition describing an event, $lik(\phi)$ describes the *likelihood* of the event. $sig(\phi)$ describes the *significance* of the event for a stakeholder's goals.

Goal. We use the concept of goal to model the desire of a stakeholder in obtaining or maintaining some business value, i.e. some state of affairs. Goals are *satisfied* if the corresponding state of affairs is achieved. If ϕ is a proposition describing a goal, $sat(\phi)$ describes the satisfaction of the goal.

Risk. A *Risk* is a composed concept which expresses a lack of knowledge about some happening and what could be the (negative) consequences. Accordingly, we define a risk as a tuple $\langle$ S, E, G $\rangle$, where:

- S are the situations, in which risk events potentially occur,
- E is the event whose occurrence impacts on stakeholder goals,
- G is the goal, which suffers a negative impact if the event occurs.

Relations. The defined relations link indicators, situations, events, and goals, and thus define implicitly the occurrence of a risk. We define six types of relations.

– ***Expose***, from a situation to an event: the higher the evidence that the situation is satisfied, the more likely the event is to happen (*evidence* represents the degree of confidence that a fact is true). For example, if the activeness of an OSS community appears to be too low, this may mean that in the future the community's activeness will fall completely, and the community will die. This is modelled through the *Expose* relation by saying that the situation of low community activeness exposes to the event of community death.
– ***Protect***, from a situation to an event: the higher the evidence that the situation is satisfied, the less likely the event is to happen. For example, if there seems to be large number of downloads of a given OSS software project, this may mean that the interest towards the OSS project is high, and this may mean that hardly the OSS community will be left die. This is modelled through the *Protect* relation by saying that the a high interest towards the OSS project protects from the event of community death.
– ***Increase***, from a situation to an event: the higher the evidence that the situation is satisfied, the bigger is the significance of the event consequence. For example, if a company, who wants to adopt a certain OSS software and needs to maintain its final product for a medium-long period of time, the presence of an active community behind the OSS project is considered important for the choice of adopting the OSS software. This is modelled through the *Increase* relation: if there is evidence that

the adopter is in the situation to need to maintain the product for a longer time, the significance of the consequence of a community death increases.

- ***Reduce***, from a situation to an event: the higher the evidence that the situation is satisfied, the smaller is the significance of the event consequence. For example, if a company wants to decrease its risk exposure to the possibility of community death, it may decide to implement a plugin architecture; so in its vision, if the OSS community will die, the plugin-based architecture will allow to quickly switch to a different component, thus reducing the significance of the event consequence. This is modelled through the *Reduce* relation by saying that a plugin-based architecture reduces the significance of the consequence of a community death.
- ***Impact***, from an event to a goal: the higher the impact, the higher is the *severity* of an event for the satisfaction of a stakeholder's goals. The impact thus defines the propagation the of risk exposure to goal satisfaction. For example, if a company has to provide support to its customers on its final product, if the event of community death will manifest, the company will be unable to provide further support. This is modelled through the *Impact* relation by saying that a community death impacts the goal of providing support to the final product.
- ***Indicate***, from an indicator to a situation: the higher the value of an indicator, the higher the evidence that the situation is satisfied. For example, the more recent the posts in a forum are, the more evidence we have for an active community. This is modelled through the *Indicate* relation by saying that the number of recent posts in a forum (possibly among others) indicates an active community.

Additionally, the ***Expose*** and ***Protect*** relationships can be defined from one event to another, propagating likelihood: this is a shortcut to say that the situations, resulting from event happening, expose other events or protect against them.

3.2 Meta-model

Figure 1 depicts the overall meta-model of the RiskML modelling language. The meta-model defines the modelling primitives of risk and the interplay between risk, goals and the ecosystem. Situations and events are the core of the meta-model. The evidence to be in a certain situation can be quantified (especially in an OSS context) by means of indicators, which were empirically evaluated or approved by experts and base on measurements of available data. Situations represent the causes for risks to appear, while events represent the manifestation of a risk. Expose, increase, protect and reduce relations from situations (and events) to events quantify the likelihood for an event to occur and the significance of its consequences. Events impact the satisfaction of goals that actors desire to be achieved. This impact may be propagated to other goals, e.g. through *i** contribution and decomposition relationships [16].

