An Approach for Decision Support on the Uncertainty in Feature Model Evolution

Le Minh Sang Tran and Fabio Massacci
DISI, University of Trento
Povo, Trento, Italy
{leminhsang.tran, fabio.massacci}@unitn.it

***Abstract*—Software systems could be seen as a hierarchy of features which are evolving due to the dynamic of the working environments. The companies who build software thus need to make an appropriate strategy, which takes into consideration of such dynamic, to select features to be implemented. In this work, we propose an approach to facilitate such selection by providing a means to capture the uncertainty of evolution in feature models. We also provide two analyses to support the decision makers. The approach is exemplified in the Smart Grid scenario.**

***Index Terms*—evolution, product lines, feature model evolution, variability, survivability, cost of reparation.**

I. Introduction

Feature-oriented modeling [17], firstly coined by Kang *et al.* in 1990, is a method capturing a software system as a hierarchy of features. It has been widely adopted in many scenarios of software product lines engineering [2] by the term *feature models*. Feature models provide a compact representation of different variants of software systems. This enables customers' personalization by deciding which features to be included or discarded in the final product.

Software keeps evolving due to various reasons (*e.g.,* compliance with new laws or standards, or changes in market needs) and companies who ship software need a long-term strategy for their products while still delivering products with the features required at present time. It is therefore necessary to have an appropriate reasoning on the evolution of software features to support the long-term planning strategy.

A key observation underpinning our work is that apart from unpredictable black swans, many of changes could be anticipated with some degree of belief because they are the results of long-term processes. Such predictions could be made with the support of expert knowledge in a particular domain of application. A paradigmatic example in the domain of power supply is the development of a Smart Grid which is an infrastructure to efficiently manage the transmission and distribution of electricity. Different expected features and development roadmap of Smart Grid have been identified [15], [20] and the important question is how to build Smart Grid software with the right features from the beginning such that it is more resilient to evolution, and the cost of adding new features in future would be minimal. There are not many studies [7], [23], [33], [28], [12], [14] concerning the evolution of software product lines represented by feature models, although there is a common consensus that this evolution is an important aspect [26], [6]. In particular, no existing work takes into consideration the uncertainty of feature model evolution.

In [30], we introduced a generic approach that captured requirements evolution by evolution rules, and two metrics to support the selection of a design alternative. In this work, we extend that approach to support modeling and reasoning on the uncertainty of feature model evolution. Our approach assists the selection of an *optimal configuration*, which is a set of features to be implemented. The system built on this optimal configuration could be able to survive as long as possible, and requires less effort to repair if evolution occurs. Our novel contributions from [30] include:

- a conceptual model for modeling evolution on feature models, which consists of two kinds of models *(i)* Evolution Possibility Model (ePM) describes potential possibilities a feature model could evolve, and *(ii)* Evolutionary Feature Model (eFM) describes the feature model with all changes due to evolution incorporated. Evolution is studied within a study period T which is divided into several milestones t_i. At every milestone, different possibilities of the original feature model are analyzed.
- two analysis techniques facilitate the decision support: *Survivability analysis* answers whether a configuration (*i.e.,* set of features) could survive during the expected evolution. *Repair cost analysis*: it is likely that few configurations remain valid for the entire study period due to changes, we need to repair them to make them valid. The question is which configuration requires less effort to get repaired. The former analysis takes the metrics' idea in [30], specializes it for feature models, and makes it a function of time to capture the survivability of a configuration over time.
- a self-validation of the proposed approach on the Smart Grid scenario, taken from the NESSoS European project.

This work aims to deal with anticipated evolution in feature models. For the uncertainty and evolution in requirements, readers could find a more detailed discussion in [8], [25], [9].

In the next, Section II presents the feature model background and related studies on feature model evolution. We discuss different evolution perspectives in Section III. Our approach is detailed in Section IV and Section V. Section VI

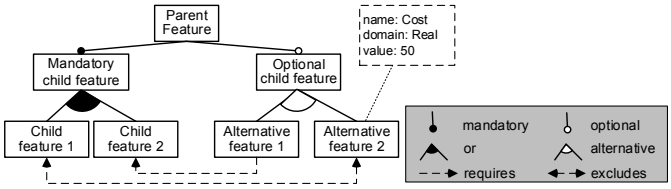


Fig. 1. An example of a feature model.

applies the proposed approach in a Smart Grid scenario. Finally, we provide an extended discussion about the approach in Section VII and conclude the paper in Section VIII.

II. Background

Previous work related to this study could fall into two categories: studies about analyses on feature models, and studies about the evolution of feature models. For the first category, readers are referred to [2] for a systematic survey. Here, we only present a background about feature models, and studies in the latter category.

A. Feature Model

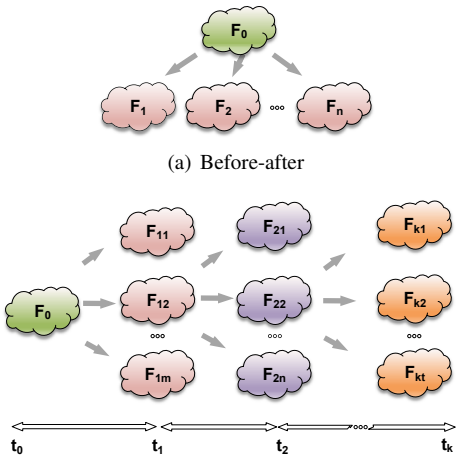
Feature model has many variants, but its basic form includes hierarchical features and relations between them [2]. The top feature represents the software system which is implemented by its children. A *configuration* is the set of features that implement the top feature. The basic feature model could be extended to include more information about the features via *attributes*. An *attribute* typically should consists of a name, a domain, and a value. The basic relations are:

- *Mandatory*: the selection of a parent feature requires the selection of its *mandatory* child features.
- *Optional*: the selection of a parent feature optionally leads to the selection of its *optional* child features.
- *Alternative (xor)*: when a parent feature is selected, exactly one of its *alternative* children ought to be selected.
- *Or*: when the parent feature is selected, at least one or more of its *or* children must be selected.
- *Requires*: a feature A requires a feature B if the selection of A implies the selection of B.
- *Excludes*: a feature A excludes a feature B if they are cannot be part of a same configuration.

Extended feature models, Fig. 1, can include complex constraints like "if an attribute of feature A is lower than a value, then feature B cannot be part of the configuration" [2].

B. Related Work on Feature Model Evolution

Botterweck et al. [7] report the need of companies to have a long-term strategy and plan product portfolios in the years ahead. They apply feature models to plan changes; evolution is considered as a sequence of feature models. They propose the Evolution Plan and Evolution Feature Model (EvoFM). The former provides an overview of evolution steps across time. The latter extends an ordinary feature model with some sub-features to support the consideration of evolution options; each EvoFM instance expresses an evolution step. Later, Pleuss et al. [23] envelop these artifacts into the EvoPL framework to plan and manage the evolution of feature models.



