Int J Software Informatics, Volume 6, Issue 3 (2012), pp. 419–434
International Journal of Software and Informatics, ISSN 1673-7288
©2012 by ISCAS. All rights reserved.
E-mail: ijsi@iscas.ac.cn
http://www.ijsi.org
Tel: +86-10-62661040

Measuring Software Requirements Evolution Caused by Inconsistency

Kedian Mu[1], Zhi Jin[2,3], and Ruqian Lu[4]

[1] (School of Mathematical Sciences, Peking University, Beijing 100871, P.R.China)

[2] (Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, P.R.China)

[3] (School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P.R. China)

[4] (Academy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing 100190, P.R.China)

Email: mukedian@math.pku.edu.cn, zhijin@sei.pku.edu.cn, rqlu@math.ac.cn

Abstract It has been widely recognized that requirements evolution is unavoidable in any sizeable software project. Moreover, if the requirement evolution is not managed properly, it may result in many troublesome problems during the process of software development. For example, poor management of requirements evolution may lead to inconsistencies in requirements and incomparability between requirements and other work products. Repairing these problems can lead to extra consumption of development resources. However, inconsistency is considered as one of the concerns of requirements evolution. In this paper, we propose a family of logic-based measures to evaluating software requirements evolution caused by inconsistency handling. Each of these measurements provides a distinctive perspective of quantitative description for the requirements evolution. At first, we provide a syntax-based measure for the change in requirements statements during the requirements evolution. Then we provide a semantics-based approach to measuring the change in the expression ability of requirements specification during the process of evolution. Finally, we characterize three special kinds of requirements evolution based on these measurements, including the evolved requirements specification with minimal change, the evolved requirements specification with minimal significance change, and the evolved requirements specification with maximal plausibility.

Key words: requirements evolution; inconsistency; requirements change; measurements

Mu KD, Jin Z, Lu RQ. Measuring software requirements evolution caused by inconsistency. *Int J Software Informatics*, **Vol.6, No.3 (2012): 419–434. http://www.ijsi.org/1673-7288/6/i135.htm**

1 Introduction

This work is sponsored by the National Natural Science Foundation of China under Grant No. 61170300, the National Basic Research 973 Program of China under Grant No. 2009CB320701, and the Key Project of National Natural Science Foundation of China under Grant No. 90818026.
Corresponding author: Kedian Mu, Email: mukedian@math.pku.edu.cn
Received 2011-11-14; Revised 2012-05-04; Accepted 2012-08-21; Published online 2012-09-25.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

It has been widely recognized that requirements evolution is inevitable for any proposed software development project. Stakeholders change their minds for many reasons, including commercial strategies updating, marketplace changes, policies and legislation changes, identifying a defect in proposed requirements, and missing some requirements. Suitable requirements changes may boost satisfactions of some stakeholders to the system-to-be and enhance the quality of software requirements specification and subsequent artifacts during the software development life cycle.

But if the requirement evolution is not managed properly, it may result in many troublesome problems during the process of software development. For example, poor management of requirements evolution may lead to incomparability between requirements and other work products. Repairing these problems can lead to extra consumption of development resources. Moreover, uncontrolled consumption of development resources such as time, funds, and human resources during requirements evolution may lead to delay in delivery of the software product, overburden on developers, difficulties in funds, and some other undetectable defects that can degrade the quality of the software product[1]. However, it has been also recognized that managing requirements evolution is often difficult for a big software project in practice[2].

Inconsistency is considered as one of the concerns of requirements evolution. As a life cycle wide process, requirements evolution must make sure that the whole software requirements specification keeps consistent, and the other work products such as design decisions and test cases accord with the requirements specification[2]. On the other hand, inconsistency has been considered as a main class of defects in requirements specifications[3]. Moreover, it has been widely recognized that inconsistencies in a requirements specification is a signal that the requirements specification should be changed[4]. Therefore, inconsistency implies requirements evolution. That is, resolving inconsistency in a requirements specification will lead to the evolution of requirements.

In this paper, we focus on the requirements evolution caused by resolving inconsistency in requirements. However, resolving inconsistency in requirements is a hard but important issue in requirements engineering[4]. One of the most important aspects for choosing appropriate actions for resolving an inconsistency is that developers need to assess the impact the actions will have on the development project and to assess the risk of resolving the inconsistency by taking the actions[5]. This makes measuring requirements evolution more necessary during the process of inconsistency handling.

With regard to inconsistency in requirements engineering, it has been increasingly recognized that it is effective to use logics to formulate management of inconsistent requirements specifications[3]. Various logic-based approaches to handling inconsistencies in requirements specications have recently been proposed[1,3−4,6−7]. Roughly speaking, these logic-based approaches formulate a software requirements specification as a knowledge base (i.e., a set of logical formulas) within the context of some appropriate logic. Then the problem of managing inconsistent software requirements can be transformed into the problem of inconsistency handling in a knowledge base. Along this line, the problem of requirements evolution may be formulated as a problem of knowledge bases evolution within some logic framework.

To address these issues, in this paper, we propose a logic-based approach to

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

measuring requirements evolution caused by inconsistency. Informally speaking, we formulate requirements specifications as knowledge bases within an appropriate logic framework. Then we propose two kinds of measures to capture the evolution from an inconsistent requirements specification to a consistent requirements specification by resolving inconsistency in that specification. Following this, we characterize some special kinds of requirements evolution processes based on these measurements. Finally, we use a small but explanatory example to illustrate our approach.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to the logical representation of requirements. We propose approaches to measuring requirements evolution caused by inconsistency in Section 3. We use a small but explanatory example to illustrate application of our approach in Section 4. We compare our approach with related work in Section 5. We conclude the paper in Section 6.

