Dealing with Known Unknowns: Towards a Game-Theoretic Foundation for Software Requirement Evolution [⋆]

Le Minh Sang Tran and Fabio Massacci

Università degli Studi di Trento, I-38100 Trento, Italy
{tran, fabio.massacci}@disi.unitn.it

Abstract. Requirement evolution has drawn a lot of attention from the community with a major focus on management and consistency of requirements. Here, we tackle the fundamental, albeit less explored, alternative of modeling the future evolution of requirements.
Our approach is based on the explicit representation of *controllable* evolutions vs *observable* evolutions, which can only be estimated with a certain probability. Since classical interpretations of probability do not suit well the characteristics of software design, we introduce a game-theoretic approach to give an explanation to the semantic behind probabilities. Based on this approach we also introduce quantitative metrics to support the choice among evolution-resilient solutions for the system-to-be.
To illustrate and show the applicability of our work, we present and discuss examples taken from a concrete case study (the security of the SWIM system in Air Traffic Management).
Keywords: software engineering, requirement evolution, observable and controllable rules, game-theoretic.

1 Introduction

"...There are known unknowns: that is to say, there are things that we now know we don't know..."

— Donald Rumsfeld, United States Secretary of Defense

In the domain of software, evolution refers to a process of continually updating software systems in accordance to changes in their working environments such as business requirements, regulations and standards. While some evolutions are unpredictable, many others can be predicted albeit with some uncertainty (e.g. a new standard does not appear overnight, but is the result of a long process).

The term *software evolution* has been introduced by Lehman in his work on laws of software evolution [17, 18], and was widely adopted since 90s. Recent studies in software evolutions attempt to understand causes, processes, and effects of the phenomenon [2, 14, 16]; or focus on the methods, tools that manage the effects of evolution [19, 25, 28].

[⋆] This work is supported by the European Commission under project EU-FET-IP-SECURECHANGE.

Requirement evolution has also been the subject of significant research [12, 15, 24, 26, 31]. However, to our understanding, most of these works focus on the issue of management and consistency of requirements. Here, we tackle a more fundamental question of modeling uncertain evolving requirements in terms of evolution rules. Our ultimate goal is to support the decision maker in answering such a question "Given these anticipated evolutions, what is a solution to implement an evolution-resilient system?".

This motivates our research in modeling and reasoning on a requirement model of a system which might evolve sometime in the future. We assume that stakeholders will know the tentative possible evolutions of the system-to-be, but with some uncertainty. For example, the Federal Aviation Authority (FAA) document of the System Wide Information Management (SWIM) for Air Traffic Management (ATM) lists a number of potential alternatives that subject to other high-level decisions (*e.g.,* the existence of an organizational agreement for nation-wide identity management of SWIM users). Such organization-level agreements do not happen overnight (and may shipwreck at any time) and stakeholders with experience and high-level positions have a clear visibility of the likely alternatives, the possible but unlikely solutions, and the politically impossible alternatives.

Our objective is to model the evolution of requirements when it is known to be possible, but it is unknown whether it would happen: the *known unknown*.

1.1 The Contributions of This Paper

We set up a game-theoretic foundation for modeling and reasoning on evolutionary requirement models:

- A way to model requirement evolutions in terms of two kinds of evolution rules: *controllable* and *observable* rules that are applicable to many requirement engineering models (from problem frames to goal models).
- A game-theoretic based explanation for probabilities of an observable evolution.
- Two quantitative metrics to help the designer in deciding optimal things to implement for the system-to-be.

This paper is started by a sketch of a case study (§2). To our purpose, we only focus on requirements of a part of the system-under-study. We distinguish which requirements are compulsory, and which are optional at design time. Based on these, we construct simple evolution scenario to illustrate our approach in subsequent sections, *i.e.* some compulsory requirements become obsoleted, and some optional ones turn to be mandatory.

Then, we discuss how to model requirement evolution (§3) using evolution rules and probabilities of evolution occurrences. We employ the game-theoretic interpretation to account for the semantic of probabilities.

We also introduce two quantitative metrics to support reasoning on rule-based evolutionary requirement models (§4). The reasoning is firstly performed

Table 1. High level requirements of ISS-ENT and ISS-BP in SWIM Security Services.

ID	Requirement	Opt.
RE1	Manage keys and identities of system entities (human, software, devices,...)	
RE2	Support Single Sign-On (SSO)	●
RE3	Support a robust Identity and Key Management Infrastructure (IKMI) that can be scaled up to large number of applications and users.	●
RE4	Intrusion detection and response	
RB1	Less cross-program dependencies for External Boundary Protection System	
RB2	More robust and scalable common security solution	●
RB3	Simpler operation of External Boundary Protection System	●
RB4	Support overall security assessment	●

The Opt(ional) column determines whether a requirement is compulsory or not at current design time. Due to evolution, optional requirements may turn to be compulsory, and current compulsory ones may no longer be needed in the future.

on a simple scenario. Then we show a programmatic way to adapt the technique to a more complex scenario (*e.g.,* large model, multiple evolutions) (§5).