As a means to achieve their goals, actors perform activities, which modify the state of the world – they are the means by which actors intentionally or unintentionally modify situations. Through the convergence of goals and tasks actors can also set up strategies to mitigate risk exposure. Actors live in an ecosystem comprised by other actors [9], and depend on them to achieve some of their goals. On the other hand, actors provide to other actors the capability to achieve their goals. Thus, an impact on one actor may also propagate its effects on the rest of the ecosystem.

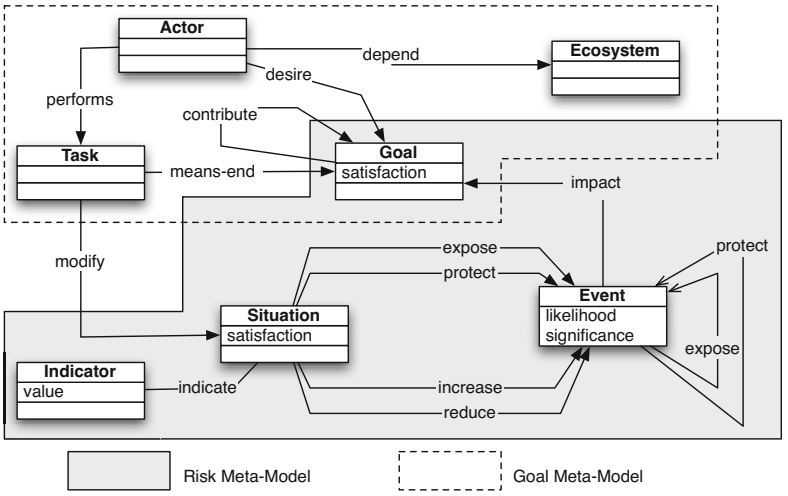


Fig. 1. Meta-model of the risk modelling language RiskML

3.3 Syntax and Semantics

Symbolic Syntax. Expressions of the language are generated via the BNF rules in Table 1. We use s for an atomic situation, e for a single event, g for an atomic goal, and i for an indicator. Expressions in the language are denoted by Greek letters. Symbols for relations are: $\wedge$ for conjunction, $\vee$ for disjunction, $\phi \xrightarrow{\text{expose}} \psi$ for Expose, $\phi \xrightarrow{\text{protect}} \psi$ for Protect, $\phi \xrightarrow{\text{increase}} \psi$ for Increase, $\phi \xrightarrow{\text{reduce}} \psi$ for Reduce, $\phi \xrightarrow{\text{impact}} \psi$ for Impact, and $\phi \xrightarrow{\text{indicate}} \psi$ for Indicate.

Semantics. A RiskML model is defined as a pair $\{\mathcal{C}; \mathcal{R}\}$, where $\mathcal{C}$ is a set of concept instances and $\mathcal{R}$ is a set of relation instances over $\mathcal{C}$. We call ϕ and ψ *propositions* describing a concept instance. If $(\phi_1, \ldots, \phi_n; op) \xrightarrow{\text{f}} \psi$ is a relation $r \in \mathcal{R}$, we call $\phi_1 \ldots \phi_n$ the *source concepts* or source propositions, *op* the logical connector of the propositions $\phi_1, \ldots, \phi_n$ and ψ the *target concept* or target proposition. We call $\alpha \in \mathcal{A}$ an attribute of a concept instance, defined on a set $\mathcal{A}$ of attributes. Each relation r

Table 1. Syntax of RiskML models

(1)	$S := s \mid S \wedge s \mid S \vee s$
(2)	$E := e \mid E \wedge e \mid E \vee e$
(3)	$G := g \mid G \wedge g \mid G \vee g$
(4)	$I := i$
(5)	$R := (S, E, G)$
(6)	$\phi := S \mid E \mid G, \psi := S \mid E \mid G$
(7)	$\alpha(\phi) := sat(\phi) \mid lik(\phi) \mid sig(\phi)$
(8)	$Rel := \phi \xrightarrow{\text{expose}} \psi \mid \phi \xrightarrow{\text{protect}} \psi \mid \phi \xrightarrow{\text{increase}} \psi \mid \phi \xrightarrow{\text{reduce}} \psi \mid \phi \xrightarrow{\text{impact}} \psi \mid \phi \xrightarrow{\text{indicate}} \psi$

modifies the attribute value $\alpha(\psi)$ of the target concept, according to the attribute values of the source concepts and the function f which defines a type of relation.

