Received: 25 November 2019 | Revised: 4 March 2020 | Accepted: 8 March 2020

DOI: 10.1002/smr.2262

RESEARCH ARTICLE - METHODOLOGY

Journal of Software: Evolution and Process WILEY

# Pattern-based software process modeling for dependability

Xuan Zhang[1,2] | Xu Wang[3] | Wei Yun[1] | Chen Gao[1] | Mengting Han[1] | Hui Liu[1]

[1]School of Software, Yunnan University, Kunming, Yunnan, China

[2]Key Laboratory of Software Engineering of Yunnan, Kunming, Yunnan, China

[3]School of Economics, Yunnan University, Kunming, Yunnan, China

**Correspondence**
Xuan Zhang, School of Software, Yunnan University, Kunming, Yunnan 650091, China.
Email: zhxuan@ynu.edu.cn

**Funding information**
Data Driven Software Engineering Innovative Research Team Funding of Yunnan Province, Grant/Award Number: 2017HC012; Science Foundation of Key Laboratory of Software Engineering of Yunnan Province, Grant/Award Number: 2015SE202; Natural Science Foundation of Yunnan Province, Grant/Award Number: 2016FB106; National Social Science Foundation of China, Grant/Award Number: 18BJL104; National Natural Science Foundation of China, Grant/Award Numbers: 61262025, 61502413, 61862063

## Abstract

Traditional process modeling focuses on modeling activities for functional requirements. For dependability requirements, a knowledge-based aspect-oriented software process modeling approach is proposed. First, we extend the pattern and context to the knowledge graph triplet structure to describe dependability-oriented knowledge patterns. By applying the patterns, dependability requirements can be organized into dependability-related activities that are integrated into the software process. Then, aspect-oriented modeling patterns based on Petri nets are introduced to support the integration of these dependability-related activities and model dependability-oriented software processes. Finally, the modeling performance and the subjective usability of the patterns are evaluated by 110 students with different degrees. The results indicate that these two indexes are on the positive track. Hence, the patterns may be the backbone of dependability-oriented software process modeling.

**KEYWORDS**

aspect-oriented modeling, dependability, knowledge graph, pattern, software process modeling

## 1 | INTRODUCTION

In the information industry, software plays an important role. The ARC advisory group, a respected consultancy and trends observer in industry, indicated that global megatrends and technology trends will drive software expenditure growth.[1] The growing ratio of software development in the costs of machinery is one of the evidences. The German Engineering Federation VDMA presented that the ratio of software had doubled in one decade[2] from 20% to 40%. Also, most of the products are software intensive. The functionality and quality of these products are largely determined by the software.[2] Facing this trend, many contributions have been made in software engineering. On the basis of Vyatkin's survey,[2] the development of software engineering areas in industry has been mostly concerned with improvement of software life cycle efficiency and dependability.

The dependability of the software is particularly important. On the basis of the International Federation for Information Processing Working Group 10.4,* Vyatkin,[2] Avižienis et al,[3] and Jin,[4] dependability is defined as the trustworthiness of a computing system, which allows reliance to be justifiably placed on the service it delivers. It encompasses a set of quality attributes, such as availability, reliability, safety, security, maintainability, and privacy. In our previous work,[5] we further proposed that functional requirements must be strictly implemented as the most critical requirements for dependability. Only when the functional requirements are strictly implemented, dependability requirements can be further realized. Facing the dependability requirements, two possible approaches exist. In the first approach, the emphasis is on techniques for developing and examining software directly. In the second approach, the emphasis is on the processes used to create these software products. The primary

*http://wg10.4.dependability.org/

benefit of focusing on techniques is that abundant useful information that is familiar to most programmers, engineers, and managers can be easily obtained. However, according to Pressman,[6] process is the glue that holds techniques together and enables rational and timely development of computer software. In addition, a process emphasis allows for the reuse of valuable previous experiences.[5] The literature is abundant about the weaknesses that emerge from jumping to code without having any straightforward modeling phase.[7] Thereby, in this paper, from the process perspective, an approach to modeling software process for dependable software is presented.

For dependability requirements, relevant knowledge needs to be accumulated to help dependability-oriented software process modeling. To assist modeling, the use of knowledge base helps model engineers to complete the model construction with high quality. In this paper, we propose to extend semantic knowledge structure as a general schema to describe dependability-oriented knowledge patterns. By applying the patterns, dependability requirements can be organized into dependability-related activities that are integrated into software processes. Then, an approach is proposed to modeling dependability-oriented software processes. It builds on our previous work presented in Zhang et al[5] where aspect-oriented process modeling was introduced. Petri net extended with aspect-oriented modeling was used as the process modeling language. The reason why Petri nets were used was to apply their formal analysis techniques to analyze the correctness of modeling. It is important to stress that the correctness of software process modeling is important in ensuring the credibility of the models, especially in the dependability-critical software domain. To ease the integration of dependability-related activities into software processes, aspect-oriented modeling patterns are defined to weave aspects of dependability-related activities into software processes. In the future, these patterns can be applied to other software process modeling languages, such as UML2.0-Based Metamodel for Software Process Modeling[8–12] or Process Lifecycle Management for Business Software,[13] in different contexts. In the rest of the paper, for the sake of simplicity, we use the knowledge pattern to represent the dependability-oriented knowledge pattern and the modeling pattern to represent the aspect-oriented modeling pattern.

The remainder of this paper is structured as follows. In Section 2, preliminary knowledge is presented. Section 3 lays out the core contributions of our paper—knowledge patterns and modeling patterns. An illustrative example for modeling a secure wireless software update process and the modeling performance and subjective usability evaluation of our approach are presented in Section 4. In Section 5, we examine related work, stress its shortcomings, and position our ideas for overcoming them. Finally, we use Section 6 to conclude this paper and outline future work.

## 2 | PRELIMINARIES

Requirements for dependability consist of both functional requirements and a set of quality attributes. Base processes are modeled for functional requirements. For multiple dependability requirements, a goal-oriented modeling and reasoning method to find the dependability-related activities was proposed in our previous work.[5] In this paper, these activities and their sources of collection are described in Section 2.1. When integrating these activities into a base process, aspect-oriented software process modeling is presented in Section 2.2.

### 2.1 | Dependability-related activities

Traditional software process activities are designed for functional requirements. For dependability requirements, the specified activities must be provided. On the basis of Vyatkin's reviews[2] before 2013 and our reviews after 2013,[5] the relevant studies provide many activities, methods, or strategies for developing and maintaining dependable software. These studies include Microsoft SDL,[14, 15] SAE J3061,[16] EU project DEWI,[17] Lyu and Musa's software reliability engineering,[18, 19] Ericson and US DoD's safety engineering,[20, 21] IEC 61508,[22] security-related activities from Anderson,[23] and the software assurance maturity model.[24] Table 1 lists part of the dependability-related activities.

These dependability-related activities need to be integrated into the traditional software process to satisfy dependability requirements. Additionally, these activities and traditional activities should not be tangled in the software process. Because of the volatile requirements, flexible and maintainable process modeling is also required. To solve this problem, aspect-oriented approach is a suitable way. In the following section, the method of the aspect-oriented process modeling[5] is summarized.

### 2.2 | Aspect-oriented software process modeling

A rigorous process is necessary for dependability but need not be a burdensome one. It should be tailored if necessary to the project.[25] The aspect-oriented paradigm provides a proper mechanism to modularization and thus reduces the complexity of models[26] and also improves reusability and maintainability. In our previous work,[5] separation of concerns was used to separate the crosscutting activities and core activities in accordance with the dependability requirements and functional requirements. Crosscutting activities are dependability-related activities, whereas

**TABLE 1** Dependability-related activities

| Dependability | Dependability-related activities |
|---|---|
| Safety | Safety scope definition, establish safety-related electrical control system (SRECS), hazard and risk analysis, determine acceptable hazard level, use hazard mitigation methods, establish hazard tracking system (HTS), establish hazard action records (HARs), safety monitoring. |
| Security | Security analysis, extract security requirements, define minimum security criteria, create quality gates/bug bars, perform security and privacy risk assessment, define misuse cases, security strategy identification, security techniques assessment, code review, satisfy minimum cryptographic design requirements, analyze attack surface, threat modeling, vulnerability management. |
| Reliability | Define functional profile, define and classify failure, fault tree analysis, reliability forecasting, reliability testing of off-the-shelf and outsourcing software, reliability planning, reliability deploying, fault tolerance design, redundancy design, reliability growth test, establish incident response plan, execute incident response plan. |
| Performance efficiency | Determine performance baseline, performance simulation, prototyping, communication performance analysis and modeling, performance tuning, user-coordinated test. |
| Maintainability | Ladder logic diagrams, more access interface design, maintainability assessment, downtimeless evolution, create maintenance scheme, software execution data analysis. |
| Interoperability | Collaboration analysis, financial impact analysis, components interoperability analysis, assembled application interoperability analysis, SOA architecture design, modbus extensions, interaction interface design, use industrial communication and protocols. |
| Flexibility | Market change analysis, reconfiguration analysis, multiagent architecture design, model-based engineering, reusable components analysis and use, scalability design, service-oriented architecture design, integrability design. |
| Dependability | Formal modeling, model-based engineering, semantic web technologies, knowledge-based requirements elicitation, generative programming |

core activities are traditional activities for functional requirements. Base processes are modeled for functional requirements. The crosscutting activities for dependability requirements are encapsulated in aspects that are woven into the base processes. An approach to modeling aspect-oriented software processes was proposed. For a detailed introduction to the method of modeling, weaving, and evaluation analysis, please refer to our previous work.[5]

Although we got a positive result from the case study in our previous work, two limitations were still found, and two future works were left. The first is extending the knowledge base. The former knowledge base stored software nonfunctional requirements (NFRs), activities, and their decompositions, implementations, and contribution relations. Query interfaces based on SPARQL were provided, but the query results did not provide support for recommending activities for specific dependability requirements. Therefore, in this paper, an extended pattern-based knowledge structure is proposed. The second is writing an easy-of-use version for practitioners to use. In our previous work,[5] formal language was used. It was good for describing the reasoning and proving the correctness of modeling but not easy for enacting in practice. After we collected and discussed the results and suggestions from the case study in our previous work, we decided to use pattern, a more easily understandable method, for the practitioners to use. The modeling performance and usability were evaluated and described in the following Section 4.

Next, the details of the new knowledge architecture, knowledge patterns, and modeling patterns were elaborated.