In addition, we discuss current limits of our work, but not the approach, as well as our plan to address them (§6). Finally, we review related works (§7) and conclude the paper(§8).

2 Case Study

Throughout this work, to give a clearer understanding of the proposed approach we draw examples taken from the design architecture of SWIM [7, 23] in ATM.

SWIM provides a secure, overarching, net-centric data network, and introduces a Service-Oriented Architecture (SOA) paradigm for airspace management. The United States FAA [7] has proposed a logical architecture of SWIM which consists of several function blocks, among which we choose to consider the Security Services block. At high level analysis of Security Services, there are five security areas: *i)* Enterprise Information Security System (ISS-ENT), *ii)* Boundary Protection ISS (ISS-BP), *iii)* SWIM Core ISS, *iv)* National Air Space (NAS) End System ISS, and *v)* Registry control. To avoid a detailed discussion on the architecture of SWIM Secure Services, which are not main topic of this work, while providing enough information for illustrating our work we refine our scope of interest on two areas: ISS-ENT and ISS-BP.

– ISS-ENT includes security requirements that are provided as part of an underlying IT/ISS infrastructure used by systems throughout the NAS.
– ISS-BP includes requirements with regard to control connections and information exchanges between internal NAS and external entities. These requirements refer to both network layer control. (*e.g.,* VPNs, firewalls) and application layer control.

Table 1 lists high level requirements of ISS-ENT and ISS-BP. For convenience, each requirement has a corresponding identifier: two characters for the

Table 2. Design elements that support requirements listed in Table 1.

ID	Element Description	RE1	RE2	RE3	RE4	RB1	RB2	RB3	RB4
A	Simple IKMI	●							
B1	OpenLDAP based IKMI	●		●					
B2	Active Directory based IKMI	●	●	●					
B3	Oracle Identity Directory based IKMI	●	●	●			●		
C	Ad-hoc SSO		●						
D	Network Intrusion Detection System				●				
E	Common application gateway for External Boundary Protection System							●	●
F	Centralized Policy Decision Point (PDP)						●		
G	Application-based solution for External Boundary Protection System					●			

Each element in this table can support (or fulfill) requirements listed in columns. To prevent useless redundancy, some elements are exclusive to due to functionality overlapping (*e.g.,* A, B1, B2 and B3 are mutual exclusive each other).

security area (RE - stands for ISS-ENT requirements, RB - stands for ISS-BP ones), and a sequence number. There are compulsory requirements (*i.e.* they are essential at the time the system is designed) and optional ones (*i.e.* they can be ignored at present, but might be critical sometime in the future). Solutions for these requirements are listed in Table 2. Each solution has an IDentifier, a short description and a checklist of requirements that it can fulfill.

3 Modeling Requirement Evolution

In this section, we describe how we model evolution, which essentially affects to any further analysis. We capture evolutions by classifying them into two groups: ***controllable*** and ***observable***. Furthermore, we include in this section the game-theoretic account for probability.

3.1 Evolution on Requirement Model: Controllable and Observable

Stakeholder requirements, mostly in textual format, are their wishes about the system-to-be. Analyzing requirements in such format is difficult and inefficient. Designer thus has to model requirements and design decisions by using various approaches (*e.g.,* model-based, goal-based) and techniques (*e.g.,* DFD, UML).

Generally, a requirement model is a set of elements and relationships, which depend on particular approach. For instance, according to Jackson and Zave [30], model elements are *Requirements*, *Domain assumptions*, *Specifications*; in a goal-based model (*e.g.,* i*), elements are goals, actors and so on.

Here we do not investigate any specific requirement model (*e.g.,* goal-based model, UML models), nor go to details about how many kinds of element and relationship a model would have. The choice of a one's favorite model to represent these aspects can be as passionate as the choice of a one's religion or football team, so it is out of scope. Instead, we treat elements at abstract meaning, and only be interested in the satisfaction relationship among elements.

In our work, we define the satisfaction relationship in terms of usefulness. That an element set X is useful to another element set Y depends on the ability to satisfy (or fulfill) Y if X is satisfied. We define a predicate useful(X, Y) which is true (1) if X can satisfy all elements of Y, otherwise false (0). The implementation of useful depends on the specific requirement model. For examples:

- Goal models [20]: useful corresponds to Decomposition and Means-end relationships. The former denotes a goal can be achieved by satisfying its subgoals. The later refers to achieving an (end) goal by doing (means) tasks.
- Problem frames [13]: useful corresponds to requirement references and domain interfaces relationships. Requirements are imposed by machines, which connect to problem world via domain interfaces. Problem world in turn connects to requirements via requirement references.

