UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM METAMODELO PARA ALINHAMENTO DE PADRÕES DE REQUISITOS E PADRÕES DE TESTES E UM FRAMEWORK PARA AVALIAÇÃO DE METAMODELOS

TACIANA NOVO KUDO

ORIENTADOR: PROF. DR. AURI MARCELO RIZZO VINCENZI

São Carlos – SP
Janeiro/2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM METAMODELO PARA ALINHAMENTO DE PADRÕES DE REQUISITOS E PADRÕES DE TESTES E UM FRAMEWORK PARA AVALIAÇÃO DE METAMODELOS

TACIANA NOVO KUDO

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação, área de concentração: Engenharia de software
Orientador: Prof. Dr. Auri Marcelo Rizzo Vincenzi

São Carlos – SP
Janeiro/2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Tese de Doutorado da candidata Taciana Novo Kudo, realizada em 22/01/2021.

Comissão Julgadora:

Prof. Dr. Auri Marcelo Rizzo Vincenzi (UFSCar)

Prof. Dr. Daniel Lucrédio (UFSCar)

Profa. Dra. Ana Cristina Ramada Paiva (U.Porto)

Prof. Dr. Katia Romero Felizardo Scannavino (UTFPR)

Prof. Dr. Sergio Teixeira de Carvalho (UFG)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.
O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Dedico ao meu esposo pelo companheirismo constante e pela admiração mesmo quando não me era possível acreditar. Aos meus filhos pela paciência e pelo amor incondicional. Aos meus pais e a minha irmã por serem meu porto seguro.

AGRADECIMENTO

Agradeço a uma Força Maior que me impulsiona sempre para as decisões mais acertadas na vida e, que durante essa jornada de doutoramento, tem-se mostrado mais presente ainda. Agradeço a cada dúvida e incerteza, pois a partir delas a pesquisa me proporcionou novos conhecimentos científicos, empíricos e filosóficos. Em especial, agradeço ao meu esposo que ficou ao meu lado durante essa trajetória e sempre me incentivou, com palavras e ações. Aos meus filhos, agradeço pela paciência e preocupação. Aos meus pais e à minha irmã, agradeço por constantemente dizerem que eu conseguiria. Ao meu orientador, em particular, agradeço pela confiança que depositou em mim, obrigada por acreditar incondicionalmente. Por fim, agradeço a todas as pessoas que cruzaram o meu caminho durante esse período, pois fazem parte direta, ou indiretamente, desse capítulo da narrativa da minha vida!

The human eye has a readiness for patterns. Much is not seen simply because the mind is blind, not eyes. The eyes see in lines, curves, and patterns. Man himself works in patterns simple or complex, and such things are often evidence of mans previous presence.

Louis L'Amour, 1972

RESUMO

Um padrão de requisito de software (PRS) é uma abordagem de reutilização viável que reúne requisitos de software recorrentes e de alta qualidade de um conjunto de aplicativos. PRS é um tema amplamente investigado porque melhora a qualidade das especificações dos requisitos e reduz o tempo de entrega e o custo do projeto. Apesar de sua importância para a Engenharia de Requisitos (ER), há uma carência de pesquisas sobre PRS nas demais fases do ciclo de vida do software. Considerando a relação intrínseca entre RE e testes, este trabalho tem como objetivo elaborar uma estratégia de reutilização abstrata e independente de domínio para o alinhamento de PRS e Padrões de Teste de Software (PTS). Um metamodelo denominado *Software Pattern MetaModel* (SoPaMM) foi produzido para que requisitos, comportamentos e casos de teste sejam relacionados, com a influência de práticas ágeis existentes como *Behavior-Driven Development* (BDD). Uma ferramenta *Terminal Model Editor* (TMEd) também foi desenvolvida para produzir catálogos de padrões seguindo a gramática do metamodelo SoPaMM. Além disso, o arcabouço *Metamodel Quality Requirements and Evaluation* (MQuaRE) foi definido para avaliar a qualidade do metamodelo SoPaMM. Em seguida, o SoPaMM foi avaliado sob a perspectiva das propriedades de qualidade definidas no MQuaRE, e os resultados da avaliação indicaram que o metamodelo apresenta boa qualidade quanto às características de Conformidade, Adequação Conceitual, Usabilidade, Manutenção e Portabilidade. As principais contribuições desta pesquisa são: (i) uma agenda de pesquisa sobre o estado da arte e o estado da prática do PRS; (ii) a identificação de carência de pesquisas envolvendo PRS em outras fases do ciclo de vida do software, além de ER; (iii) o metamodelo SoPaMM; (iv) a ferramenta TMEd; (v) o arcabouço MQuaRE; e (vi) a avaliação da qualidade do metamodelo SoPaMM usando a estrutura MQuaRE. Lições aprendidas e propostas de trabalhos futuros concluem esta tese.

Palavras-chave: Padrão de requisito de software; Padrão de teste de software; Metamodelagem; BDD; Catálogo; Avaliação de qualidade.

ABSTRACT

A *Software Requirement Pattern* (SRP) is a feasible reuse approach that joins recurrent and high-quality software requirements from a set of applications. SRP is a widely investigated theme because it improves the quality of requirements specifications and reduces delivery time and project cost. Despite its importance for Requirements Engineering (RE), there is a lack of research on SRP over other software life cycle phases. Considering the intrinsic relation between RE and testing, this work aims to elaborate on an abstract and domain-independent reuse strategy for aligning SRP and *Software Test Pattern* (STP). A metamodel called *Software Pattern MetaModel* (SoPaMM) was produced so that requirements, behaviors, and test cases are related, with the influence of existing agile practices as *Behavior-Driven Development* (BDD). A *Terminal Model Editor* (TMEd) tool was also developed to produce pattern catalogues following the SoPaMM metamodel grammar. Moreover, the *Metamodel Quality Requirements and Evaluation* (MQuaRE) framework was defined to evaluate the SoPaMM metamodel quality. Then, SoPaMM was evaluated from the perspective of the quality properties defined in MQuaRE, and the evaluation results indicated that the metamodel has good quality concerning Compliance, Conceptual Suitability, Usability, Maintenance, and Portability characteristics. The main contributions of this research are: (i) a research agenda on the state of the art and state of the practice of SRP; (ii) the identification of a lack of research involving SRP in other phases of the software life cycle, beyond RE; (iii) the SoPaMM metamodel; (iv) the TMEd tool; (v) the MQuaRE framework; and (vi) the quality evaluation of the SoPaMM metamodel using the MQuaRE framework. Lessons learned and proposals of future work conclude this research.