2 Preliminaries

We use classical logic-based language to represent requirements in this paper. First order logic may be considered as a promising tool to represent requirements, since most tools and notations for representing requirements could be translated into formulas of first order logic[4]. Moreover, in a logic-based framework for representing requirements, consistency checking is always associated with certain scenarios with regard to the requirements specification[4], or some specific domain knowledge. That is, we must add further relevant facts (e.g., domain knowledge) to model each scenario. Then reasoning about requirements is always based on these certain facts. It implies that checking the consistency of requirements considers only ground formulas. Furthermore, if we assume a universally quantified formula is just an abbreviation for the conjunction of formulas that can be formed by systematically instantiating the variables of the quantified formula with the constants in the language, then we may restrict the first order language to the propositional case[4]. It will render consistency checking decidable. This gives some computational advantages. However, restricting first order logic to propositional logic in some way is a useful and practical way of balancing the computational advantages of propositional logic against its limited expressive power in requirements engineering as well as software engineering[3]. For these reasons, we assume a classical first order language without function symbols and existential quantifiers. This classical first order logic is the most convenient to illustrate our approach, as will be shown in the rest of the paper.

Let $\mathscr{P}$ be a set of predicate symbols, $\mathscr{V}$ be a set of variable symbols, and $\mathscr{C}$ a set of constant symbols. We call $\mathscr{A} = \{p(q_1, \cdots, q_n) | p \in \mathscr{P} \text{ and } q_1, \cdots, q_n \in \mathscr{V} \cup \mathscr{C}\}$ the set of atoms. Let $\mathscr{F}$ be the set of classical formulas formed from a set of atoms $\mathscr{A}$ and logical connectives $\{\vee, \wedge, \neg, \rightarrow\}$. In particular, we call $p(q_1, \cdots, q_n)$ a ground atom if and only if $q_1, \cdots, q_n$ are all constant symbols. Let $\mathscr{A}_0$ be a set of ground atoms. Let $\mathscr{F}_0$ be the set of classical formulas formed from a set of atoms $\mathscr{A}_0$ and logical connectives $\{\vee, \wedge, \neg, \rightarrow\}$. Let $\mathscr{G}$ be the set of formulas formed from $\mathscr{F}$, where if $\alpha \in \mathscr{F}$, and $X_1, \cdots, X_n$ are the free variables of α, then $\forall X_1, \cdots, \forall X_n \alpha \in \mathscr{G}$. Essentially, the set $\mathscr{G}$ contains only universally quantified formulas (in which the quantifiers are outermost) and ground formulas[4].

A classical knowledge base K is a finite set of formulas in $\mathscr{F}_0$. K is inconsistent if there is a formula α in $\mathscr{F}_0$ such that $K \vdash \alpha$ and $K \vdash \neg\alpha$. We abbreviate $\alpha \wedge \neg\alpha$ as $\bot$

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

if there is no confusion. Then an inconsistent knowledge base K is denoted by $K \vdash \bot$. Moreover, an inconsistent knowledge base K is called a *minimal inconsistent set* if none of its proper subset is inconsistent. If $K' \subseteq K$ and K' is a minimal inconsistent set, then we call K' a *minimal inconsistent subset* of K.

Let $\mathsf{MI}(K)$ be the set of all the minimal inconsistent subsets of K, i.e.,

$$\mathsf{MI}(K) = \{K' \subseteq K | K' \vdash \bot \text{ and } \forall K'' \subset K', K'' \nvdash \bot\}.$$

The minimal inconsistent subsets can be considered as the purest form of inconsistency for conflict resolution where the syntactic representation of the information is important, since removing one formula from each minimal inconsistent subset would be sufficient to resolve the inconsistency[8]. In contrast, *a free formula* of a knowledge base K is referred to as a formula of K that does not belong to any minimal inconsistent subset of K. In this paper, we use $\mathsf{FREE}(K)$ to denote the set of free formulas of K.

Example 2.1. Consider $K_1 = \{a, \neg a, \neg a \vee b, \neg b, c\}$. Then $\mathsf{MI}(K_1) = \{\{a, \neg a\}, \{a, \neg a \vee b, \neg b\}\}$ and $\mathsf{FREE}(K_1) = \{c\}$.

A consistent subset of K is called a maximal consistent subset of K if there is no consistent subset of K that can subsume it. Let $\mathsf{MC}(K)$ be the set of all the maximal consistent subsets of K, i.e.,

$$\mathsf{MC}(K) = \{\emptyset \subset K' \subseteq K | K' \nvdash \bot \text{ and } \forall K'' \supset K', K'' \vdash \bot\}.$$

The maximal consistent subsets of a knowledge base can be considered as plausible perspectives of the base.

Example 2.2. Consider $K_2 = \{a, \neg a, b\}$. Then $\mathsf{MC}(K_2) = \{\{a, b\}, \{\neg a, b\}\}$.

We can use formulas in $\mathscr{G}$ to formulate requirements expressed in natural language. For example, we can represent a requirement,“ *if an authorized user requests to borrow a book and the book is available, then the user can borrow the book*”, as

$$\forall \text{User} \forall \text{Book } (\text{auth}(\text{User}) \wedge \text{requ}(\text{User}, \text{Book}) \wedge \text{avai}(\text{Book}) \rightarrow \text{borr}(\text{User}, \text{Book})).$$

However, to check inconsistency of requirements collections, the universally quantified formulas are always instantiated by the constants in certain scenarios. For example, given the following facts: “*Alice is an authorized user, and she applies to borrow the book of software engineering* ; *The book of software engineering is available*”. Then we use the following ground formula as a substitute for the universally quantified formula above:

$$\text{auth}(\text{Alice}) \wedge \text{requ}(\text{Alice}, \text{Soft\_eng}) \wedge \text{avai}(\text{Soft\_eng}) \rightarrow \text{borr}(\text{Alice}, \text{Soft\_eng})$$

Generally, if ground formulas $\alpha_1, \alpha_2, \cdots, \alpha_n$ are the instantiations of the universally quantified formula α by using different facts in a scenario, then we may use $\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$ as a substitute for α in the scenario. Thus, we concentrate on the instantiated requirements in the rest of this paper. That is, we assume that an individual set of requirements can be formulated by a classical knowledge base. With this, we restrict the first order logical representation of requirements to the propositional case.