For evolutionary software systems which may evolve under some circumstances (*e.g.,* changes in requirements due to changes in business agreements, regulations, or domain assumption), their requirement models should be able to express as much as possible the information about known unknowns *i.e.* potential changes. These potential changes are analyzed by evolution assessment algorithms to contribute to the decision making process, where a designer decides what would be in the next phase of the development process.

Based on the actor who can decide which evolution would happen, we categorize requirement evolutions into two classes:

- *controllable evolution* is under control of designer to meet high level requirements from stakeholder.
- *observable evolution* is not under control of designer, but its occurrence can be estimated with a certain level of confidence.

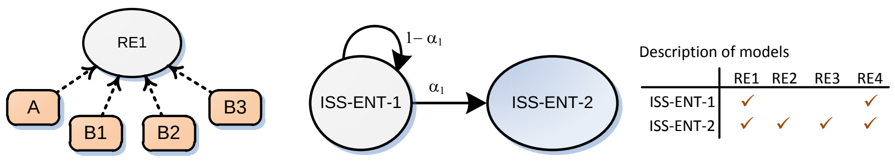
Controllable evolutions, in other words, are designer's moves to identify different design alternatives to implement a system. The designer then can choose the most "optimal" one based on her experience and some analyses on these alternatives. In this sense, controllable evolution is also known as design choice.

Observable ones, in contrast, correspond to potential evolutions of which realization is outside the control of the designer. They are moves of reality to decide how a requirement model looks like in the future. Therefore, the stakeholder and designer have to forecast the reality's choice with a level of uncertainty. The responses are then incorporated into the model.

We capture the evolution in terms of evolution rule. We have *controllable rule* and *observable rule* corresponding to controllable and observable evolution.

Definition 1. *A controllable rule r_c is a set of tuples $\langle RM, RM_i \rangle$ that consists of an original model RM and its possible design alternative RM_i.*

$$r_c = \bigcup_{i}^{n} \left\{ RM \xrightarrow{*} RM_i \right\}$$



(a) Controllable rule (b) Observable rule

Fig. 1. Example of controllable rule (a), and observable rule (b).

Definition 2. *An observable rule r_o is a set of triples $\langle RM, p_i, RM_i\rangle$ that consists of an original model RM and its potential evolution RM_i. The probability that RM evolves to RM_i is p_i. All these probabilities should sum up to one.*

$$r_o = \bigcup_{i=1}^{n} \left\{ RM \xrightarrow{p_i} RM_i \right\}$$

Fig. 1 is a graphical representation of evolution rules taken from SWIM case study. Left, Fig. 1(a) describes a controllable rule where a requirement model containing IKMI (RE1) has four design choices: A, B1, B2, and B4 (see Table 1 and Table 2). Right, Fig. 1(b) shows that the initial model ISS-ENT-1 (including RE1 and RE4) can evolve to ISS-ENT-2 (including RE1 to RE4), or remain unchanged with probabilities of α and $1-\alpha$. These rules are as follows:

$$r_c = \left\{ RE1 \xrightarrow{*} A, RE1 \xrightarrow{*} B1, RE1 \xrightarrow{*} B2, RE1 \xrightarrow{*} B3 \right\}$$

$$r_o = \left\{ \text{ISS-ENT-1} \xrightarrow{\alpha_1} \text{ISS-ENT-2}, \text{ISS-ENT-1} \xrightarrow{1-\alpha_1} \text{ISS-ENT-1} \right\}$$

3.2 Game-Theoretic Account for Probability

Here, we discuss about why and how we employ game-theoretic (or betting interpretation) to account for probabilities in observable rules.

As mentioned, each potential evolution in an observable rule has an associated probability; these probabilities sum up to 1. However, who tells us? And what is the semantic of probability? To answer the first question, we, as system Designers, agree that Stakeholder will tell us possible changes in a period of time. About the second question, we need an interpretation for semantic of probability.

Basically, there are two broad categories of probability interpretation, called "physical" and "evidential" probabilities. Physical probabilities, in which frequentist is a representative, are associated with a random process. Evidential probability, also called Bayesian probability (or subjectivist probability), are considered to be degrees of belief, defined in terms of disposition to gamble at certain odds; no random process is involved in this interpretation.

To account for probability associated with an observable rule, we can use the Bayesian probability as an alternative to the frequentist because we have no event to be repeated, no random variable to be sampled, no issue about measurability (the system that designers are going to build is often unique in some respects). However, we need a method to calculate the value of probability as well as to explain the semantic of the number. Since probability is acquired from the requirement eliciting process involving the stakeholder, we propose using the game-theoretic method in which we treat probability as a price. It seems to be easier for stakeholder to reason on price (or cost) rather than probability.