(a) Before-after

(b) Continuous

Fig. 2. Two kinds of evolution perspective of feature models.

Other studies about the impact of evolution in feature models are [33], [28], [12], [14]. In these studies, evolution is captured by the addition, removal, modification of entities in feature models. Thüm et al. [28] study the impact of changes by presenting an algorithm to classify changes (*e.g.,* refactoring, specialization, generalization). Gamez and Fluentes [12] work on cardinality-based feature models and evaluate the change impact by computing the differences between previous configuration and new evolved configuration. Ye and Zhang [33] provide formal specifications to detect any conflicting dependencies introduced by adding or removing features. Guo et al. [14] also analyze the impact of evolution on the consistency of feature models. They provide an ontology-based formalization of feature models and a set of consistency constraints to identify inconsistencies in feature model evolution.

In our work, evolution also happens in time as a sequence of feature models. We also capture the evolution uncertainty and provide reasoning techniques to select an optimal configuration while they do not do it.

Besides, there are a number of studies that have addressed the issues of uncertainty modeling of requirements and architecture. Our own work [30] addresses the problem of the known unknowns. Paper [9] tackles the issue of adaptation when the domain is known but the events are unknown. See also [8], [25] for additional references.

III. Evolution Perspectives

Evolution appears on feature models due to changes from the environment. Changes possibly compete with each other, for example, to do with raising standards. This is also mentioned as *multiple change*, one of the key issues in evolution, identified by Lam and Loomes [19]. The likely winners cannot be identified precisely, but could be foreseen at some levels of certainty based on expert knowledge in the system domain. This uncertainty should be addressed in order to improve the sustainability of the system.

Evolution is usually examined in a determined and finite study period. Evolution can be analyzed at once during this period *i.e.,* how the feature model looks like at the beginning – the *before* model, and how it potentially looks like at the end – the *after* ones. Many *after* models with different likelihood of occurrence are allowed to capture the evolution uncertainty. However, exactly one *after* model will occur in reality. We call this evolution perspective *before-after evolution*.

A different perspective divides the study period into several milestones at which we study how the feature model looks like. At the beginning (milestone t_0) the feature model appears with no change – the *original* feature model. At milestone t_1, the original model could evolve into some $after_1$ models with some probability, similar to the above *before-after* evolution. These models could further evolve into some other $after_2$ models at milestone t_2 and so on. We call this kind of evolution perspective *continuous evolution*.

Fig. 2 visualizes these two kinds of evolution perspective where a feature model is depicted as a cloud. Fig. 2(a) illustrates the *before-after* evolution perspective where a feature model might evolve into other models. Fig. 2(b) exemplifies the *continuous evolution* perspective.

Clearly, the *before-after evolution* perspective is a special case of the *continuous evolution* perspective where there is only a single milestone in the entire study period. Moreover, the long-term usage period of a system in practice is often over years. During such period, many changes might occur, and the winner might depend on the previous ones. The *continuous evolution* perspective thus looks better than its sibling to analyze the evolution in long-term period. Thus we aim to support continuous evolution of feature models.

IV. Modeling The Feature Model Evolution

This section discusses our proposed approach to model the evolution in feature models. The EVE framework [19] categorized changes related to evolution into four types: environmental changes (*e.g.,* the introduction of new laws), requirement changes (*e.g.,* new requirements derived from environmental changes), viewpoint changes (*e.g.,* introduction of a new technology), and design changes (*e.g.,* introduction/removal of a feature). Among those, design changes reflect changes in an original feature model. The first three types of changes are mostly the original cause behind design changes.

Evolution in feature models is captured by two kinds of models: *Evolution Possibility Model (*ePM*)* and *Evolutionary Feature Model (*eFM*)*. The *ePM* captures changes outside a feature model, a.k.a external changes, *i.e.,* environmental changes, requirements changes, and viewpoint changes. These changes will incite design changes in a feature model, which are captured by the *eFM*.

A. Evolution Possibility Model

The Evolution Possibility Model (*ePM*) captures all possible anticipated external changes during a study period, and arranges them into potential situations that might occur in future due to evolution. The *ePM* model enables the traceability

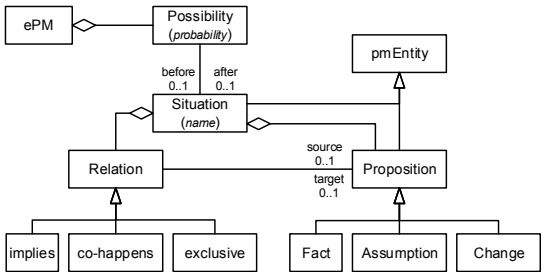


Fig. 3. The meta-model of Evolution Possibility Model.

of external changes and design changes. It helps to answer questions like *"why should this new feature be introduced?"*.

Fig. 3 presents the meta-model for an *ePM*. The *ePM* consists of a set of *Possibilities*. A *Possibility* captures how (external) changes happen. It describes the *Situations* that exist before, and after changes happen. The likelihood that the corresponding changes happen is depicted as the *probability* attribute of the *Possibility*. A *pmEntity* represents an entity (either *Situation* or *Proposition*) in an *ePM* model. It is used to link between an *ePM* model and an *eFM* model. A *Situation* consists of a set of *Propositions* which are either *Facts*, or *Assumptions*, or potential *Changes*. Each *Situation* has a *name* to distinguish itself among others. The *Relation* between these *Propositions* includes: *1) implies* relation denotes a likely causal relation between two *Propositions*. For example, a *Fact*, or an *Assumption* might incite a *Change*; or a *Change* could also incite another *Change*. *2) co-happens* relation denotes that two *Changes* should happen together. *3) exclusive* relation denotes the mutual exclusion between two *Changes*. If one change happens, the other ones will not.

The situation where no change exists is referred to as *before* situation, and the others are referred to as *after* situations. For every situation, we could always construct a corresponding feature model. The feature models of *before*, and *after* situations are respectively referred to as *before*, and *after* feature model. Hereafter we abuse the name of a situation for its corresponding feature model, and versa.

In order to formally express the evolution possibilities, we adopt the *observable rule* proposed in [30] for dealing with requirements evolution. It has been adopted to address the evolution of risks in long-lived software system [29].

Let F be a situation, and F_i be $after_i$ situations which F might evolve into. The observable rule $r_o$of F is below:

$$r_o(F) = \left\{ F \xrightarrow{p_i} F_i \;\middle|\; \sum_{i=1}^{n} p_i = 1 \right\} \quad (1)$$

where p_i is the *evolution probability* that F evolves into F_i. We assume all known possibilities are anticipated and mutually exclusive. Additionally, we ignore all unknown possibilities. Therefore the sum of all p_i equals 1.