## 3 | PATTERNS IN DEPENDABILITY-ORIENTED SOFTWARE PROCESS MODELING

Semantic technologies are aiming at knowledge representation and automatic manipulation and can be useful for enhancing the performance of software development processes.[2] On the basis of the semantic technologies, the construction of large-scale knowledge bases has been extensively studied. Notable endeavors of this kind include DBpedia,[27] YAGO/YAGO2,[28–30] Freebase,[†] and WolffranAlpha.[‡] Most of these knowledge bases represent knowledge framework in the form of knowledge graph. A knowledge graph is a semantic graph consisting of nodes and edges. The nodes represent concepts or entities. The edges represent the semantic relationships between concepts and entities.[31] These knowledge bases contain many millions of entities and concepts (or nodes), their mappings into semantic classes and individual instances, and relationships between nodes. In our previous work,[5] dependability-related knowledge from different open data and documents is modeled into nodes and edges. Then, with the help of knowledge refinement techniques, the knowledge from the preceding knowledge bases was integrated to further model the knowledge as knowledge base. The logical structure of our knowledge base is presented in Figure 1.

[†]https://developers.google.com/freebase/
[‡]https://www.wolframalpha.com/

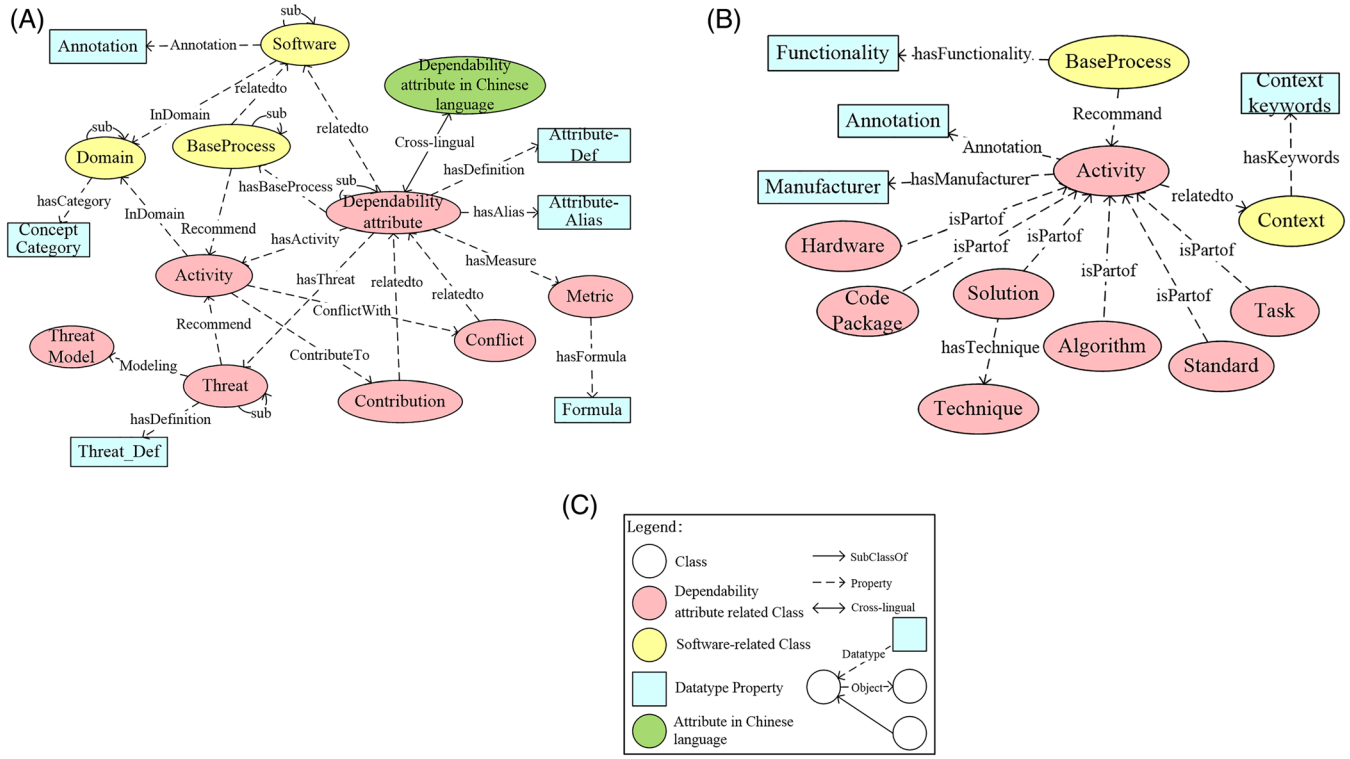



**FIGURE 1** A, Logical structure for dependability attribute in knowledge base. B, Logical structure for activity in knowledge base. C, Legend of the logical structure

In Figure 1, the logical structure of our knowledge base is composed of two types of nodes: classes and properties. Classes are represented by circles, and properties are represented by rectangles. For classes, different colors are used to distinguish different node types. Red nodes represent dependability-related classes, yellow nodes represent software-related classes, and green nodes represent classes in Chinese language. The lines with arrows between nodes represent the relationships. A solid line with a one-way arrow indicates a decomposition relationship; a dashed line indicates a property that a class has; a solid line with a double-headed arrow indicates the same classes described by a different language. Figure 1C shows the legend of the logical structure.

For clarity, the logical structure of the knowledge base is drawn in two parts of the figure. Figure 1A,B depicts the logical structures for dependability attributes and activities, respectively. In Figure 1A, the activity node stores the dependability-related activities, as shown in Table 1. These activities, as shown in Figure 1B, involve the knowledge of hardware, code packages, techniques, algorithms, standards, and tasks, which related to dependability attributes. In addition to the activities, knowledge of threats and metrics is also stored in the knowledge base. Conflicts and contributions that may exist between dependability attributes are also stored.

The purpose of the knowledge modeling is to provide dependability improvement knowledge for software process modeling. Knowledge structure in Figure 1 is the base of flexible, sharing, and reusable modeling. But query of the knowledge is not enough. Treatment of the dependability-related knowledge is difficult and is highly dependent on the experience of system analysts. Moreover, a dependable system will not offer uniform levels of dependability across all functions; for example, a dependable radiotherapy system may become unavailable but cannot be allowed to overdose a patient.[25] In software engineering, a pattern is characterized as a general reusable solution to a commonly occurring problem that arises within a specific context. The most important advantage of a pattern is that it describes a solution in a more readily accessible form.[4] Therefore, to help improve dependability capability by adding dependability-related activities to software processes, we define knowledge patterns and modeling patterns.

Referring to YAGO2[30] and Jin,[4] we first propose knowledge patterns to provide recommendations for dependability-oriented software process modeling. Afterward, modeling patterns are applied to aspect-oriented software process modeling. When extending these patterns, the logical structure of the dependability-oriented software process modeling is presented in Figure 2. Figure 2 consists of three layers. The bottom layer is the knowledge base consisting of subject, predicate, and object (SPO) triples. The logical structure of the knowledge base is shown in Figure 1. The middle layer in Figure 3 encapsulates the SPO triples into knowledge patterns for the upper layer. The upper layer uses the knowledge patterns to model the dependability-oriented software processes. In the following, the patterns are defined, and the guidelines for the use of these patterns are present.

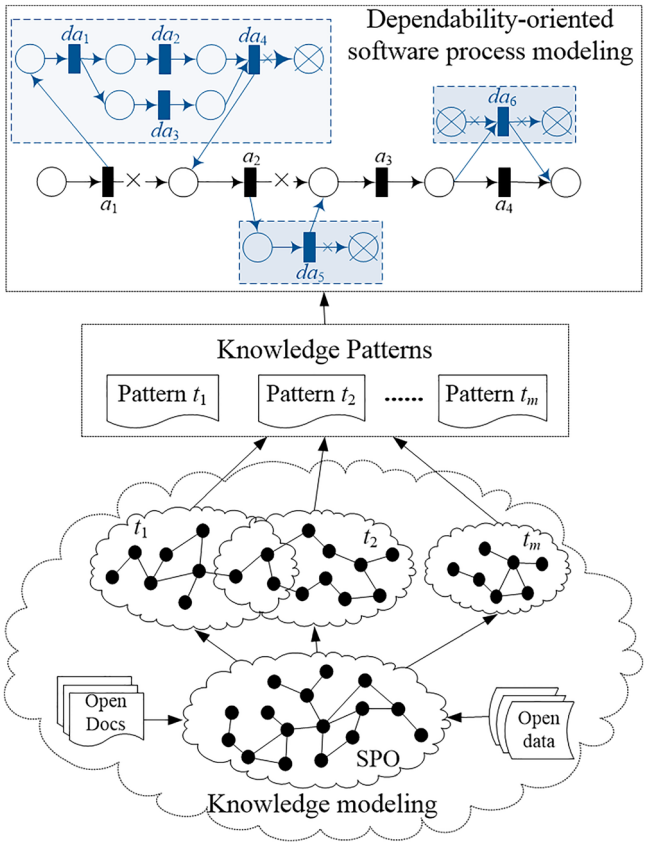



**FIGURE 2** Logical structure for knowledge modeling, pattern extension, and process modeling

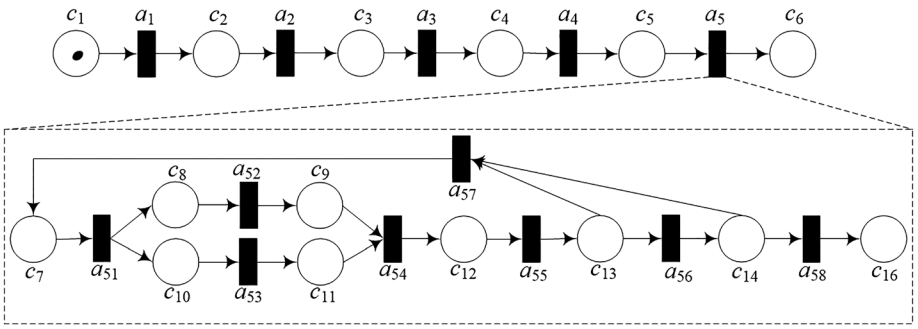


**FIGURE 3** Software maintenance process

## 3.1 | Knowledge patterns

Referring to the concept of infobox[§] in Wikipedia and Cucumber feature file,[¶] a detailed description of each knowledge pattern is provided in an indented infobox template, which consists of a summary of the unifying aspect that the patterns share. The following is the knowledge patterns' infobox template.

```
{{Infobox knowledge_pattern
|Pattern_name =
|Dependability_attribute =
|hasContext =
|hasDomain =
|hasBaseProcess =
```

[§]https://en.wikipedia.org/wiki/Infobox
[¶]https://cucumber.io/

```
|hasActivity =
|hasTask =
|hasSolution =
|hasStandard =
|hasAlgorithm =
|hasCode_package =
|hasHardware =
|hasTechnique =
|hasContribution =
|hasConflict =
}}
```

In a knowledge pattern, *hasContext* describes a particular problem treated by the pattern. A context characterizes the situations in which the pattern tries to solve. *hasDomain* defines the domain used by the pattern. *hasActivity* includes tasks, solutions, standards, algorithms, code packages, or hardware. They describe the alternatives for satisfying the dependability attribute. The indented of *hasTask*, *hasSolution*, *hasStandard*, *hasAlgorithm*, *hasCode_package*, and *hasHardware* means that they are one part of *hasActivity*. *hasContribution* lists all the dependability attributes that are positively affected by the execution of the activity. In contrast, *hasConflict* lists attributes that are negatively affected. These fields of the knowledge pattern structure are obtained by summarizing and discussing the relevant literature and various online sources.