In particular, we call a knowledge base R a (partial) requirements specification if each formula of R represents a requirement. If there is no confusion we make no

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

distinction between a classical knowledge base and a requirements specification in the rest of this paper.

3 Measuring the Evolution of Inconsistent Requirements

From a syntax sensitive perspective of inconsistency handling, each minimal inconsistent subset of a requirements specification can be considered as a potential requirements sets to be changed for restoring consistency. In contrast, each maximal consistent subset of that requirements specification may be considered as a plausible perspective that describing the stakeholders' demands. We assume that evolving requirements driven by resolving inconsistency in a requirements specification is performed by revising some requirements (at least one requirement) in each minimal inconsistent subset of that requirements specification. Therefore the evolution of requirements driven by inconsistency handling is exactly a process of enlarging or contracting the consistent perspective of the original requirements specification. Just for simplicity of discussion, we assume that there is at least one maximal consistent subset for any requirement specification in this paper, i.e., $\mathsf{MC}(R) \neq \emptyset$ for each R.

Definition 3.1. Let R be a requirements specification. Then the requirements specification R^c is called a requirements specification evolved from R by inconsistency handling, if

(a1) $R^c \nvdash \bot$;

(a2) $\mathsf{FREE}(R) \subseteq R^c$;

(a3) $\forall r \in R \setminus R^c$, there exists $M \in \mathsf{MI}(R)$ such that $r \in M$.

Obviously, if R is consistent, then $R^c = R$. Note that (a1) states that R^c should be a consistent requirements specification. (a2) makes sure that the requirements free from inconsistency should not be changed during the process of evolution. In contrast, (a3) states that only requirements in minimal inconsistent subsets are considered as requirements to be changed for resolving inconsistency in R. However, how to change the chosen requirements is a context-sensitive issue. For example, deletion, weakening, and splitting are considered as three actions for the stepwise inconsistency resolution process[10].

Example 3.1. Consider $R_1 = \{a, \neg a, b\}$. R_1 is inconsistent because $R_1 \vdash a$ and $R_1 \vdash \neg a$. Then

$$\mathsf{MI}(R_1) = \{\{a, \neg a\}\}, \text{ and } \mathsf{FREE}(R_1) = \{b\}.$$

Evidently, each of the following requirements specifications can be considered as a requirements specification evolved from R_1:

- $\{b\}$, $\{a, b\}$, $\{\neg a, b\}$;
- $\{a, \neg a \vee c, b\}$, $\{a \vee c, \neg a, b\}$;
- $\{a \vee c, \neg a \vee d, b\}$.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

But neither of $\{a\}$ and $\{\neg a\}$ is the requirements specification evolved from R_1.

The evolution of requirements due to inconsistency handling is essentially a process of enlarging some consistent perspectives reflected by the original one in some sense. To address this, we focus on assessing how far the evolved requirements specification is from maximal consistent subsets of the specification. Note that any two maximal consistent subsets of a knowledge base may convey different information reflected by the base. This makes more necessary to consider only the closest maximal inconsistent subsets to the base in the case of knowledge bases merging or belief revision[16]. However, in requirements engineering, how to revise requirements for inconsistency resolving is rather a context sensitive issue. It is not always that the requirements of one of the closest maximal consistent subsets remain unchanged. Any maximal consistent subset has a chance to be enlarged during the evolution. So, we are concentrated on the problem of an evolved requirements specification is closest to which maximal consistent subset and how far from the subset the evolved specification is. That is, we are more interested in the distance between the evolved specification and the maximal consistent subsets rather than that between the original specification and the maximal consistent subsets. This distinguishes the role of maximal consistent subsets in our measures below from that in knowledge base merging and belief changes such as Ref. [16].

3.1 Syntax-Based approach to measuring evolution

Intuitively, the evolution can be captured by comparing the maximal consistent perspectives of the original requirements and the revised requirements. Then we can define a drastic measure for the degree of evolution as follows:

Definition 3.2 (Degree of Evolution DE). Let R be a requirements specification and R^c the consistent evolution of R. Then the degree of evolution R^c w.r.t. R, denoted $DE(R^c|R)$, is defined as

$$DE(R^c|R) = \min_{R' \in \mathsf{MC}(R)} \frac{|R^c|}{|R'|}.$$

Note that $DE(R^c|R)$ captures the degree of enlargement of the maximal consistent perspective of R when R evolves to R^c. Evidently, if R is consistent, then $R^c = R$ and

$$DE(R^c|R) = \frac{|R^c|}{|R|} = 1.$$

Generally, if $DE(R^c|R) > 1$, then the requirements specification evolved from R provides a bigger consistent perspective of requirements about the system-to-be. In contrast, if $DE(R^c|R) \leqslant 1$, then the perspective reflected by the revised requirements is not bigger than the maximal plausible perspective reflected by the original requirements.

Example 3.2. Consider $R_1 = \{a, \neg a, b\}$ again. Then

- $DE(\{b\}|R_1) = \frac{1}{2}$,
- $DE(\{a, b\}|R_1) = DE(\{\neg a, b\}|R_1) = 1$,
- $DE(\{a, \neg a \vee c, b\}|R_1) = DE(\{a \vee c, \neg a, b\}|R_1) = \frac{3}{2}$,

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

- $DE(\{a \vee c, \neg a \vee d, b\}|R_1) = \frac{3}{2}$.

Note that $\{b\}$ is a revised result by deleting the two requirements involved in minimal inconsistent subsets of R_1. Only the requirement b remains. Actually, it is the smallest requirements specification evolved from R_1. Both $\{a, b\}$ and $\{\neg a, b\}$ are the revised results by deleting as few as possible requirements involved in minimal inconsistent subsets of R_1. In contrast, $\{a, \neg a \vee c, b\}$, $\{a \vee c, \neg a, b\}$, and $\{a \vee c, \neg a \vee d, b\}$ are the revised results by weakening some requirements involved in minimal inconsistent subsets of R_1, respectively. Each of them indeed enlarge the consistent perspective of the original requirements. In this sense, the values of DE accord with intuition.

