Requirements Engineering (2023) 28:97–115
https://doi.org/10.1007/s00766-022-00377-5

ORIGINAL ARTICLE

Aligning requirements and testing through metamodeling and patterns: design and evaluation

Taciana Novo Kudo[1,2] · **Renato de Freitas Bulcão-Neto**[2] · **Valdemar Vicente Graciano Neto**[2] · **Auri Marcelo Rizzo Vincenzi**[1]

Received: 17 December 2020 / Accepted: 23 February 2022 / Published online: 1 June 2022

Abstract
Poorly executed requirements engineering activities profoundly affect the deliverables' quality and project's budget and schedule. High-quality requirements reuse through requirement patterns has been widely discussed to mitigate these adverse outcomes. Requirement patterns aggregate similar applications' behaviors and services into well-defined templates that can be reused in later specifications. The abstraction capabilities of metamodeling have shown promising results concerning the improvement of the requirement specifications' quality and professionals' productivity. However, there is a lack of research on requirement patterns beyond requirements engineering, even using metamodels as the underlying structure. Besides, most companies often struggle with the cost, rework, and delay effects resulting from a weak alignment between requirements and testing. In this paper, we present a novel metamodeling approach, called Software Pattern MetaModel (SoPaMM), which aligns requirements and testing through requirement patterns and test patterns. Influenced by well-established agile practices, SoPaMM describes functional requirement patterns and acceptance test patterns as user stories integrated with executable behaviors. Another novelty is the evaluation of SoPaMM's quality properties against a metamodel quality evaluation framework. We detail the evaluation planning, discuss evaluation results, and present our study's threats to validity. Our experience with the design and evaluation of SoPaMM is summarized as lessons learned.

Keywords Requirement · Testing · Pattern · Metamodel · Quality · Evaluation

1 Introduction

The value of requirements engineering (RE) strongly impacts software projects whenever requirements-related activities are poorly executed. Incorrect, omitted, misinterpreted, or conflicting requirements usually result in extrapolated budget and delivery times [16, 44].

In the last decade, requirements reuse [12, 13, 17, 19, 22] has been a feasible alternative to mitigate those issues, making the RE tasks more prescriptive and systematic while facilitating the reuse of existing requirements artifacts. A fairly discussed reuse approach is the requirement pattern (RP) concept, which is an abstraction that aggregates behaviors and services observed in multiple similar applications [46]. Usually, RP guides requirements elicitation and specification through well-defined templates that can be reused in later specifications [1, 14, 29].

A promising approach for representing RP is through metamodeling [8] because it raises the level of abstraction at which software is conceived, implemented, and evolved. As related work, Franch et al. [18] propose a metamodel that defines a structure of requirement patterns themselves, the relationships among them, and classification criteria for grouping them. Ya'u et al. [49]'s metamodel comprises a reusable structure, variability modeling, and traceability of software artifacts for software product line engineering. Metamodeling provides a general representation structure for RP toward improving the quality of specifications and the requirements engineers' productivity. Despite those benefits to the RE process, the traceability between RP and software artifacts produced in other development phases (e.g., other types of software patterns) is still a road to pave [7].

✉ Taciana Novo Kudo
taciana@ufg.br

[1] Departamento de Computação, Universidade Federal de São Carlos (UFSCar), SP, São Carlos, Brazil

[2] Instituto de Informática, Universidade Federal de Goiás (UFG), GO, Goiânia, Brazil

Table 1 Metamodeling approaches for requirement patterns

References	Pattern	Formalism	Tool	Evaluation
[45]	(N)FRP	CNL	–	–
[49]	NFRP	NL	–	–
[18]	(N)FRP	NL	PABRE-Man	Catalogues
SoPaMM	(N)FRP ATP	CNL	TMEd	Catalogues and quality

For instance, consider the alignment between RE and testing. The popular V-model [39] highlights the influence of requirements activities in the software development life cycle (SDLC) by interrelating the user acceptance testing and requirement analysis phases to determine whether a software system satisfies the requirements specified. However, most software companies still struggle with the cost, rework, and delay effects resulting from a weak alignment between requirements and testing [10, 15].

In this paper, our general goal is to extend RP's benefits to other SDLC stages beyond RE. We describe the design and the evaluation of a metamodeling strategy to relate different software patterns, called Software Pattern MetaModel (SoPaMM) [2]. Currently, SoPaMM aligns functional requirement patterns (FRP) and acceptance test patterns (ATP), which structure generic testing solutions to recurrent behaviors from different scenarios [31] and help a tester understand the context of a testing practice [30]. Potential candidates as an ATP include repetitive, alike, and high-value test practices.

SoPaMM defines how FRP and ATP can be written, organized, related, and classified. SoPaMM borrows concepts and practices from the Behavior-Driven Development (BDD) agile methodology [11] by describing FRP via user stories and associating it with behaviors through the Gerkhin language. Integrating with existing agile methodologies leverages the use of SoPaMM, as these methodologies are common practices in today's software industry. Furthermore, the description of FRPs with BDD concepts allows, with adequate tool support, to automate acceptance tests associated with FRPs.

As metamodel quality impacts the terminal models' quality,[1] we evaluated SoPaMM using the Metamodel Quality Requirements and Evaluation (MQuaRE) framework [3], comprising an evaluation process, metamodel quality requirements and measures, and a quality model. Evaluation results report the SoPaMM's levels of compliance, conceptual suitability, usability, maintainability, and portability under six evaluators' perspective.

This work's contributions are threefold:

1. A metamodeling solution for aligning requirements and testing;
2. The joint use of requirement patterns and an agile testing methodology;
3. The metamodel quality evaluation.

This paper is organized as follows: Sect. 2 discusses related work; Sect. 3 presents background; Sects. 4 and 5 describe the SoPaMM metamodel and how to use it to build a catalogue containing requirement patterns and test patterns; Sect. 6 presents the evaluation of the metamodel using a metamodel quality assessment framework, and Sect. 7 presents conclusions and future work.

2 Related work

This section compares metamodeling approaches for software requirement patterns. We examine related work considering the metamodel patterns, the formalism for patterns representation, tool support, and metamodel evaluation. Table 1 summarizes the comparison between SoPaMM and related work.

Franch et al. [18] define the structure of an RP, the types of relationships among RPs, and classification criteria for grouping them. The main idea behind using metamodeling is to provide more flexibility on how to model RPs by decoupling the types of RPs and allowing the types of relationships more configurable. Videira and Da Silva [45] develop a requirements specification language based on identifying the most frequent linguistic patterns used in requirements documents. Such specification language relies on metamodel concepts and mapping rules between these and sentences found in requirements documentation. In turn, Badamasi et al. [49] present a metamodel-based representation for RPs, including variability modeling and software artifacts traceability. The main goal is to improve the systematic reuse of RPs by integrating concepts of a software product line and model-driven engineering.

Two of these proposals [18, 45] cover both functional and nonfunctional requirement patterns (FRP and NFRP, respectively). In contrast, one work represents NFRP only [49]. To the best of the authors' knowledge, the SoPaMM metamodel we propose is the only one proposing the alignment

[1] Terminal models are metamodel instances as defined in the Meta-Object Facility (MOF) architecture [33].

of functional requirement patterns (FRP) and acceptance test patterns (ATP).

Multiple representation formats have been used: traditional natural language (NL) [18, 49], and controlled natural language (CNL) with a subset of meaningful terms for pattern representation [45]. Comparatively, our metamodeling solution goes further by using two easy-to-use and widely accepted controlled natural languages in the agile software industry: user stories and the Gherkin language [40] for requirements and behaviors specification, respectively.

Tools are an effective strategy to assist practitioners' practices in the use of requirement patterns. In previous work [4], we report the development of the Terminal Model Editor (TMEd), a tool that facilitates the definition of software patterns (i.e., instances of SoPaMM) and the maintenance and evolution of a patterns catalogue. What differentiates TMEd and PABRE-Man [18, 35] is the type of software pattern covered by the latter, i.e., only requirement patterns, whereas TMEd also handles test patterns.

Finally, a single-related work evaluates its metamodel approach elaborating on catalogues of FRP and NFRP as metamodel instances [14, 18, 36]. Application domains covered by these catalogues of patterns include content management and call for tender processes. In our proposal, we developed a catalogue containing NFRP as well as FRP aligned to ATP for the certification of electronic health record systems [2, 4]. In this paper, we go further by evaluating the quality of the SoPaMM metamodel using a quality evaluation framework [3, 5] as the existing metamodeling-based pattern approaches lack such an evaluation perspective.

In brief, the SoPaMM metamodel has distinct but significant aspects compared with related work:

- it is the only solution that bridges RPs to other software patterns such as ATPs; this allows representing more complex and valuable pattern-based artifacts such as a test traceability matrix;
- it borrows usual agile practices (user story and the Gherkin notation) and takes advantage of tool support; this may reduce the SoPaMM's learning curve.
- it is the only one evaluated from a quality perspective; this allows anticipating software patterns' quality built upon it regarding multiple characteristics.

3 Background

This section presents two key components of our metamodeling approach aligning requirement and test patterns: the Behavior-Driven Development (BDD) methodology and the Metamodel Quality Requirements and Evaluation (MQuaRE) framework.

3.1 Behavior-driven development (BDD)

BDD describes a software process widely adopted in agile software engineering practices [32]. BDD's main goal is to close the gap between business and technical teams regarding understanding the expected behavior of the software to be developed [11]. Therefore, the key element in BDD is the software's behavior.