Loosely speaking, *ePM* could be seen as a collection of observable rules where each situation has at most one observable rule. In an observable rule, we call the situation

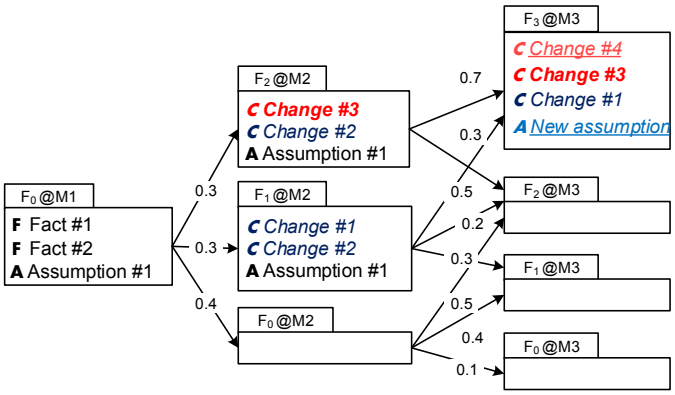


Fig. 4. An example of Evolution Possibility Model

on the left hand side of the arrow as *before* situation, and the ones on the right hand side as *after* situations. A situation could be an *after* situation in an observable rule, and could be a *before* situation in another observable rule. The situation which is not an *after* situation in any observable rules is also referred to as the *original* situation. Similarly, the situations which are not *before* situations in any rules are referred to as *leaf* situations. We additionally use the empty set notation ($\emptyset$) to denote the observable rules of *leaf* situations as follows:

$$r_o(F) = \emptyset \Leftrightarrow F \text{ is a leaf situation} \quad (2)$$

Example 1. Fig. 4 illustrates an example of *ePM*. In this figure, situations are illustrated as folder shapes where situation names are on the top, followed by the milestones, the situation's propositions are located inside. An evolution possibility is depicted by an arrow connecting a *before* situation to an *after* situation. Evolution probability is a label decorated to the arrow. To keep the *ePM* simple, this example does not consider relations among propositions, but presents them as a list. In this list, the first letter denotes the proposition type, *i.e.,* F – fact, A – assumption, C – change. The original situation is $F_0@M1$, and the leaf situations are $F_i@M3$ ■

B. Evolutionary Feature Model

The Evolutionary Feature Model (*eFM*) captures all feature models in all situations of the *ePM* model. The *eFM* incorporates all design changes due to evolution. Syntactically, the *eFM* is an attributed feature model which is enriched with a new relation, namely *traces*, connecting an entity in *ePM* (*i.e.,* a situation, or a proposition) to an entity (*i.e.,* a feature, or a feature attribute, or a complex constraint) in *eFM*. By modeling this way, design changes made to a feature model are directly or indirectly associated with situations in *ePM*.

Fig. 5 presents the meta-model of the *eFM*. The meta-model includes notions for a basic attributed feature model, and a new type of relation – *traces*. An *eFM* consists of a set of *Elements*. An *Element* could be either an *fmEntity*, or an *fmRelation*, or a *traces* relation. An *fmEntity* represents a *Feature*, or an

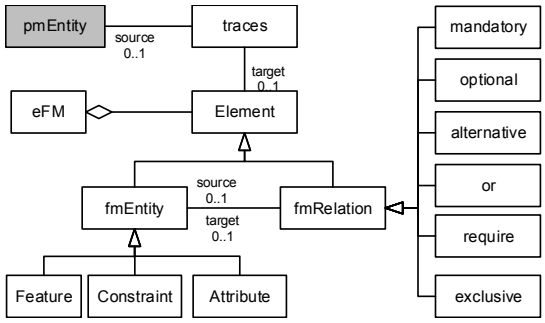


Fig. 5. The meta-model of Evolutionary Feature Model.

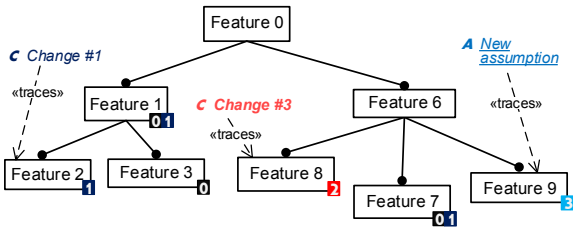


Fig. 6. An example of Evolutionary Feature Model (*eFM*).

Attribute of a feature, or a complex *Constraint*. A *Relation* denotes one of basic relations discussed in Section II. A *traces* relation connecting a *pmEntity* of an *ePM* to an *fmEntity* to denote the situation that an *fmEntity* belongs to.

The feature model of a given situation in *ePM* could be generated easily from the *eFM* by filtering out all entities and their related relations which are explicitly marked as belonging to other situations, but not the given situation.

Example 2. Fig. 6 exemplifies an *eFM* model which incorporates all design changes due to external changes presented in the corresponding *ePM* in Fig. 4. Dashed arrows with label *traces* represent the link from external changes in *ePM* to elements in *eFM*. Elements belonging to some (not all) situations are decorated with small solid boxes with situation's index inside, *i.e.,* **0–3** stand for $F_0 - F_3$ in all milestones. Other elements without such decoration belong to all situations. The F_0 will include features 0, 1, 3, 6, and 7; and F_1 will include features 0, 1, 2, 6, 7. ■

Even though an *eFM* will increase the complexity of the feature model, we still adopt it due to its following advantages:

- Intuitively, the *before* and *after* feature models share a large portion of the features. These shared features need to be replicated among models. Any modifications to the shared part consequently need to be replicated in all models, which may be costly and error-prone. The *eFM* merges *before* and *after* models in a single one, thus it avoids this replication problem.
- The *eFM* captures all design changes in a single model which facilitates a global view of the evolution, and the difference of design changes among situations.

V. Decision Support on Feature Model Evolution

This section describes the analyses that exploit *ePM* and *eFM* to provide more information about the survivability and cost of repair of a configuration.

A. Survivability Analysis

The survivability analysis measures whether a configuration could be still operational without any modifications despite of the occurrence of evolution. The analysis computes the following quantitative metrics: Max Belief, Residual Disbelief, and Max Disbelief. The first two metrics are based on our previous work on requirements evolution [30] in order to assess a configuration at a point in time.

- *Max Belief (MB)* is a time series measuring the maximum belief that a configuration will still be a valid configuration represented by any *after* feature models at certain milestone t within the study period.
- *Residual Disbelief (RD)* is a time series of the complements of total belief that a configuration will still be a valid configuration represented by any *after* feature models at certain milestone t.
- *Max Disbelief (MD)* is a time series measuring the maximum belief that a configuration will not be a valid configuration represented by any *after* feature models at certain milestone t.