However, $DE(R^c|R)$ focuses on only the ratio of the size of R^c to that of the biggest maximal consistent subsets of R. This makes $DE(R^c|R)$ insufficient to capture a more fine-grained comparison between R and R^c. To illustrate this, consider R_1 again, $\{a \vee c, \neg a, b\}$ is the result of weakening only one requirement involved in inconsistency in R_1, in contrast, $\{a \vee c, \neg a \vee d, b\}$ is the result of weakening all the requirements involved in inconsistency in R_2. Intuitively, $\{a \vee c, \neg a \vee d, b\}$ should have higher level of evolution than that $\{a \vee c, \neg a, b\}$ should have. But $DE(\{a \vee c, \neg a, b\}|R_1) = DE(\{a \vee c, \neg a \vee d, b\}|R_1) = \frac{3}{2}$.

To address this, we define a more fine-grained measure for the degree of evolution as follows:

Definition 3.3 (Degree of Evolution DE_f). Let R be a requirements specification and R^c the consistent evolution of R. Then the degree of evolution R^c w.r.t. R, denoted $DE_f(R^c|R)$, is defined as

$$DE_f(R^c|R) = \begin{cases} \frac{|R \setminus R^c|}{|\bigcup \mathsf{MI}(R)|}, & R \vdash \bot \\ 0, & R \nvdash \bot \end{cases},$$

where $\bigcup \mathsf{MI}(R) = \cup_{R' \in \mathsf{MI}(R)} R'$.

Note that $DE_f(R^c|R)$ captures the normalized number of requirements changed during the evolution of R.

Example 3.3. Consider $R_1 = \{a, \neg a, b\}$ again. Then

- $DE_f(\{b\}|R_1) = 1$,
- $DE_f(\{a, b\}|R_1) = DE_f(\{\neg a, b\}|R_1) = \frac{1}{2}$,
- $DE_f(\{a, \neg a \vee c, b\}|R_1) = DE_f(\{a \vee c, \neg a, b\}|R_1) = \frac{1}{2}$,
- $DE_f(\{a \vee c, \neg a \vee d, b\}|R_1) = 1$.

Note that $\{b\}$ and $\{a \vee c, \neg a \vee d, b\}$ are the revised results by deleting and weakening all the requirements involved in inconsistency, respectively, so they have the highest level of evolution. In contrast, each of the other requirements specifications evolved from R keeps one requirements involved in inconsistency unchanged.

Evidently, $DE_f(R^c|R)$ satisfies the following intuitive properties:

- $0 \leqslant DE_f(R^c|R) \leqslant 1$ for all R^c.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

- $DE_f(R^c|R) \leqslant DE_f(R^{c'}|R)$ if $|R \backslash R^c| \leqslant |R \backslash R^{c'}|$, that is, the more requirements changed, the higher the degree of evolution is.
- $DE_f(R^c|R) = 0$ if and only if R is consistent.
- $DE_f(R^c|R) = 1$ if and only if all the requirements involved in inconsistency have been changed.

Based on the two measures, we can define two corresponding inconsistency handling strategies or evolution processes, respectively.

Definition 3.4 (Evolved requirements specification with minimal change) Let R be an inconsistent requirements specification and R^c a requirements specification evolved from R. Then R^c is called an evolved requirements specification with minimal change if

$$DE_f(R^c|R) \leqslant DE_f(R^{c'}|R) \text{ for all } R^{c'}.$$

Roughly speaking, the evolved requirements specification with minimal change requires that a desirable action for resolving inconsistency should make sure that the number of requirements in minimal inconsistent subsets to be changed is the minimum among all the possible proposals. This accords with the principle of minimal change, and is appropriate for the case that the cost of changing requirements is expensive. From now on, we use R^c_{minc} to denote the evolved requirements specification with minimal change of R.

Example 3.4. Consider $R_1 = \{a, \neg a, b\}$ again. Note that

$$\mathsf{MI}(R_1) = \{\{a, \neg a\}\}.$$

Then

$$DE_f(R^c|R_1) \in \{\frac{1}{2}, 1\}.$$

So, $\{a, b\}$, $\{\neg a, b\}$, $\{a, \neg a \vee c, b\}$, and $\{a \vee c, \neg a, b\}$ are evolved requirements specifications with minimal change. But neither of $\{b\}$ and $\{a \vee c, \neg a \vee d, b\}$ is the evolved requirements specification with minimal change.

Example 3.5. Consider $R_2 = \{a, \neg a, \neg a \vee b, \neg b, c\}$. Then $\mathsf{MI}(R_2) = \{\{a, \neg a\}, \{a, \neg a \vee b, \neg b\}\}$ and $\mathsf{FREE}(R_2) = \{c\}$. Note that $\{a\} = \{a, \neg a\} \cap \{a, \neg a \vee b, \neg b\}$. Intuitively, if we choose a as the requirement to be changed, then both $\{a, \neg a\}$ and $\{a, \neg a \vee b, \neg b\}$ can be eliminated. So, the evolved requirements specification of R_2 with minimal change satisfies the following conditions:

- $DE_f(R^c_{2_{minc}}|R_2) = \frac{1}{4}$;
- $\{\neg a, \neg a \vee b, \neg b, c\} \subseteq R^c_{2_{minc}}$;
- $R^c_{2_{minc}} \nvdash a$.

Definition 3.5 (Evolved requirements specification with maximal plausibility). Let R be an inconsistent requirements specification and R^c a requirements specification evolved from R. Then R^c is called an evolved requirements specification with maximal plausibility if

- $|R^c| = |R|$;

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

- $DE(R^c|R) \geqslant DE(R^{c'}|R)$ for all $|R^{c'}| \leqslant |R|$.