The game-theoretic approach, discussed by Shafer et al. [27] in Computational Finance, begins with a game of three players, *i.e.* Forecaster, Skeptic, and Reality. Forecaster offers prices for tickets (uncertain payoffs), and Skeptic decides a number of tickets to buy (even a fractional or negative number). Reality then announces real prices for tickets. In this sense, probability of an event E is the initial stake needed to get 1 if E happens, 0 if E does not happen. In other words, the mathematics of probability is done by finding betting strategies.

In this paper, we do not deal with stock market but the design of evolving software, *i.e.* we extend it for software design. We then need to change rules of the game. Our proposed game has three players: *Stakeholder*, *Designer*, and *Reality*. For the sake of brevity we will use "he" for Stakeholder, "she" for Designer and "it" for Reality. The sketch of this game is denoted in protocol 1.

Protocol 1

```
Game has n round, each round plays on a software $C_i$
FOR $i = 1$ to $n$
    Stakeholder announces $p_i$
    Designer announces her decision $d_i$: believe, don't believe
    If Designer believes
        $K_i = K_{i-1} + M_i \times (r_i - p_i)$
    Designer does not believe
        $K_i = K_{i-1} + M_i \times (p_i - r_i)$
    Reality announces $r_i$
```

The game is about Stakeholder's desire of having a software C. He asks Designer to implement C, which has a cost of $M\$$. However, she does not have enough money to do this. So she has to borrow money from either Stakeholder or National Bank with the return of interest (ROI) p or r, respectively.

Stakeholder starts the game by announcing p which is his belief about the minimum ROI for investing $M\$$ on C. In other words, he claims that r would be greater than p. If M equals 1, p is the minimum amount of money one can receive for 1\$ of investment. Stakeholder shows his belief on p by a commitment that he is willing to buy C for price $(1 + p)M$ if Designer does not believe him and borrow money from someone else.

If Designer believes Stakeholder, she will borrow M from Stakeholder. Later on, she can sell C to him for $M(1+r)$ and return $M(1+p)$ to him. So, the final amount of money Designer can earn from playing the game is $M(r - p)$.

If Designer does not believe Stakeholder, she will borrow money from National Bank, and has to return $M(1 + r)$. Then, Stakeholder is willing to buy C with $M(1 + p)$. In this case, Designer can earn $M(p - r)$.

Suppose that Designer has an initial capital of K_0. After round i-*th* of the game, she can accumulate either $K_i = K_{i-1} + M(r-p)$ or $K_i = K_{i-1} + M(p-r)$, depend on whether she believes Stakeholder or not. Designer has a winning strategy if she can select the values under her control (the $M\$$) so that she always keeps her capital never decrease, intuitively, $K_i >= K_{i-1}$ for all rounds.

The law of large numbers here corresponds to say that if unlikely events happen then Designer has a strategy to multiply her capital by a large amount. In other words, if Stakeholder estimates Reality correctly then Designer has a strategy for costs not to run over budget.

4 Making Decision: What Are the Best Things to Implement

One of the main objectives of modeling evolution is to provide a metric (or set of metrics) to indicate how well a system design can adapt to evolution. Together with other assessment metrics, designers have clues to decide what an "optimal" solution for a system-to-be is.

The major concern in assessment evolution is answering the question: "Whether a model element (or set of elements) becomes either useful or useless after evolution?". Since the occurrence of evolution is uncertain, so the usefulness of an element set is evaluated in term of probability. In this sense, this work proposes two metrics to measure the usefulness of element set as follows.

Max Belief (MaxB): of an element set X is a function that measures the maximum belief supported by Stakeholder such that X is useful to a set of top requirements after evolution happens. This belief of usefulness for a set of model element is inspired from a game in which Stakeholder play a game together with Designer and Reality to decide which elements are going to implementation phase.

Residual Risk (RRisk): of an element set X is the complement of total belief supported by Stakeholder such that X is useful to set of top requirements after evolution happens. In other words, residual risk of X is the total belief that X is not useful to top requirements with regard to evolution. Importantly, do not confuse this notion of residual risk with the one in risk analysis studies which are different in nature.