Keywords: Software requirement pattern; Software test pattern; Metamodeling; BDD; Behavior-Driven Development; Catalogue; Quality evaluation.

LISTA DE PUBLICAÇÕES

A seguir é apresentada a lista de publicações originadas desse trabalho até o momento.

- **Artigo completo em conferência:**
  - KUDO, T. N.; BULCÃO-NETO, R. F.; MACEDO, A. A.; VINCENZI, A. M. R. Padrão de requisitos no ciclo de vida de software: Um mapeamento sistemático. In: Proceedings of the Conference: Congresso Ibero Americano em Engenharia de Software, CIbSE 2019, p. 420–433, Havana, Cuba. 2019. (*Prêmio de 3º melhor artigo da trilha de Engenharia de Requisitos)*.
  - KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. A conceptual metamodel to bridging requirement patterns to test patterns. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering, SBES 2019, p.155-160, Salvador, Brazil. 2019.
  - KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Uma Ferramenta para Construção de Catálogos de Padrões de Requisitos com Comportamento In: Anais do Workshop em Engenharia de Requisitos, WER 2020, p. 1-14, São José dos Campos, SP, Brasil, August. 2020.
  - KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Toward a Metamodel Quality Evaluation Framework: Requirements, Model, Measures, and Process. In: Proceedings of the XXXIV Brazilian Symposium on Software Engineering, SBES 2020, Rio Grande do Norte, Brazil, October. 2020.
- **Artigo completo em periódico:**
  - KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Requirement patterns: A tertiary study and a research agenda. IET Software, v. 14, n. 1, p. 18–26. 2020.
  - KUDO, T. N.; BULCÃO-NETO, R. F.; MACEDO, A. A.; VINCENZI, A. M. R. A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle. Journal of Software Engineering Research and Development, v. 7, p. 9:1-9:11, 2019.
- **Artigo submetido para avaliação:**
  - KUDO, T. N.; BULCÃO-NETO, R. F.; GRACIANO NETO, V. V.; VINCENZI, A. M. R. Aligning requirements and testing through metamodeling and patterns: Design and evaluation. Requirements Engineering v. XX. n. XX, p. 1-20. 2020. (submetido para).

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

BDD	*Behaviour Driven Development*
CVS	*Ciclo de Vida do Software*
EMF	*Eclipse Modeling Framework*
ER	*Engenharia de Requisitos*
MQuaRE	*Metamodel Quality Requirements and Evaluation*
MDE	*Model Driven Development*
MOF	*Meta Object Facility*
OMG	*Object Management Group*
PRS	*Padrão de Requisito de Software*
PRBC	*Padrão de Requisito Baseado em Comportamento*
PTS	*Padrão de Teste de Software*
SoPaMM	*Software Pattern MetaModel*
S-RES	*Sistemas de Registro Eletrônico de Saúde*
TMEd	*Terminal Model Editor*
TS	*Teste de Software*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Neste capítulo é apresentada uma introdução com contextualização sobre o tema, o problema e a hipótese de pesquisa, bem como o objetivo geral e os objetivos específicos do trabalho. Por fim, é explicado o corpo desta tese que é composta por uma série de artigos publicados ao longo do doutoramento.

1.1 Contextualização

A Engenharia de Requisitos (ER) é a primeira fase no Ciclo de Vida do Software (CVS) e de extrema importância, pois as funcionalidades e restrições do software precisam ser identificadas e bem compreendidas. O objetivo da ER é servir de direcionamento para o restante do ciclo de vida do software. Os requisitos são a base de todo o desenvolvimento, pois definem as necessidades das partes interessadas (clientes, analistas, projetistas, desenvolvedores e testadores) no software. (Bourque e Fairley, 2014).

Embora se reconheça a importância da Engenharia de Requisitos no processo de desenvolvimento de software, estudos (Franch, 2015; Tockey, 2015) demonstram que ainda há uma grande porcentagem de projetos de software que extrapolam o orçamento ou os prazos de entrega, ou que precisam ser cancelados. Esses estudos identificam que a principal causa de baixo desempenho é resultado de requisitos incorretos, omitidos, mal interpretados, ou conflitantes.

Uma alternativa eficaz para tratar problemas de qualidade de especificação é a prática de reúso de requisitos (Chernak, 2012; Irshad et al., 2018). O reúso de requisitos é a forma de utilizar, em um novo projeto, requisitos que foram escritos em projetos anteriores (Palomares, 2014). Quando uma empresa gera várias especificações de requisitos ao longo do tempo, uma parte dos requisitos é específica para cada software, mas uma porção significativa dos requisitos se repete (Withall, 2007). Segundo Palomares (2014), o reúso de requisitos é possível porque nem todos os requisitos que

definem um software são específicos somente para ele, e aproveitar o conhecimento adquirido em projetos anteriores é uma estratégia adequada para melhorar a qualidade dos requisitos e a eficiência do processo de ER. Conforme declarado por Wiegers e Beatty (2013), a reutilização eficaz de requisitos confiáveis pode promover consistência dentro e entre os aplicativos, menos defeitos, retrabalho reduzido, maior produtividade da equipe, menores custos de desenvolvimento e entrega mais rápida.