To facilitate the formulation of these metrics, we employ the operation valid [2] that takes a feature model and a configuration as inputs and returns 1 if the configuration is represented by the feature model, 0 otherwise.

$$\mathsf{valid}(F, C) = \begin{cases} 1 & \text{if } C \text{ is represented by } F \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

We define an operation $\mathsf{validpos}$ that takes a feature model and a configuration as inputs, and returns a set of evolution possibilities where the given configuration is valid in *after* model evolution:

$$\mathsf{validpos}(F, C) = \left\{ F \xrightarrow{p_i} F_i \in r_o(F) \,|\mathsf{valid}(F_i, C) = 1 \right\} \tag{4}$$

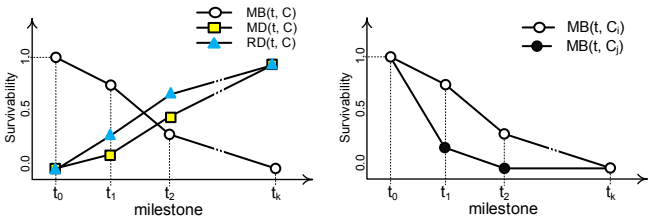
We further define an operation age, which takes a feature model and returns the milestone when the feature model supposes to be. The formula of age is as follows:

$$\mathsf{age}(F) = \begin{cases} 1 + \mathsf{age}(F') & \text{if } \exists F' : \langle F' \xrightarrow{p} F \rangle \in r_o(F') \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The cumulative values of Max Belief, Residual Disbelief, Max Disbelief at milestone t for a configuration C, given a feature model F, are depicted as follows:

$$MB(t, C|F) = \begin{cases} \mathsf{valid}(F, C) & \text{if } \mathsf{stable}(F, t) \\ \max\limits_{F \xrightarrow{p_i} F_i \in \mathsf{validpos}(F,C)} p_i \cdot MB(t, C|F_i) & \text{otherwise} \end{cases} \tag{6}$$

$$RD(t, C|F) = \begin{cases} 1 - \mathsf{valid}(F, C) & \text{if } \mathsf{stable}(F, t) \\ 1 - \sum\limits_{F \xrightarrow{p_i} F_i \in \mathsf{validpos}(F,C)} p_i \cdot (1 - RD(t, C|F_i)) & \text{otherwise} \end{cases} \tag{7}$$



(a) One configuration (b) *MB* of two configurations

Fig. 7. The survivability diagram.

$$MD(t, C|F) = \begin{cases} \mathsf{valid}(F, C) & \text{if } \mathsf{stable}(F, t) \\ \max\limits_{F \xrightarrow{p_i} F_i \notin \mathsf{validpos}(F,C)} p_i \cdot MD(t, C|F_i) & \text{otherwise} \end{cases} \tag{8}$$

where $\mathsf{stable}(F, t) = (r_o(F) = \emptyset \vee \mathsf{age}(F) \geq t)$.

The equation (6) means that, if the given F is stable before milestone t, the validity of the given configuration C within F does not change. As the result, the Max Belief of C in F, captured by as $MB(t, C|F)$, is determined by the validity of C in F. If F evolves, its evolution is denoted in the observable rule $r_o(F)$, the $MB(t, C|F)$ is the maximum belief p_i of all possibilities where C is a valid configuration in their *after* models, multiplied by the Max Belief of C in these *after* models if they continue to evolve. The ideas behind (7), (8) are similarly explained.

Given that F_0 is the original feature model, the time series of Max Belief, Residual Disbelief, and Max Disbelief for a varying t are defined as follows:

$$MB(t, C) = MB(t, C|F_0) \tag{9}$$

$$RD(t, C) = RD(t, C|F_0) \tag{10}$$

$$MD(t, C) = MD(t, C|F_0) \tag{11}$$

These time series could be plotted on a diagram called *Survivability Diagram*, see Fig. 7, showing the development of these series of a configuration over time. In Fig. 7(a) at milestone t_0, an arbitrary configuration C is valid in F_0, its $MB(t_0, C)$ is obviously 100%, whereas its $RD(t_0, C)$, and $MD(t_0, C)$ are zero. With subsequent milestones t_i, when evolution occurs the value of $MB(t_i, C)$ might decrease, while $RD(t_i, C)$, and $MD(t_i, C)$ might increase.

The survivability diagram can contain only one or two series of the above metrics of two or more configurations, see Fig. 7(b) for the Max Belief of two configurations C_i and C_j. Such information could provide extra visual information to support the decision making process.

The above metrics can be used to relatively compare among configurations. We compare configurations based on individual metrics. In particular, the comparison between configurations using Max Belief follows the rule: *the higher Max Belief, the better configuration*. Meanwhile, the comparison using *RD* and *MD* is done in the opposite: *the lower Residual Disbelief/Max Disbelief, the better configuration*.

B. Repair Cost Analysis

Few configurations could survive from the beginning to the very end of the study period. Such configurations might require a large investment on features that are only required in later milestones. Hence, decision makers have to consider a trade-off between two strategies: whether to implement all features at the beginning, or postpone some features later. To support such decision, this section provides the analysis on cost of reparation which takes into account the expense to repair an invalid configuration at a certain milestone.

We define two operations, namely repcst and repair, that both take a feature model and a configuration as inputs. The former returns a value, called *repair cost*, to represent the minimum cost to make the given configuration become a valid one represented by the given feature model. The latter returns a set of repaired configurations, which have minimal costs.

A straightforward definition of these operators is to enumerate all possible configurations of the input feature model and compute the minimum. Such enumeration has been supported by many studies with automated implementations (see Table 3 in [2]). We also use it because it makes the formal definition easier to grasp. Given a set of features C and a feature model F, let $\mathcal{C}_F$ be the set of all configurations of F. We select configurations C' in $\mathcal{C}_F$ such that the cost to migrate C to C' (*e.g.,* cost to implement new features) is minimized. The formulation of repcst and repair is as follows:

$$\mathsf{repcst}(F,C) = \min\{\overbrace{\mathsf{cst}(C_i \backslash C)}^{\text{cost of adding new features}} + \underbrace{\mathsf{cst}'(C \backslash C_i)}_{\text{cost of removing obsoleted features}} | C_i \in \mathcal{C}_F\} \quad (12)$$

$$\mathsf{repair}(F,C) = \mathrm{argmin}_{C_i \in \mathcal{C}_F}(\mathsf{cst}(C_i \backslash C) + \mathsf{cst}'(C \backslash C_i)) \quad (13)$$

where $\mathsf{cst}(C_i \backslash C)$ denotes the cost of adding new features which are in a valid configuration, but not in the given one; $\mathsf{cst}'(C \backslash C_i)$ is the cost of removing obsoleted features in the given configuration, but not in a valid one. For the actual implementation, a more efficient result is obtained by computing directly the minimal number of fixes for the configurations [2]. This can be done with local search algorithms.