Given an evolutionary requirement model $RM = \langle RM, r_o, r_c \rangle$ where $r_{o=} \bigcup_i \left\{ RM \xrightarrow{p_i} RM_i \right\}$ is an observable rule, and $r_c = \bigcup_{ij} \left\{ RM_i \xrightarrow{*} RM_{ij} \right\}$ is a controllable rule, the calculation of max belief and residual risk is illustrated in Eq. 1, Eq. 2 as follows.

$$MaxB(X) = \max_{RM \xrightarrow{p_i} RM_i \in \mathcal{S}} p_i \tag{1}$$

$$RRisk(X) = 1 - \sum_{RM \xrightarrow{p_i} RM_i \in \mathcal{S}} p_i \tag{2}$$

where $\mathcal{S}$ is set of potential evolutions in which X is useful.

$$\mathcal{S} = \left\{ RM \xrightarrow{p_i} RM_i | \exists (RM_i \xrightarrow{*} RM_{ij}) \in r_c \; st.\mathsf{useful}(X, RM_{ij}) \right\}$$

One may argue about the rationale of these two metrics. Because he (or she) can intuitively measure the usefulness of an element set by calculating the *Total Belief* which is exactly the complement of our proposed *Residual Risk*. However, using only Total Belief (or *Residual Risk*) may mislead designers in case of a *long-tail* problem.

The long-tail problem, firstly coined by Anderson [1], describes a larger population rests within the tail of a normal distribution than observed. A long-tail example depicted in Fig. 2 where a requirement model RM might evolve to several potential evolutions with very low probabilities (say, eleven potential evolutions with 5% each), and another extra potential evolution with dominating probability (say, the twelfth one with 45%). Suppose that an element A appears in the first eleven potential evolutions, and an element B appears in the last twelfth potential evolution. Apparently, A is better than B due to A's total belief is 55% which is greater than that of B, say 45%. However, at the end of the day, only one potential evolution becomes effective (*i.e.* chosen by Reality) rather than 'several' potential evolutions are together chosen. If we thus consider every single potential evolution to be chosen, the twelfth one (45%) seems to be the most promising and *Max Belief* makes sense here. Arguing that A is better than B or versa is still highly debatable. Ones might put their support on the long tail [1], and ones do the other way round [5]. Therefore, we introduce both *Residual Risk* and *Max Belief* to avoid any misleading in the decision making process that can be caused when using only Total Belief.

For a better understanding of *Max Belief* and *Residual Risk*, we conclude this section by applying our proposed metrics to the evolution of SWIM Security Services discussed in previous section. In Fig. 3, here we have an initial requirement model RM0(ISS-ENT-1,ISS-BP-1) that will evolve to RM1(ISS-ENT-2,ISS-BP-1), RM2(ISS-ENT-1,ISS-BP-2), and RM3(ISS-ENT-2,ISS-BP-2) with

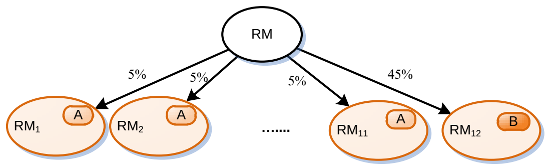


Fig. 2. The long-tail problem.

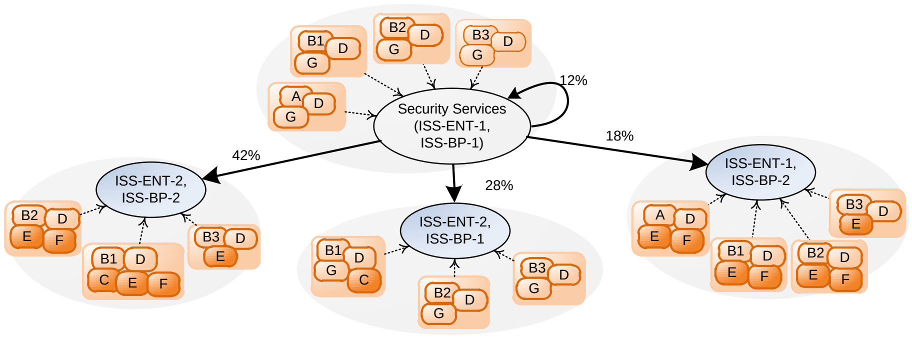


Fig. 3. Evolution of the SWIM Security Service.

probabilities of 28%, 18% and 42%, respectively. There are 12% that RM0 stays unchanged. Each requirement model is represented as a bubble in which there is a controllable rule with several design alternatives. Each design alternative is an element set represented as a rounded rectangle that contains elements (such as A, D, and G) to support (fulfill) requirements of that requirement model.

Table 3 shows some examples, where the first column displays element sets, and the two next columns show values of max belief and residual risk. Notice that the max belief and residual risk in the first row, where the element set is $\{A, D\}$, are n/a which means that we are unable to find any potential evolution that $\{A, D\}$ can support all top requirements.