To achieve collaboration and shared comprehension between people with likely different expectations, two BDD practices deserve further special attention: software functional specification as user stories and the alignment of each user story with executable scenarios.

User stories describe software features using a natural language syntax: "AS a <role>, I CAN <capability>, SO THAT <receive benefit>)". As such, user stories are more closely related to business goals, facilitating communication in a project.

Furthermore, domain experts, testers, and developers collaborate on describing scenarios as features' expected behaviors written in the Gherkin language. The natural order of a scenario is: *Given* one or more preconditions, *When* a set of actions is performed, *Then* an outcome is obtained.

Next, we present the *Notifications* feature and the Gherkin syntax describing one of this feature's desired behavior (*someone likes a post*). Observe the level of details expressed in each scenario's component (preconditions, execution steps, and results), including test data examples.

```
Feature: Notifications
  As a system user
  I can get notifications
  So that I can see received notifications
Scenario: someone likes my post
  Given a user with email "john@doe" is connected
  with "mary@ann"
        And "mary@ann" has a public post with text
        "check this out!"
  When I sign in as "john@doe"
        And I am on "mary@ann"'s page
        And I follow "Like"
        And I sign out
  When I sign in as "mary@ann"
        And I follow "Notifications" in the header
  Then the notification dropdown should be visible
        And I should see "john@doe liked your post!"
        And I should have 1 email delivery
```

This easy-to-use but powerful behavior specification syntax enables BDD-oriented tools can automatically generate technical and end-user documentation, such as test case specifications. Our metamodeling proposal benefits from how BDD describes features as user stories integrated with behaviors represented as executable scenarios.

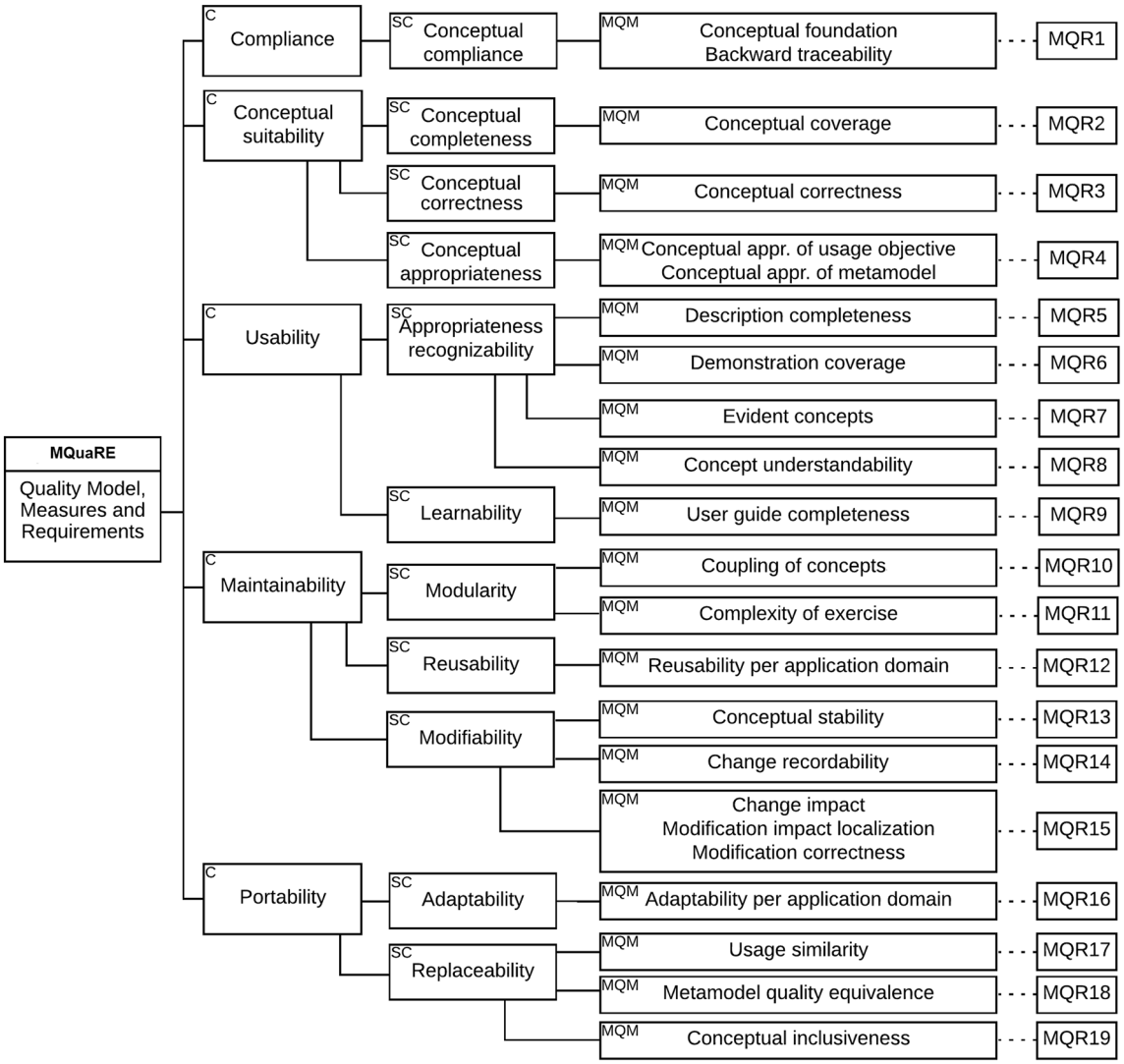

Fig. 1 The MQuaRE quality model, measures and requirements

3.2 The MQuaRE framework

MQuaRE is an integrated framework composed of an evaluation process that arranges metamodel quality requirements (MQR) and measures (MQM) and a metamodel quality model with activities, tasks, input and output artifacts, and users' roles [3]. The metamodel quality characteristics and sub-characteristics were set by compiling and comparing related research contributions [27, 28, 38, 43] with international standards' quality models, such as ISO/IEC 25010 [24] and ISO/IEC 9126 [23], with minor amendments.

MQR may comprise multiples aspects of a metamodel, e.g., whether it is easy to use and maintain or compliant to specific standards. The current version of MQuaRE provides 19 MQRs that meet those preconditions and can be reused by metamodel users [5].

The MQuaRE's quality model categorizes the MQRs into five characteristics (C) subdivided into eleven sub-characteristics (SC) described in Fig. 1. MQuaRE also includes 23 MQMs bound to its quality model, i.e., these are quantifications of quality characteristics and sub-characteristics of a metamodel under evaluation (see Fig. 1). For the sake of brevity, we outline each metamodel quality characteristic in terms of its respective quality measures. Further information can be found elsewhere [3, 5].

1. *Compliance*: the degree to which the conceptual foundation of a metamodel complies with theories, regulations, standards, and conventions.
2. *Conceptual suitability*: the degree to which the set of metamodel concepts covers all the specified requirements, is correctly modeled, facilitates the accomplish-

ment of modeling tasks, and is appropriate for performing these tasks.
3. *Usability*: the degree to which users can recognize whether a metamodel is appropriate for their needs considering the metamodel specifications' particularities. For instance, the completeness and demonstration capability of usage scenarios, the clearness and correct understanding of metamodel concepts, and the documentation's guidance degree for metamodel usage.
4. *Maintainability*: the degree to which changes result in minimal impact on the metamodel structure, the metamodel can be used in more than one application domain, and the metamodel can be effectively and efficiently modified without introducing inconsistencies or degrading its quality.
5. *Portability*: the degree to which a metamodel can effectively and efficiently be adapted for different application domains and which a metamodel can replace another metamodel for the same purpose.

The MQuaRE's evaluation process contains five main activities performed by an evaluation requester or an evaluator, as follows:

1. Establish the metamodel evaluation requirements: the evaluation requester defines the MQRs according to the general evaluation purpose (e.g., estimate the final quality of metamodel, compare between distinct metamodels for the same domain, or assess the positive and negative aspects of a metamodel).
2. Specify the metamodel evaluation: based on the MQRs defined, the evaluation requester selects the MQMs, establishes the respective target value and acceptable tolerance value, and defines formulas to calculate the quality grades of characteristics and sub-characteristics. This amount of information constitutes a high-level evaluation plan.
3. Design the metamodel evaluation: taking the previous evaluation plan as a starting point, the evaluation requester elaborates on a detailed metamodel quality evaluation plan (MQEP), containing target metamodel specifications, the measurement functions for each MQM, an evaluation schedule, and others.
4. Execute the metamodel evaluation: the metamodel evaluator uses the information of MQEP to calculate metamodel quality measurements, apply the target value and acceptable tolerance value, compute the quality grades of quality characteristics and sub-characteristics, and make observations about likely problems during the evaluation.
5. Conclude the metamodel evaluation: the metamodel evaluator and the evaluation requester shall carry out a joint review of the evaluation results. All documentation generated must be reassessed, and adaptations can be made when justified and documented.

Regarding our metamodel's quality evaluation, we systematically performed the MQuaRE process and its activities. Further details are found in Sect. 6.

4 The SoPaMM metamodel

The SoPaMM's core idea is the specification of functional requirement patterns (FRP) linked to acceptance test patterns (ATP). Figure 2 depicts the most significant components of an FRP related to an ATP. For the sake of conciseness, some FRP and ATP metadata are not shown.