The repair cost analysis measures the weighted average reparation costs to repair a configuration due to evolution within the study period. The analysis takes into account all costs to repair a configuration in *after* situations of possibilities in observable rules, and averages them with weights that are equal to the probabilities of the evolution possibilities. The outcome is a time series representing the cumulative reparation cost of a configuration at every milestone.

Let F be a feature model, we define the Cost-of-Reparation (*CoR*) of a configuration C at a certain milestone t as follows:

$$\mathit{CoR}(t, C|F) = \begin{cases} \mathsf{repcst}(F,C) & \text{if } \mathsf{stable}(F,t) \\ \mathit{CoR}'(t, C|F) & \text{otherwise} \end{cases} \quad (14)$$

$$\mathit{CoR}'(t, C|F) = \sum_{F \xrightarrow{p_i} F_i \in r_o(F)} p_i \cdot \min_{C_j \in \mathsf{repair}(F_i, C)} (\mathsf{repcst}(F_i, C) + \mathit{CoR}(t, C \cup C_j | F_i))$$

Given that F_0 is the original feature model, the Cost-of-Reparation of a configuration C at a certain milestone t is defined as follows:

$$\mathit{CoR}(t, C) = \mathit{CoR}(t, C|F_0) \quad (15)$$

By considering both cost-of-reparation and the initial implementation cost of each configuration, we can calculate the cumulated implementation cost of a configuration at each milestone as follows:

$$\mathit{Cost}(t, C) = \begin{cases} \textit{initial implementation cost} & \text{if } t = 0 \\ \mathit{Cost}(t-1, C) + \mathit{CoR}(t, C) & \text{otherwise} \end{cases} \quad (16)$$

The plot of cumulated implementation cost is referred to as *Cost Diagram* which shows the development of the implementation cost of configuration. We do not give example of Cost diagram here but in the next section. This diagram and *Survivability Diagram* previously discussed could be useful hints to facilitate the configuration selection.

VI. Application of the Proposed Approach

This section exemplifies the proposed approach in the Smart Grid scenario taken from the NESSoS European project. The evolution described in this section is real, and is taken from the Smart Grid evolution road maps [21], [22]. The evolution probabilities are invented for the illustrative purpose.

A. The Smart Grid Scenario

According to the European Technology Platform [10], the Smart Grid is "an electricity network that can intelligently integrate the actions of all users connected to it–generators, consumers and those that do both–in order to efficiently deliver sustainable, economic and secure electricity supplies". A Smart Grid uses ICT technologies and operates in parallel with an electric power grid to optimize the transmission and distribution of electricity. According to [15], the evolution of Smart Grid has three milestones. The future infrastructure will be based on one-way Automated Meter Reading (AMR) technology, which allows utilities to remotely collect meter data. Next, AMR is replaced or supplemented by the Automated Metering Infrastructure (AMI) to enable utilities to remotely change parameters inside meters, or turn on/off switches. Finally, customers are allowed to actively choose appropriate price options, or utilities that best suit their needs. Different European countries have started to roll out the different capabilities of the Smart Grid infrastructure. For example Britain is currently discussing the roll out of AMR technology. Italy has already rolled out AMR and the authors own buildings have operational elements of AMI allowing utilities to switch off electricity if consumptions exceed a contractual threshold by 20%.

This scenario is very appropriate to capture software product lines: software for different components of the Smart Grid must be customized and shipped based on present needs (*e.g.,* a metering software using 3G connection features versus other network features) but be able to cope with future changes.

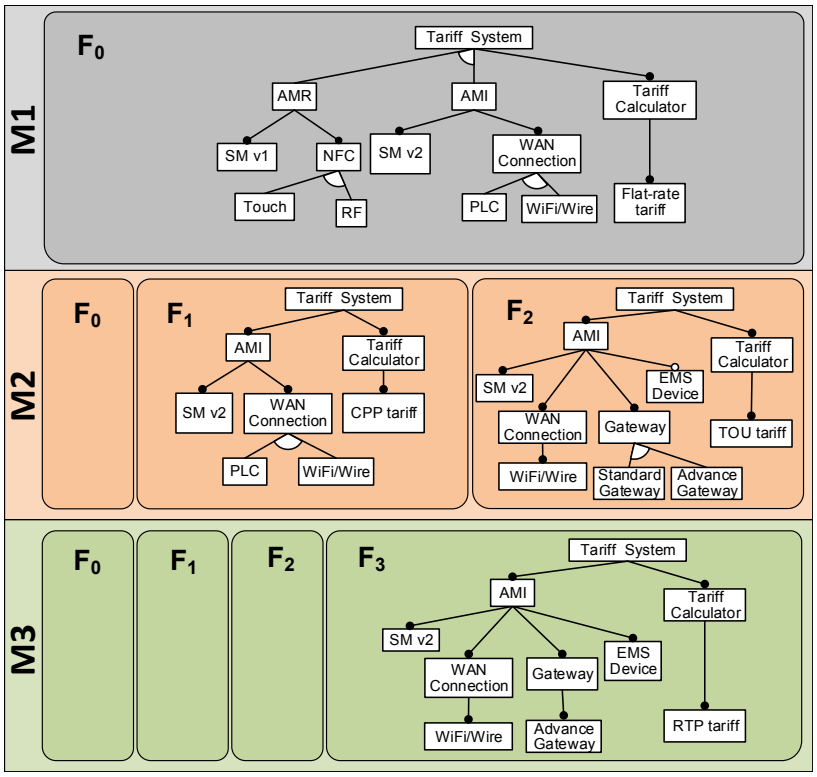


Fig. 8. The evolution of the tariff system in the Smart Grid scenario.

To save space, we do not repeat the feature models F_0, F_1, F_2 in M2, M3. Instead, we use round rectangles with label on top to represent the existence of these models.

Acronyms in the diagram: AMR–Automated Meter Reading, AMI–Automated Metering Infrastructure, SM v1–Smart Meter version 1, SM v2–Smart Meter version 2, NFC–Near Field Connection, EMS–Energy Management System, RF–Radio Frequency, PLC–Power Line Connection, WAN–Wide Area Network, TOU–Time-of-Use, CPP–Critial-Peak-Period, RTP–Real-time Period.

Here we focus on the evolution of the tariff system that calculates electric invoices based on collected meter data. The time frame is divided into three milestones: M1, M2, and M3. The evolution of the feature model representing the tariff system is exhibited in Fig. 8, which is described below.