Compared to the evolved requirement specification with minimal change, the evolved requirements specification with maximal plausibility aims to elicit stakeholders' demands and to satisfy them as much as possible during the process of inconsistency resolving. This may boost satisfactions of some stakeholders involved in inconsistencies to the software product. However, this kind of evolution may bring expensive costs for changing the requirements and the other work products in the life cycle of software development. From now on, we use R^c_{maxp} to denote the evolved requirements specification with maximal plausibility of R.

Example 3.6. Consider $R_1 = \{a, \neg a, b\}$ again. Note that

$$\mathsf{MC}(R_1) = \{\{a, b\}, \{\neg a, b\}\}.$$

Then

$$DE(R^c|R_1) \in \{\frac{3}{2}, 1\} \text{ for } |R^c| \leqslant |R_1|.$$

So, $\{a \vee c, \neg a \vee d, b\}$, $\{a, \neg a \vee c, b\}$, and $\{a \vee c, \neg a, b\}$ are evolved requirements specifications with maximal plausibility. But $\{a, b\}$, $\{\neg a, b\}$, and $\{b\}$ are not the evolved requirements specification with maximal plausibility.

It has been increasingly recognized that the relative importance of requirements can help stakeholders to make some necessary trade-off decisions for resolving inconsistency. To address this, we need to attach a weight or qualitative priority level to each formula that represents an individual requirement. For convenience and simplicity and without losing generality, we assume that the set of priorities used in this paper is $(0, 1]$. Let R be a requirements specification, then a prioritization over R is a function P_R from R to $(0, 1]$ such that the bigger the priority value of a requirement, the more preferred is the requirement. By this, we can use $\langle R, P_R\rangle$ to formulate prioritized requirements specification(or a prioritized knowledge base). Note that this kind of prioritized knowledge base is exactly Type-I prioritized knowledge base defined in Ref. [9].

To adapt the degree of evolution to the setting that takes the priority of each requirement into account, we define the significance of a set of requirements as follows:

Definition 3.6 (Significance of a requirements specification). Let $\langle R, P_R\rangle$ be a prioritized requirements specification, then the significance of R, denoted $S(R)$, is defined as

$$S(R) = \sum_{r \in R} P_R(r).$$

Roughly speaking, $S(R)$ describes the relative importance of R based on the relative importance of each requirement in R. Obviously, if $P_R(r) = 1$ for each $r \in R$, then $\langle R, P_R\rangle$ can be considered as a classical requirements specification.

Definition 3.7 (Degree of Evolution DE_w). Let $\langle R, P_R\rangle$ be a prioritized requirements specification and $\langle R^c, P_{R^c}\rangle$ the consistent evolution of $\langle R, P_R\rangle$. Then the degree of evolution R^c w.r.t. R, denoted $DE_w(R^c|R)$, is defined as

$$DE_w(R^c|R) = \begin{cases} \frac{S(R \setminus R^c)}{S(\bigcup \mathsf{MI}(R))}, & R \vdash \bot \\ 0, & R \nvdash \bot \end{cases},$$

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

where $\bigcup \mathsf{MI}(R) = \cup_{R' \in \mathsf{MI}(R)} R'$.

Informally speaking, $DE_w(R^c|R)$ captures the normalized significance of the set of requirements changed during the evolution of R. It takes into account the number as well as the priority level of requirements changed for resolving inconsistency in R.

Example 3.7. Consider $\langle R_1, P_{R_1}\rangle$, where $R_1 = \{a, \neg a, b\}$ and $P_{R_1}(a) = 0.8$, $P_{R_1}(\neg a) = 0.4$, $P_{R_1}(b) = 0.9$. Then

- $DE_w(\{b\}|R_1) = 1$,
- $DE_w(\{a, b\}|R_1) = \frac{1}{3}$,
- $DE_w(\{\neg a, b\}|R_1) = \frac{2}{3}$,
- $DE_w(\{a, \neg a \vee c, b\}|R_1) = \frac{1}{3}$,
- $DE_w(\{a \vee c, \neg a, b\}|R_1) = \frac{2}{3}$,
- $DE_w(\{a \vee c, \neg a \vee d, b\}|R_1) = 1$.

Compared to DE_f, $DE_w(\{a, b\}|R_1) < DE_w(\{\neg a, b\}|R_1)$, although both $\{a, b\}$ and $\{\neg a, b\}$ are the evolved requirements specifications by deleting only one requirements involved in inconsistency. Allowing for $P_{R_1}(a) = 0.8 > P_{R_1}(\neg a) = 0.4$, the comparison result is intuitive.

Definition 3.8 (Evolved requirements specification with minimal significance change). Let $\langle R, P_R\rangle$ be an inconsistent prioritized requirements specification and $\langle R^c, P_{R^c}\rangle$ a requirements specification evolved from R. Then R^c is called an evolved requirements specification with minimal significance change if

$$DE_w(R^c|R) \leqslant DE_w(R^{c'}|R) \text{ for all } \langle R^{c'}, P_{R^{c'}}\rangle.$$

Essentially, the evolved requirements specification with minimal significance change requires that a desirable action for resolving inconsistency should make sure that the significance of requirements in minimal inconsistent subsets to be changed is the minimum among all the possible proposals. Note that the evolved requirements specification with minimal change only concerns the minimal number of requirements to be changed. In contrast, the evolved requirements specification with minimal significance change concerns the priority of requirements to be changed as well as the number of such requirements. From now on, we use R^c_{mins} to denote the evolved requirements specification with minimal significance change of R.

Similarly, it is not difficult to adapt the measure DE and the evolved requirements specification with maximal plausibility to prioritized requirements specifications if we use the significance of a set of requirements instead of the number of requirements in the set.