In knowledge base, the dependability-related knowledge is represented as triples of subject (S), predicate (P), and object (O), in compatibility with the Resource Description Framework data model. When a knowledge pattern is constructed, referred to YAGO2[30] and Jin,[4] the pattern is defined as an extended 5-tuple, as defined in Definition .

**Definition 1** A knowledge pattern is a 5-tuple (*S*,*P*,*O*,*T*,*X*), where (1) (*id*,*S*,*P*,*O*) represents dependability-related knowledge as a triple of subject (S), predicate (P), and object (O). Each SPO triple is given an identifier *id*, (2) (*id*, *T*) associates a pattern *t* (*t* ∈ *T*) with an SPO triple that is identified by *id*, and (3) (*id*, *X*) defines a context *x* (*x* ∈ *X*) that the pattern is used.

Borrowing from the method of YAGO2[30] that expanded spatial and temporal dimensions, we extended the pattern and context in the original knowledge structure. In addition, Jin's nonfunctionality patterns[4] are used to describe nonfunctional problems and introduce appropriate NFR extensions into the function models. Referring to her method, our knowledge patterns introduce dependability-related knowledge. Therefore, in Definition , every SPO triple is identified by an identifier *id*, and the associated pattern is given the same identifier *id*. A pattern here is constructed by adding the identifiers of the associated SPO triples. It is worth noting that different patterns may share the same SPO triples. As shown in Figure 3, the SPO triples surrounded by pattern "$t_1$" might be defined in different pattern "$t_2$."

For each parameter in pattern infobox, if its value has been defined in SPO triple, add an identifier *id* to this triple. Otherwise, add (*S*,*P*,*O*,*T*) to knowledge base. After all the parameters in the pattern infobox are traversed, pattern context *X* is extended to (*S*,*P*,*O*,*T*), and we get (*S*,*P*,*O*,*T*,*X*). Algorithm 1 provides the pseudo code for the pattern extension.

**Algorithm 1**

**Pseudo code for the pattern extension.**

```
Algorithm 1: Extending patterns to knowledge base
Input:
(S, P, O): Triple entities in the knowledge base;
T_f: a set of pattern described in infobox data format;
Output:
(S, P, O, T, X): knowledge pattern;
Begin
 For ∀u ∈ t_f do /*t_f ∈ T_f*/
 /*u is the value of each parameter in a pattern infobox */
 If u_i ∈ S then /*u_i is the ith value of u */
  For ∀p ∈ P in (u_i, P, O) do
```

```
If u_j ∈ t_f and u_j matching o_k then /*o_k ∈ O*/
 (u_i, p, o_k) → (id, u_i, p, o_k) ∨ (id, t)
Else if u_j ∈ t_f and no matching of u_j in O then
 Add (u_i, p, u_j)
 (u_i, p, u_j) → (id, u_i, p, u_j) ∨ (id, t)
Else if u_i ∉ S and u_i is the value of Dependability_attribute parameter then
 s := u_i
For ∀u_j ∈ t_f and u_j ≠ u_i do
 p := r_j /* r_j is the parameter label of u_j*/
 If u_j matching o_k then /*o_k ∈ O*/
 (s,p,o_k) → (id,s,p,o_k) ∨ (id, t)
 If no matching of u_j in O then
  Add (u_i, p, u_j)
  (s,p,u_j) → (id,s,p,u_j) ∨ (id, t)
 Add the value of hasContext parameter to x /*x ∈ X*/
Return (S,P,O,T,X)
End
```

To avoid duplication of SPO, when we add patterns, we have to match them to the existing SPO. Thereby, we used the matching algorithm of YAGO2[30] to match SPO.

Next, modeling patterns are defined to support the dependability-oriented software process modeling.

## 3.2 | Modeling patterns

Process models are considered as a communication medium. Noncomplying or nonexact diagrams negatively impact this communication, therefore leading to misunderstandings that cost more time and money.[5] In the augmented age of software, making processes simpler and more repeatable is an inevitable trend. Facing these aims, an approach to using patterns in software process modeling represents an alternative to model dependability issues.

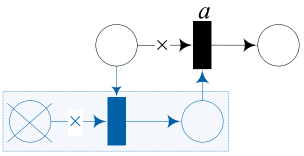
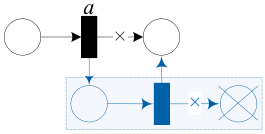
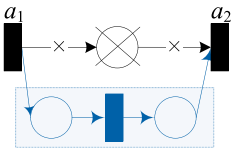
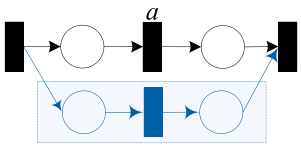
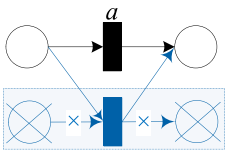
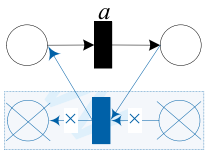
In our previous work,[5] the aspect-oriented paradigm provides a proper mechanism to weave the aspects into the base processes. All the possible base-aspect weaving structures can be summarized into 10 types. On the basis of the correctness definition in the previous work, only six weaving operations are used because the others violate the correctness definitions of weaving and can be replaced. In this paper, modeling patterns are proposed to represent these six weaving operations. Table 2 lists these six patterns, and Definition  provides a definition of the pattern.

**Definition 2** A modeling pattern defines a dependability-oriented aspect $d = (v,W)$ being woven into a base process $b = (C,A; F,M_0)$, where (1) $v$ is a triple advice ($DC$, $DA$; $DF$). It is a Petri net in which $DC$ is a set of conditions, $DA$ is a set of dependability-related activities, and $DF$ is a set of flow relations. It augments or constraints a base process $b$. (2) $W$ is a set of tuples, and $\forall w \in W$ is a 2-tuple $w = (pc, wt)$ in which $pc$ is a pointcut that represents a weaving position in the base process $b$. $pc$ is a condition $pc \in C$ or an activity $pc \in A$ or a flow relation $pc \in F$ in $b$. $wt$ is a weaving type for the advice $v$; before, after, between, concurrency, selection, and iteration are six weaving types. (3) $M_0$ is the initial marking, where a marking is a mapping $M$: $C \mapsto \{0,1\}$.

A Petri net is a set of nodes and arcs. There are two types of nodes: places and transitions, which represent the state of the system and the occurrence of events, respectively. Arcs are directed and connect places with transitions or transitions with places. In Definition , a condition is a place in Petri nets. An activity is a transition, and a flow relation is an arc in Petri nets. The definition in terms of conditions, activities, and flow relations is for a better understanding of software process. The state of a base process is defined by a marking, which puts zero or one token (graphically represented by a dot) on each condition. The firing process induces a token's flow among conditions; when an activity fires, token from all its input conditions is move to the activity output conditions. An activity can only be fired if there are tokens at its input conditions.

These modeling patterns are the outcome of an analysis we conducted in the field of human–computer interaction[5] to examine how the weavings are actually deployed. In Section 4, the patterns' usability is examined via a user experiment.

**TABLE 2** Modeling patterns

| Before activity weaving | After activity weaving | Between activities weaving |
|---|---|---|
|  |  |  |
| **Concurrent activity weaving** | **Selective activity weaving** | **Iterative activity weaving** |
|  |  |  |

Next, we introduce an example from the industrial software world to share the use of the patterns to the example to consolidate the explanation.

## 4 | ILLUSTRATIVE EXAMPLE AND EVALUATION

One of the main driving forces of the software developments is dependability.[2] The goal of this paper is to solve the dependability issue. We explored methods for enhancing, improving, and innovating software process techniques to support the development and production of dependable software. An approach to modeling dependability-oriented software processes was proposed, and knowledge pattern was constructed to provide dependability-related activities and the related tasks, solutions, techniques, and so on. Next, a modeling result of using patterns is illustrated in Section 4.1, and the approach is evaluated in Section 4.2.

### 4.1 | Illustrative example

Consider a process in Figure 3, where there is a software maintenance process, which is described in Petri net language. This process starts with *planning maintenance* activity (i.e., $a_1$ in Figure 3) and ends up with the *software update* activity (i.e., $a_5$ in Figure 3). The *software update* activity is

**TABLE 3** Activities in Figure 3

| Symbol | Activity |
|---|---|
| $a_1$ | Planning maintenance |
| $a_2$ | Analyze modification |
| $a_3$ | Modification implementation |
| $a_4$ | Maintenance review |
| $a_5$ | Software update |
| $a_{51}$ | Define update |
| $a_{52}$ | Develop update packages |
| $a_{53}$ | Conversion of software and data |
| $a_{54}$ | Update execution |
| $a_{55}$ | Update verification |
| $a_{56}$ | Assess new environment |
| $a_{57}$ | Redefine update |
| $a_{58}$ | Achieve old environment |

decomposed into a fine-grained process, the *software update* process, which is framed by a dotted line. The detailed information of activities in the processes is shown in Table 3.