- *Milestone M1*. Customers could only choose a utility at the beginning. The meter data will be monthly sent to the utility automatically. The flat-rate tariff is applied, which mean the same price applied for each unit of consumption. For this purpose, either AMR or AMI could be deployed. The AMR requires smart meter with near-field communication technology such as Touch, or Radio Frequency (RF) to send meter data. The AMI requires advanced smart meter which uses either Power Line Connection (PLC) or other Wide Area Network (WAN) connection to send data. The feature model represents this milestone is F_0.
- *Milestone M2*. At this milestone, the tariff system could stay the same as M1 *i.e.,* F_0. It could also evolve into F_1. Particularly, more fine-grain tariff could be applied, *i.e.,* Critial-Peak-Period (CPP) where the higher peak-price is restricted to a small number of peak-hours per year, and much higher price for other peak hours. For this purpose, more meter data will be sent to utility in fine-grain intervals. This requires some functionalities of the AMI infrastructure. F_0 could evolve into F_2 which is very similar to F_1, except a new security requirement to protect customer data arises, and Time-of-Use (TOU) tariff is applied. In this tariff, prices vary between peak- and off-peak hours. The security requirement requires a home Gateway system to be deployed.
- *Milestone M3*. Similarly, all previous feature models $F_0 - F_2$ have some chances to be the case in this milestone. Besides, the tariff system could evolve from either F_1 or F_2 into F_3. In F_3, there are more utilities in the market, enabling customers to actively choose the utilities as well as different payment options. The tariff calculation is now based on real-time usage of meter data.

B. *Construct the* ePM *and the* eFM

Fig. 9 presents the *ePM* of the tariff system of the Smart Grid scenario. Similar to Fig. 4, situations are illustrated as folder shapes where situation names are on the top, followed by the milestones. The propositions are located inside each situation.

The observable rule of the original situation F_0 at milestone M1 is described as follows:

$$r_o(F_0@M1) = \Big\{F_0@M1 \xrightarrow{0.3} F_0@M2,\ F_0@M1 \xrightarrow{0.4} F_1@M2,\ F_0@M1 \xrightarrow{0.3} F_2@M2\Big\}$$

Fig. 10 shows the *eFM* model of the tariff system which incorporates all design changes due to external changes presented in the corresponding *ePM* in Fig. 9.

C. *Perform Survivability Analysis*

Table I reports different configurations of the Smart Grid tariff system. Due to the lack of space, we do not report parent features in a configuration. They could be inferred by their

Fig. 9. The *ePM* of the tariff system of the Smart Grid.

Situations are illustrated as folder shapes where situation names F_i followed by the milestones $@M_i$. The situation's propositions are located inside and denoted as **F** – Fact, **A** – Assumption, **C** – Change. An evolution possibility is depicted by an arrow, with probability attached, to connect a *before* situation to an *after* situation.

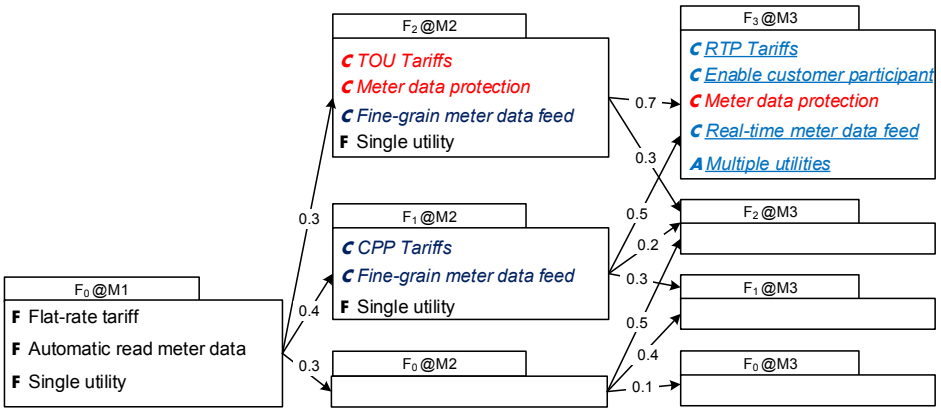


Fig. 10. The *eFM* of the tariff system of the Smart Grid.

Dashed arrows with label *traces* represent the link from external changes in *ePM* to elements in *eFM*. Elements that belong to some (not all) situations are decorated with small solid boxes with situation's index inside, *i.e.,* **0–3** stand for $F_0 - F_3$ in all milestones. Other elements without such decoration belong to all situations.

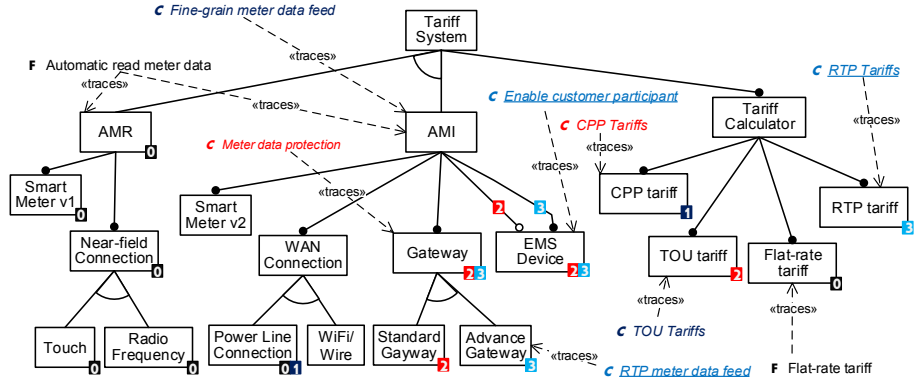


Table I
SURVIVABILITY METRICS FOR THE CONFIGURATIONS OF THE SMART GRID TARIFF SYSTEM.

Configuration(*)	Valid in F_0	F_1	F_2	F_3	M2 *MB*	M2 *RD*	M2 *MD*	M3 *MB*	M3 *RD*	M3 *MD*
C1 {SMv1,RF,Flat-rate tariff}	✓				0.30	0.70	0.40	0.03	0.97	0.21
C2 {SMv2,Wifi,Flat-rate tariff}	✓				0.30	0.70	0.40	0.03	0.97	0.21
C3 {SMv2,WiFi,CPP tariff,Flat-rate tariff}	✓	✓			**0.40**	0.30	0.30	0.12	0.73	0.21
C4 {SMv2,WiFi,Standard Gateway,TOU tariff, Flat-rate tariff}	✓		✓		0.30	0.40	0.40	0.15	0.73	0.21
C5 {SMv2,WiFi,Standard Gateway,TOU tariff, CPP tariff, Flat-rate tariff}	✓	✓	✓		**0.40**	**0.00**	0.00	0.15	0.41	0.21
C6 {SMv2,WiFi,Advance Gateway,EMS,RTP tariff,TOU tariff, CPP tariff,Flat-rate tariff}	✓	✓	✓	✓	**0.40**	**0.00**	0.00	**0.21**	**0.00**	0.00

(*): To save space we do not show parent features in the configurations.

selected children *e.g.,* the configuration $C1$ in Table I is fully reported as {Tariff System, AMR, SMv1, NFC, RF, Tariff Calculator, Flat-rate tariff}. For each configuration, the table reports situations where it is valid, and its survivability values at milestone M2 and M3. The survivability metrics at M1 are the same for all $C1$–$C6$, *i.e., MB* $= 100\%$ and *RD* $=$ *MD* $= 0\%$, and are not reported. The survivability diagrams of these configurations of Smart Grid are reported in Fig. 11.