Example 3.8. Consider $\langle R_2, P_{R_2}\rangle$, where $R_2 = \{a, \neg a, \neg a \vee b, \neg b, c\}$ and $P_{R_2}(a) = 0.9$, $P_{R_2}(\neg a) = 0.3$, $P_{R_2}(\neg a \vee b) = 0.6$, $P_{R_2}(\neg b) = 0.3$, $P_{R_2}(c) = 0.9$. Then $\mathsf{MI}(R_2) = \{\{a, \neg a\}, \{a, \neg a \vee b, \neg b\}\}$ and $\mathsf{FREE}(R_2) = \{c\}$. Note that

$$S(\{a\}) = 0.9 > S(\{\neg a, \neg b\}) = 0.3 + 0.3 = 0.6.$$

Intuitively, if we choose $\neg a$ and $\neg b$ as the requirement to be changed, then both $\{a, \neg a\}$ and $\{a, \neg a \vee b, \neg b\}$ can be eliminated. So, the evolved requirements specification of R_2 with minimal significance change satisfies the following conditions:

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

- $DE_w(R^c_{2_{mins}}|R_2) = \frac{2}{7}$;
- $\{a, \neg a \vee b, c\} \subseteq R^c_{2_{mins}}$;
- $R^c_{2_{mins}} \nvdash \neg a$ and $R^c_{2_{mins}} \nvdash \neg b$.

For example, $\{a, \neg a \vee d, \neg a \vee b, \neg b \vee d, c\}$ is the evolved requirements specification of R_2 with minimal significance change.

3.2 Semantics-Based approach to measuring evolution

These measures focus on the aspect of syntax-based or syntax sensitive inconsistency resolving actions. They don't take the semantics of a knowledge base into account. However, some different evolved requirements specifications have the same set of models. To illustrate this, consider $R = \{a, \neg a, b\}$. Evidently, $\{a, b\}$ and $\{a, \neg a \vee b, b\}$ are two different evolved requirements specifications from the syntax-based perspective. But the two sets of requirements have the same model. In some sense, $\neg a \vee b$ may be considered as a redundant requirement in $\{a, \neg a \vee b, b\}$ since the information conveyed by $\{a, \neg a \vee b, b\}$ cannot change when $\neg a \vee b$ was removed. In fact, the set of models of a knowledge base can be considered as a description of expression ability of that base. Informally, the bigger the set of models, the lower the expression ability of the knowledge base is.

To address this, we define the following semantics-based measures for the degree of evolution of requirements.

Definition 3.9 (Degree of evolution DE_{s1}). Let R be a requirements specification and R^c the consistent evolution of R. Then the degree of evolution R^c w.r.t. R, denoted $DE_{s1}(R^c|R)$, is defined as

$$DE_{s1}(R^c|R) = \min_{R' \in \mathsf{MC}(R)} \{||\mathsf{Mod}(R')| - |\mathsf{Mod}(R^c)||\},$$

where $\mathsf{Mod}(R')$ is the set of models of R'.

Actually, $DE_{s1}(R^c|R)$ captures the extent that the models of the maximal consistent perspective of R has been contracted or extended when R evolves to R^c. Evidently, if R is consistent, then $R^c = R$ and

$$DE_{s1}(R^c|R) = |\mathsf{Mod}(R)| - |\mathsf{Mod}(R^c)| = 0.$$

Example 3.9. Consider $R_1 = \{a, \neg a, b\}$ again. Then

- $DE_{s1}(\{b\}|R_1) = 1$, since $\mathsf{Mod}(\{a, b\}), \mathsf{Mod}(\{\neg a, b\}) \subset \mathsf{Mod}(\{b\})$.
- $DE_{s1}(\{a, b\}|R_1) = DE_s(\{\neg a, b\}|R_1) = 0$, both $\{a, b\}$ and $\{\neg a, b\}$ are the maximal consistent subsets of R_1.
- $DE_{s1}(\{a, \neg a \vee b, b\}|R_1) = 0$ because $\mathsf{Mod}(\{a, \neg a \vee b, b\}) = \mathsf{Mod}(\{a, b\})$. So, $\neg a \vee b$ could be considered as an unnecessary weakening.
- $DE_{s1}(\{a, c, b\}|R_1) = 1$.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

Definition 3.10 (Degree of evolution DE_{s2}). Let R be a requirements specification and R^c the consistent evolution of R. Then the degree of evolution R^c w.r.t. R, denoted $DE_{s2}(R^c|R)$, is defined as

$$DE_{s2}(R^c|R) = \min_{R' \in \mathsf{MC}(R)} \{|\mathsf{Mod}(R') \ominus \mathsf{Mod}(R^c)|\},$$

where $\mathsf{Mod}(R') \ominus \mathsf{Mod}(R^c)$ is the symmetric difference of $\mathsf{Mod}(R')$ and $\mathsf{Mod}(R^c)$.

Compared to $DE_{s1}(R^c|R)$, $DE_{s2}(R^c|R)$ captures the difference between the evolved requirements and the original requirements in semantics in some sense.

Example 3.10. Consider $R_1 = \{a, \neg a, b\}$ again. Then

- $DE_{s2}(\{b\}|R_1) = 1$.
- $DE_{s2}(\{a, b\}|R_1) = DE_s(\{\neg a, b\}|R_1) = 0$.
- $DE_{s2}(\{a, c, b\}|R_1) = 1$.

4 A Small Case Study

We use the following small but explanatory example to illustrate some measures presented in this paper.

Example 4.1. Consider the following requirements for updating an existing software system. A representative of the sellers of the new system, provides the following demands:

(a) The system-to-be should be open, that is, the system-to-be could be extended easily.

(b) The system-to-be should be secure.

(c) The user interface of the system-to-be should be fashionable.

A representative of the users of the existing system, provides the following demands:

(d) The system-to-be should be developed based on the techniques used in the existing system;

(e) The user interface of the system-to-be should maintain the style of the existing system, i.e., it does not need to be fashionable.

The domain expert in requirements engineering provides the following constraint, which is a consequence of (b) above:

(f) To guarantee the security of the system-to-be, openness (or ease of extension) should not be considered.

With regard to the prioritization over these requirements, suppose that both (b) and (f) are assigned to 0.9. Both (a) and (c) are assigned to 0.6, and (e) is assigned to 0.4. (d) is assigned to 0.7.