If S is a proposition describing a situation, $sat(S)$ is an attribute of S describing the fact that the situation is satisfied or not satisfied. If E describes an event, $lik(E)$ describes its *likelihood* and $sig(E)$ describes its *significance*. If G is a proposition describing a goal, $sat(G)$ describes the *satisfaction* of this goal.

Table 2 summarises the semantics of a RiskML model. To evaluate how risky an event is, we calculate the exposure $exp(E)$ to its occurrence through a function f – the *risk aversion* function – of its likelihood to occur and the significance of its occurrence (A1). For sake of simplicity, the function is currently implemented as the average of the two values. The relations A2 to A7 propagate values across a risk model with the following meaning:

- Expose: evidence of satisfaction of the source Situation (S) is positively propagated to the target Event's (E) likelihood.
- Protect: evidence of satisfaction of the source Situation (S) is negatively propagated to the target Event's (E) likelihood.
- Increase: evidence of satisfaction of the source Situation (S) is positively propagated to the target Event's (E) significance.
- Reduce: evidence of satisfaction of the source Situation (S) is negatively propagated to the target Event's (E) significance.
- Impact: exposure (combining likelihood and significance) of the source risk event (E) is negatively propagated to the target Goal's (G) satisfaction.
- Indicate: the source Indicator (I) value $\mu(I)$, defined as an assignment of values (coming from a set of available data M, obtained, for example, from measurements, or expert opinions) is propagated to the evidence of being satisfied for the target situation S.

A8 states the general propagation rule in a risk model: given a node of the model ψ and a set of relations entering into the node $\alpha_1(\Phi_1) \xrightarrow{w_1} \beta(\psi), ..., \alpha_n(\Phi_n) \xrightarrow{w_n} \beta(\psi)$, the value propagated to $\beta(\psi)$ is the maximum among the values propagated by the single relations. Each relation can have multiple source nodes and one target node (A9). Source nodes can be put in AND- or OR- connection. In the first case, what is propagated is the product of the source values (attenuation), since propagation takes place only if both are present; in the second case, the difference between sum and product of source nodes (reinforcement), resembling a propagation of probabilities as used in [5].

3.4 Example

Figure 2 depicts a sample RiskML model. In this model, two indicators, [commit frequency] and [current/past commit frequency ratio], are used to capture available measures of the OSS project. The indicators are used to detect the condition of a low activeness in the community, represented by means of the [Low activeness] situation. Situations are the highest level representation of what is observed to be true or false at time t_0. At any time t_x, with $x > 0$ (i.e., at any time in the future), some events are possible to happen, such as [the community will die]. The *expose* relations inform that the more

Table 2. Axioms and propagation rules for RiskML models

Axioms

A1	$exp(E) = f(lik(E), sig(E))$

Relations

	Relation	Definition
A2	$S \xrightarrow{\text{expose}} E$	$sat(S) \xrightarrow{+} lik(E)$
A3	$S \xrightarrow{\text{protect}} E$	$sat(S) \xrightarrow{-} lik(E)$
A4	$S \xrightarrow{\text{increase}} E$	$sat(S) \xrightarrow{+} sig(E)$
A5	$S \xrightarrow{\text{reduce}} E$	$sat(S) \xrightarrow{-} sig(E)$
A6	$E \xrightarrow{\text{impact}} G$	$exp(E) \xrightarrow{-} sat(G)$
A7	$I \xrightarrow{\text{indicate}} S$	$\mu(I) \xrightarrow{+} sat(S)$

Rules

	Rule	Definition
A8	$\alpha_1(\Phi_1) \xrightarrow{w_1} \beta(\psi), ..., \alpha_n(\Phi_n) \xrightarrow{w_n} \beta(\psi)$	$\beta(\psi) = max(w_1 * \alpha(\Phi_1), ..., w_n * \alpha(\Phi_n))$
A9	$\alpha(\Phi) \xrightarrow{w} \beta(\psi)$	$\alpha(op(\phi_1, ..., \phi_n)) \xrightarrow{w} \beta(\psi)$
A10	$(and)\ op = \wedge$	$\alpha(\Phi) = \alpha(\phi_1) + ... + \alpha(\phi_n)/n$
A11	$(or)\ op = \vee$	$\alpha(\Phi) = 1 - (\alpha(\phi_1) + ... + \alpha(\phi_n))/n$