Uma das abordagens para reúso de requisitos é a de Padrão de Requisitos de Software (PRS), uma abstração que agrega comportamentos e serviços comuns a vários sistemas e podem ser reutilizados em software similares (Franch et al., 2020; Withall, 2007). Com o tempo, propostas de PRS, como a definida por (Withall, 2007), foram surgindo e ainda continuam recebendo atenção de pesquisadores (Barros-Justo et al., 2018; Irshad et al., 2018; Palomares et al., 2017).

Propostas de PRS para diversos domínios são encontradas na literatura, por exemplo: sistemas embarcados (Konrad e Cheng, 2002), gerenciamento de conteúdo (Palomares et al., 2013) e computação em nuvem (Beckers et al., 2014). Além disso, PRS tem sido utilizado para melhorar a comunicação entre as equipes de desenvolvimento incluindo detalhes técnicos no seu conteúdo (Macasaet et al., 2019) e auxiliar em processos de licitação de software (Costal et al., 2019). Segundo Palomares et al. (2017) entre os benefícios obtidos ao se utilizar PRS, estão:

- *completude* das especificações dos requisitos;
- *uniformidade* dos requisitos;
- *consistência* dos requisitos;
- *clareza* na especificação dos requisitos.

Uma abordagem promissora para representar PRS é por meio de metamodelagem, pois esta aumenta o nível de abstração em que o software é criado (Baudry et al., 2007). Um metamodelo pode fornecer uma estrutura de representação geral para PRS, visando melhorar a qualidade das especificações e a produtividade dos engenheiros de requisitos (Assar, 2014; Baudry et al., 2007; Loniewski et al., 2010). Franch et al. (2010) propõem um metamodelo que define a estrutura de um PRS, relações entre padrões e critérios de classificação para agrupar padrões. Ya'u et al. (2016) propõem um metamodelo para representar PRS baseado na engenharia de linha de produto de software, com uma estrutura reutilizável que trata variabilidade de modelos e rastreabilidade de artefatos de software.

1.2 Problema

Apesar dos benefícios de PRS para o processo de RE, poucas pesquisas abordam o alinhamento entre PRS e os artefatos produzidos em outras fases de desenvolvimento (Kudo et al., 2019b,c). Em um estudo terciário apresentado por Kudo et al. (2020a) os estudos secundários mapeados demonstram os benefícios do uso de PRS especificamente na fase de ER e não ampliam a pesquisa para o uso de PRS nas demais fases do CVS. Além disso, no estudo secundário apresentado por Kudo et al. (2019c), apenas 10 (dez) estudos propõem o uso do PRS além da etapa de ER, e desses, apenas 1 estudo integra PRS com a fase de testes.

Esta observação se dá em função da relação intrínseca entre as atividades de testes e de requisitos, como é definido no modelo V apresentado na Figura 1.1. Este modelo associa a fase de teste de aceitação à fase de especificação de requisitos e do sistema, a fim de determinar se o software atende a seus requisitos ou descobrir situações em que o software se comporta de maneira diferente das especificações de requisitos (Myers e Sandler, 2004; Rook, 1986). Apesar da importante relação entre as fases de requisitos e de testes, a maioria das empresas ainda luta com os efeitos de custo, retrabalho e atraso resultantes de um fraco alinhamento entre requisito e teste (Bjarnason e Borg, 2017).

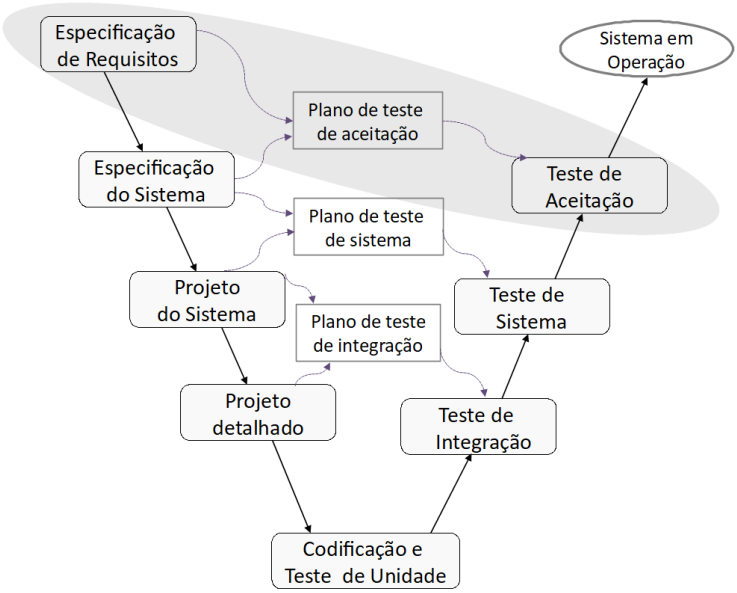

Figura 1.1: Ciclo de vida com etapas sucessivas de teste [Adaptado de (Rook, 1986)]

Ainda no tocante a reuso de especificações de software de alta qualidade, existe a abordagem de Padrão de Teste de Software (PTS), que descreve soluções genéricas para testar comportamentos recorrentes (Moreira e Paiva, 2014). Um PTS tem como

objetivo auxiliar o testador a entender o contexto de uma prática de teste e decidir por padrões de testes alternativos. Boas práticas de testes que se repetem tendem a ser documentadas como um PTS.

A partir dos estudos de Kudo et al. (2019b,c), foi possível identificar uma *carência de pesquisas que apoiam a integração de PRS ao longo de todo CVS*. Ao considerar a ligação natural entre os requisitos e os testes de software (Figura 1.1) e o pequeno número de pesquisas que integram PRS com a fase de testes, observa-se *uma oportunidade de pesquisa em investigar o uso conjunto de PRS com PTS*.