To help assimilate using the patterns, let us consider an example of a secure and dependable wireless software update process. This example is extracted from Steger et al.[32] They proposed a generic framework SecUp to enable secure and efficient wireless automotive software updates. SecUp is designed to fulfill the requirements of wireless software update for a modern vehicle as it allows us to securely enable new features on the vehicle and to fix software bugs by updating the software over the air.

Applying the pattern infobox definition and employing the SecUp's security concept, *secure software update* pattern is described below.

```
{{Infobox pattern
|Pattern_name = secure software update
|Dependability_attribute = Internet security
|hasContext = secure wireless software update, Prevent from unauthorized access
|hasDomain = automotive industry
|hasBaseProcess = software update
|hasActivity =security analysis
 |hasTask = extraction security requirements
 |hasSolution =dependable smart gateway interface
|hasActivity =security strategy identification
 |hasTask = security strategy definition
 |hasSolution = authentication
 |hasTechnique = NFC smartcard, PIN code
 |hasConflict = time performance
 |has Algorithm = RSA, SHA
 |hasConflict = time performance
 |hasStandard = IEEE 802.11s
 |hasContribution = reliability, availability
 |hasStandard = Trusted Platform Module
 |hasConflict = time performance
 |hasHardware = AURIX
 |hasManufacturer = Infineon
 |hasContribution = Safety, time performance
|hasActivity =STRIDE threat modeling
 |hasTask = evaluate threats
|hasActivity = establish security transport design
 |hasTask = employ security strategy
 |hasSolution = diagnostic tester
 |hasSolution = VPN tunnel
|hasActivity = perform security design review
|hasActivity = security test
 |hasTask = data verification
}}
```

Secure software update pattern is used to satisfy Internet security requirement in automotive industry domain. It contains six activities: *security analysis*, *security strategy identification*, *STRIDE threat modeling*, *establish security transport design*, *perform security design review*, and *security test*. Different tasks, solutions, algorithms, standards, or hardware are used for different activities. Applying the preceding Algorithm 1, this pattern is added to the knowledge base. Because the visualization of the actual knowledge base is not conductive to show the extension of the patterns, we depicted the partial logical structure of *secure software update* pattern in Figure 4.

In Figure 4, the dark ellipse surrounds the SPO triples of the extended pattern, and we use the upper left corner to write the pattern name in a circle to identify the range of the pattern. The outside nodes in the left are connected to the pattern, and they belong to the pattern. The nodes in the upper right corner are not connected to the pattern. They do not belong to the pattern but belong to the knowledge base. When using this knowledge pattern, SPARQL is used to query, and the query results are used to generate the output form, as shown in Figure 5.

As shown in Figure 5, Internet security is the dependability attribute of the secure software update pattern. The base process is the software update process. There are six dependability-related activities in the pattern. These activities will be integrated into the base process,

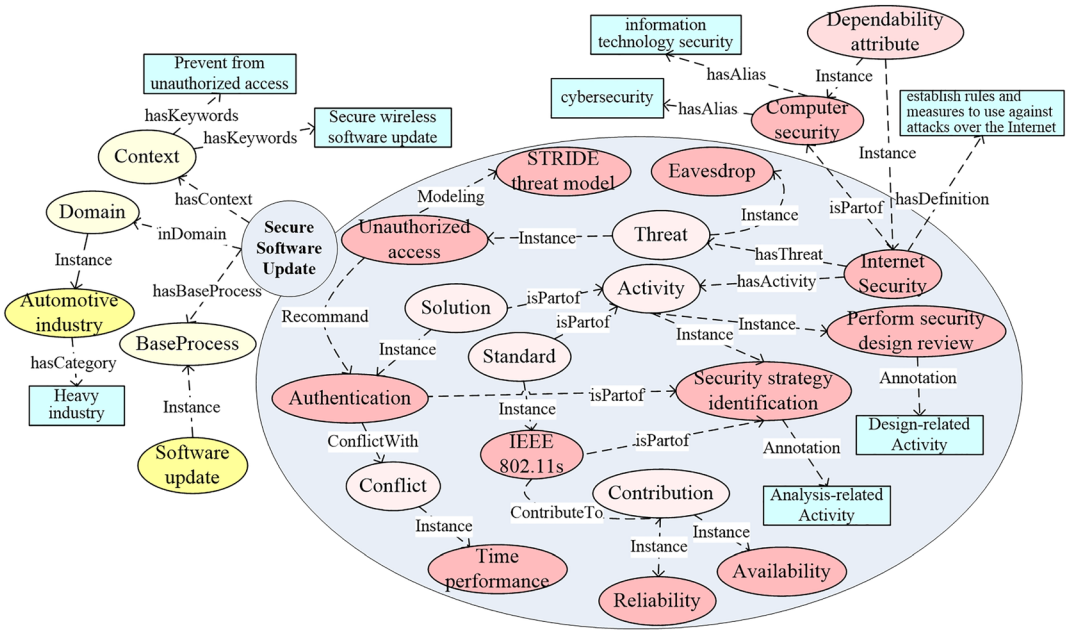



**FIGURE 4** Secure software update pattern in knowledge graph

and the integrated process is shown in Figure 6. The blue underlined content indicates that there are associated triple links in the knowledge base. These associated triples are available by clicking on the links. On the basis of this knowledge pattern, using the modeling patterns in Table 2, a security requirement-oriented software update process is modeled (see Figure 6). The detailed information of security-related activities is shown in Table 4. Please note that the knowledge in the knowledge pattern needs to be tailored in a specific software project. Moreover, dependability-related activities require different weaving positions when weaving to different base processes. Therefore, software processes still require process engineers to complete modeling. However, we defined the modeling patterns in this paper and developed a process modeling aided tool in our previous work.[5] Process engineers can use the modeling patterns and the tool to assist them in aspect-oriented process modeling. Because we allow process engineers to tailor knowledge patterns to specific project needs, we do not perform alignment constraints. But we need to ensure the correctness of process modeling. This correct control and evaluation method was introduced in detail in the previous work.[5]

The integrated process is initiated by adding security-related activities, that is, *security analysis*, *security strategy identification*, *STRIDE threat modeling*, and *establish security transport design*, before *defining update*. Before the update is defined, to ensure that the update contains the security strategies, the necessary security analysis and security strategy identification work needs to be done. Then, before *update execution* activity, *perform security design review* activity is added to ensure that the security strategies can be enforced. Finally, after *update verification* activity is completed, perform *security test* activity to ensure the security of the software update.

Because the vehicles will be wirelessly connected to the Internet for upgrading the software or fixing bugs. This secure wireless remote update process is beneficial over the entire life cycle of a modern vehicle and will significantly reduce the time needed for vehicle maintenance.

## 4.2 | Evaluation and discussion

Through the preceding example, a secure software update pattern is used, and a corresponding secure software update process model is created. However, in practice, the patterns must be useful and ease to learn. When integrating dependability-related activities into a base process, the execution of the process must not be interrupted, and the dependability-related activities must validly provide the expected dependability capability.

In our previous work,[5] the correctness, performance, and effectiveness of aspect-oriented software process modeling had been evaluated. The evaluation results show that our process modeling is correct and effective and the dependability of the software can be improved. For the newly defined patterns, in this paper, we evaluate the modeling performance and the subjective usability of them. As mentioned in the preceding Section 3, providing an easy-of-use version for practitioners to use our approach is the primary goal of this paper.

**FIGURE 5** Secure software update pattern

**Secure Software Update Pattern**

**Dependability attribute**

- Internet security

**BaseProcess**

- Software update

**Domain**

- Automotive industry

**Context**

- Secure wireless software update
- Prevent from unauthorized access

**Activity**

1. Security analysis # Analysis-related activities
   - ✓ Task: extraction security requirements
   - ✓ Solution: dependable smart gateway interface
2. Security strategy identification #Analysis-related activities
   - ✓ Task: security strategy definition
   - ✓ Solution: authorization
   - ✓ Related to: Role-based access control
   - ✓ Solution: authentication
   - ✓ Technique: NFC smartcard, PIN code
     - Conflict: time performance
   - ✓ Algorithm: RSA, SHA
     - Conflict: time performance
   - ✓ Standard: IEEE 802.11s
   - ✓ Contribution: reliability, availability
   - ✓ Standard: Trusted Platform Module
   - ✓ Alias: ISO/IEC 11889
     - Conflict: time performance
   - ✓ Hardware: AURIX
   - ✓ Manufacturer: Infineon
   - ✓ Contribution: Safety, time performance
3. STRIDE threat modeling # Analysis-related activities
   - ✓ Task: evaluate security strategy
   - ✓ Technique: unauthorized access
   - ✓ Technique: external eavesdropping
   - ✓ Technique: external tamper
   - ✓ Technique: internal rogue
   - ✓ Technique: internal spoof
4. Establish security transport design # Design-related activities
   - ✓ Task: Employ security strategy
   - ✓ Solution: diagnostic tester
   - ✓ Solution: VPN tunnel
5. Perform security design review # Design-related activities
6. Security test /* Test-related activities */
   - ✓ Task: data verification

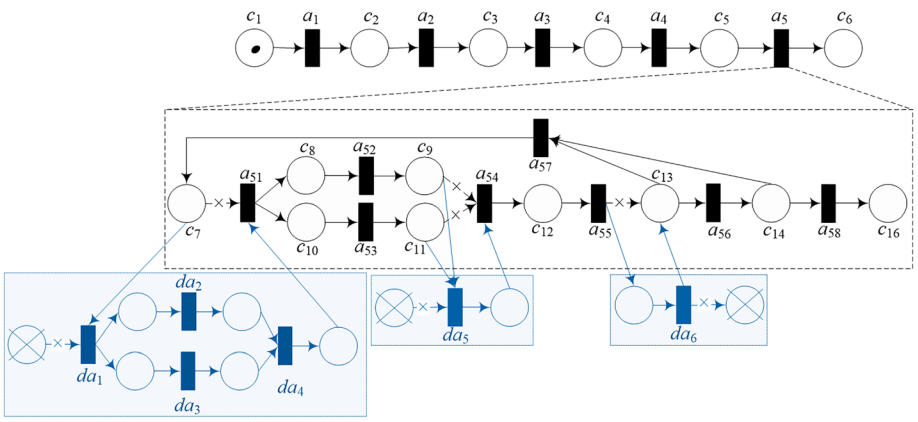



**FIGURE 6** Security-oriented software maintenance process

**TABLE 4** Security-related activities in Figure 6

| Symbol | Security-related activity |
|---|---|
| $da_1$ | Security analysis |
| $da_2$ | Security strategy identification |
| $da_3$ | STRIDE threat modeling |
| $da_4$ | Establish security transport design |
| $da_5$ | Perform security design review |
| $da_6$ | Security test |

In order to evaluate the proposed patterns, an experimental study was conducted. Our primary experiment goal is to evaluate the modeling performance and subjective usability of our knowledge patterns and modeling patterns. These are evaluated in the simulated context of a classroom experiment. The study took place from November 2018 to April 2019 at the School of Software in Yunnan University.