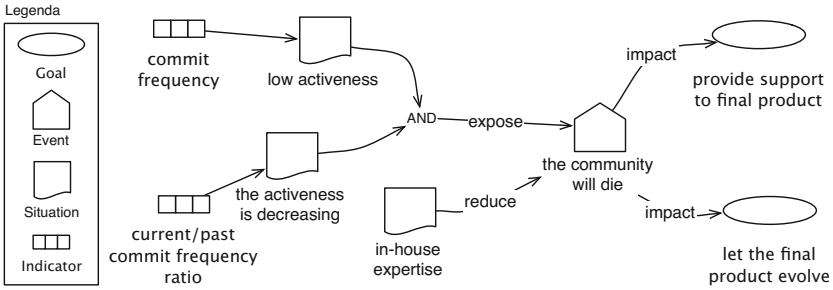
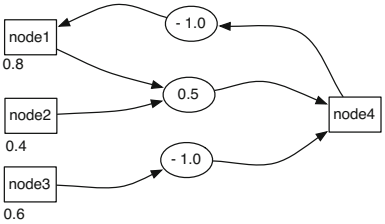


Fig. 2. Example of RiskML Risk Exposure Model

there is evidence that the community is low active, the higher is the likelihood of a future community death. Notice that in this paper we omit reasoning on the quantification of weights and assume a default weight of 0.5, if not differently specified. Also, if not differently specified, events have by default a significance of 0.5. In the example, the [In-house expertise] situation models the organisational fact that skilled programmers are available to the adopter, and this lowers the significance of a possible community death. If the events occurs, the *impact* relations show the affected business goals.

4 Reasoning with Risk Models

RiskML models are meant to be used to support automated reasoning for risk assessment. To do this we assign *evidence* values to known elements of the models (typically the indicators) and apply a label propagation algorithm to infer knowledge about unknown elements (typically the goals). An *evidence* is a numerical quantification of a certain truth value, expressed as a real number in the range [0..1]. An evidence represents the degree of confidence that the truth value holds, or does not hold. For example,



Output		
	sat	den
node1	0.8	0.4
node2	0.4	0.0
node3	0.0	0.6
node4	0.4	0.6

Fig. 3. The label propagation technique

with reference to "timeliness" (defined as the capability of a certain OSS community to deliver new releases according to its roadmap timetable), we may express as having evidence 1 is that the releases are always delivered on the announced day.

Positive vs. Negative Evidence. To each node two evidence values are associated: a value for satisfaction evidence, and a value for denial evidence. For example, for a node "The door is locked", a satisfaction evidence indicates that there is evidence that the door is locked; a denial evidence indicates that there is evidence that the door is not locked. Both values can hold at the same time, indicating that there is some positive evidence but also some negative evidence about the node. For example, a closed door with a key inserted into the keyhole may highlight that the door is locked (because the door is actually closed) but at the same time there is a possibility that the door is not locked (because of the presence of the key).

Model Stability. Separating satisfaction from denial evidence allows us to ensure monotonicity of label propagation algorithms. In turn, this allows us to manage cycles by calculating only stable models. Figure 3 illustrates how this technique works. `node1` has an input value of +0.8; `node2` has an input value of +0.4; `node3` has an input value of +0.6; `node1` and `node2` propagate positively to (reinforce) `node4` with a weight of 0.5. `node3` propagates negatively to (inhibits) `node4` by -1.0. `node4` results in a satisfaction evidence of 0.4 (= max(0.8 * 0.5, 0.4 * 0.5)). Additionally, it has a denial evidence of 0.6 due to `node3`. `node4`, in turn, inhibits `node1`, which also assumes a denial evidence of 0.4. This value is not further propagated because, although it is in relation with `node4`, the denial evidence of `node4` is 0.6, which is greater than 0.4. This example explains how the monotonicity achieved by means of decoupling the satisfaction from the denial evidence allows us to solve cycles.

Propagation Algorithm. Label propagation [12] is a forward reasoning techniques applied to graph analysis. Starting from the knowledge about some known nodes of the graph, it has the objective of inferring knowledge about unknown nodes. Label propagation has been successfully applied in [6] to support forward reasoning in goal analysis. We adapt their approach to our purposes, generalising it and supporting risk assessment. Label propagation is implemented using a forward quantitative reasoning algorithm, listed in Algorithm 1. The algorithm iterates over every node of the label graph, and evaluates the incoming propagation arcs. If the value(s) of the node is lower that the values of the source nodes, weighted by the arc's weight, the value(s) of the node is