Table 3. Examples of Max Belief and Residual Risk

Element set	Max Belief	Residual Risk
{A, D,}	n/a	n/a
{A, E, D, G, F}	18%	70%
{B3, D, G}	28%	60%
{B1, D, G, C}	28%	60%
{B3, D, E, G}	42%	0%
{B2, D, E, F, G}	42%	0%

In Table 3, $\{B3, D, E, G\}$ and $\{B2, D, E, F, G\}$ seem to be the best choices, since they have a high max belief (42%) and low residual risk (0%). The zero residual risk means these element sets are surely still useful after evolution. If the cost of implementation is the second criteria and assume that each element has equal cost, then $\{B3, D, E, G\}$ seems to be better.

5 Handling Complex Evolution

If a model is too large and complex, instead of dealing with the evolution of the whole model, we can consider evolution in each subpart. If a subpart is still too large and complex, we can recursively divide it into smaller ones, each with its local evolution rule, until we are able to deal with.

We then need to combine these local rules together to produce a global evolution one for the whole model. For simplicity, we assume that:

ASS-1: Independence of evolutions All observable rules are independent. It means that they do not influent each other. In other words, the probability that an evolution rule is applied does not affect to that of other rules.

ASS-2: Order of evolutions Controllable evolutions are only considered after observable evolutions.

As discussed, observable rules are analyzed on independent subparts. Prevailing paradigms of software development (*e.g.,* Object-Oriented, Service-Oriented) encourage encapsulation and loosely coupling. Evolutions applying to subparts, therefore, are often independent. Nevertheless, if there are two evolution rules which influent each other, we can combine them into a single one. We assume that dependent evolutions do happen, but not a common case. Hence manual combination of these rules is still doable.

The second assumption is the way we deal with controllable rules. If we apply controllable rules before observable ones, it means we look at design alternatives before observable evolutions happen. This makes the problem more complex since under the effect of evolution, some design alternatives are no longer valid, and some others new are introduced. Here, for simplicity, we look at design alternatives for evolved requirement models that will be stable at the end of their evolution process.

After all local evolutions at subparts are identified, we then combine these rules into a global evolution rule that applies to the whole model. The rationale of this combination is the effort to reuse the notion of Max Belief and Residual Risk (§4) without any extra treatment. In the following we discuss how to combine two independent observable evolution rules.
Given two observable rules:

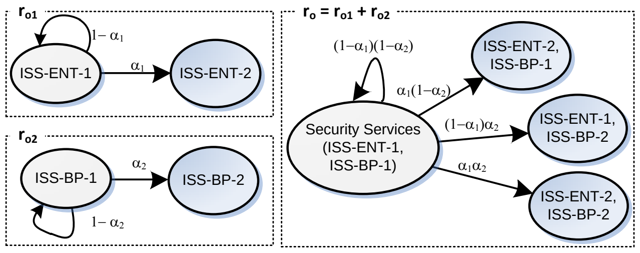
$$r_{o1} = \bigcup_{i=1}^{n} \left\{ RM1 \xrightarrow{p1_i} RM1_i \right\} \text{ and } r_{o2} = \bigcup_{j=1}^{m} \left\{ RM2 \xrightarrow{p2_j} RM2_j \right\}$$

Let r_o is combined rule from r_{o1} and r_{o2}, we have:

$$r_o = \bigcup_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} \left\{ RM1 \cup RM2 \xrightarrow{p1_i * p2_j} RM1_i \cup RM2_j \right\}$$

Fig. 4 illustrates an example of combining two observable rules into a single one. In this example, there are two subparts of SWIM Security Service: ISS-ENT and ISS-BP. The left hand side of the figure displays two rules for these parts, and in the right hand side, it is the combined rule.

In general case, we have multiple steps of evolution *i.e.* evolution happens for many times. For the ease of reading, *step 0* will be the first step of evolution, where no evolution is applied. We use RM_i^d to denote the i-*th* model in step d, and $r_{od,i}$ to denote the observable evolution rule that applies to RM_i^d , *i.e.* $r_{od,i}$ takes RM_i^d as its original model.



Description of models

	RE1	RE2	RE3	RE4
ISS-ENT-1	✓			✓
ISS-ENT-2	✓	✓	✓	✓

	RB1	RB2	RB3	RB4
ISS-BP-1	✓			
ISS-BP-2		✓	✓	✓

Fig. 4. Example of combining two observable evolution rules.

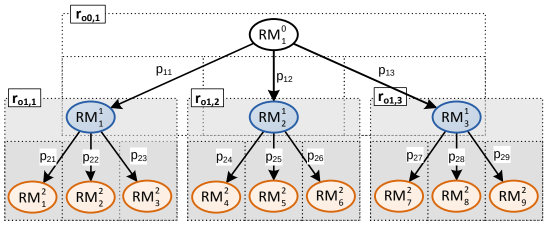


Fig. 5. Multiple steps (phases) evolving requirement model.