### 4.2.1 | Evaluation experiment setup

To evaluate our approach, the sample of 110 participants consisted of 6 doctoral students, 20 master students, 36 third-year undergraduate students, and 48 second-year undergraduate students who major in software engineering. Four doctoral students had working experience, and two doctoral students have published papers in software process modeling. Eight master students were about to graduate, and some of them have jobs in information industry. Of the other master students, six had experience of software process modeling. Thirty-seven third-year undergraduates had software project development experiences. Forty-eight second-year undergraduates had learned software engineering and were participating in software project development. All doctoral students and master students have studied Petri nets language, whereas the undergraduate students did not. The experiences of the doctoral students and master students with software process modeling range from 1 to 4 years, with a median of 2.5 years. All the undergraduates had basic concepts of software process modeling and played SimSE game[#] to practice a "virtual" software engineering process, which enabled them to learn the complex cause and effect relationships underlying the processes of software engineering.

### 4.2.2 | Evaluation experiment design

In the evaluation, Goal Question Metrics (GQM) paradigm[33] was used to define evaluation goals, questions, and the needed metrics. Goals are defined first in the hierarchy structure of the GQM paradigm. A goal specifies purpose of measurement, object to be measure, issue to be measured, and viewpoint from which the measure is taken. In the following, the purpose and scope of our experiment are described by the goals (indicated in G*):

G1. Analyze the dependability-oriented software processes modeling approach for the purpose of evaluating the modeling performance with respect to its complexity of use and applicability from the point of view of the process modeling engineer and stakeholders in the context of industrial software development, maintenance, and evolution process.
G2. Analyze the proposed patterns for the purpose of evaluating their subjective usability with respect to its understandability, ease of use, usefulness, and workload from the point of the process modeling engineer and stakeholders in the context of industrial software development, maintenance, and evolution process.

According to the goals of our evaluation experiment, the following questions (indicated in Q*) are formulated that the experiment should help to answer.

Q1. Can real users model a dependability-oriented software process completely and correctly?
Q2. Are the patterns easy for real users to understand and use?
Q3. Do real users feel that the patterns are useful to aid them to generate a dependability-oriented software process?
Q4. How long does it take for real users to use the patterns to complete a dependability-oriented software process?

[#]https://www.ics.uci.edu/~emilyo/SimSE/

Q1 is related to the first goal, whereas Q2–Q4 refine the second goal. The corresponding metrics of GQM are defined below to answer the questions. We describe each metric, denoted by $M_{ij}$, where $i$ corresponds to the question identifier and $j$ is a counter in the case that more than one metric is defined per question.

$M_{11}$. Model completion rate. This metric is the percentage of the dependability-related activities that are modeled in the processes and the dependability-related activities that are defined in the knowledge patterns. It reflects the proportion of the knowledge patterns that can be fully used to model a dependability-oriented software process. This metric is defined as follows:

$$C = \frac{DA_m + A_m}{DA_t + A} \times 100. \tag{1}$$

$DA_m$ and $A_m$ are the number of dependability-related activities and base activities that are modeled in the process, respectively. $DA_t$ is the number of dependability-related activities that are defined in the patterns. $A$ is the number of base activities that should be modeled.

$M_{12}$. Modeling correctness rate. This metric is the percentage of the satisfied correctness and the defined correctness. When integrating dependability-related activities into a base process, ensuring the correctness of the integrated process models is a prerequisite. As a supplement to $M_{11}$, this metric mainly judges the correct rate of modeling. It is defined as follows:

$$R = \frac{R_A}{DA_m + A_m} \times 100. \tag{2}$$

$R_A$ is the number of activities that satisfied correctness. On the basis of the model properties of Petri nets, the aspect–aspect correctness, structural correctness, and dynamic correctness are defined in our previous work.[5] $DA_m$ and $A_m$ are defined in metric $M_{11}$.

$M_{21}$. Understandability of patterns. This metric assesses the perception of the users regarding their difficulty in understanding the proposed patterns. Similar to the work presented by Briand et al,[34] for this metric, we designed a scaled question, which is represented on the aspects of user acceptance, that is, "Are the patterns easy to understand?" We measured the level of agreement with the questions on a five-point Likert scale.[35] For analysis, the values are projected onto a numerical scale ranging from 0 to 4, which represent complete disagreement, disagreement, neutral, agreement, and complete agreement, respectively.

$M_{22}$. Ease of use of the patterns. Similar to $M_{21}$, this metric assesses the perception of the users regarding their difficulty in using the proposed patterns. The scaled question is "Is it easy to use the patterns?" The measure of the agreement level is the same as $M_{21}$.

$M_{31}$. Usefulness of the patterns in modeling. This metric assesses the perception of the users regarding the pattern capacity, which significantly aids the users in modeling a dependability-oriented software process. Similarly, we designed two scaled questions: "Will you use the patterns and recommend them?" and "Are improvements being made by using the patterns?" The measure of the agreement level is the same as $M_{21}$ except that we averaged the values to the two questions.

$M_{41}$. Workload on using the patterns to model. This metric is measured by the users' spending time of using the patterns to model a dependability-oriented software process. To be consistent with metric $M_{21}$, $M_{22}$, and $M_{31}$, we projected the users' spending time onto the same numerical scale ranging from 0 to 4, which represent very slow, slow, medium, fast, and very fast, respectively.

Metrics $M_{11}$ and $M_{12}$ reflect the modeling performance, whereas $M_{21}$, $M_{22}$, $M_{31}$, and $M_{41}$ measure the subject usability of the proposed patterns.

The evaluation was conducted over a period of 8 weeks. Doctoral students, master students, third-year undergraduates, and second-year undergraduates each took 2 weeks to participate in the experiment. At the beginning of the experiment, we asked the participants to perform a modeling task without the use of our approach. After that, we held a lecture about our approach and gave the participants two assignments of using patterns to create two dependability-oriented software process models. The first assignment came from the industrial practice,[36, 37] whereas the second was the illustrative example in Section 4.1.

For the first modeling assignment, our participants were asked to model a software test reporting process that exhibits software dependability and testing effectiveness improvements. Test report inspection pattern in Figure 7 provides the activities that can be used to improve the process of software defect reporting. Software testing is an important part of project development because it could limit a number of potential errors. Improving test quality can improve software dependability to a certain extent. According to the work of Wang et al,[36] unclear or invalid defect reporting usually causes the communication gap between testers and developers. Through their empirical study on measuring and

**Test Report Inspection Pattern**

**Dependability attribute**

- Dependability
- Effectiveness

**BaseProcess**

- Software testing

**Domain**

- Industrial automation software

**Context**

- Unclear or invalid test report
- Improve test efficiency

**Activity**

1. Inspect test report #Test-related activities
   - ✓ Standard: IEC61499, IEC61508, IEC62061
2. Mark clear test report #Test-related activities
3. Mark invalid test report #Test-related activities
4. Withdraw unclear test report #Test-related activities
5. Modify unclear test report #Test-related activities

**FIGURE 7** Defect report inspection pattern

improving the process of software defect reporting, checking test reports and training are ways to improve test reports and further improve the test quality. Using this pattern, the improved process model is depicted in Figure 8. The corresponding activities in the model are described in Table 5.

Of course, it is important to note that just checking the test report is not enough to greatly improve the dependability of the software. Improving software dependability requires the joint execution of multiple dependability-related activities. The following two subsections explain the outcomes in the context of the experiment.

### 4.2.3 | Modeling performance evaluation

To obtain the first metric, $M_{11}$, we counted the number of activities that were modeled into the processes. Using Equation 1, we calculated the model completion rate for all participants. Regarding the metric about the correctness of modeling, we analyzed the correctness of each activity in the models. Using Equation 2 as well, we obtained the modeling correctness rate. Then, we use a line chart (see Figure 9) to depict the participants' modeling completion and correctness ratio.

All results for two assignments are presented in relation to the experience level of the participants. As can be seen from the results, the modeling performance is related to the model complexity. The second assignment had more activities (19) than the first one (13), and there is an

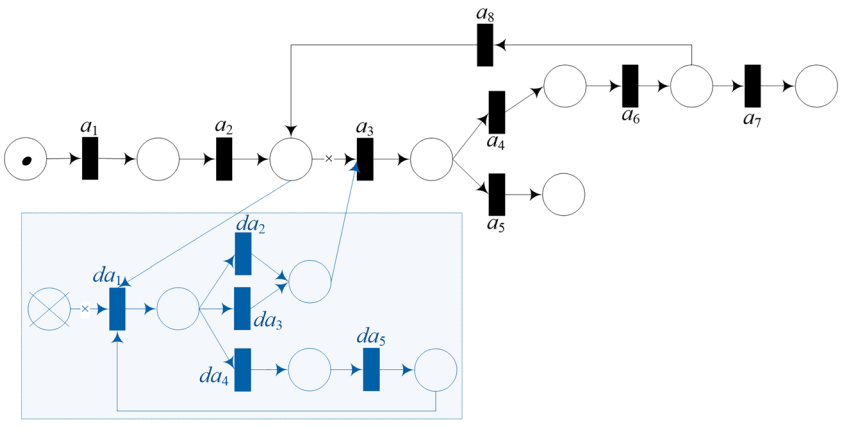
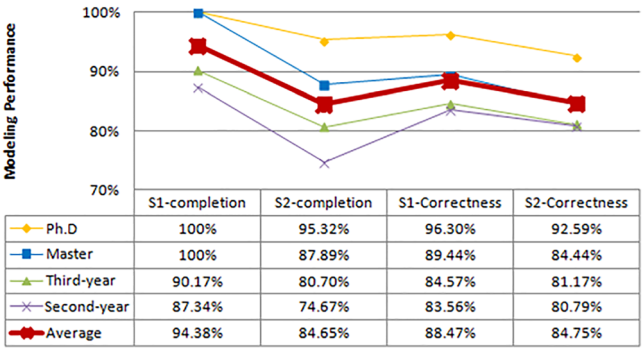