1.3 Hipótese

Este trabalho defende a tese de que é possível fornecer um alinhamento entre requisitos e testes por meio de uma representação abstrata, como metamodelos, independente de domínio e que permite o reúso integrado de PRS e PTS. Desta forma, especificações de requisitos, derivadas desta representação abstrata, teriam comportamentos associados para execução de testes.

1.4 Objetivos

Considerando o exposto acima e, dado que existe uma carência de pesquisas que apoiam o uso de PRS em outras etapas do CVS, o objetivo geral desta tese é definir uma abordagem de reuso que relacione PRS a PTS e, assim, possa estabelecer um forte alinhamento entre atividades de engenharia de requisitos e de testes.

Para atingir o objetivo geral, são objetivos específicos deste trabalho:

1. definir uma representação abstrata e independente de domínio, baseada em metamodelagem, que permita relacionar PRS e PTS;
2. criar catálogos de PRS integrado a PTS para domínios de aplicação específicos;
3. avaliar propriedades de qualidade da representação em metamodelo.

1.5 Organização do Texto

O Capítulo 2 apresenta um estudo exploratório sobre padrões de requisitos. A princípio, foi realizado um estudo terciário sobre padrões de requisitos de software (Kudo et al.,

2020a). O estudo terciário foi realizado de maneira sistemática com busca automática em cinco fontes de dados, além de *snowballing* (Wohlin, 2014). A contribuição deste trabalho foi uma visão geral sobre o estado da arte e da prática em padrões de requisitos e a sugestão de uma agenda de pesquisa nesta área. O artigo completo deste estudo terciário foi publicado no Journal IET Software e encontra-se disponível no Capítulo 2 desta tese.

Os Capítulos 3 e 4 apresentam o estudo sobre a rastreabilidade entre padrões de requisitos e outros artefatos produzidos nas demais fases do ciclo de desenvolvimento de software. Foi realizado um mapeamento sistemático para tentar identificar em quais fases do ciclo de vida de software, além da etapa de requisitos, padrões de requisitos são usados: projeto, construção, teste e/ou manutenção. Neste estudo, identificou-se uma lacuna do uso de padrões de requisitos nas etapas de construção, teste e manutenção. Alguns trabalhos já propõem soluções de padrões de requisitos integrados com a fase de projeto de software. Este mapeamento sistemático foi publicado na *Conferencia Iberoamericana de Software Engineering* - CIbSE'2019 (Kudo et al., 2019b) e ganhou o prêmio de 3o melhor artigo, sendo convidado a enviar uma versão estendida para o *Journal of Software Engineering Research and Development* JSERD (Kudo et al., 2019c). Os dois artigos estão disponíveis nos Capítulos 3 e 4, respectivamente.

O Capítulo 5 apresenta a proposta do metamodelo SoPaMM (Software Pattern MetaModel) para relacionar PRS com PTS, de forma a viabilizar a integração de padrões de requisitos na etapa de testes. Uma vez que a metodologia (*Behavior-Driven Development*) (Chelimsky et al., 2010) é uma alternativa para definir requisitos e permitir a geração de testes, integramos os conceitos de BDD na proposta do metamodelo. Uma primeira versão do metamodelo SoPaMM foi publicada no Simpósio Brasileiro de Engenharia de Software SBES'2019 (Kudo et al., 2019a). Esse artigo com a descrição completa do SoPaMM encontra-se disponível no Capítulo 5.

O capítulo 6 apresenta uma ferramenta *Terminal Model Editor* (TMEd) para apoiar a construção de catálogos de padrões de requisitos usando o metamodelo SoPaMM como modelo de referência. Um artigo descrevendo essa ferramenta, juntamente com um estudo de caso mostrando a utilização da mesma, foi publicado no Workshop de Engenharia de Requisitos WER'2019 (Kudo et al., 2020b). Esse artigo encontra-se disponível no Capítulo 6.

O Capítulo 7 apresenta a proposta de um framework para avaliação de qualidade de metamodelos denominado MQuaRE (*Metamodel Quality Requirements and Evaluation*). O MQuaRE compreende um modelo com cinco características de qualidade para metamodelos, 19 requisitos e 23 medidas de qualidade para metamodelos, e um processo de avaliação composto por cinco atividades que guiam a avaliação de qualidade de um metamodelo de software. O MQuaRE foi proposto com influência na série ISO/IEC

25000. Uma descrição detalhada do MQuaRE juntamente com um estudo de caso avaliando o SoPaMM foi publicado no Simpósio Brasileiro de Engenharia de Software SBES'2020 (Kudo et al., 2020d). Esse artigo encontra-se disponível no Capítulo 7.

O Capítulo 8 apresenta uma avaliação de qualidade completa do metamodelo SoPaMM, com todos os requisitos e medidas de qualidade oferecidas pelo framework de avaliação de metamodelos MQuaRE. Participaram dessa avaliação, seis especialistas em engenharia de software, qualidade de software, engenharia de requisitos e/ou metamodelagem. Os resultados dessa avaliação foram submetidos para o *Requirements Engineering Journal* e está sob avaliação. A versão submetida desse artigo com todo o detalhamento da avaliação encontra-se disponível no Capítulo 8.

Por fim, as conclusões, contribuições, lições aprendidas e propostas de trabalhos futuros são apresentadas no Capítulo 9.

Capítulo 2

UM ESTUDO TERCIÁRIO DA LITERATURA SOBRE PADRÕES DE REQUISITOS

Artigo publicado no Journal IET Software.

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Requirement patterns: A tertiary study and a research agenda. IET Software, v. 14, n. 1, p. 18–26. 2020.