If we

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

- use the predicate Open(sys) to denote that the system is open;
- use the predicate Fash(int_f) to denote that the interface is fashionable;
- use the predicate Exis(sys) to denote that the system will be developed based on the techniques used in the existing system;
- use the predicate Secu(sys) to denote that the system is secure.

Then we have a prioritized knowledge base $\langle R, P_R\rangle$ for the requirements above, where

$$R = \{\text{Open(sys)}, \text{Secu(sys)}, \text{Fash(int\_f)}, \text{Exis(sys)}, \neg\text{Fash(int\_f)}, \text{Secu(sys)} \rightarrow \neg\text{Open(sys)}\},$$

and $P_R : R \mapsto [0, 1]$ such that

$$P_R(\text{Open(sys)}) = 0.6, \quad \text{P}_\text{P}(\text{Fash(int\_f)}) = 0.6, \text{P}_\text{R}(\neg\text{Fash(int\_f)}) = 0.4,$$
$$P_R(\text{Exis(sys)}) = 0.7, \quad \text{P}_\text{R}(\text{Secu(sys)}) = \text{P}_\text{R}(\text{Secu(sys)} \rightarrow \neg\text{Open(sys)}) = 0.9.$$

Clearly, the following inconsistencies can be identified from these requirements:

$$R \vdash \text{Open(sys)} \wedge \neg\text{Open(sys)}, \quad \text{R} \vdash \text{Fash(int\_f)} \wedge \neg\text{Fash(int\_f)}.$$

And the set of minimal inconsistent subsets of K is

$$\mathsf{MI}(R) = \{\{\text{Open(sys)}, \text{Secu(sys)}, \text{Secu(sys)} \rightarrow \neg\text{Open(sys)}\}, \{\text{Fash(int\_f)}, \neg\text{Fash(int\_f)}\}\}.$$

The set of free formulas of K is $\mathsf{FREE}(R) = \{\text{Exis(sys)}\}$.

Evidently, to resolve inconsistencies in R, at least two requirements should be changed. Suppose that there are three potential actions for resolving inconsistency in R:

(A1) abandon (a) and (e);

(A2) abandon (b) and (c);

(A3) abandon (a) and (c).

The corresponding evolved requirements are R_1^c, R_2^c, and R_3^c, respectively, where

$$R_1^c = \{\text{Secu(sys)}, \text{Fash(int\_f)}, \text{Exis(sys)}, \text{Secu(sys)} \rightarrow \neg\text{Open(sys)}\},$$
$$R_2^c = \{\text{Open(sys)}, \text{Exis(sys)}, \neg\text{Fash(int\_f)}, \text{Secu(sys)} \rightarrow \neg\text{Open(sys)}\},$$
$$R_3^c = \{\text{Secu(sys)}, \text{Exis(sys)}, \neg\text{Fash(int\_f)}, \text{Secu(sys)} \rightarrow \neg\text{Open(sys)}\}.$$

If we use the degree of evolution DE, then

$$DE(R_1^c|R) = DE(R_2^c|R) = DE(R_3^c|R) = 1.$$

This means that the perspective reflected by each evolved requirements specification is not bigger than the maximally plausible perspective reflected by R.

If we use the degree of evolution DE_f, then

$$DE_f(R_1^c|R) = DE_f(R_2^c|R) = DE_f(R_3^c|R) = \frac{2}{5}.$$

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

Moreover, each of the three evolved requirements specification can be considered as an evolved requirements specification with minimal change.

If we take into account the priority of each requirement and use the degree of evolution DE_w, then

$$DE_w(R_1^c|R) = \frac{5}{17} < DE_w(R_3^c|R) = \frac{6}{17} < DE_w(R_2^c|R) = \frac{15}{34}.$$

Actually, R_1^c is an evolved requirements specification with minimal significance change. So, action A1 is preferred. This result accords with intuition.

On the other hand, if we use DE_{s1} and DE_{s2}, respectively, then

$$DE_{s1}(R_1^c|R) = DE_{s1}(R_2^c|R) = DE_{s1}(R_3^c|R) = 0.$$

$$DE_{s2}(R_1^c|R) = DE_{s2}(R_2^c|R) = DE_{s2}(R_3^c|R) = 0.$$

This implies that each evolved requirements specification have the same expression ability as R. That is, this evolution brings no new information about requirements. However, each evolved requirements specifications mentioned above is exactly a maximal consistent subset of R.

Note that each of these measures describes the same evolution of requirements caused by inconsistency from a distinctive aspect. Syntax-based measures such as DE, DE_f and DE_w focus on counting the number and the significance of requirements involved in evolution, whilst semantics-based measures are concentrated on the changes in the models due to evolution. In applications such as the case study illustrated above, the choice of measures depends on which aspect(s) we are preferred.

5 Discussion and Comparison

Requirements evolution is a pervasive issue in requirements engineering. There are a number of approaches to managing requirements evolution have been presented both in general[1,11−13,15] and in particular application domains[14]. It has been increasingly recognized that the use of logic is effective to manage requirements evolution, especially in the presence of inconsistency[1,3,13]. For example, Zowghi et al formulated the problem of requirements evolution in the framework of nonmonotonic logics such as default logic and AGM belief revision[3,13], whilst Mu et al modeled the problem of changing requirements as a negotiation-style belief revision[1]. However, there is relatively fewer approaches to measuring or assessing the evolution of requirements in the software development life cycle. In this paper, we focus on a particular kind of evolution, i.e., requirements evolution caused by inconsistency resolving. In the following, we compare our approaches with some of closely related proposals.