The multi-step evolution begins with an original model RM_1^0. This model can evolve to one of the potential evolutions RM_i^1. In the second step, each RM_i^1 then also evolves to one of many potential evolutions RM_j^2. The evolution stops after k steps. If we represent a model as a node, and connect a model to its potential evolutions as we have done as aforementioned, then we have a tree-like graph, called *evolution tree* with k-depth.

Fig. 5 illustrates a two-step evolution, in which observable rules are denoted as dotted boxes. The original model lays on top part of a box, and all potential evolutions are in sub boxes laid at the bottom. There are directed edges connecting the original model to potential evolutions. The label on each edge represents the probability such that original model evolves to target model.

In Fig. 5, an initial requirement model RM_1^0 evolves to either RM_1^1, RM_2^1 or RM_3^1. Likewise, RM_i^1 evolves to RM_j^2, where i=1..3 and j=1..9. Here, we have a ternary complete tree of depth 2. Generally, the evolution tree of a k-step consecutive evolution is a complete k-depth, m-ary tree.

We can always collapse a k-step evolution into an equivalent 1-step one in terms of probability by letting the original model evolve directly to the very last models with the probabilities that are multiplication of probabilities of intermediate steps. Therefore, any k-step evolution has an equivalent 1-step evolution. Hence all analyses discussed in §4 are applicable without any modification.

6 Limitation

Obviously there are limitations in this work:

- *Real world applicability.* Even though we work on a real world case study, this work is still pure theory. It needs to be elaborated and then evaluated with the industry. We plan to prove our work in the field of Air Traffic Management (ATM), where we interact with designers and stakeholder of an ATM system, and get their feedback for validation.
- *Obtaining probability.* Since evolution probabilities are obtained from stakeholder, they are individual opinions. To deal with the problem, we shall work on an interaction protocol with stakeholder to minimize inaccuracy, as well as equip an appropriate mathematic foundation (*e.g.,* Dempster and Shafer's theory) for our reasoning.
- *Independence of evolution.* Complex models may require many probabilities that are not independent. This breaks the assumptions discussed in §5. Even though designers can solve this problem by manually combining dependent evolutions, we still need a more systematic way to deal with them.

7 Related Works

A majority of approaches to software evolution has focused on the evolution of architecture and source code level. However, in recent years, changes at the requirement level have been identified as one of the drivers of software evolution [4, 12, 31]. As a way to understand how requirements evolve, research in PROTEUS [24] classifies changing requirements (that of Harker et al [11]) into five types, which are related to the development environment, stakeholder, development processes, requirement understanding and requirement relation. Later, Lam and Loomes [15] present the EVE framework for characterizing changes, but without providing specifics on the problem beyond a meta model.

Several approaches have been proposed to support requirements evolution. Zowgi and Offen [31] work at meta level logic to capture intuitive aspects of managing changes to requirement models. Their approach involves modeling requirement models as theories and reasoning changes by mapping changes between models. However, this approach has a limitation of overhead in encoding requirement models into logic.

Russo et al [26] propose an analysis and revision approach to restructure requirements to detect inconsistency and manage changes. The main idea is to allow evolutionary changes to occur first and then, in the next step, verify their impact on requirement satisfaction. Also based on this idea, Garcez et al [4] aim at preserving goals and requirements during evolution. In the analysis, a specification is checked if it satisfies a given requirement. If it does not, diagnosis information is generated to guide the modification of specification in order to satisfy the requirement. In the revision, the specification is changed according to diagnosis information generated. Similar to Garcez et al, Ghose's [9] framework is based on formal default reasoning and belief revision, aims to address the problem of inconsistencies due to requirement evolution. This approach is supported by automated tools [10]. Also relating to inconsistencies, Fabrinni et al [6] deal with requirement evolution expressed in natural language, which is challenging to capture precisely requirement changes. Their approach employs formal concept analysis to enable a systematic and precise verification of consistency among different stages, hence, controls requirement evolution.

Other notable approaches include Brier et al.'s [3] to capture, analyze, and understand how software systems adapt to changing requirements in an organizational context; Felici et al [8] concern with the nature of requirements evolving in the early phase of systems; Stark et al [29] study the information on how change occurs in the software system and attempt to produce a prediction model of changes; Lormans et al [21] use a formal requirement management system to motivate a more structural approach to requirement evolution.

8 Conclusion

We have discussed a rule-based representation of evolutions on requirement models. We proposed game-theoretic approach to explain the uncertainty of evolutions. We also introduced two notions of max belief and residual risk to reason on evolutionary models, in which the higher max belief and lower residual risk models seem to be more evolution-resilient than others. Together with other analyses (*e.g.,* cost, risk), these values can help designers in making decision.

During the discussion, we provided many examples taken from a real world project, SWIM. These examples not only help to explain better our idea, but also show the promising applicability of our approach.