IET Software

Review Article

IET Journals
The Institution of Engineering and Technology

Requirement patterns: a tertiary study and a research agenda

ISSN 1751-8806
Received on 18th January 2019
Revised 5th July 2019
Accepted on 9th September 2019
doi: 10.1049/iet-sen.2019.0016
www.ietdl.org

Taciana N. Kudo[1,2] ✉, Renato F. Bulcão-Neto[1], Auri M.R. Vincenzi[2]

[1]*Instituto de Informática, Universidade Federal de Goiás, Goiânia-GO, Brazil*
[2]*Departamento de Computação, Universidade Federal de São Carlos, São Carlos-SP, Brazil*
✉ *E-mail: taciana@dc.ufscar.br*

Abstract: The low performance of software projects generally arises from erroneous, omitted, misinterpreted, or conflicting requirements. To produce better quality specifications, the practise of requirements reuse through requirement patterns has been widely debated in the secondary literature. However, a tertiary study that provides an overview of secondary studies on the state of the art and the practise of requirement patterns does not exist. This study describes a study of secondary literature on requirement patterns under a perspective on research and practise. The identification and selection methods of secondary studies include automatic search on five sources, inclusion, and exclusion criteria, the snowballing technique, and the quality assessment of those studies. Four secondary studies are considered relevant according to the purpose of this research from a 26-distinct-study group. The authors' contribution is two-fold: the tertiary study itself and a preliminary research agenda dealing with state of the art and practise on requirement patterns.

1 Introduction

While recognising the importance of requirements engineering in the software development process, recent studies [1, 2] demonstrate that a large percentage of software projects still extrapolate budget or deadlines or need to be cancelled. The primary cause of this poor performance is mainly due to the dependency on the customer and the interdisciplinary nature and inherent uncertainty of the requirements engineering process. This results in the bulk of incorrect, omitted, misinterpreted, or conflicting requirements.

This kind of problem can be addressed through the requirements reuse practise [3], in which a new software project makes use of requirements knowledge obtained from previous projects. As stated by Wiegers and Beatty [4], effectively reusing trusted requirements can promote consistency both within and across applications, fewer defects, reduced rework, higher team productivity, lower development costs, and faster delivery.

Among several proposals dealing with requirements reuse, the adoption of requirement patterns [5] has been much discussed in primary and secondary studies [6–9]. Instead of writing a requirement in natural language, a requirement pattern offers a systematic way to specify a particular type of requirement in the form of a template with categories of information. The structure and content of a requirement written as a pattern facilitate reuse.

Increasingly, secondary studies in software engineering have been used as a valuable research tool [10–12] for providing knowledge about a subject as well as allowing the identification of gaps and themes for future research. However, before starting work on a secondary study, it is essential to know if such type of study on a particular topic already exists. If this occurs, a tertiary review may be conducted, in which the subject of interest includes secondary studies on that topic [13].

This paper describes a tertiary study aiming at investigating the comprehensiveness of research on requirement patterns, from the selection and synthesis of relevant secondary studies on this subject. The methodology used relies on both classic tertiary studies in software engineering [14, 15] and the tool support [16] on the planning and conduction of the study protocol.

A gap between the state of the art and practise concerning requirement patterns is suggested by the results of our in-depth analysis of four relevant secondary studies. Such evidence allows us to propose a research agenda as a prelude to drive current and future research toward addressing this gap. As a tertiary study on requirement patterns does not exist, we elaborate and report the study protocol in such a way that one can easily replicate or augment it.

The organisation of this paper is as follows. Section 2 details the study protocol and Section 3 reports the data extraction of the secondary studies selected. Section 4 presents the synthesis of such secondary studies considering the research questions of this tertiary study and Section 5 discusses the threats to the validity of this research. Section 6 introduces a research agenda to gear future works on requirement patterns and Section 7 outlines the contributions of this work. Finally, the Appendix shows the list of studies identified by our search strategy.

2 Tertiary study protocol

As a tertiary review can adopt the same procedures used in secondary studies [17], we defined a process composed of three phases: planning, conducting, and publishing the results (see Fig. 1).

The planning phase of this tertiary study starts with the statement of the objective, followed by the definition of keywords. To find an adequate set of keywords, a pilot search has to be executed as many times as necessary. Next, the study protocol is created that, in turn, is evaluated to compose possible modifications, if necessary.

The conduction phase encompasses the activities of identification and selection of secondary studies, data extraction, and synthesis. The search strategy as part of the study protocol allows the identification of the studies, whereas the selection of these relies on inclusion and exclusion criteria (EC) and assessment quality criteria for secondary studies, both previously defined in the study protocol. The data extraction activity starts as soon as the relevant secondary studies are selected. Next, a synthesis of these studies is performed to answer the research questions of the tertiary study.

Then, synthesis results are reported as technical reports or scientific papers combining multiple formats such as textual, tabular, and graphical descriptions to name a few. In general, these documents are later disseminated and evaluated by expert readers.

The iterative aspect of the whole process in Fig. 1 allows the refinement of phases and activities in such a way that the study protocol contains every information. The management of

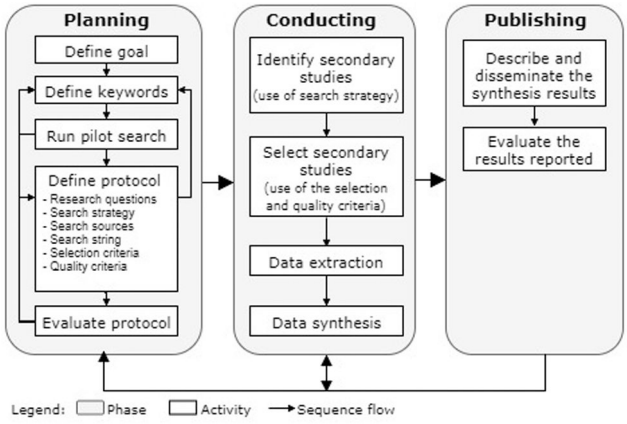

Fig. 1 *Phases and activities of this tertiary study, adapted from [17]*

systematic studies generally counts on tool support; in our case, we have been using the *StArt* tool [16].