Influenced by the BDD agile methodology, an FRP is a composition of *Feature* elements described through the user story syntax, as such:

As: the stakeholder who benefits from the Feature;
I_can: the Feature itself;
So_that: the Feature's aggregated value.

Using the BDD's Gherkin syntax, one or more *Scenarios* represent Feature's behaviors, where:

Given describes, in one or more clauses, the Scenario's initial context;
When describes the events that trigger a Scenario;
Then describes, in one or more clauses, the Scenario's expected outcomes.

In the example of Fig. 2, *FRP_User_Creation* describes an excerpt of a user creation feature. Observe the FRP structure in which an administrator user is the stakeholder who benefits from this feature in order that a new user is registered for the system. This feature has two behaviors represented: a successful and an unsuccessful scenario, both linked to a same precondition, i.e., the attempt of creating a new user. But, the scenarios' execution steps are distinct regarding the user data's validity. Similarly, one different outcome is represented for each scenario, i.e., the new user registration and the display of an error message.

Finally, the *Example* concept allows defining and linking multiple data to each scenario. Observe that each scenario has two data instances so that the one scenario registers a new user successfully, whereas the other scenario does not due to invalid examples of user identification number. This FRP-Feature-Scenario-Example representation is what defines our *behavior-driven functional requirement pattern* approach.

Now, consider the representation of *ATP_User_Creation* in Fig. 2. The ATP is composed of two test cases, each

Fig. 2 The structure and contents of an FRP associated with an ATP

```
FUNCTIONAL REQUIREMENT PATTERN
Name: FRP_User_Creation
Problem: The user must be registered for the system
Forces: Ensuring that the user is successfully created.

  FEATURE USER CREATION
  As: an administrator
  I_can: create a new user
  So_that: he/she is registered for the system

    SCENARIO SUCCESSFUL USER CREATION
    Given: I am trying to create a new user
    When: I enter <ITRN>, <name>, <gender>, <date of birth>,
    <father's name>, <mother's name> and <role>
    Then: the system should register a new user with these data

      EXAMPLE
      735.101.320-92 | Carlos Chagas | Male | 01.01.2000 | Jose
      Chagas | Carla Chagas | Doctor

      342.052.020-40 | Jose Mouro Brasil | Male | 01.01.2000 | Jose
      Brasil | Josefa Brasil | Ophthalmologist Doctor

    SCENARIO NOT SUCCESSFUL USER CREATION - INVALID ITRN
    Given: I am trying to create a new user
    When: I enter an invalid <ITRN>
    Then: The system should display the <message> error message

      EXAMPLE
      735.101.320-00 | "This is an invalid individual taxpayer
      registration number!"
      342.052.020-00 | "This is an invalid individual taxpayer
      registration number!"


ACCEPTANCE TEST PATTERN
Name: ATP_User_Creation
Problem: The user creation feature must be tested against its
scenarios and respective example data.
Forces: Ensuring that all user creation scenarios are successfully
tested.

  Test Case SUCCESSFUL USER CREATION
    InputData ← Example of Scenario SUCCESSFUL USER CREATION
    OutputData ← Example of Scenario SUCCESSFUL USER CREATION

  Test Case NOT SUCCESSFUL USER CREATION - INVALID ITRN
    InputData ← Example of Scenario NOT SUCCESSFUL USER CREATION -
    INVALID ITRN
    OutputData ← Example of Scenario NOT SUCCESSFUL USER CREATION
    - INVALID ITRN
```

related to a particular test scenario, i.e., (un)successful user creation. Each test case contains preconditions, expected results, and postconditions (scenario's *Given-When-Then* clauses) as well as input and output test data from the respective *Example* of *Scenario*.

Once presented the FRP and ATP concepts, we outline the entire SoPaMM metamodel illustrated in Fig. 3. Similar to the PABRE metamodel [18, 35, 36], the *Catalogue* element is a means of systematically gathering patterns, usually addressing the most common problems for a particular application domain. The *Catalogue* concept is the coarsest grained reuse unit in SoPaMM.

A *Software Pattern Bag* (SPB) is a composition of multiple *Software Pattern* (SP) elements that, in turn, represent an extensible point to accommodate different types of SP, such as requirement, test, or even design patterns, with minimal impact on the structure defined. Unlike other pattern catalogues found in the literature, the SPB concept in SoPaMM

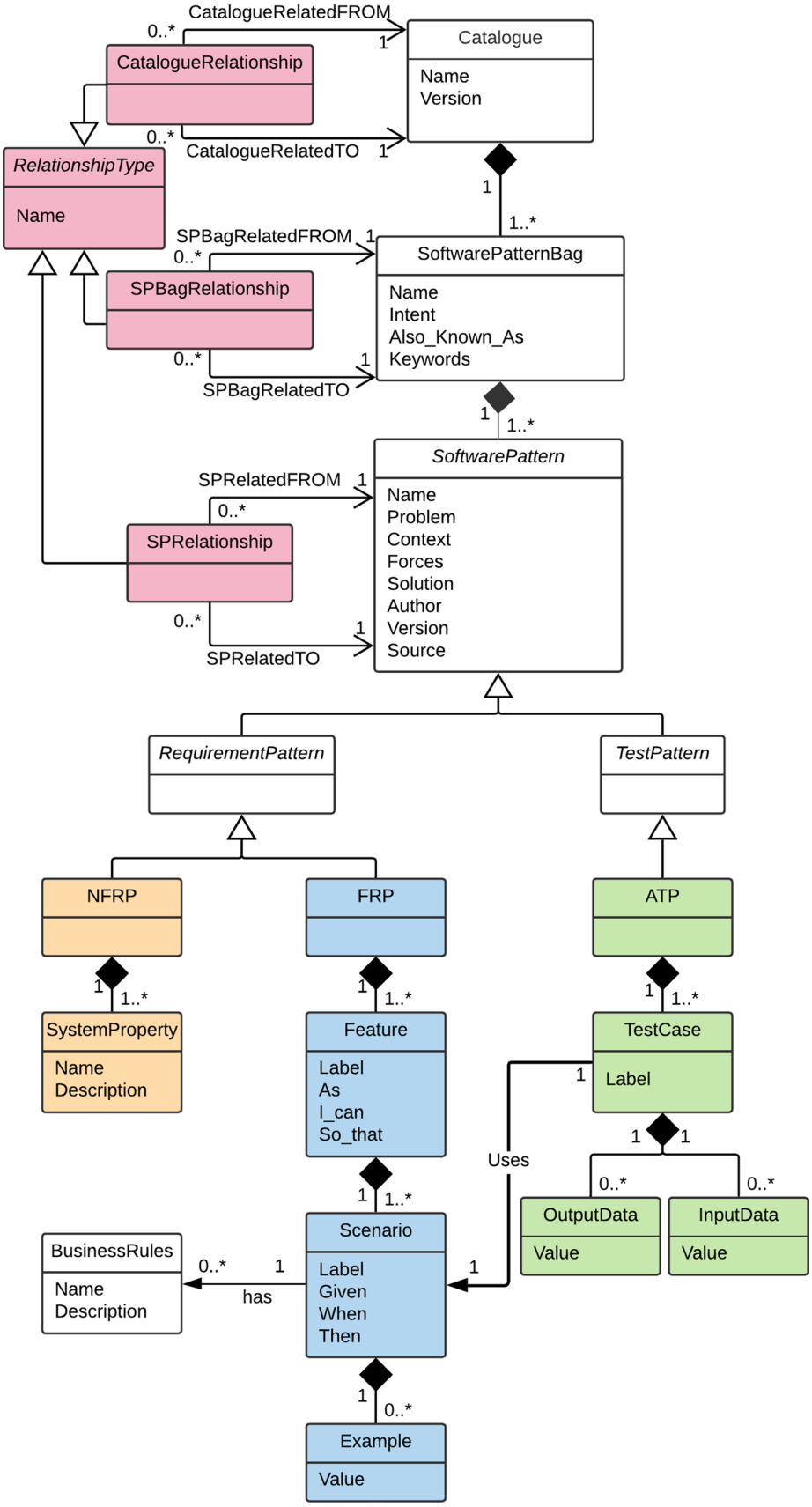

Fig. 3 The SoPaMM meta-model

allows organizing, in a same catalogue, software patterns for problems at different stages of the SDLC. In comparison with *Catalogue*, an SPB works as a fine-grained reuse unit of multiple types of related patterns (e.g., FRP and ATP) and consequently may be of great utility during the development of software project documentation.

Note in Fig. 3 that industry standards [34] and classic literature on software patterns [20, 21, 37, 46] contribute to SP metadata's definition (e.g., problem, context, forces, solution).

Although the literature defines relationship types between requirement patterns (e.g., *extends*, *has*, *uses*) [46], we implement a broader definition of relationship types in SoPaMM. Due to flexibility reasons, these are not predefined (attribute *Name* in *RelationshipType*) and allow relating catalogues, software pattern bags, and software patterns in general (i.e., *CatalogueRelationship*, *SPBagRelationship*, and *SPRelationship*, respectively).

Non-functional requirement pattern (NFRP) is a composition of software system properties (behavioral constraints or quality attributes) described by textual attributes (i.e., name and description). As an example of NFRP for several existing applications, user credentials must be validated by an authentication server, forbidding user authentication on the client-side.