**FIGURE 8** Improved software test reporting process

**TABLE 5** Activities in Figure 8

| Symbol | Dependability-related activity |
|---|---|
| $a_1$ | Test software |
| $a_2$ | Submit test report |
| $a_3$ | Assign test report |
| $a_4$ | Verify and debug |
| $a_5$ | Revoke test report |
| $a_6$ | Review software modification |
| $a_7$ | Close test report |
| $a_8$ | Reopen test report |
| $da_1$ | Inspect test report |
| $da_2$ | Mark clear test report |
| $da_3$ | Mark invalid test report |
| $da_4$ | Withdraw unclear test report |
| $da_5$ | Modify unclear test report |

**FIGURE 9** Results on the modeling performance




activity ($a_5$) that is further decomposed into processes, and the dependability-oriented activities that need to be woven are scattered. The completion of its modeling is probably harder. From the results, the completion rate of the second assignment is indeed much lower than that of the first assignment. In addition, the results also show that the modeling performance is related to the level of experience of participants. Among them, the completion rate and correctness rate of doctoral students are significantly higher. However, although the completion rate and correctness rate of other students are lower, the gap is relatively small. Compared with the model completion rate, the correctness rate is relatively high. Even for the third-year and second-year students who have relatively less modeling experience, as long as they model the processes, the correctness can be guaranteed. This can prove that our modeling pattern has a good ability to ensure correctness. However, when participants submitted the model, some participants suggested that it would be much easier to model only the dependability-oriented activity into the base process, because the patterns only describe the dependability-related activities. Although the modeling of the base process adds a little complexity and makes the modeling completion rate lower, it is still necessary to model a complete process in the experiment.

In summary, as depicted in Figure 9, the completion and correctness rates of modeling for both assignments are relatively high, and the average rate is all higher than 84%. This can explain to some extent that our modeling approach has good modeling performance.

### 4.2.4 | Subjective usability evaluation

Subjective usability was considered the ability of the user to use the patterns to carry out a software process modeling successfully. Here, we use four metrics: *understandability* ($M_{21}$), *ease of use* ($M_{22}$), *usefulness* ($M_{31}$), and *workload* ($M_{41}$). These are important indicators for the quality of a modeling language extension as well as the user acceptance.[5, 38–40] In reaching these goals, the patterns could be claimed to ease the modeling of software processes. We use the boxplot to show the results on the subjective usability evaluation.

As shown in Figure 10, the results show that the patterns' understandability (median 3.13 and mean 3.03) and ease of use (median 3.06 and mean 2.89) scored higher among the participants, followed by workload (median 2.81 and mean 2.62) and usefulness (median 2.75 and mean 2.72). The standard deviation is lowest for understandability (0.997) and highest for workload (1.138), whereas ease of use (1.02) and usefulness

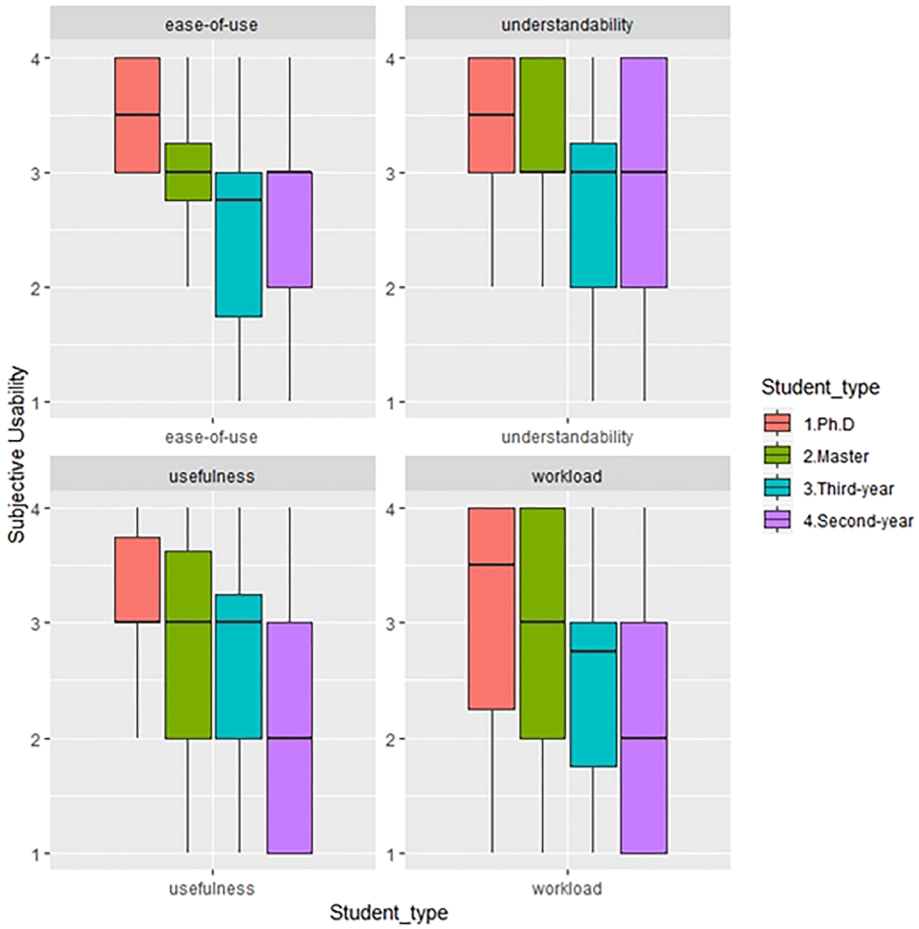



**FIGURE 10** Results on the subjective usability

(1.108) rank in between. The correlation between experience of participants and the judgment of understandability (0.775), usefulness (0.775), and ease of use (0.718) is lower than the participants' workload (0.981). The participants' assessments of understandability, usefulness, and ease of use are relevant to their experience. Their workload in modeling has a more significant correlation. The correlation results show that although participants with less experience agree that the patterns are understandable, easy to use, and useful, they still require more time to complete the modeling.

Then, we counted the proportion of participants who gave positive assessments, that is, answers with a value higher than 2, in Table 6. Except for second-year students whose workload and usefulness assessment is less than 50%, other positive assessments are at least greater than 53%.

With respect to modeling dependability-oriented software process, the evaluation examines the modeling performance and subjective usability of our proposed approach and patterns. Summarizing the relevant results concerning our stated goals, we can observe that despite the benefits of the patterns, its use is not trivial and proper training is required to take advantage of it. Besides the preceding evaluation results, the observation also raised an interesting fact that even with few days of training, it is possible to start using the patterns to model dependability-oriented software processes even with users with no previous knowledge on software process modeling.

## 5 | RELATED WORK

The proposed knowledge-based aspect-oriented model for dependability improvement in software processes is relevant to dependability assurance method, semantic knowledge, and pattern. We, therefore, divide the related work into corresponding three parts as follows.

### 5.1 | Dependability assurance for software

Functional safety in the standard IEC 61508[22] is an important quality characteristic for automation systems software. Standard IEC 62061 specifies requirements and makes recommendations for the design, integration, and validation of safety-related electronic control systems for

**TABLE 6** Positive evaluation

| Metric | Students | Positive | Positive% |
|---|---|---|---|
| Workload | Doctoral | 4 | 67 |
| | Master | 12 | 60 |
| | Third year | 19 | 53 |
| | Second year | 23 | 48 |
| Understandability | Doctoral | 6 | 100 |
| | Master | 17 | 85 |
| | Third year | 21 | 58 |
| | Second year | 27 | 56 |
| Usefulness | Doctoral | 5 | 83 |
| | Master | 14 | 70 |
| | Third year | 19 | 53 |
| | Second year | 19 | 40 |
| Ease of use | Doctoral | 6 | 100 |
| | Master | 15 | 75 |
| | Third year | 20 | 56 |
| | Second year | 28 | 58 |

machines.[41] Stemming from various safety requirements and certification procedures, dependability requirements have become the new increasing demand for software in industrial automation systems.[2]

On the basis of the review of software engineering in industrial automation, Vyatkin indicated that the increasing demands of dependability would increase the demand for formal method application and corresponding tools in automation software engineering.[2] A recent popular quality assurance method is formal verification. However, the surveys from Frey and Litz,[42] Johnson,[43] and Hanisch et al[44] showed that such methods were hindered by their high computational complexity, as well as lack of user-friendly tools. Afterward, for dependability requirements, different methods were provided. Kim et al[45] extended the notion of timed action to statecharts to specify a timed and prioritized behavior in using a resource for embedded software's real-time and resource constraints in different execution environments. For the robustness of automation systems, Schütz et al[46] presented a modeling approach and notation that support the development of soft sensors, which are used to detect and compensate sensor failures during runtime on a programming logic controller with IEC 61131-3 language elements. Commercial off-the-shelf (COTS) equipment in industrial systems are not robust and safe. For detecting transient faults of COTS, Asghari et al[47] designed a software-based method for detection of control flow errors because about 33%–77% of the transient faults cause control flow errors. Givehchi et al[48] proposed an intermediate interoperability layer to improve interoperability in industrial system. Bhatti et al[49] presented a unifying approach to analyze quantitative aspects in hardware and qualitative aspects in software and proposed a semantics-preserving transformation of IEC 61499 function blocks to PRISM models for achieving safety of industrial automation systems. Steger et al[32] defined a secure and dependable wireless software update process for different automotive application scenarios. They focused on comprising security mechanisms to prevent abuse and attacks. All of these methods can be described uniformly in terms of patterns. The patterns can be further stored as reusable knowledge. In this paper, we define Steger's security mechanisms as a knowledge pattern and use the modeling patterns to weave the security-related activities into the base process. The security mechanisms in pattern provide a clearer presentation. The woven software process model is easy for complying.

The above methods are proposed to solve individual attribute in the dependability. When it comes to the multiple dependability attributes of industrial automation systems, the relevant work draws on the aspect-oriented approach in the field of software engineering and uses crosscutting concerns to describe dependability requirements. Freitas et al,[50] Wehrmeister et al,[51] and Binotto et al[52] introduced the aspect-oriented paradigm to improve the treatment of NFRs for distributed embedded real-time systems. Wehrmeister et al[53] combined unified modeling language (UML) with concepts of aspect-oriented software design to cope with specific NFRs. Roque et al[54] used aspect-oriented concepts to model faults in early-design phases of distributed vehicular control systems. Their early fault modeling could improve the control system modeling mapping fault behaviors in order to mitigate the diagnosis the fault impacts in critical tasks. Using the aspect-oriented paradigm, they improved the encapsulation of dependability requirements handling and avoiding scattering and tangling of crosscutting concerns.