2.1 Research questions and pilot search

The primary goal of this tertiary study is to investigate the scope of research on requirement patterns, which is compiled in the following research questions (RQs):

RQ1: What is state of the art in requirement patterns?
RQ2: What are the most searched topics on requirement patterns?
RQ3: What are the current gaps in requirement patterns research?

A proper search strategy is required to answer these research questions based on the maximum of relevant secondary studies. The first step is to perform a pilot search to find a search string that balances search comprehensiveness and accuracy in the relevance of the retrieved studies [17]. During the pilot search, a good practise is the search string analysis to find the gold-standard set of papers as well as to aid in the definition of a more consistent protocol. Sensitivity and precision are two critical metrics [18] in the search string analysis: a highly sensitive search strategy retrieves most of the relevant studies, but it can also retrieve many irrelevant ones; a highly accurate search strategy retrieves a small number of irrelevant papers, but it can miss important ones. The ideal strategy is one that can be both sensitive and precise, capturing precisely the gold standard without unnecessary items.

For that reason, our pilot search exploits the following synonyms related to the central research theme: *requirement pattern*, *requirement reuse*, *requirement template*, *systematic review*, *systematic mapping*, *systematic literature review*, *systematic literature mapping*, *review of studies*, *structured review*, *literature analysis*, *in-depth survey*, *literature survey*, *meta-analysis*, *past studies*, *subject matter expert*, *analysis of research*, *empirical body of knowledge*, *overview of existing research*, and *body of published research*.

After the analysis of the pilot search results, we re-examine the set of keywords, as suggested in [17], from which we made the following observations:

- The term ‘requirement reuse’ often returns studies whose focus is not requirement patterns, but is requirements reuse. For instance, to support requirements reuse, some studies retrieved propose the reuse of use cases, requirements traceability, and the extraction of terms from requirements.
- Requirement patterns focused studies always have the term ‘requirement pattern’ in their content, but seldom have the term ‘requirement reuse’.
- The term ‘requirement template’ shows up as keyword during the pilot search from studies in requirement patterns.
- The pilot search returns papers that describe literature surveys on requirement patterns.
- Dozens of papers whose knowledge area is not computer science arise because of some variants of the terms ‘systematic literature’, ‘systematic mapping’, and ‘survey’. Besides, some information sources do not provide users with a filtering mechanism by knowledge area.
- Despite using a long list of synonyms for systematic literature study, as suggested in [13], many of these synonyms do not change search results – an iterative refinement was necessary to obtain a more concise search string.

The analysis of the original set of keywords of the pilot search produces a 6.25% precision, i.e. three relevant papers from a group of 48 studies returned. After the test of keywords possibilities, a more balanced search string regarding sensitivity and precision is achieved: 4 relevant papers from a group of 26 papers retrieved, which means 15.4% of precision. Comparatively, one more relevant paper is retrieved, and almost half of unnecessary papers are rejected.

On the basis of this analysis, we reach the final set of keywords arranged as the following search string:

((‘*requirement pattern*’ OR ‘*requirement template*’) AND (*survey* OR ‘*systematic review*’ OR ‘*systematic literature review*’ OR ‘*systematic mapping*’ OR ‘*systematic literature mapping*’))

2.2 Search strategy

Once defined the string search, the next step is the choice of the ideal set of study sources applicable to the requirement patterns theme. Here is the list of sources selection criteria defined in this protocol:

- Sources considered relevant for the software engineering area.
- Sources with a search mechanism available on the web and logical expressions support.
- Sources that allow search over study abstract, at least.

As a result, the sources chosen for this tertiary study include the following search engines and digital libraries: ACM Digital Library, Engineering Village, IEEE Xplorer, Science Direct, and Scopus.

When necessary, the search string is tailored in response to the search capabilities of each source. In some sources, for example, variations of the terms ‘requirement pattern’ and ‘requirement template’ in the plural form are necessary. Owing to indexing a broader collection of papers, the search scope of the ACM Digital Library must be configured to *The ACM Guide to Computing Literature*. We also decide not to use temporal and idiom filtering on search results.

The Appendix presents a list of 26 studies, which are referenced from now on as [Sn], resulting from this search strategy. In general, including duplicates, 40 studies are found and distributed as follows: ACM Digital Library (4), Engineering Village (13), IEEE Xplorer (2), Science Direct (11), and Scopus (10).

2.3 Studies selection

This section describes the method used to identify which of the 26 non-duplicate studies remaining are relevant to answer the research questions of this tertiary study. The selection method is composed of inclusion criteria (IC) and EC [13], the snowballing technique [19], and quality assessment criteria (QC) [20].

2.3.1 Selection criteria: A set of six ECs were defined and applied to those 26 studies. The ECs are as follows:

EC1: Study not published in journals or conference proceedings.
EC2: The study does not cover computer science-related subjects.
EC3: Requirement patterns are not the main subject of the study.
EC4: It is not a secondary study.
EC5: It is an informal study since this does not define research questions, the search strategy, and the processes of data extraction and synthesis.
EC6: Full text not accessible.