The survivability analysis shows that both $C5$ and $C6$ are two winners at milestone M2 because they both have the highest *MB*, and the lowest *RD* and *MD*. In a longer term – milestone M3, $C6$ is the only winner. For other configurations, $C3$ is better than $C4$ at M2, but it is worse than $C4$ at M3. Both $C1$ and $C2$ are equivalent.

D. Perform Repair Cost Analysis

To illustrate the cost calculation, we simplify the cost of each feature as 1 unit; and assume the cost of removing obsoleted features while repairing a configuration is zero. For actual costs and more detailed information, one can consult [13] and a number of websites[1]. We assume an extra 20% of costs for each late implementation of a feature on a deployed system.

Fig. 12 reports the cost diagram for the configurations listed in Table I. This figure, instead of reporting absolute numbers, shows the normalized value to the baseline, which is the smallest value of configurations' costs at the first milestone. In this scenario, the cost of $C1$ (or $C2$) is chosen as the baseline.

[1]www.smartgrids.eu, www.netw.doe.gov, www.supergen-networks.org.uk

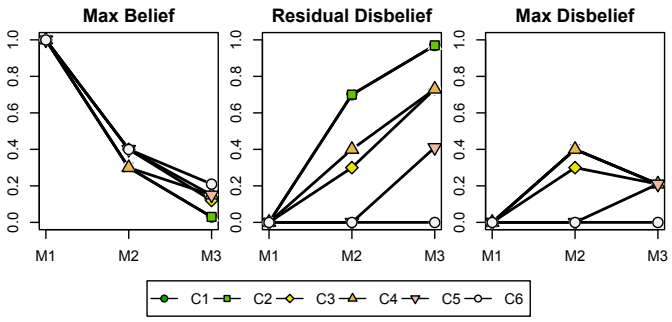
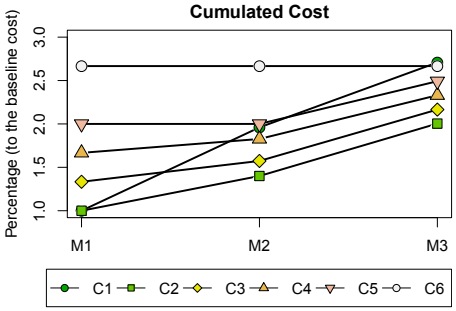


Fig. 11. Survivability diagram for Smart Grid configurations



(a) Cumulated cost diagram

	M1		M2		M3
Configuration	Cost	CoR	Cost	CoR	Cost
C1	**1.00**	0.96	1.96	0.75	2.71
C2	**1.00**	0.40	**1.40**	0.60	**2.00**
C3	1.33	0.24	1.57	0.59	2.17
C4	1.67	0.16	1.83	0.50	2.33
C5	2.00	0.00	2.00	0.49	2.49
C6	2.67	0.00	2.67	0.00	2.67

(b) Data table

All costs are normalized to the baseline value, which is the smallest cost of all configurations at the first milestone M1.

Fig. 12. The cost diagram for configurations in Table I.

The medium expensive configuration is $C3$ whose cost is 1.33 times higher than $C1$'s; and the most expensive configuration is $C6$ whose cost is 2.67 times higher than $C1$'s.

The diagram shows that universal configurations which try to address all possible evolution in advance are not necessary good choices. Such configurations are $C5$ and $C6$ (see Table I). These configurations are winners with respect to the survivability analysis in Section VI-C because they anticipate more potential changes in advance than others. The costs for these configurations are always among the highest. In contrast, a naive configuration *i.e.,* $C1$ which does not take into account evolution may cost less at the beginning, but might turn to be the most expensive at the end. The configuration $C2$ is the most appropriate choice because it has the smallest pace of cost development during the entire study period.

VII. Discussion

Applicability. The applicability of the proposed approach depends on the ability to elicit potential evolution as well as evolution possibilities. Both could be elicited with the aid of domain knowledge. The Smart Grid scenario, and our previous work [30] are examples where evolution could be anticipated. For eliciting probabilities, we can employ a classical approach by Boehm [5] in software risk estimation; a more recent survey can be found in Khan *et al.* [18]. Different approaches can be used to elicit such estimates like use case diagrams [24], lists and questionnaires. Different opinions collected from stakeholders can be combined using ranking [3], analytical hierarchy processes [16], or cumulative sorting [4].

Our approach lacks an empirical validation with external experts. There are two ways of doing it. At first, one could analyze an existing, possibly open-source, system for historical data, then asks developers of the system to assess their decisions in the past. A second alternative is to engage a group of experts and ask their opinion on the input parameters such as probabilities. Then the validity of the prediction results could be validated by them. We leave these for future work.

Scalability. The scalability of the approach is in two folds: the ability to capture and manage big feature models, and the ability to perform reasoning on big feature models. The former fold is a pure technical problem which mostly depends on capability the GUI tool to divide a big model into several diagrams, such as our previous work to manage large model in requirements evolution [31]. The latter fold concerns the scalability of the proposed analyses.

The scalability of the survivability analysis mostly relies on the operator valid which has been implemented efficiently in past studies [32], [11].

Similarly, the scalability of the repair cost analysis relies on two operators repair and repcst. From hypergraph theory [1] we know that polynomial time algorithms exist for *MB* or alternatively for *MD* because they are monotone functions. However, this is not true for *RD* which may require exponential time. So, an alternative would be to approximate the results by using only *MB* (or *MD*). The naive implementations for these operators, see (13) and (12), require to enumerate all possible configurations of a feature model. This could be computationally expensive, especially when a feature model consists of a huge amount of features. More efficient approaches could be obtained by adopting artificial intelligence planning techniques. Two promising approaches were proposed by Soltani *et al.* [27] and Ernst *et al.* [9]. The former generates a configuration for a feature model given a set of core features, and set of non-functional preferences and constraints (*e.g.,* minimize of implementation cost). The latter studied a class of algorithms using AI Trust Maintenance Systems to find new configurations that use as much as possible of the old configuration, and minimize the number of new features that need to be implemented.