Contrary to existing works only focusing on techniques, methods, or strategies, our work supports the injection of dependability into the software entire life cycle, including development, assembly, maintenance, and evolution. Furthermore, the corresponding knowledge is stored and used by inexperienced organizations and supports future reuse.

### 5.2 | Semantic knowledge

Applying semantic knowledge to dependability requirements is often used to describe single or parts of the attributes. Lastra and Delamer[55] used ontologies and semantic web services in manufacturing systems to infer knowledge on the classification of processes and on how to execute and compose those processes to enhance autonomy, interoperability, and rapid reconfigurability. Niemelä et al[56] defined metrics for reliability and security attributes as quality ontologies to model quality properties as an integrated part of software architecture. Lopez et al[57] translated softgoal interdependency graph models to instances of the NFRs and design rationale ontology to store the NFRs and design knowledge as machine-readable semantic graphs. This enabled analysis of alternatives, search, exploration, and reuse of NFR knowledge. Elahi et al[58] proposed a modeling ontology for integrating vulnerabilities into conceptual modeling frameworks (misuse case diagrams, i* models, and CORAS risk models) to enable a finer-grained security analysis, assess the risks of vulnerabilities, and decide on proper countermeasures. Gandhi and Lee[59] used ontological domain modeling techniques to discover the multidimensional correlations among regulatory security requirements to improve the understanding of the potential security risk. Saeki et al[60] proposed the technique to extract knowledge for eliciting security requirements from Common Criteria and to use it to security requirements analysis. Souag et al[61] used security and domain ontologies to guide domain-specific security requirements elicitation. Balushi et al[62] provided a standardized quality terminology and ontological constructs to capture NFRs throughout the requirements engineering activities. Hästbacka and Kuikka[63] applied semantics and reasoning to models to infer generalized classifications and detect specific structure, flaws, and error-prone designs. The semantic descriptions of models also allowed linking of related engineering knowledge to support engineering tasks. Zhou et al[64] and Provenzano et al[65] introduced ontological approaches for safety requirements elicitation based on environment ontology and hazard ontology, respectively. They all aimed at empowering the requirements analysts with knowledge that helped in the process of capturing, prioritizing, understanding, and reusing dependability requirements.

The dealing with dependability requirements requires knowledge. Our work is also based on semantic knowledge to build reusable patterns. But, compared with the above efforts, our goal is to satisfy multiple attributes. In addition, in order to effectively use knowledge, based on traditional knowledge base, we provide a more convenient way to use knowledge in the form of patterns.

#### 5.2.1 | Patterns

Pattern-based approaches have been popular for describing software development knowledge, for example, the software design patterns. Some efforts have also been made to extend pattern-based approaches to provide knowledge about elaborating and refining NFRs.[4] As stated by Palomares et al,[66] patterns are proposed in 47% of the requirements reuse proposal identified between 2010 and 2015.

Supakkul et al[67] presented a pattern-based approach for capturing and reusing knowledge of NFRs using objective, problem, alternative, and selection patterns. Salini and Kanmani[68] designed security requirement patterns for creating security requirements ontology for an e-voting system. Similarly, Daramola et al[69] also presented the pattern-based security requirements using ontologies. Their use of security requirement patterns assisted software developers in incorporating security mechanisms and techniques into the software development process. Jin defined some NFR patterns in a problem-oriented way and used them to describe nonfunctional problems and introduce appropriate solutions into the function models of a system.[4] Yousfi et al used the concept of patterns to propose an approach to represent the ubiquitous computing features for ubiquitous business process modeling. Their evaluation of the patterns showed that the patterns are easy to understand and apply even with a shallow understanding of ubicomp.[5]

An important research method in industrial is to rely on proven solutions rather than reinventing a new one.[2] In the context of engineering, the term "pattern" was introduced by architect Christopher Alexander. In his view, "each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over."[70] This formulation is so generic that fitted well in requirements engineering.[71] Therefore, referring to the pattern concepts and design methods in requirements engineering, we propose knowledge patterns and modeling patterns based on knowledge-based and aspect-oriented methods. These patterns not only provide dependability-related knowledge but also support dependability-oriented software process modeling.

## 6 | CONCLUSIONS AND FUTURE WORK

Reflecting on the trend of the growing importance of software, there are a great number of research projects and corresponding publications addressing various aspects of software development process. However, dependability is still a challenge.[2] For dependability, a credible process is certainly necessary. In the extreme, no credible process means that there is no reason to believe the software is certified.[25] In this paper, our aims are to explore methods for enhancing, improving, and innovating software process techniques and to support the development and production of dependable software. The knowledge patterns and the modeling patterns were proposed for providing dependability-related knowledge and

integrating dependability-related activities into software processes. In reality, our approach can bring benefits to enterprises at least in the following aspects:

(1) A pattern-based knowledge model is proposed for dependability requirements in a flexible and reusable way.
(2) A knowledge-based aspect-oriented software process modeling is introduced for dependability improvement. It is helpful for the process improvement in an incremental way at build time. Also, it can facilitate software process control and risk reduction.

Although we got a positive result from the evaluation, we still found three limitations that are needed to be solved in the future.

(1) Although we have built 37 knowledge patterns, the number is still too small. The building of a large number of knowledge patterns is limited by the acquisition of pattern data. The next step is to study the document-based automatic extraction and crowdsourcing collection mode. At the same time, build the model and propose an evaluation method.
(2) From the practice survey of Palomares et al[66] of answering the question about requirements reuse techniques, only 10% of participants from IT companies answered that they have used requirement patterns. The critical factors and barriers are the existence of a well-defined reuse method, the tool support, and the involvement of people in the software development team in the reuse process. Therefore, our other future work will take advantage of knowledge graph representation in pattern recommendation and use recommendation to provide the practical reuse method and develop the corresponding tool.
(3) Petri net-based language is the first-generation software process modeling language. It focuses on process execution and formality, which makes it become complex, inflexible, and difficult to understand.[72] Some new-generation languages have been proposed and consolidated in industrial environments. Therefore, in the future, language like UML4SPM[8–12] or PLM4BS[13] will be applied. Furthermore, we will draw on the experience and problems of software industry experiments summarized by Yan et al[31] and use expert consultation method Delphi[8] for future real industrial project experiments to validate our proposal.

## ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under Grants 61862063, 61502413, and 61262025; the National Social Science Foundation of China under Grant 18BJL104; the Natural Science Foundation of Yunnan Province under Grant 2016FB106; the Science Foundation of Key Laboratory of Software Engineering of Yunnan Province under Grant 2015SE202; and the Data Driven Software Engineering Innovative Research Team Funding of Yunnan Province under Grant 2017HC012.

## ORCID


*Xuan Zhang* https://orcid.org/0000-0003-2929-2126

## REFERENCES

1. ARC Advisory Group. Automation and software expenditures for discrete industries. 2018. https://www.arcweb.com/market-studies/automation-software-expenditures-discrete-industries.
2. Vyatkin V. Software engineering in industrial automation: state-of-the-art review. *IEEE Trans Ind Inform*. Aug. 2013;9(3):1234-1249.
3. Avižienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans Dependable Secur Comput*. 2004;1(1):1-23.
4. Jin Z. *Environment Modeling-Based Requirements Engineering for Software Intensive Systems*. Cambridge, MA, USA: Elsevier Inc.; 2018.
5. Zhang X, Wang X, Kang YN. Trustworthiness requirement-oriented software process modeling. *J Softw Evol Proc*. 2018;30(12):1-28.
6. Pressman RS. *Software Engineering: A Practitioner's Approach*. 5th ed. New York: McGraw-Hill Companies, Inc.; 2001.
7. Yousfi A, Hewelt M, Bauer C, Weske M. Toward uBPMN-based patterns for modeling ubiquitous business processes. *IEEE Trans Ind Inform*. Aug. 2018;14(8):3358-3367.
8. Bendraou R, Gervais MP, Blanc X. UML4SPM: a UML2.0-Based Metamodel for Software Process Modelling. In: Briand L, Williams C, eds. *Model Driven Engineering Languages and Systems. International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Vol.3713 Berlin, Heidelberg: Lecture Notes in Computer Science, Springer; 2005:17-38.
9. Bendraou R, Gervais MP, Blanc X. UML4SPM: an executable software process modeling language providing high-level abstractions. *The 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC), IEEE*. 2006;297-306.
10. Bendraou R, Sadovykh A, Gervais MP, Blanc X. *Software process modeling and execution: the UML4SPM to WS-BPEL approach*. Lubeck, Germany: The 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA); 2007:314-321.
11. Bendraou R, Jezequel JM, Fleurey F. Combining aspect and model-driven engineering approaches for software process modeling and execution. *Trustworthy Software Development Processes, Springer*. 2009;148-160.
12. Bendraou R, Jezequel JM, Fleurey F. Achieving process modeling and execution through the combination of aspect and model-driven engineering approaches. *J Softw-Evol Proc*. 2012;24(7):765-781.
13. Garcia-Garcia JA, Garcia-Borgonon L, Escalona MJ, Mejias M. A model-based solution for process modeling in practice environments: PLM4BS. *J Softw-Evol Proc*. 2018;30(12):e1982,1-23.
14. Potter B. Microsoft SDL threat modelling tool. *Netw Security*. 2009;2009:15-18.