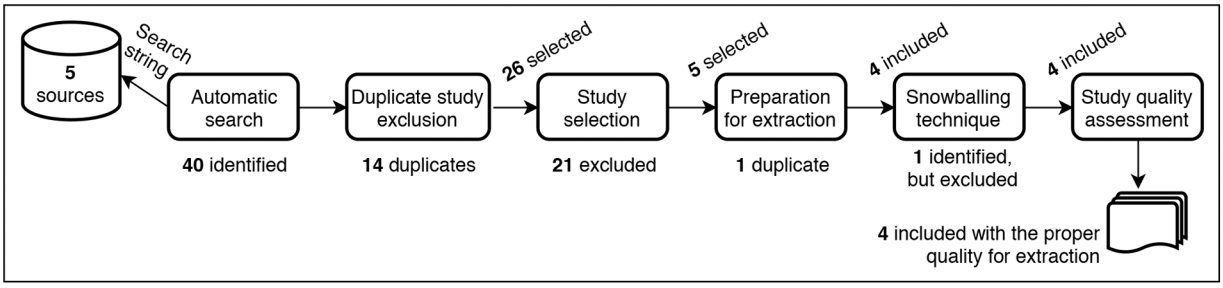

Fig. 2 *Detailed view of the identification and selection processes of secondary studies*

Table 1 Duplicate studies per information source

Study	ACM Digital Library	Engineering Village	IEEE Xplorer	Scopus
[S1]	—	∘	∘	•
[S2]	—	∘	—	•
[S4]	—	∘	—	•
[S15]	—	∘	—	•
[S20]	—	∘	—	•
[S21]	—	•	∘	—
[S22]	—	∘ ∘	—	∘ •
[S23]	•	∘	—	∘
[S24]	•	∘	—	∘

Table 2 Studies excluded and the respective criterion

Criterion	Studies excluded
EC1	[S1–S4]
EC2	[S5–S7]
EC3	[S8–S12]
EC4	[S13–S19]
EC5	[S20, S21]

The exclusion of a study occurs when it falls into at least one of such ECs. If not excluded, the study should meet one of the following inclusion criteria (IC):

IC1: The study features a systematic literature review.
IC2: The study features a systematic literature mapping.
IC3: The study features a literature survey.

After reading the title, abstract, and keywords of every study, the IC and EC drive which study should be rejected or not. In cases when the reading of papers' metadata is not sufficient for decision making, a further reading of their introduction and conclusion sections is needed.

As a result of the application of the IC and EC of the 40 studies identified in the automatic search, 14 are duplicates, 21 removed by EC, and 5 included. The number of papers identified, duplicated, excluded, and evaluated before data extraction is found in Fig. 2, which depicts the activities of identification and selection of secondary studies on requirements patterns of this tertiary study.

Regarding duplicate studies, Table 1 lists the 14 duplicate papers per information source. The ∘ symbol represents each study instance excluded (14) because of its copies in more than one bibliographic database. The • symbol, in turn, represents the instance of a duplicate study kept for the studies selection phase. Therefore, of the 26 studies identified through automatic search, 9 of them (•) have duplicates, and 17 do not. Observe also that automatic searches on Engineering Village and Scopus returned two instances of the study [S22] each, but only one of them went to the selection phase. This is because there are two versions of the same study [S22], both indexed by these two databases.

Concerning the 21 excluded studies Table 2 presents them grouped by EC. By analysing this table, the exclusion of five secondary studies is due to the fact they do not focus on requirement patterns (EC3). By also analysing their abstracts, we observe that these studies are about requirements reuse or specification, or software patterns, in general. None of these five secondary studies excluded by the EC3 criterion, therefore, deepens their research on requirement patterns, but only cite this term in their abstracts.

After the application of the EC, five studies remain, among them, a study [S23], which after being read in detail is also considered as duplicate, since the research questions and experiment are very similar to a same authors' another study [S24]. Finally, after applying the inclusion and EC, the following four secondary studies remain: [S22, S24–S26], which refer to [8, 9, 21, 22].

2.3.2 Snowballing: The snowballing technique [19] allows the identification of relevant studies by scanning the list of bibliographic references or citations of a paper. When you examine the former, this is called backward snowballing; in the case of the latter, it is called forward snowballing.

The backward snowballing technique applied to the four secondary studies selected [S22, S24–S26] complements the search strategy of this tertiary study, besides automatic searching. The list of bibliographic references of these studies is then analysed to check whether other relevant secondary studies on requirement patterns exist or not. From this analysis, one identifies that [S25] see the [S22] and [S24] works, which demonstrates consistency among these three studies.

Fig. 3 illustrates a new secondary study [3] found after backward snowballing over [S24] work. This new study is missing in the initial selection phase because of the search string used (only the term 'survey' is found). Even after reading the title, abstract, and keywords of this study, the decision for its inclusion or not happens only after reading the introduction and conclusion sections. Chernak's work [3] reports research on requirements reuse in practise, but it does not cite requirement patterns at any point in its analysis. Therefore, this study was excluded by the EC3 criterion because it did not have requirement patterns as the central theme.

2.3.3 Quality assessment of secondary studies: An important decision when performing systematic literature studies is to check the quality of primary or secondary studies. The Centre for Reviews and Dissemination (CDR) [20] maintains a database of systematic reviews in medicine selected according to pre-established quality criteria. These same quality criteria can also be used to assess systematic studies regardless of knowledge area, as some works reuse them in software engineering [13, 15].

In this tertiary study, we analyse each secondary study from the light of four questions such as QC based on the CDR criteria. The acceptable responses for each question are Y (Yes), P (Partially), and N (Not). The following is the list of quality assessment-related questions.

QC1: Are the IC and EC appropriately described?

Y: Criteria are explicit.
P: Criteria are implicit.
N: Criteria are not defined or not easily identified.

QC2: Does the search cover all relevant studies?

Y: It uses four or more sources relevant and one additional search strategy.
P: It uses three sources relevant, but no extra search strategy.
N: It uses at most two sources relevant to the area of interest.

QC3: Is the quality or validity of the included primary studies assessed?

Y: Quality criteria are explicit and associated with each primary study.
P: Research questions from the secondary study address the quality of primary studies.
N: No quality assessment.