VIII. Conclusion

In this work we have addressed the challenge of dealing with feature model evolution. We have proposed a modeling approach to capture the uncertainty of feature model evolution in terms of two models: Evolution Possibility Model and Evolutionary Feature Model. The former captures different possibilities that a feature model could evolve. The latter captures all changes made to the original feature model due to evolution. The proposed approach allows users to study the continuous evolution of a feature model in several milestones.

We also develop two analyses that exploit the uncertainty of evolution. The first analysis aims to study to what extent a configuration (*i.e.,* set of features to implement) could be resilient to the evolution within the study period. The second analysis aims to understand the development of reparation cost to repair a configuration which is invalid due to evolution. With these analyses, we aim to support the decision makers in selecting an 'optimal' configuration regards to the evolution.

Acknowledgement

This work is partly supported by EU-FP7-IST-NoE-NESSOS, and EMFASE. We would like to thank the anonymous reviewers for useful comments.

References

[1] G. Ausiello, P. Franciosa, and D. Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. In *Theoretical Computer Science*, volume 2202 of *LNCS*, pages 312–328. Springer, 2001.

[2] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems*, 35(6):615–636, 2010.

[3] P. Berander and A. Andrews. Requirements prioritization. In *Engineering and Managing Software Requirements*, pages 69–94. Springer, 2005.

[4] P. Berander and M. Svahnberg. Evaluating two ways of calculating priorities in requirements hierarchies - an experiment on hierarchical cumulative voting. *Journal of Systems and Software*, 82(5):836–850, 2009.

[5] B. W. Boehm. Software risk management: Principles and practices. *IEEE Software*, 8(1):32–41, 1991.

[6] J. Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Software Product Lines*, volume 2379 of *LNCS*, pages 257–271. Springer, 2002.

[7] G. Botterweck, A. Pleuss, D. Dhungana, A. Polzer, and S. Kowalewski. EvoFM: Feature-driven Planning of Product-line Evolution. In *Proc. of Product Line Approaches in Software Engineering*, pages 24–31, 2010.

[8] B. H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*, volume 5525 of *LNCS*, pages 1–26. Springer, 2009.

[9] N. A. Ernst, A. Borgida, and I. Jureta. Finding incremental solutions for evolving requirements. In *Proc. of the 19th IEEE Intl. Requirements Engineering Conference (RE)*, pages 15–24, 2011.

[10] European Commission. European smart grids technology platform: vision and strategy for Europes electricity. Technical report, 2006.

[11] D. Fernandez-Amoros, R. H. Gil, and J. C. Somolinos. Inferring information from feature diagrams to product line economic models. In *Proc. of 13th Intl. Software Product Line Conference (SPLC)*, pages 41–50, 2009.

[12] N. Gamez and L. Fuentes. Software product line evolution with cardinality-based feature models. In *Top Productivity through Software Reuse*, volume 6727, pages 102–118. 2011.

[13] C. W. Gellings. *The Smart Grid: Enabling Energy Efficiency and Demand Response*. The Fairmont Press, Inc., 2009.

[14] J. Guo, Y. Wang, P. Trinidad, and D. Benavides. Consistency maintenance for evolving feature models. *Expert Systems with Applications*, 39(5):4987 – 4998, 2012.

[15] F. Hassan. The path of the smart grid. *IEEE Power and Energy Magazine*, 8(1):18–28, 2010.

[16] S.-M. Huang, I.-C. Chang, S.-H. Li, and M.-T. Lin. Assessing risk in erp projects: identify and prioritize the factors. *Industrial Management and Data Systems*, 104(8):681–688, 2004.

[17] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-Oriented domain analysis (FODA) feasibility study. Technical report, CMU/SEI-90-TR-21, Carnegie Mellon University, 1990.

[18] M. A. Khan, S. Khan, and M. Sadiq. Systematic review of software risk assessment and estimation models. *Intl. Journal of Engineering and Advanced Technology (IJEAT)*, 1(4):298–305, 2012.

[19] W. Lam and M. Loomes. Requirements evolution in the midst of environmental change: a managed approach. In *Proc. of the 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR)*, pages 121–127, 1998.

[20] J. Momoh. *Smart Grid: Fundamentals of Design and Analysis*. Wiley-IEEE Press, 2012.

[21] National Energy Technology Laboratory. A vision for the smart grid. Technical report, 2009.

[22] NESSoS project. Selection and documentation of the two major application case studies. NESSoS Deliverable 11.2, 2011.

[23] A. Pleuss, G. Botterweck, D. Dhungana, A. Polzer, and S. Kowalewski. Model-driven support for product line evolution on feature level. *Journal of Systems and Software*, 85(10):2261–2274, 2012.

[24] M. Sadiq, M. Rahmani, M. Ahmad, and S. Jung. Software risk assessment and evaluation process (SRAEP) using model based approach. In *Proc. of Intl. Conference on Networking and Information Technology (ICNIT)*, pages 171–177, 2010.

[25] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-aware systems: A research agenda for re for self-adaptive systems. In *Proc. of 18th IEEE Intl. Requirements Engineering Conference (RE)*, pages 95–103, 2010.

[26] S. Schach and A. Tomer. Development/maintenance/reuse: Software evolution in product lines. In *Software Product Lines*, volume 576 of *Engineering and Computer Science*, pages 437–450. Springer, 2000.

[27] S. Soltani, M. Asadi, D. Gašević, M. Hatala, and E. Bagheri. Automated planning for feature model configuration based on functional and non-functional requirements. In *Proc. of 16th Intl. Software Product Line Conference (SPLC)*, pages 56–65, 2012.

[28] T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *Proc. of the 31st Intl. Conference on Software Engineering (ICSE)*, pages 254–264, 2009.

[29] L. M. S. Tran. Early dealing with evolving risks in long-life evolving software systems. In *Proc. of Advanced Information Systems Engineering Workshops – CAiSE Workshops*, pages 518–523, 2013.

[30] L. M. S. Tran and F. Massacci. Dealing with known unknowns: Towards a game-theoretic foundation for software requirement evolution. In *Proc. of 23rd Intl. Conference on Advanced Information Systems Engineering (CAiSE)*, pages 62–76, 2011.

[31] L. M. S. Tran and F. Massacci. Unicorn: A tool for modeling and reasoning on the uncertainty of requirements evolution. In *Proc. of CAiSE Forum*, pages 161–168, 2013.

[32] J. White, D. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortes. Automated diagnosis of product-line configuration errors in feature models. In *Proc. of 12th Intl. Software Product Line Conference (SPLC)*, pages 225–234, 2008.

[33] H. Ye and W. Zhang. Formal definition of feature models to support software product line evolution. In *Proc. of Software Engineering Research and Practice*, pages 349–355, 2008.