Algorithm 1: Label propagation algorithm.

```
Data: Graph G(C, R)
Result: Labels propagated in G
begin
    boolean graph_changed = false;
    repeat
        for (c in C) do
            double max_positive = 0;
            double max_negative = 0;
            for (r in R) do
                double positive_val = r.evalPositivePropagation();
                double negative_val = r.evalNegativePropagation();
                if ( positive_val > max_positive ) then
                    c.setPositiveVal(r.attribute, positive_val);
                    max_positive = positive_val;
                    graph_changed = true;
                if ( negative_val > max_negative ) then
                    c.setNegativeVal(r.attribute, negative_val);
                    max_negative = negative_val;
                    graph_changed = true;
    until ( graph_changed == false );
```

updated. When a complete iteration over the graph nodes is completed, without having done any propagation, the algorithm terminates. Because of the monotonicity property, the algorithm always terminates.

5 Application

We validated our approach by taking advantage from the experience of the industrial partners of the RISCOSS project (`www.riscoss.eu`), in which the present work is performed.

Scenario. A typical software company addressed in our project (referred to as "the adopter", from now on) produces software ("the final product") for end users, by using third party OSS components. While choosing a component to adopt, it seeks to fulfil requirements, which may come from both itself and its clients, and concern the business as well as the technical needs. For example, an adopter produces an end-user accounting software, which allows accountants and other employees together to edit online reports and publish them to the public audience. Once deployed, it is planned to run for a long time without being replaced. Thus, it needs to be possible to maintain it and to let it continuously evolve without having to replace it. Moreover, the effort of integration with the adopter's accounting software should be minimised. So the adopter is searching for a Wiki OSS component with the required characteristics. In such a setting, technical qualities of available OSS components are easily evaluable by using existing tools. Having done this for our validation scenario, three candidate Wikis have been identified as having the desired technical properties, namely: (i) Mediawiki; (ii) Dokuwiki; and (iii) XWiki. To choose the right one, we need to analyse their risks.

Risk Model. Figure 4 depicts an excerpt of a RiskML model for the Wiki case. The model represents the structural knowledge available to the adopter about possible events,

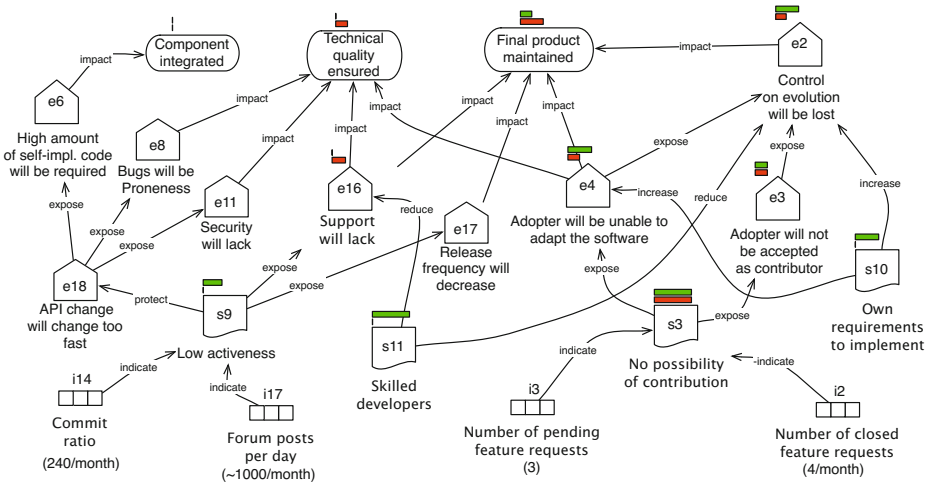


Fig. 4. A graphical representation of a RiskML model for the XWiki component, annotated with the analysis results

the conditions of their occurrence, and their possible impact on the business goals. It is worth noticing that the topology of the model is not fixed. Part of the ongoing project consists in gathering the information to build the most correct model, but in any case the model or some of its parts can be customised but the adopter. The depicted **goals** have been identified by means of an overview on large corporates and small enterprises. The model depicts in particular the goals that have been selected for the described scenario: [Component integrated], [Technical quality ensured], and [Final product maintained]. To goals are associated the risk **events** that impact on them. Events have been selected after a systematic literature review and an analysis of several industrial case studies; e.g. for the goal [Final product maintained] the risk events [Release frequency will decrease], [Adopter will be unable to adapt the software], and [Control on evolution will be lost] and [Support will lack] are modelled. The defined **situations** influence event likelihood ([Low activeness]) or significance ([Skilled developers]). Identified **indicators** mainly refer to code metrics and mailing lists statistics; e.g. the two indicators [Commit ratio] and [Forum posts per day] give evidence to the situation [Low activeness]. The measures defined by the indicators can be fed manually by the user or automatically from existing repositories. The indicator numbers are translated into evidence degrees by means of a normalising function. It is worth noticing that the topology of the model is not fixed. Part of the ongoing project consists in gathering the information to build the most correct model, but in any case the model or some of its parts can be customised but the adopter.