For future work, we plan to instantiate our approach to a concrete modeling language (*e.g.,* goal-based language) and apply to a more convincing case study. We shall interact with stakeholder and designers, show them our approach and get their feedback to validate the usability of proposed approach.

References

1. C. Anderson. The long tail. *Wired*, October 2004.
2. A. Anton and C. Potts. Functional paleontology: The evolution of user-visible system services. *TSE*, 29(2):151–166, 2003.
3. J. Brier, L. Rapanotti, and J. Hall. Problem-based analysis of organisational change: a real-world example. In *Proc. of IWAAPF '06*. ACM, 2006.
4. A. d'Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer. Combining abductive reasoning and inductive learning to evolve requirements specifications. In *IEE Proceedings - Software*, volume 150(1), pages 25–38, 2003.
5. A. Elberse. Should you invest in the long tail? *Harvard Business Review*,2008.
6. F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. Controlling requirements evolution: a formal concept analysis-based approach. *ICSEA '07*, 2007.
7. FAA. System wide information management (SWIM) segment 2 technical review. Tech. report, 2009.
8. M. Felici. *Observational Models of Requirements Evolution.* PhD thesis, 2004.

9. A. Ghose. A formal basis for consistency, evolution and rationale management in requirements engineering. *ICTAI '99*, 1999.
10. A. Ghose. Formal tools for managing inconsistency and change in re. In *IWSSD '00*, Washington, DC, USA, 2000. IEEE Computer Society.
11. S. Harker, K. Eason, and J. Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *RE '01*, 1993.
12. J. Hassine, J. Rilling, J. Hewitt, and R. Dssouli. Change impact analysis for requirement evolution using use case maps. In *IWPSE '05*, 2005.
13. M. Jackson. *Problem Frames: Analysing & Structuring Software Development Problems*. Addison-Wesley, 2001.
14. C. F. Kemerer and S. Slaughter. An empirical approach to studying software evolution. *TSE*, 25(4):493–509, 1999.
15. W. Lam and M. Loomes. Requirements evolution in the midst of environmental change: a managed approach. In *CSMR '98*, 1998.
16. M. LaMantia, Y. Cai, A. MacCormack, and J. Rusnak. Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory: Two exploratory cases. In *Proc. of WICSA '08*, pages 83–92, 2008.
17. M. Lehman. On understanding laws, evolution and conservation in the large program life cycle. *J. of Sys. and Soft.*, 1(3):213 –221, 1980.
18. M. Lehman. Programs, life cycles, and laws of software evolution. *Proc. IEEE 68*, 9:1060 –1076, September 1980.
19. L. Lin, S. Prowell, and J. Poore. The impact of requirements changes on specifications and state machines. *SPE*, 39(6):573–610, 2009.
20. L. Liu and E. Eric Yu. Designing information systems in social context: A goal and scenario modelling approach. *Info. Syst*, 29:187–203, 2003.
21. M. Lormans, H. van Dijk, A. van Deursen, E. Nocker, and A. de Zeeuw. Managing evolving requirements in an outsourcing context: an industrial experience report. In *IWPSE '04*, pages 149 –158, 2004.
22. T. Mens, J. Ramil, and M. Godfrey. Analyzing the evolution of large-scale software. *J. of Soft. Maintenance and Evolution: Research and Practice*, 16(6):363–365,2004.
23. Program SWIM-SUIT. D1.5.1 Overall SWIM users requirements. Tech. report, 2008.
24. Project PROTEUS. Deliverable 1.3: Meeting the challenge of chainging requirements. Tech. report, Centre for Soft. Reliab., Univ. of Newcastle upon Tyne, 1996.
25. R. Ravichandar, J. Arthur, S. Bohner, and D. Tegarden. Improving change tolerance through capabilities-based design: an empirical analysis. *J. of Soft. Maintenance and Evolution: Research and Practice*, 20(2):135–170, 2008.
26. A. Russo, B. Nuseibeh, and J. Kramer. Restructuring requirements specifications. In *IEE Proceedings: Software*, volume 146, pages 44 – 53, 1999.
27. G. Shafer, V. Vovk, and R. Chychyla. How to base probability theory on perfect-information games. *BEATCS*, 100:115 – 148, February 2010.
28. P. Soffer. Scope analysis: identifying the impact of changes in business process models. *J. of Soft. Process: Improvement and Practice*, 10(4):393–402, 2005.
29. G. Stark, P. Oman, A. Skillicorn and A. Ameele. An examination of the effects of requirements changes on software maintenance releases. *J. of Soft. Maintenance: Research and Practice*, pages 293–309, 1999.
30. P. Zave and M. Jackson. Four dark corners of req. eng. *TSEM*, 6(1):1–30, 1997.
31. D. Zowghi and R. Offen. A logical framework for modeling and reasoning about the evolution of requirements. *ICRE '97*, 1997.