15. Howard M, Lipner S. *The Secure Development Life-Cycle*. Redmond: Microsoft Press; 2006.
16. SAE. *SAE J3061: surface vehicle recommended practice-cybersecurity guidebook for cyber-physical vehicle systems*. Warrendale, PA, USA, Tech. Rep. 01-2016: SAE Int.; 2016.
17. DEWI. EU project dependable embedded wireless infrastructure. 2019. http://www.dewiproject.eu/.
18. Lyu R. *Handbook of Software Reliability Engineering*. Washington: IEEE Computer Society; 1996.
19. Musa D. *Software Reliability Engineering*. McGraw-Hill: Columbus; 1999.
20. Ericson A. *Hazard Analysis Techniques for System Safety*. Hoboken, NJ: John Wiley & Sons; 2005.
21. United Stated Department of Defense. Standard practice for system safety, MIL-STD-882D. Feb., 2000. http://www.system-safety.org/Documents/MIL-STD-882D.pdf.
22. *International standard: IEC 61508 functional safety of electrical electronic programmable electronic safety-related systems, in Part1–Part7*. Geneva, Switzerland: *Int*. Electrotech. Commission (IEC), 2010-2016.
23. Anderson R. *Security Engineering*. Hoboken, NJ: John Wiley & Sons; 2008.
24. OWASP (the open web application security project). Software assurance maturity model—a guide to building security into software development. 2009. http://www.opensamm.org/.
25. Jackson D. A direct path to dependable software. *Commun ACM*. Apr. 2009;52(4):78-88.
26. Akkaya I, Derler P, Emoto S, Lee EA. Systems engineering for industrial cyber-physical systems using aspects. *Proc IEEE*. 2016;104(5):997-1012.
27. Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives ZG. *DBpedia: a nucleus for a web of open data*. Busan, Korea: Proc. Int. Semantic Web Conf., Asian Semantic Web Conf; 2007:722-735.
28. Suchanek FM, Kasneci G, Weikum G. *YAGO: a core of semantic knowledge*. Banff, Canada: Proc. Int. conf. World Wide Web; 2007:697-706.
29. Suchanek FM, Kasneci G, Weikum G. YAGO: a large ontology from Wikipedia and WordNet. *Web Semantics: Sci Serv Agents World Wide Web*. 2008;6:203-217.
30. Hoffart J, Suchanek FM, Berberich K, Weikum G. YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. *Artif Intell*. 2013;194:28-61.
31. Yan JH, Wang CY, Cheng WL, Gao M, Zhou AY. A retrospective of knowledge graphs. *Front Comp Sci*. 2018;12(1):55-74.
32. Steger M, Boano CA, Niedermayr T, et al. An efficient and secure automotive wireless software update framework. *IEEE Trans Ind Inform*. May, 2018;14(5):2181-2193.
33. Basili VR, Caldiera G, Rombach HD. *Goal Question Metric Paradigm*. New York, NY, USA: Wiley; 1994.
34. Briand L, Penta MD, Labiche Y. Assessing and improving state-based class testing: a series of experiments. *Trans Softw Eng*. 2004;30(11):770-793.
35. Oppenheim AN. *Questionnaire Design, Interviewing and Attitude Measurement*. London: Pinter; 1992.
36. Wang DD, Wang Q, Yang Y, Li Q, Wang HT, Yuan F. Is it really a defect?—an empirical study on measuring and improving the process of software defect reporting. In Proc IEEE Int Symp Empirical Softw Engineering and Measurement, pp. 434-443, 2011.
37. Jamro M. POU-oriented unit testing of IEC 61131-3 control software. *IEEE Trans Ind Inform*. Oct. 2015;11(5):1119-1129.
38. Nysetvold AG, Krogstie J. *Assessing business process modeling languages using a generic quality framework*. 5 Hershey, PA, USA: Advanced Topics in Database Research. IGI Global; 2006:79-93.
39. Davis FD. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q*. 1989;13(3):319-340.
40. Albert W, Tullis T. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. San Francisco, CA, USA: Newnes; 2013.
41. *International standard: IEC 62061 safety of machinery—functional safety of safety-related electrical electronic and programmable electronic control systems*. Geneva, Switzerland: *Int*. Electrotech. Commission (IEC), 2015.
42. Frey G, Litz L. Formal methods in PLC programming. *Proc. IEEE Int. Conf. Syst. Man, Cybern*. 2000;4:2431-2436.
43. Johnson TL. Improving automation software dependability: a role for formal methods? *Control Eng Practice*. 2007;15:1403-1415.
44. Hanisch HM, Hirsch M, Missal D, Preuße S, Gerber C. One decade of IEC 61499 modeling and verification—results and open issues. *Proc. Inf. Control Problems Manuf. Conf., Moscow, Russia*. 2009;211-216.
45. Kim J, Kang I, Choi JY, Lee I. Timed and resource-oriented statecharts for embedded software. *IEEE Trans Ind Inform*. Nov. 2010;6(4):568-578.
46. Schütz D, Wannagat A, Legat C, Vogel-Heuser B. Development of PLC-based software for increasing the dependability of production automation systems. *IEEE Trans Ind Inform*. Nov. 2013;9(4):2397-2406.
47. Asghari SA, Taheri H, Pedram H, Kaynak O. Software-based control flow checking against transient fault in industrial environments. *IEEE Trans Ind Inform*. Feb., 2014;10(1):481-490.
48. Givehchi O, Landsdorf K, Simoens P, Colombo AW. Interoperability for industrial cyber-physical systems: an approach for legacy systems. *IEEE Trans Ind Inform*. Dec., 2017;13(6):3370-3378.
49. Bhatti ZE, Roop PS, Sinha R. Unified functional safety assessment of industrial automation systems. *IEEE Trans Ind Inform*. Feb., 2017;13(1):17-26.
50. Freitas EP, Wehrmeister MA, Pereira CE, Wagner FR, Silva ET Jr, Carvalho FC. DERAF: a high-level aspects framework for distributed embedded real-time systems design. *In Proc. 10th Int. Workshop on Early Aspects*, pp. 55-74, 2007.
51. Wehrmeister MA, Freitas EP, Pereira CE, Rammig F. GenERTiCA: a tool for code generation and aspects weaving. *In Proc. 11th IEEE Int. Symp. Object Oriented Real-Time Distrib*., pp. 234-238, 2008.
52. Binotto A, Freitas EP, Pereira CE, Larsson T. Towards dynamic task scheduling and reconfiguration using an aspect oriented approach applied on real-time concerns of industrial systems. *In Proc. 13th IFAC Symp. Inf. Control Problems Manuf., Moscow, Russia*, pp. 1423-1428, June, 2009.
53. Wehrmeister MA, Pereira CE, Ramming FJ. Aspect-oriented model-driven engineering for embedded systems applied to automation systems. *IEEE Trans Ind Inform*. Nov. 2013;9(4):2373-2386.
54. Roque AS, Pohren D, Freitas EP, Pereira CE. An approach to address safety as non-functional requirements in distributed vehicular control systems. *J Control Autom Electr Syst*. 2019;30(10):700-715.
55. Lastra JLM, Delamer M. Semantic web services in factory automation: fundamental insights and research roadmap. *IEEE Trans Ind Inform*. Feb. 2006;2(1):1-11.
56. Niemelä E, Evesti A, Savolainen P. Modeling quality attribute variability. In *Proc. Int. Conf. Evaluation of Novel Approaches to Software Engineering*, pp. 169-176, Jan. 2008.

57. Lopez C, Astudillo H, Cysneiros LM. Semantic-aided interactive identification of reusable NFR knowledge fragments. In: Meersman R, Tari Z, Herrero P, eds. *On the Move to Meaningful Internet Systems: OTM 2008 Workshops. Lecture Notes in Computer Science*. Vol.5333 Berlin, Heidelberg: Springer; 2008.
58. Elahi G, Yu E, Zannone N. A modeling ontology for integrating vulnerabilities into security requirements conceptual foundations. In: Laender AHF, Castano S, Dayal U, Casati F, de Oliveira JPM, eds. *Conceptual Modeling—ER 2009. Lecture Notes in Computer Science*. Vol.5829 Berlin, Heidelberg: Springer; 2009.
59. Gandhi RA, Lee SW. Discovering multidimensional correlations among regulatory requirements to understand risk. *ACM Trans Softw Eng Methodol*. Sep., 2011;20(4, article 16):1-37.
60. Saeki M, Ashi SH, Kaiya H. Enhancing goal-oriented security requirements analysis using common criteria-based knowledge. *Int J Softw Eng Knowl Eng*. 2013;23(5):695-720.
61. Souag A, Salinesi C, Wattiau I, Mouratidis H. Using security and domain ontologies for security requirements analysis. *proc. IEEE Annual Computer Software and Applications Conference Workshops*. 2013;101-107.
62. Balushi THA, Sampaio PRF, Loucopoulos P. Eliciting and prioritizing quality requirements supported by ontologies: a case study using the ElicitO framework and tool. *Expert Syst*. May, 2013;30(2):129-151.
63. Hästbacka D, Kuikka S. Semantics enhanced engineering and model reasoning for control application development. *Multimed Tools Appl*. Jul. 2013; 65(1):47-62.
64. Zhou JL, Hänninen K, Lundqvist K, Lu Y, Provenzano L, Forsberg K. An environment-driven ontological approach to requirements elicitation for safety-critical systems. *proc. IEEE Int. Req. Eng. Conf., Ottawa, ON, Canada*. 2015;247-251.
65. Provenzano L, Hänninen K, Zhou JL, Lundqvist K. An ontological approach to elicit safety requirements. *proc. IEEE Asia-Pacific Sofw. Eng. Conf*. 2017; 713-718.
66. Palomares C, Quer C, Franch X. Requirements reuse and requirement patterns: a state of the practice survey. *Empir Softw Eng*. 2017;22:2719-2762.
67. Supakkul S, Hill T, Chung L, Tun TT, do Prado Leite JCS. An NFR pattern approach to dealing with NFRs. *proc. IEEE Int. Req. Eng. Conf*. 2010;179-188.
68. Salini P, Kanmani S. A knowledge-oriented approach to security requirements engineering for e-voting system. *Int J Comput Appl*. July, 2012;49(11): 21-25.
69. Daramola O, Sindre G, Stalhane T. Pattern-based security requirements specification using ontologies and boilerplates. *In Proc. IEEE 2nd Int. Workshop on requirements patterns, Chicago*, pp. 54-59, 2012.
70. Alexander C. *The Timeless Way of Building*. New York: Oxford University Press; 1979.
71. Franch X, Quer C, Renault S, Guerlain C, Palomares C. Constructing and using software requirement patterns. In: Maalej W, Thurimella A, eds. *Managing Requirements Knowledge*. Berlin, Heidelberg: Springer; 2013.
72. García-Borgoñón L, Barcelona MA, García-García JA, Alba M, Escalona MJ. Software process modeling languages: a systematic literature review. *Inform Softw Technol*. 2014;56:103-116.


**How to cite this article:** Zhang X, Wang X, Yun W, Gao C, Han M, Liu H. Pattern-based software process modeling for dependability. *J Softw Evol Proc*. 2020;e2262. https://doi.org/10.1002/smr.2262