Results. On the models we applied the described label propagation algorithm, which has been implemented in a reasoning tool developed in Java. The tool takes as input a risk model, the functions and weights, and the input evidences, and produces as output the risk evidences. The results of the analysis on a single component, XWiki, are illustrated in Fig. 4. In the model, we see for example that the situation s3 receives

Table 3. Risk evidence for the three OSS components under analysis; notice that the reported values do not imply probabilities and do not have a meaning as standalone values: they are only intended to be used for comparison.

	Xwiki	MediaWiki	DokuWiki
Component integrated	0 / 0	0 / 0	0 / 0.2
Technical quality ensured	0 / 0.28	0 / 0.4	0 / 0.2
Final product maintained	0.38 / 0.5	0.3 / 0.4	0.1 / 0.6
Control on evolution will be lost	0.5 / 0.38	0.4 / 0.25	0.6 / 0.2
Adopter will be unable to adapt the software	0.5 / 0.38	0.4 / 0.3	0.6 / 0.1
…			

strong positive as well as negative evidence from the two indicators i2 and i3. In turn, the situation raises the likelihood of events e3 and e4. However, e3 for example does not present high risk exposure levels because its significance is not increased by explicit factors (there are no entering *increase* relations). Viceversa, e2 is likely to happen and, due to the need represented by s10, [Own requirements to implement], it presents a rather high risk exposure level. Finally, this exposure level impact on the [Final product maintained] goal, which is at danger of not being satisfied (red bar).

Table 3 shows how to use analysis results, by reporting a comparison among the results of the three components. On the first column are reported the goals and the impacting events. On the right part of the table, the impact values on goals are reported for the three candidate OSS components. Worth noticing that the values are intended to be used only for comparing the components with each other, so they can not be interpreted, for example, as probabilities. For each component, and for each element (goal/event), both the positive and negative values for satisfaction (or exposure) are reported. This allows us to make finer-grained comparisons among the components, in both their strength and weakness points. For example, we see that, with respect to the [Final product maintained] goal, MediaWiki seems to have less evidence of satisfaction than XWiki (0.3 vs. 0.38) but at the same time less evidence of not satisfaction (0.4 vs 0.5). Going in the details of each associated event allows to have an idea of *why* these numbers are produced, and consequently to make a better decision.

6 Related Work

The Goal-Risk framework [1] is a goal-oriented method based on *i** [16] that aims at capturing, analysing and assessing risk at an early stage of the requirements engineering process. A Goal-Risk model consists of nodes, relations and a set of impact relations. The nodes can be characterised by two properties: *Sat* and *Den*, indicating the evidence that the node is satisfied/fulfilled/present or denied/failed/absent, respectively. The model is comprised by an *asset layer*, modelling anything that has value to an organization (such as goals); an *event layer*, used to model phenomena likelihood through satisfiability/deniability evidences and severity, which indicates their capability to prevent goal achievement; a *treatment layer*, containing the countermeasures set up to mitigate the risks. KAOS [15] goal-oriented analysis methodology deals with risk management by complementing goal modelling and analysis with obstacle analysis that