Noteworthy that this is an enhanced version of SoPaMM. The main differences from its previous versions [2, 4] are the insertion of the *Catalogue* concept, the redefinition of the SPB's and SP's attributes, and the reformulation of how to handle test cases. In early versions, a test case was a composition of steps and UI elements (for web and mobile applications) represented by the *Page Object* design pattern [26, 42]. However, the removal of UI elements in the current version makes SoPaMM more flexible and technology-independent.

5 How to use SoPaMM

The purpose of this section is twofold. Firstly, it introduces a pattern catalogue building method using SoPaMM. Secondly, it reports an instantiation of this method on developing a catalogue for electronic health record (EHR) systems.

5.1 A method for developing SoPaMM-based pattern catalogues

This section presents a general method for elaborating on pattern catalogues based on the SoPaMM grammar. Preferably, this method should be performed by a requirements engineer supported by a domain expert, if necessary. However, knowledge about the SoPaMM constructs (e.g., SPB, FRP, and ATP) is mandatory.

Two phases are present: one in which candidate patterns are found (steps 1 to 5) and another in which final patterns are organized and written (steps 6 and 7).

1. Gather as much as possible available requirements documentation, test case specifications, and pattern catalogues.
2. Study each requirement and test case and classify it: which type would it be?
3. If an existing pattern can be applied to it, record that fact and go on.
4. If an existing pattern does not quite fit, study the requirement or test case to see if you could develop a new, more specialized pattern for it. If so, classify it as a functional (FRP) or nonfunctional requirement pattern (NFRP) or acceptance test pattern (ATP), suggest a name, and add it to the list of candidates.
5. When you have gone through all specifications, review the candidate patterns looking for duplicates or overlaps, and resolve these inconsistencies.
6. Then, group the resulting software patterns (SP) into software pattern bags (SPB). Note that an SPB allows the composition of multiple and different SP types, such as FRP, NFRP, and ATP.
7. Write each software pattern.
   (a) If the pattern is an NFRP, create one System Property element for each behavioral constraint or quality attribute found in the specifications. Thus, a particular NFRP may be a composition of System Property elements.
   (b) If the pattern is an FRP, then create a Feature element in it using the user story syntax. For each Feature element, search for scenarios describing its behavior in the specifications available. For each feature's behavior found in the specifications, elaborate on it through the Gherkin language syntax. Otherwise, create a proper scenario from scratch, in Gherkin, if no scenario is linked to a feature. Search in the specifications for data statements that might be used as data examples to execute each scenario describing a feature's behavior. If no data examples are available, create and assign them to each scenario stated.
   (c) If the pattern is an ATP, create and associate a test case with a proper FRP scenario. For each test case defined, classify the scenario's data examples as input and output data. As a result, there must be one test case for each scenario with preconditions, actions, expected outcomes, and the corresponding input and output data examples.

In *step 1*, test case specifications have been given special attention because SoPaMM allows the creation of test patterns and their alignment with requirement patterns. This is a new feature of our approach compared to related work.

In *step 4*, it is not a concern if one cannot foresee a pattern being useful in specifying a requirement or test case. Requirements and test cases do not always conveniently fit the patterns. Unfortunately, there are no strict rules for identifying requirement patterns [46] and test patterns [31].

5.2 SoPaMM instantiation

According to the MOF architecture [33], metamodel instances are called terminal models, which, in turn, describe real-world application models. In our approach, SoPaMM instances are software pattern catalogues serving as guidance for real-world software specifications, e.g., test case specifications. Using the 7-step method previously presented, we instanced SoPaMM to build a pattern catalogue for electronic health record (EHR) systems.

The Brazilian Health Informatics Society (BHIS) offers a quality certification process of EHR systems based on an extensive list of requirements and test scripts elaborated by BHIS's skilled staff. These specifications result from a compilation of several EHR projects and experiences with firm adherence to international standards.

The BHIS's requirements for quality certification of an EHR-System are organized into two Security Assurance Levels (SAL). The first level (SAL1) determines mandatory requirements for exchanging information on supplementary health. The second level (SAL2) allows replacing paper health records with their electronic equivalents. Complimentary, BHIS also provides an operational manual of tests and analysis for EHR system certification composed of test scripts.

Regarding *step 1* of the SoPaMM instantiation process, these requirements and test scripts are the principal information sources, with no support of catalogues.

In *step 2*, a 17-year experienced software engineer with a background in requirements and testing and strong knowledge of the SoPaMM grammar built the pattern catalogue for EHR systems. Aiming at the minimum legal requirements for EHR systems certification, she took into account the 84 SAL1 requirements and the 40 respective test scripts. There was no need for a domain expert due to the discussion of such information by specialists in Health Informatics.

For better comprehension, the BHIS documentation contains the *SAL1.04.08* legal requirement classified as a functional requirement. This requirement describes who has access to the patient's medical record (i.e., only the patient or his/her legal guardian). The BHIS documentation also includes the *SAL1.S018* test script. It describes the set of procedures a user must perform to meet the *SAL1.04.08* requirement. The following is an excerpt of the SAL1.04.08 requirement and the respective SAL1.S018 test script:

```
LEGAL REQUIREMENT CODE: SAL1.04.08
DESCRIPTION: Access to patient's medical record - the EHR system provides the
patient or his/her legal guardian with access to the former's medical record.
The system user must have access only to his/her medical record and can not have
access to other patients' medical records. Should the system user is the
patient's legal guardian, he/she can visualize both his/her medical record and
that patient's medical record.

TEST PROCEDURE CODE: SAL1.S018
RELATED TO: SAL1.04.08
DESCRIPTION: the EHR system allows the patient or his/her legal guardian to
access the former's medical record.

  PROCEDURE 1: a patient must log on to the EHR system and try to access another
  patient's medical record. Then, he/she should view his/her medical record
  contents.
  PROCEDURE 2: a patient's legal guardian must log on to the EHR system and try
  to access another patient's medical record. Next, he/she should view the
  patient's medical record under his/her legal liability.
```

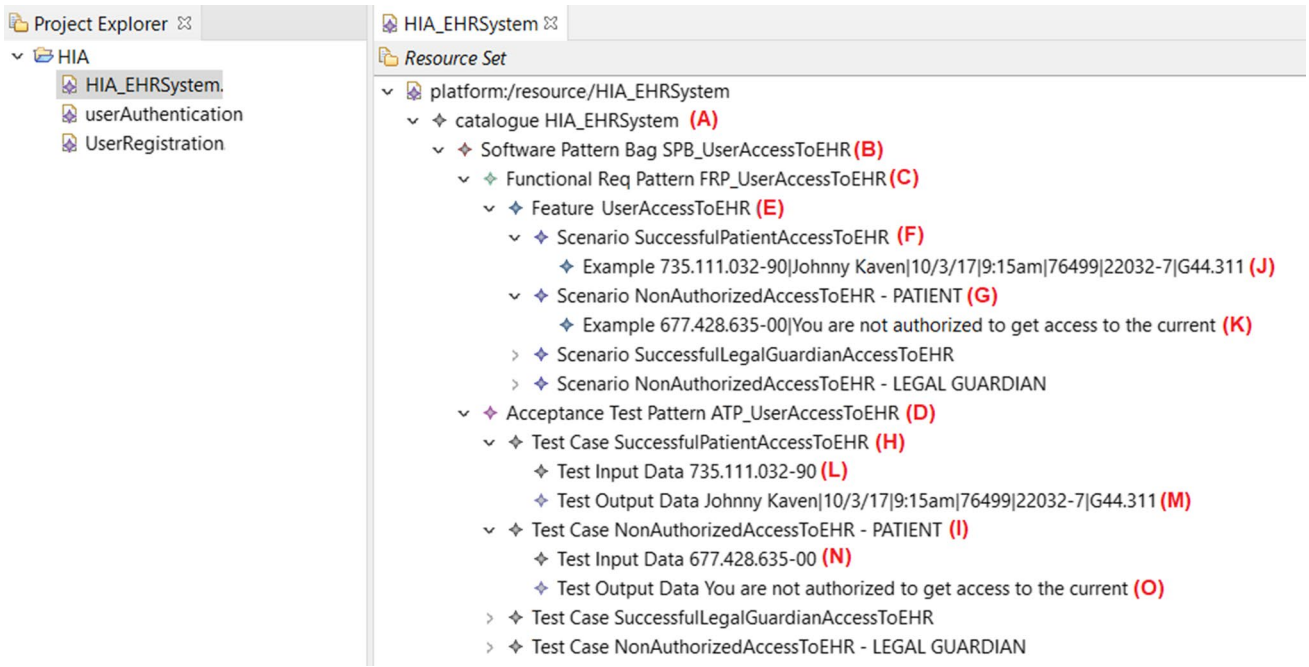

Fig. 4 The pattern catalogue for EHR systems in the TMEd user interface

In the first iterations of the method, *step 3* had no result due to the nonexistence of previous patterns, as described in *step 1*. In other words, the association between a software pattern and an input requirement/test procedure was possible only when the engineer started identifying software patterns.

In the last iteration of classifying software patterns (*step 4*), the engineer recognized 65 FRP, 36 NFRP, and 40 ATP. Excluding similarities and overlaps, the total number of patterns found was 18 FRP, 36 NFRP, and 18 ATP (*step 5*). Access to the patient's medical record is a mandatory function in every EHR system. Thus, it may be specified as an FRP, called *FRP_UserAccessToEHR*, and be reused in various situations of use in an EHR system. For instance, when a patient needs to schedule a clinical exam, view imaging exam reports, or print the medication prescription. Furthermore, regarding the test procedures required to validate that requirement, these may be developed in parallel as a set of test cases constituent of an ATP named *ATP_UserAccessToEHR*.