The Requirements Maturity Index (RMI) presented in Ref. [11] aims to quantify the readiness of requirements during requirements evolution. It is defined as

$$RMI = \frac{R_T - R_C}{R_T},$$

where R_T is the number of software requirements in the current delivery; R_C is the number of software requirements in the current delivery that are added, deleted or modified from a previous delivery[11]. Also, by taking into account the cumulative number of requirements changes CR_C and the average number of requirements

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

changes AR_C, they presented two refinements of RMI in Ref. [15], namely Requirements Stability Index RSI and Historical Requirements Maturity Index $HRMI$. That is,

$$RSI = \frac{R_T - CR_C}{R_T},\ HRMI = \frac{R_T - AR_C}{R_T}.$$

Roughly speaking, RMI captures the relative change in the number of requirements in the current stage of evolution, whilst $HRMI$ and RSI describe the relative change in the number of the requirements in the whole history of evolution and how the requirements keep stable in some sense, respectively. The syntax-based measures presented in this paper accord with RMI in the sense of using the (relative) number of requirements changed in evolution to measure or the evolution.

However, all the syntax-based measures presented in this paper focus on the evolution caused by inconsistency handling. These are stemmed from the specific characteristics of inconsistent requirements specification, i.e., maximal consistent subsets and minimal inconsistent subsets. That is, measures for such evolution are more interested in capturing the changes of requirements in minimal inconsistent subsets rather than all the requirements. Moreover, some syntax-based measures such as DE_w take into account the priority of each requirement as well as the number of requirements. In addition, we also attempt to assess the evolution by capturing the changes of the set of models of related set of requirements. This semantics-based approach may be more appropriate for capturing the change of expression ability of a requirements specification.

Besides the Requirements Maturity Index, the similarity measure in information retrieval is also introduced to requirements engineering to measuring the similarity between requirements sentences[12]. Compared to similarity measure, the measures presented in this paper are more interested in difference between the original inconsistent requirements specification and the evolved requirements specification. Moreover, this difference may be used to capture the impact of inconsistency handling actions on the development project.

6 Conclusions

We have presented approaches to measuring the requirements evolution caused by inconsistency. This paper presented the following contributions to managing requirements evolution caused by inconsistency.

(a) We presented two syntax-based measures for the evolution of an inconsistent requirements specification. One aims to capture the degree of enlargement of the biggest maximal consistent perspective of an inconsistent requirements specification in inconsistency handling, and another aims to capture the normalized number of requirements involved in inconsistency to be changed for resolving inconsistency.

(b) We characterized two particular kinds of requirements evolution by using the two syntax-based measures, respectively, i.e., evolved requirements specifications with minimal change and evolved requirements specifications with maximal plausibility.

(c) We presented a syntax-based measure for the evolution of inconsistent prioritized requirements specification, which considers both the number and the priority levels of requirements to be changed for resolving inconsistency. We also characterized the evolved requirements specification with minimal significance change based on the

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved

measure.

(d) We presented a semantics-based measure for the evolution of inconsistent requirements specification, which may be more appropriate for capturing the change of expression ability of a requirements specification.

References

[1] Mu K, Liu W, Jin Z, Hong J, Bell D. Managning software requirements changes based on negotiation-style revsion. Journal of Computer Science and Technology, 2011, 26(5): 890–907.

[2] Lormans M. Monitoring requirements evolution using views. Proc. of the 11th European Conference on Software Maintenance and Reengineering, Amsterdam, The Netherlands. March 2007. 349–352.

[3] Gervasi V, Zowghi D. Reasoning about inconsistencies in natural language requirements. ACM Trans. on Software Engineering and Methodologies, 2005, 14(3): 277–330.

[4] Hunter A, Nuseibeh B. Managing inconsistent specification. ACM Trans. on Software Engineering and Methodology, 1998, 7(4): 335–367.

[5] Nuseibeh B, Easterbrook S, Russo A. Leveraging inconsistency in software development. IEEE Computer, 2000, 33(4): 24–29.

[6] Mu K, Jin Z, Zowghi D. A priority-based negotiations approach for handling inconsistency in multi-perspective software requirements. Journal of Systems Science and Complexity, 2008, 21(4): 574–596.

[7] Mu K, Liu W, Jin Z, Yue A, Lu R, Bell D. Handling inconsistency in distributed software requirements specifications based on prioritized merging. Fundamenta Informaticae, 2009, 91(3-4): 631–670.

[8] Reiter R. A theory of diagnosis from first priniciples. Artificial Intelligence, 1987, 32(1): 57–95.

[9] Mu K, Liu W, Jin Z. Measuring the blame of each formula for inconsistent prioritized knowledge bases. Journal of Logic and Computation, 2012, 22(3):481–516.

[10] Hunter A, Grant J. Measuring consistency gain and information loss in stepwise inconsistency resolution. In: Liu W, ed. ECSQARU 2011, LNAI 6717. 2011. 362–373.

[11] Anderson S, Felici M. Requirements evolution from process to product oriented management. Proc. of Profes 2001, 3rd International Conference on Product Focused Software Process Improvement. Kaiserslautern, Germany, September 10-13, 2001. LNCS 2188, Springer-Verlag. 2001. 27–41.

[12] Park S, Kim H, Ko Y, Seo J. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. Information and Software Technology, 2000, 42(6): 429–438.

[13] Zowghi D, Ghose A, Peppas P. A framework for reasoning about requirements evolution. In: Foo NY, Goebel R, eds. PRICAI'96: Topics in Artificial Intelligence, 4th Pacific Rim International Conference on Artificial Intelligence. Cairns, Australia, August 26-30, 1996. Lecture Notes in Computer Science 1114, Springer. 1996. 157–168.

[14] Lutz R, Mikulski I. Operational anomalies as a cause of safety-critical requirements evolution. Journal of Systems and Software, 2003, 65(2): 155–161.

[15] Anderson S, Felici M. Quantitative aspects of requirements evolution. 26th International Computer Software and Applications Conference (COMPSAC 2002). Prolonging Software Life: Development and Redevelopment, 26-29 August, 2002. Oxford, England, IEEE Computer Society. 2002. 27–32.

[16] Konieczny S. On the difference between merging knowledge bases and combining them. In: Cohn AG, Giunchiglia F, Selman B, eds. KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000. Morgan Kaufmann Publishers. 2000. 135–144.

Copyright by Institute of Software, Chinese Academy of Sciences. All rights reserved