consists in identifying and modelling the adverse conditions that may prevent a goal to be achieved [3]. KAOS has a formal representation of the goal model, relies on strictly measurable variables and takes into account partial or probabilistic values for goal satisfaction, but does not integrate concrete measures and indicators. Adverse conditions are identified and modelled; then risk assessment is performed to reduce the likelihood or the severity of each obstacle, and mitigation is performed by applying countermeasure patterns in the goal model. EKD [13] is a methodology for mapping organizational processes that can serve as a basis for identification risk management tasks. It provides a controlled way of analysing and documenting an enterprise, its objectives and support systems and allow for the specification of enterprise knowledge models. During the model developing, the participant parties engage in tasks that involve deliberation and reasoning. The objective is to provide a clear, unambiguous picture of how enterprise processes function currently in an "as is" model or in a modified "should be" model. EKD proposes to exploit the concept of patterns in order to recognise risks and revise the practices in order to mitigate them. Finally, CORAS [8] is a model-based approach based on UML for performing security risk analysis and assessment based on security analysis techniques, CORAS is comprised by three different components: a risk modelling language; a method, for the security analysis process; and a tool for documenting, maintaining and reporting risk analysis results. It limits to a defensive risk analysis to protect company assets. The method does not rely on a particular reasoning technique but is flexible to be supported by several techniques such as graph label propagation and logic based techniques. The problem of evaluating the qualities and characteristics of OSS software components and development processes via metrics and statistical analysis for decision-making purposes has been considered in several projects. For example, the QualOSS project[1] had the purpose of defining a method to assess the quality of OSS projects, via a quality model composed of three types of interrelated elements: *quality characteristics* of a product or community, *metrics*, concrete aspects that can be measured, and *indicators* that define how to aggregate and evaluate the measurement values to obtain consolidated information. The recently started OSSMETER[2] project aims to develop a platform that will support decision makers in the process of discovering, assessing and monitoring the health, quality and activity of OSS.

7 Conclusion

In this paper we presented a modelling framework for assessing risk in open source software (OSS) components adoption. The framework relies on a language, tailored to describe available risk indicators, risk events, and their impact on business goals. A label propagation algorithm has been used to infer risk exposure values, starting form available indicators. Also, working risk models have been produced to asses risk in a controlled scenario. We acknowledge that the reliability of the framework in a production environment ultimately depends on the quality of the risk models. Currently, the model is instantiated with risks and indicators retrieved from a systematic literature

[1] http://www.qualoss.eu/
[2] http://www.ossmeter.eu

review and an analysis of several industrial case studies [10]. More work is ongoing in this direction.

Acknowledgement. This work is a result of the RISCOSS project, funded by the EC 7th Framework Programme FP7/2007-2013, agreement number 318249.

References

1. Asnar, Y., Giorgini, P., Mylopoulos, J.: Goal-driven risk assessment in requirements engineering. Requir. Eng. 16(2), 101–116 (2011)
2. Barone, D., Jiang, L., Amyot, D., Mylopoulos, J.: Reasoning with key performance indicators. In: Johannesson, P., Krogstie, J., Opdahl, A.L. (eds.) A Calculus of Communication Systems. LNBIP, vol. 92, pp. 82–96. Springer, Heidelberg (1980)
3. Cailliau, A., van Lamsweerde, A.: Assessing requirements-related risks through probabilistic goals and obstacles. Requir. Eng. 18(2), 129–146 (2013)
4. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A.: Sweetening wordnet with dolce. AI Magazine 24(3), 13–24 (2003)
5. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 167–181. Springer, Heidelberg (2002)
6. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Formal reasoning techniques for goal models. J. Data Semantics 1, 1–20 (2003)
7. Kenett, R.S., Zacks, S.: Modern Industrial Statistics: with applications in R, MINITAB and JMP, 2nd edn. John Wiley and Sons (2014) With contributions by D. Amberti
8. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis - The CORAS Approach. Springer (2011)
9. Messerschmitt, D.G., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. The MIT Press (2003)
10. Morandini, M., Siena, A., Susi, A.: Systematic literature review: Risks in oss adoption. Technical report, FBK, Trento (2013), `http://selab.fbk.eu/riscoss_ontology/riskSLR.html`
11. Morandini, M., Siena, A., Susi, A.: A context-specific definition of risk for enterprise-level decision making. In: The 8th International Workshop on Value Modeling and Business Ontology (2014)
12. Nilsson, N.J.: Problem-solving Methods in Artificial Intelligence. McGraw-Hill, New York (1971)
13. Rolland, C., Nurcan, S., Grosz, G.: Enterprise knowledge development: the process view. Information & Management 36(3), 165–184 (1999)
14. Siena, A., Jureta, I., Ingolfo, S., Susi, A., Perini, A., Mylopoulos, J.: Capturing Variability of Law with *Nómos* 2. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012 Main Conference 2012. LNCS, vol. 7532, pp. 383–396. Springer, Heidelberg (2012)
15. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. IEEE Trans. Software Eng. 26(10), 978–1005 (2000)
16. Yu, E.S.-K.: Modelling strategic relationships for process reengineering. PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada (1996)