In *step 6*, the engineer organized the final software patterns into software pattern bags (SPB). Considering both the FRP and the ATP cited, these are interrelated and organized into an SPB named *SPB_UserAccessToEHR*.

Finally, the engineer started the patterns writing process using the SoPaMM constructs (*step 7*), as shown next. The FRP representation comprises a feature (*UserAccessToEHR*) and its respective behaviors as successful and unsuccessful scenarios. Each scenario has input and/or output test data (see EXAMPLE constructs). Besides, observe that the value of the *SOURCE* attribute refers to the requirement and the test procedure (SAL1.04.08 and SAL1.S018) that contribute to the specifications of FRP and ATP.

```
FUNCTIONAL REQUIREMENT PATTERN
NAME: FRP_UserAccessToEHR
PROBLEM: A patient's medical record is accessible solely by him/her or his/her
legal guardian.
FORCES: Ensure non-authorized people do not access the patient's medical record.
SOURCE: SAL1.04.08 and SAL1.S018

  FEATURE UserAccessToEHR
  As an EHR user system
  I can access the patient's medical record
```

```
So that I view the record information.

  SCENARIO SuccessfulPatientAccessToEHR
  Given I am a patient
  When I try to access my medical record using my <patientID>
  Then the system shows my record: <patientName> <admissionDate> <admissionTime>
  <medicalProcedureCodes> <laboratoryExaminationsCodes> <diagnosisCodes>
    EXAMPLE
    735.111.032-90|Johnny Kaven|10/3/17|9:15am|76499|22032-7|G44.311

  SCENARIO NonAuthorizedAccessToEHR - PATIENT
  Given I am a patient
  When I try to access other patient's medical record using a <patientID>
  Then the system should exhibit a message warning about non-authorized
  access to the patient's medical record, as <MESSAGE>
    EXAMPLE
    677.428.635-00|You are not authorized to get access to the current
    patient's medical record!

  SCENARIO SuccessfulLegalGuardianAccessToEHR
  Given I am the patient's legal guardian
  When I try to access that patient's medical record using the <patientID>
  Then the system shows the person's record for which I am legally
  liable, with the following information: <patientName> <admissionDate>
  <admissionTime> <medicalProcedureCodes> <laboratoryExaminationsCodes>
  <diagnosisCodes>
    EXAMPLE
    735.111.032-90|Johnny Kaven|10/3/17|9:15am|76499|22032-7|G44.311

  SCENARIO NonAuthorizedAccessToEHR - LEGAL GUARDIAN
  Given I am the patient's legal guardian
  When I try to access a patient's medical record for which I am not
  legally liable using his/her <patientID>
  Then the system should exhibit a message warning about non-authorized
  access to the patient's medical record, as <MESSAGE>
    EXAMPLE
    735.101.020-40|You are not authorized to get access to the current
    patient's medical record!


ACCEPTANCE TEST PATTERN
NAME: ATP_UserAccessToEHR
PROBLEM: The patient's medical record access should be tested considering
all scenarios and the respective input and output data examples.
FORCES: Ensure that all scenarios of accessing the patient's medical record
are successfully tested.
SOURCE: SAL1.S018

  TEST CASE SuccessfulPatientAccessToEHR
  InputData  EXAMPLE of SCENARIO SuccessfulPatientAccessToEHR
  OutputData  EXAMPLE of SCENARIO SuccessfulPatientAccessToEHR

  TEST CASE NonAuthorizedAccessToEHR - PATIENT
  InputData  EXAMPLE of SCENARIO NonAuthorizedAccessToEHR - PATIENT
  OutputData  EXAMPLE of SCENARIO NonAuthorizedAccessToEHR - PATIENT

  TEST CASE SuccessfulLegalGuardianAccessToEHR
  InputData  EXAMPLE of SCENARIO SuccessfulLegalGuardianAccessToEHR
  OutputData  EXAMPLE of SCENARIO SuccessfulLegalGuardianAccessToEHR

TEST CASE NonAuthorizedAccessToEHR - PATIENT
InputData  EXAMPLE of SCENARIO NonAuthorizedAccessToEHR - LEGAL GUARDIAN
OutputData  EXAMPLE of SCENARIO NonAuthorizedAccessToEHR - LEGAL GUARDIAN
```

As the patterns catalogue specification in *step 7* is not a trivial task, a software engineer usually builds patterns through tool support. Therefore, we developed the Terminal Model Editor (TMEd) tool [4] on top of the Eclipse Modeling Framework (EMF). TMEd allows the creation and editing of every SoPaMM construct (e.g., SPB, FRP, and ATP) and an XML output file generation following the SoPaMM grammar. Besides, it is the only tool supporting the parallel specification of FRP and ATP to the best of the authors' knowledge.

Figure 4 illustrates the TMEd user interface. There is a list of SoPaMM-based pattern catalogues on the left-hand side, including the *HIA_EHRSystem* presented in this section. On the right-hand side is that pattern catalogue contents structured as a tree of SoPaMM elements.

The *Catalogue* (A) is at the highest hierarchical level, and the contents of the *SPB_UserAccessToEHR* bag (B) comprise the *FRP_UserAccessToEHR* and *ATP_UserAccessToEHR* (C and D, respectively). Two patient-related scenarios of the *UserAccessToEHR* feature (E) are highlighted: successful (F) and unsuccessful scenarios (G) associated with the respective test cases (H and I) in the ATP.

Those test cases use the corresponding test data (J and K) defined in the FRP's scenarios (F and G). The distinction between input and output data (L, M, N, and O) from those scenarios is implemented in the test cases. The remaining scenarios involving the patient's legal guardian are also represented in the catalogue.

In brief, we claim the following benefits of the parallel development of FRP and ATP using the SoPaMM metamodel:

1. the earlier the FRP's feasibility is analyzed, the higher is the FRP's quality;
2. the number of errors in an FRP-ATP alignment is likely lower because they are specified together;
3. the combination of FRP and ATP works as a testable reuse unit because scenarios have preconditions and postconditions that can be validated and automatically processed;
4. traceability is visible in earlier development stages;
5. if the reuse of the whole catalogue is not suitable, one may reuse one or more SPBs that meet the project's needs;
6. the reuse of FRP-ATP in software projects should reduce the development time and cost.

6 Quality evaluation

This section details the quality evaluation of the SoPaMM metamodel using the MQuaRE framework described in Sect. 3. First, we describe the evaluation planning and design, including the evaluation purpose, the evaluators' profiles, the evaluation supporting artifacts, the MQuaRE's activities performed by the participants, and the evaluation period. Then, we present and discuss the evaluation results and threats to the validity of this work.

Table 2 Relation between MQuaRE's quality measures and supporting artifacts

ID	Metamodel quality measure (MQM)	Supporting artifact
CCc-1	Conceptual foundation	1
CCc-2	Backward Traceability	1
CCp-1	Conceptual coverage	1, 2
CCr-1	Conceptual correctness	1, 2, 3
CAp-1	Conceptual appropriateness of usage objective	1, 3
CAp-2	Conceptual appropriateness of metamodel	1, 3
UAp-1	Description completeness	1, 3
UAp-2	Demonstration coverage	1, 3
UAp-3	Evident concepts	1, 2
UAp-4	Concept understandability	1, 2
ULe-1	User guide completeness	1, 3
MMo-1	Coupling of concepts	2
MMo-2	Complexity of exercise	2
MRe-1	Reusability per application domain	1, 3, 5
MMd-1	Conceptual stability	1, 4
MMd-2	Change recordability	1, 4
MMd-3	Change impact	1, 4
MMd-4	Modification impact localization	1, 4
MMd-5	Modification correctness	1, 4
PAd-1	Adaptability per application domain	3, 5
PRe-1	Usage similarity	3, 6
PRe-2	Metamodel quality equivalence	1, 2, 6
PRe-3	Conceptual inclusiveness	1, 2

6.1 Evaluation planning and design

As evaluation requesters, the SoPaMM's developers performed the first three activities of the MQuaRE's evaluation process described in Sect. 3.2. As a result, the evaluation requesters elaborated on a detailed metamodel quality evaluation plan (MQEP) containing the following information:

Evaluation purpose: the main goal is *to estimate the SoPaMM's final quality* regarding the MQuaRE's quality model.

Evaluation specification: according to the evaluation purpose, the MQuaRE quality model shall support SoPaMM's evaluation, including 11 quality sub-characteristics

associated with 23 quality measures (MQM) and 19 quality requirements (MQR), as in Fig. 1. Besides, one defined the SoPaMM's quality grades using the arithmetic mean of both its MQMs and sub-characteristics. Consider the Usability characteristic in Fig. 1, comprising the appropriateness recognizability and the learnability sub-characteristics. The SoPaMM's usability grade shall be the arithmetic mean between these sub-characteristics. In turn, the former's grade is given by the arithmetic mean of its MQMs: description completeness, demonstration coverage, evident concepts, and concept understandability. Finally, the SoPaMM's learnability grade is given by the measurement value of its only MQM: user guide completeness. Target and acceptable tolerance values were also established for each MQM. For instance, target and tolerance values were set to 1 and 0.75, respectively, for all MQMs whose values closer to 1 are the better.

Evaluation design: the evaluation requesters attached to the MQEP a set of metamodel artifacts and an association table between these artifacts and each MQM (including its ID[2]) to fully support evaluators' tasks (see Table 2). A brief description of each supporting artifact is presented next.

1. SoPaMM's requirements and design specification: a comprehensive guide about metamodel concepts, semantics, and modeling decisions under the analysis and design viewpoints;
2. SoPaMM implementation in an Ecore[3] format file;
3. SoPaMM's user documentation: a detailed description of SoPaMM's use cases to manage software pattern catalogues;
4. SoPaMM's version history: this document describes the commonalities and differences between the three existing versions of SoPaMM;
5. SoPaMM-based patterns catalogues: one catalogue supports the certification of electronic health record systems [2, 4]; another represents behavior-driven requirements of IoT systems, and two catalogues separately describe general functionalities and behaviors for user authentication and registration. The SoPaMM's developers built all these catalogues, which are useful for metamodel reusability and adaptability;
6. The PABRE metamodel specification: the SoPaMM's most similar metamodel but focused on software requirement patterns. This artifact helps evaluate the replacement of SoPaMM by another metamodel with the same purpose in the same application domain.

Six participants with different expertise evaluated SoPaMM. Referring to the evaluators as *E1* to *E6*, their profiles are as follows:

- *E1* has ten or more years of expertise in software quality;
- *E2* and *E3* own ten or more years of software requirements expertise, being three in requirement patterns;
- *E4* holds ten or more years of expertise in software quality, software requirements, and software metamodeling;
- *E5* and *E6* own practical experience in software engineering in general.

[2] ID consists of an abbreviated alphabetic code with the initial letter in uppercase of the quality characteristic followed by two letters representing the sub-characteristic and an ordinal number of the sequential order within a quality sub-characteristic. For instance, the UAp-2 represents the second measure of appropriateness recognizability (Ap), which is sub-characteristic of Usability (U).

[3] Ecore is the core metamodel of the *Eclipse Modeling Framework* and describes models and runtime support for them. Available at https://wiki.eclipse.org/Ecore.

Table 3 SoPaMM's quality evaluation results

Characteristic	Grade	Sub-characteristic	Grade	MQM	Value
Compliance	0.95	Conceptual compliance	0.95	CCc-1	1.00
				CCc-2	0.90
Conceptual suitability	1.00	Conceptual completeness	1.00	CCp-1	1.00
		Conceptual correctness	0.99	CCr-1	0.99
		Conceptual appropriateness	1.00	CAp-1	1.00
				CAp-2	1.00
Usability	0.99	Appropriateness recognizability	0.99	UAp-1	1.00
				UAp-2	0.98
				UAp-3	1.00
				UAp-4	0.98
		Learnability	0.98	ULe-1	0.98
Maintainability	0.91	Modularity	0.94	MMo-1	1.00
				MMo-2	0.88
		Reusability	0.96	MRe-1	0.96
		Modifiability	0.84	MMd-1	0.71
				MMd-2	0.98
				MMd-3	0.93
				MMd-4	0.77
				MMd-5	0.80
Portability	0.86	Adaptability	1.00	PAd-1	1.00
		Replaceability	0.72	PRe-1	1.00
				PRe-2	0.78
				PRe-3	0.37

The execution of the SoPaMM evaluation was carried out from *September 23 to October 15, 2020*. Due to the COVID-19 pandemic, the evaluators were further assisted by two explanatory videos: one video overviews MQuaRE, whereas the other describes how to calculate a particular MQM and consequently the respective quality sub-characteristic and characteristic.

Thus, the SoPaMM's evaluation kit included a detailed textual evaluation plan enriched with evaluation supporting artifacts and two tutorial videos. Also, the evaluators received a template report document for registering the evaluation results.

After completing the metamodel evaluation, the participants also filled in a questionnaire that assessed their perception of MQuaRE and SoPaMM. As MQuaRE details are out of this work's scope, we emphasize here only the questions about SoPaMM. Three questions in a five-point Likert scale asked how much the SoPaMM metamodel could help write requirement patterns and test patterns and generate high-quality requirement specifications and test specifications.

6.2 Evaluation results

Table 3 summarizes SoPaMM's evaluation results. The more the quality characteristics' and sub-characteristics' grades are closer to 1, the better.

In general, the SoPaMM metamodel was well-judged regarding its quality characteristics and sub-characteristics. Except for the Portability category, characteristics' and sub-characteristics' grades were higher than 0.8.

Concerning the Conceptual compliance sub-characteristic, the evaluators concluded that all SoPaMM's foundations are easily identified through the *conceptual foundation* measure (CCc-1): OMG's MOF (Metamodel Object Facility) and SPMS (Structured Patterns Metamodel Standard), and the BDD (Behavior-Driven Development) methodology. The participants also traced each metamodel concept back to its conceptual foundation (CCc-2), and the result was very satisfactory again (0.9). A caveat should be made regarding the variation of the measurement value of CCc-2. The evaluator *E3* was the only one that assigned a lower grade (0.69) because in his opinion, the SoPaMM's specification causes misunderstandings about what a metamodel concept is.

The evaluators assigned high scores for measures concerning conceptual completeness, correctness and appropriateness (1.0, 0.99, and 1.0, respectively). These scores indicate that SoPaMM's supporting documentation models all of SoPaMM's requirements, and less than 1% of the concepts modeled presents modeling mistakes (revealed by the evaluator *E3*). Despite that, all SoPaMM's concepts, as described in the supporting documentation, allow achieving specific

usage objectives defined by the evaluation requesters (e.g., creating a software pattern catalogue).

Regarding the Usability measures, evaluators agreed that users might easily recognize that SoPaMM is appropriate for their needs (UAp-1 to 4). Besides, they also concluded that SoPaMM might be quickly learned for a given context of use (ULe-1). The corresponding measurement values showed a very low variation among the six evaluators, similar to the Conceptual suitability measures.

From the Maintainability viewpoint, the participants judged that changes on SoPaMM's concepts have minimal impact on other concepts (MMo-1 and 2). Further, they concluded that SoPaMM's usage scenarios, present in its specifications, can be reused in multiple application domains (MRe-1). However, evaluations deferred regarding the degree to which SoPaMM can be effectively and efficiently modified without introducing inconsistencies or degrading its quality (MMd-1 to 5). In particular, the *conceptual stability* measure (MMd-1) is the only one whose value (0.71) is less than the tolerance value (0.75).

Finally, Portability reached the lowest grade (0.86), specifically influenced by the *conceptual inclusiveness* measure (PRe-3 = 0.37). In MQuaRE, this measure partially analyzes metamodel's replaceability. Both this measure and metamodel quality equivalence (PRe-2) presented a high contrast among evaluators' judgments. On the other hand, the participants concluded that SoPaMM is flexible enough to be adapted in multiple application domains (PAd-1). Furthermore, SoPaMM is fully capable of replacing an equivalent metamodel for the same purpose in the same application domain (PRe-1).

Regarding the three questionnaire items about SoPaMM, the evaluators were unanimous that it certainly helps specify requirement and test patterns and the production of requirement and test specifications. Observe that this result is solely based on the participants' experience with the SoPaMM's evaluation process. We are also aware that the small number of evaluators does not convey statistical significance.

6.3 Discussion

Evaluation results suggest that the SoPaMM has a good quality regarding Compliance, Conceptual suitability, Usability, Maintainability, and Portability. Evaluators' comments reported positive and negative aspects about SoPaMM and MQuaRE.

The evaluators *E1*, *E2*, and *E5* concluded that "*SoPaMM satisfactorily meets quality requirements*." *E3* assigned a lower grade to the Compliance measures because in his opinion, "*the SoPaMM documentation is not clear regarding what a metamodel concept is*." Moreover, *E3* and *E4* suggested that a software tool would facilitate metamodel's evaluation using MQuaRE. They agreed that managing multiple documents without tool support is cumbersome (e.g., evaluation plan, metamodel specifications, pattern catalogues, and evaluation report).

Analyzing evaluators' observations, we believe that the evaluation support artifacts (e.g., requirements and design specification) and the explanation of how to calculate each MQM contributed positively to SoPaMM's performance regarding Compliance, Conceptual suitability, and Usability. The variation between the respective values of measures was very low (zero, in most cases), even though there is no statistical evidence. *E3* stated that "*usability measures were the easiest to calculate*."

Some Maintainability measures had that same variation pattern, as did one Portability measure (MMo-1, MMo-2, MRe-1, and PAd-1, in this order). In particular, the scores of MRe-1 and PAd-1 (0.96 and 1.0, respectively) demonstrate that the alignment between FRP and ATP described in SoPaMM-based pattern catalogues can be easily reused and adapted for different application domains.

Still concerning Maintainability, *E3* and *E4*, however, reported that Modifiability measures are challenging to understand and calculate for those who are not the metamodel developer (MMd-1 to 5). *E4* did not feel comfortable computing the MMd-3, MMd-4, and MMd-5 measures, so he left them blank. Although *E3* has assessed the SoPaMM's modifiability, he reported not feel confident about it, particularly regarding MMd-4 and MMd-5.

As evaluation results shown in Table 3, all the participants agreed that Portability is troublesome to measure, particularly PRe-2 and PRe-3. The evaluator *E4* reported that Portability is not relevant to metamodels. In his opinion, "*metamodels are considered Domain-Specific Languages (DSL), i.e., they are inherently domain-specific. Hence, the proposition of a metric that measures if the users must recognize whether a metamodel contains concepts whose purpose is understood correctly without prior training is questionable. If the concepts hold by a metamodel are domain-specific, only users related to that domain will probably understand the concepts with no training*."

Moreover, *E4* advised not to use *Replaceability* as an essential criterion. According to him, "*if one metamodel already exists, it makes sense to adapt it, but replace it with a new one sounds not productive*." *E4* also reinforced that the MQuaRE's quality evaluation model is comprehensive about a metamodel's characteristics. However, he is "*not confident that every single metamodel should exhibit all these quality properties*." For this work, *E4*'s opinion is entirely relevant because of his ten-year metamodeling expertise.

In brief, we consider applying an evaluation framework to measure metamodel's Compliance, Conceptual suitability, among other quality characteristics, is not trivial. Usually, there is an ecosystem of organizations involving standardization, certification, and evaluation in which the certifying

organization provides training courses on quality models, for instance. Conversely, SoPaMM's evaluation was the first MQuaRE use case. Also, tutorial videos were the only training support the evaluators had. For these reasons, we believe that MQuaRE may have negatively influenced the results, specifically regarding Maintainability and Portability. However, we reinforce that SoPaMM's quality properties' grades were higher than 0.85, and they could be better if a comprehensive training course preceded the evaluation.

6.4 Threats to validity

The validity of experiment results depends on experiment settings, and it can be of four types [47]: internal, external, construction, and conclusion. We discuss the threats to validity managed and mitigated, as follows.

Conclusion validity refers to the statistical relation between the initial data and the outcomes. The number of participants might have negatively affected the SoPaMM's quality analysis. However, to minimize this threat, we selected evaluators with multiple specialties varying from general to specific software engineering knowledge, such as requirements engineering, software quality, and metamodeling. Also, most MQMs values' low variation conveys more reliability to conclusions (except for the Modifiability and Replaceability measures). Furthermore, the evaluation process's implementation was as standard as possible; all subjects received the same treatment (e.g., the evaluation kit) and could be helped if demanded. Only the participants *E5* and *E6* requested further support with minimal intervention of the evaluation requesters.

Concerning internal validity, it refers to factors affecting the outcomes, not being independent variables. The decision for not using a control group was counterbalanced with the group heterogeneity. Besides, the evaluators *E5* and *E6* experienced difficulties in understanding MQuaRE. They reported frequent access to the complete MQuaRE documentation to obtain further details, mostly about interpreting some MQMs (e.g., Portability-related) and the PABRE metamodel. Despite tailoring measures borrowed from ISO/IEC standards for metamodel quality purposes, this missing information in the evaluation plan may have hampered the SoPaMM's portability results. Furthermore, impacted mainly by the COVID-19 outbreak, the participants did not receive extensive training but only the evaluation plan, supporting artifacts, tutorial videos, and an evaluation report.

Construct validity indicates the extent to which measures accurately reflect the theoretical concepts intended to measure. From the need for a comprehensive metamodel quality evaluation framework, MQuaRE arose after the SoPaMM proposal. Therefore, we understand that the metamodel quality perspective of SoPaMM's creators may have influenced the definition of both the MQuaRE's quality model and measures. However, to mitigate a likely bias, MQuaRE compiles related work on metamodel quality [27, 28, 38, 43] and international standards for software quality, such as ISO/IEC 25010 [24] and ISO/IEC 25023 [25].

External validity concerns the generalization of research findings outside the experiment setting. Once again, we selected a heterogeneous group as a representative population. On the other hand, we know both novice evaluators in MQuaRE and supporting tutorial videos do not represent the industrial practice that usually includes highly trained evaluators.

7 Conclusions and future work

Influenced by well-accepted agile practices and international standards for metamodeling, the Software Pattern MetaModel (SoPaMM) provides a general structure for software pattern specification. The novelty is that SoPaMM links requirements to testing through requirement patterns and test patterns as a reuse approach of higher-quality software artifacts produced in these phases. Furthermore, given that metamodels' quality may affect the software specifications' quality, we also estimate multiple quality facets of SoPaMM through an evaluation framework called Metamodel Quality Requirements and Evaluation (MQuaRE).

The following are lessons learned with the SoPaMM's development and quality evaluation:

1. Our requirements and testing alignment approach through patterns provide reusable, testable, and traceable software artifacts, which should reduce a project's development time and cost.
2. Most of SoPaMM's quality properties were well-judged, namely conceptual compliance, completeness, correctness, appropriateness, and learnability, appropriateness recognizability, reusability, and adaptability.
3. Although the quality in use evaluation was not performed, the participants experienced and approved the alignment of requirements and testing through pattern catalogues built upon SoPaMM. High scores of SoPaMM's reusability and adaptability suggest evaluators' approval.
4. The evaluation kit might bring additional details about quality measures and the PABRE metamodel to support evaluators' tasks thoroughly. The MQuaRE documentation should be revised as well for the same purpose.
5. Maybe not all metamodels should exhibit all quality properties present in MQuaRE, as noted by the expert subject.
6. MQuaRE-aware tool support would undoubtedly be helpful.

As future work, we plan to enhance SoPaMM's capabilities by bridging nonfunctional requirement patterns (NFRP) to test patterns (TP). New types of NFRP and TP can be created and aligned, e.g., performance NFRP and effort TP or reliability NFRP and fault recovery TP. Despite being widely investigated in the literature [1, 9, 41, 48], NFRP is often restricted to requirements engineering and not other software life cycle phases as we have been studied.

Besides, we aim to extend our TMEd tool with new functionalities, such as creating a public repository of SoPaMM-based pattern catalogues and manual search for patterns across catalogues. The goal is further widespread our proposal of behavior-driven functional requirement patterns. In the long term, TMEd will also empower professionals with software patterns mining features, including automatic discovery and recommendation.

The catalogues generated by TMEd are input for another tool we have been working on, called behavior-DRivEn Application Model generator (DREAM). From a SoPaMM-based pattern catalogue, DREAM allows the automatic generation of requirements and test case specifications with traceability support. This initiative will enable us to demonstrate the benefits of using requirement patterns aligned to test patterns in the software industry projects. We are currently working on validating the pattern catalogue for Brazilian electronic health record systems certification with experts. All these efforts have origins from a research agenda on requirement patterns we published elsewhere [6].

Finally, we learned that a software tool could better assist MQuaRE users' tasks. A wizard would guide evaluation requesters toward a more effective metamodel evaluation plan. Similarly, it would also instruct evaluators on which metamodel artifacts are applicable, how to compute each measure, and the evaluation report's generation. This metamodel evaluation supporting tool will be under development soon.

Acknowledgements The work described in this manuscript has not been published before. It is not also under consideration for publication anywhere else. Besides, its publication has been approved by all co-authors.

Author Contributions Taciana Novo Kudo, Renato de Freitas Bulcão-Neto were involved in conceptualization and writing—original draft preparation. Taciana Novo Kudo, Renato de Freitas Bulcão-Neto, Valdemar Vicente Graciano Neto contributed to methodology. All authors were involved in results analysis and threats to validity. Renato de Freitas Bulcão-Neto, Auri Marcelo Rizzo Vincenzi contributed to writing—review and editing. Taciana Novo Kudo was involved in funding acquisition. Auri Marcelo Rizzo Vincenzi contributed to supervision.

Funding Partial financial support was received from the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior−Brazil (CAPES)−Finance Code 001.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Amorndettawin M, Senivongse T (2019) Non-functional requirement patterns for agile software development. In: Proceedings of the 2019 3rd International Conference on Software and E-Business, ICSEB 2019, pp 66–74. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3374549.3374561
2. Taciana N. Kudo, Renato F. Bulcão-Neto, Auri M.R. Vincenzi (2019) A Conceptual Metamodel to Bridging Requirement Patterns to Test Patterns. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering, SBES 2019, pp. 155–160. ACM, Salvador, Brazil. https://doi.org/10.1145/3350768.3351300
3. Taciana N. Kudo, Renato F. Bulcão-Neto, Auri M.R. Vincenzi (2020) Toward a Metamodel Quality Evaluation Framework: Requirements, Model, Measures, and Process. In: Proceedings of the XXXIV Brazilian Symposium on Software Engineering, SBES 2020, pp. 102–107. ACM, Natal,Brazil. https://doi.org/10.1145/3422392.3422461
4. Taciana N. Kudo, Renato F. Bulcão-Neto, Auri M.R. Vincenzi (2020) Uma Ferramenta para Construção de Catálogos de Padrões de Requisitos com Comportamento. In: Workshop em Engenharia de Requisitos, WER 2020, pp. 1–14. Editora PUC-Rio, São José dos Campos, Brazil. http://wer.inf.pucrio.br/WERpapers/artigos/artigos_WER20/12_WER_2020_paper_16.pdf
5. Taciana N. Kudo, Renato F. Bulcão-Neto, Auri M.R. Vincenzi (2020) Metamodel Quality Requirements and Evaluation (MQuaRE). CoRR abs/2008.09459. arxiv.org/abs/2008.09459
6. Taciana N. Kudo, Renato F. Bulcão-Neto, Auri M.R. Vincenzi (2020) Requirement patterns: a tertiary study and a research agenda. IET Softw 14(1):18–26. https://doi.org/10.1049/iet-sen.2019.0016
7. Taciana N. Kudo, Renato F. Bulcão-Neto, Auri M.R. Vincenzi, Alessandra A. Macedo (2019) A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle. J. Softw. Eng. Res. Dev. 7:9. https://doi.org/10.5753/jserd.2019.458
8. Baudry B, Nebut C, Traon YL (2007) Model-driven engineering for requirements analysis. In: 11th IEEE international enterprise distributed object computing conference (EDOC 2007), p 459
9. Beckers K, Côté I, Goeke L (2014) A catalog of security requirements patterns for the domain of cloud computing systems. In: Proceedings of the ACM symposium on applied computing, pp 337–342. ACM, Gyeongju, Republic of Korea
10. Bjarnason E, Borg M (2017) Aligning requirements and testing: working together toward the same goal. IEEE Softw 34(1):20–23. https://doi.org/10.1109/MS.2017.14
11. Chelimsky D, Astels D, Helmkamp B, North D, Dennis Z, Hellesoy A (2010) The RSpec book: behaviour driven development with Rspec, cucumber, and friends, 1st edn. Pragmatic Bookshelf, Raleigh, NC
12. Cheng BHC, Atlee JM (2009) Current and future research directions in requirements engineering. In: Lyytinen K, Loucopoulos P, Mylopoulos J, Robinson B (eds) Design requirements engineering: a ten-year perspective. Springer, Berlin, pp 11–43
13. Chernak Y (2012) Requirements reuse: the state of the practice. In: 2012 IEEE international conference on software science. Technology and Engineering, SWSTE 2012, Herzlia, Israel, June

12–13, 2012. IEEE Computer Society, Los Alamitos, CA, USA, pp 46–53
14. Costal D, Franch X, López L, Palomares C, Quer C (2019) On the use of requirement patterns to analyse request for proposal documents. In: Laender AHF, Pernici B, Lim E, de Oliveira JPM (eds) conceptual modeling—38th international conference, ER 2019, Salvador, Brazil, November 4–7, 2019, Proceedings. *Lecture Notes in Computer Science*, vol. 11788, pp. 549–557. Springer . https://doi.org/10.1007/978-3-030-33223-5_45
15. Ebert C, Ray R (2021) Test-driven requirements engineering. IEEE Softw 38(1):16–24. https://doi.org/10.1109/MS.2020.3029811
16. Franch X (2015) Software requirements patterns: a state of the art and the practice. In: Proceedings of the 37th international conference on software engineering, vol 2, ICSE'15, pp 943–944. IEEE Press, Piscataway
17. Franch X, Palomares C, Quer C (2020) Industrial practices on requirements reuse: an interview-based study. In: Madhavji NH, Pasquale L, Ferrari A, Gnesi S (eds) Requirements engineering: foundation for software quality—26th international working conference, REFSQ 2020, Pisa, Italy, March 24–27, 2020, Proceedings [REFSQ 2020 was postponed], *Lecture Notes in Computer Science*, vol 12045, pp 78–94. Springer. https://doi.org/10.1007/978-3-030-44429-7_6
18. Franch X, Palomares C, Quer C, Renault S, De Lazzer F (2010) A metamodel for software requirement patterns. In: Wieringa R, Persson A (eds) Requirements engineering: foundation for software quality. Springer, Berlin, pp 85–90
19. Fricker S, Grau R, Zwingli A (2015) Requirements engineering: best practice, pp 25–46. Springer, Berlin
20. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman, Boston
21. Haskins C (2003) Using patterns to share best results—a proposal to codify the Sebok. INCOSE Int Symp 13(1):15–23
22. Irshad M, Petersen K, Poulding S (2018) A systematic literature review of software requirements reuse approaches. Inf Softw Technol 93(C):223–245
23. ISO/IEC: ISO/IEC 9126-1:2001 Software engineering—product quality—Part 1: quality model. ISO/IEC 9126-1:2001 1:1–25 (2001)
24. ISO/IEC: ISO/IEC 25000:2014 Systems and software engineering—systems and software Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE. ISO/IEC 25000:2014 2:1–27 (2014)
25. ISO/IEC: ISO/IEC 25023:2016 systems and software engineering—systems and software quality requirements and evaluation (SQuaRE)—measurement of system and software product quality. ISO/IEC 25023:2016 1:1–45 (2016)
26. Leotta M, Clerissi D, Ricca F, Spadaro C (2013) Improving test suites maintainability with the page object pattern: an industrial case study. In: ICST workshops, pp 108–113. IEEE Computer Society, Washington, DC
27. Ma H, Shao W, Zhang L, Ma Z, Jiang Y (2004) Applying OO metrics to assess UML meta-models. In: Baar T, Strohmeier A, Moreira A, Mellor SJ (eds) UML 2004—the unified modeling language. Modeling languages and applications, pp 12–26. Springer, Berlin
28. Ma Z, He X, Liu C (2013) Assessing the quality of metamodels. Front Comput Sci 7(4):558
29. Macasaet RJ, Noguera M, Rodríguez ML, Garrido JL, Supakkul S, Chung L (2019) Micro-business requirements patterns in practice: remote communities in developing nations. J Univ Comput Sci 25(7):764–787. http://www.jucs.org/jucs_25_7/micro_business_requirements_patterns
30. Meszaros G (2006) XUnit test patterns: refactoring test code. Prentice Hall, Upper Saddle River
31. Moreira RMLM, Paiva ACR (2014) A GUI modeling DSL for pattern-based GUI testing—PARADIGM. In: ENASE 2014—Proceedings of the 9th international conference on evaluation of novel approaches to software engineering, Lisbon, Portugal, 28–30 April, 2014, pp 126–135. IEEE, Lisbon, Portugal
32. Oliveira G, Marczak S, Moralles C (2019) How to evaluate BDD scenarios' quality? In: do Carmo Machado I, Souza R, Maciel RSP, Sant'Anna C (eds) Proceedings of the XXXIII Brazilian symposium on software engineering, SBES 2019, Salvador, Brazil, September 23–27, pp 481–490. ACM. https://doi.org/10.1145/3350768.3351301
33. OMG: Meta object facility (mof) specification, version 1.4. Object Management Group, Inc. (2002)
34. OMG: Structured patterns metamodel standard. OMG—Object Management Group (2017)
35. Palomares C, Quer C, Franch X (2011) Pabre-man: management of a requirement patterns catalogue. In: RE 2011, 19th IEEE international requirements engineering conference, Trento, Italy, August 29 2011–September 2, 2011, pp 341–342. IEEE Computer Society. https://doi.org/10.1109/RE.2011.6051666
36. Palomares C, Quer C, Franch X, Renault S, Guerlain C (2013) A catalogue of functional software requirement patterns for the domain of content management systems. In: Proceedings of the 28th annual ACM symposium on applied computing, SAC '13, pp 1260–1265. ACM, New York
37. Rising L (1999) Patterns: a way to reuse expertise. IEEE Commun Mag 37(4):34–36
38. Rocco J, Di Ruscio D, Iovino L, Pierantonio A (2014) Mining metrics for understanding metamodel characteristics. In: Proceedings of the 6th international workshop on modeling in software engineering (MiSE 2014), pp 55–60. ACM, New York
39. Rook P (1986) Controlling software projects. Softw Eng J 1:7
40. Smart JF (2014) BDD in action: behavior-driven development for the whole software lifecycle, 1st edn. Manning Publications
41. de Souza Cunha H, do Prado Leite JCS, Duboc L, Werneck V (2013) The challenges of representing transparency as patterns. In: Third IEEE International Workshop on Requirements Patterns, RePa 2013, Rio de Janeiro, Brazil, July 16, 2013, pp 25–30. IEEE Computer Society . https://doi.org/10.1109/RePa.2013.6602668
42. Stocco A, Leotta M, Ricca F, Tonella P (2015) Why creating web page objects manually if it can be done automatically? In: Proceedings of the 10th international workshop on automation of software test, AST '15, pp 70–74. IEEE Press, Piscataway
43. Strahonja V (2007) The evaluation criteria of workflow metamodels. In: 29th international conference on information technology interfaces, pp 553–558. IEEE, New York
44. Tockey S (2015) Insanity, hiring, and the software industry. Computer 48:96–101
45. Videira C, da Silva AR (2005) Patterns and metamodel for a natural-language-based requirements specification language. In: Belo O, Eder J, Cunha JF, Pastor O (eds) The 17th conference on advanced information systems engineering (CAiSE '05), Porto, Portugal, 13–17 June, 2005, CAiSE Forum, Short Paper Proceedings, CEUR Workshop Proceedings, vol 161. CEUR-WS.org. http://ceur-ws.org/Vol-161/FORUM_31.pdf
46. Withall S (2007) Software requirement patterns. Best practices. Microsoft Press, Redmond
47. Wohlin C, Runeson P, Hst M, Ohlsson MC, Regnell B, Wessln A (2012) Experimentation in software engineering. Springer, Berlin
48. Xuan X, Wang Y, Li S (2014) Privacy requirements patterns for mobile operating systems. In: Zhao L, do Prado Leite JCS, Supakkul S, Chung L, Wang Y (eds) 4th IEEE international workshop on requirements patterns, RePa 2014, Karlskrona, Sweden,

August 26, 2014, pp. 39–42. IEEE Computer Society. https://doi.org/10.1109/RePa.2014.6894842
49. Ya'u B, Nordin A, Salleh N (2016) Software requirements patterns and meta model: a strategy for enhancing requirements reuse (rr). In: 2016 6th international conference on information and communication technology for the muslim world, pp 188–193. ICT4M, Jakarta, Indonesia