QC4: Are primary studies adequately described?

Y: Details of each primary study are explicit.
P: There is only a summary of each primary study.
N: There are no results of the primary studies analysed.

Considering the response to each question, we assign the following score: 1 (one) for each 'Yes' response; 0.5 (medium) to 'Partly'; and 0 (zero) for 'No'. This benchmark evaluates the scores of the secondary studies selected for data extraction [S22, S24–S26].

The final quality score of each secondary study is the arithmetic mean of the scores for its quality questions. The importance of a final quality score is two-fold: to reject studies that do not meet quality criteria defined in the study protocol and to guide the synthesis activity of the secondary studies. In this tertiary study, we reject those studies that scored 0 (zero) on all quality criteria described (QC1–QC4).

Table 3 presents the final quality score for each secondary study selected. Note that there is no study excluded since all studies score higher than zero on at least one of the quality questions. It is worth mentioning that the highest quality score study [S24] includes a literature review that explicitly defines the search strategy, search string, information sources, studies selection criteria, analysis, and synthesis.

3 Data extraction

This section describes information extracted from the full-text reading of the secondary studies selected, which includes:

- The main objective and respective research questions.
- The selection methods of primary studies.
- The evidence collected from the synthesis of these studies.

The four secondary studies selected for data extraction and synthesis dates from 2011, 2017, and 2018: three of them published in international journals and one in an international workshop. Following the order of quality score of the secondary studies, Table 4 presents the respective objectives, search strategies (A for automatic, M for manual, and S for snowballing), and the number of sources and primary studies analysed.

Palomares *et al.* [S24] map the state of practise of requirements reuse and patterns through a literature review that presents search strategy, search string, studies selection criteria, synthesis, and a conclusion of the analysis made. The sources of studies used are ACM Digital Library, IEEE Xplorer, Science Direct, and Springer Link.

The review results indicate an unusual practise of requirements reuse and patterns in industry. Authors also look for surveys and interviews on the state of the art of requirements reuse in companies and identify the need for work that gives more evidence on the positive and negative points of the different abstractions and artefacts in the requirements reuse proposals.

For this reason, authors carry out an exploratory survey with 71 requirements engineers that answer an online questionnaire on both requirements reuse and the use of requirement patterns in industry practise. As a result, the respondents identify problems that the use of requirement patterns can solve such as specification completeness and requirements uniformity, inconsistency, and ambiguity.

In other secondary study relevant for this tertiary review, Barros-Justo *et al.* [S25] present a systematic mapping to find the software patterns generally reported in scientific research, the requirements engineering activities impacted by the use of these patterns, and which software development or end product properties are affected by the use of software patterns. The authors do not specify the type of software pattern (e.g. requirement pattern or design pattern); the point is whether the industry makes use of software patterns or not. The search strategy for the selection of primary studies includes manual and automatic search, the snowballing technique, and the sources listed next: ACM Digital Library, IEEE Xplorer, Scopus, and Web of Science.

The next secondary study analysed is Irshad *et al.* [S26] that presents a systematic review of research on requirements reuse. The main result is the creation of 11 groups with different reuse approaches, of which three of these focus on requirement patterns: (i) a structure group, with studies on requirements organisation to facilitate reuse; (ii) a template group, with studies that provide a guide for writing requirements in a specific format; and (iii) a hybrid group, with studies that use a combination of different approaches to support requirements reuse.

In this recent study, the main source of primary studies is the Google Scholar search engine, and the search strategy also includes snowball sampling. However, these authors carry out a sanity check searching on the Scopus database. The point is to check the

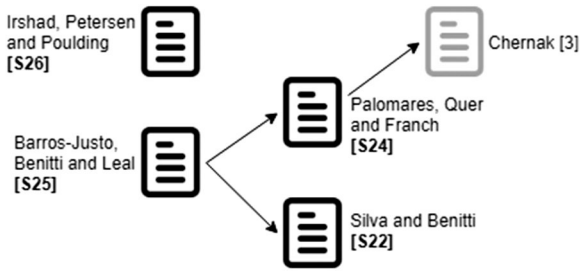

Fig. 3 *Backward snowballing result: a new secondary study [3] is identified from [S24]*

Table 3 Quality score of secondary studies for extraction, in descending order

Secondary study	QC1	QC2	QC3	QC4	Score
Palomares *et al.* [S24]	1	1	1	1	**1**
Barros-Justo *et al.* [S25]	1	1	0.5	1	**0.87**
Irshad *et al.* [S26]	1	0	1	0.5	**0.62**
Silva and Benitti [S22]	1	0.5	0	0	**0.37**

existence of new primary studies not found through Google Scholar.

These authors also recognise that metamodels can be used to structure and facilitate the reuse of requirement patterns. They conclude that most of the studies proposing the use of templates also suggest the use of natural language. Additionally, there are studies on ontologies and templates of use cases intended for requirement patterns reuse. As for the hybrid approach, one primary study uses a catalogue of requirements to facilitate requirements traceability, whereas other study makes use of families of applications to identify requirement patterns and, in turn, for easier requirements reuse.

Finally, Silva and Benitti [S22] identify and compare the available approaches to requirements writing. Regarding the set of evidence found as a result of this systematic mapping, requirement patterns support the specification of both non-functional and functional requirements: non-functional requirements writing based on categories; functional requirements writing in the form of information systems requirements; and the occurrence of composite requirement patterns. The search strategy of this work includes automatic search on 14 sources including ACM Digital Library, IEEE Xplorer, Science Direct, Springer Link, and Brazilian thesis and dissertations digital libraries, to name a few.

4 Data synthesis

This section presents a synthesis of the data extracted concerning the research questions of this tertiary study.

4.1 About the research question 1

To answer the question 'What is the state of the art in requirement patterns?', we made the following observations.

4.1.1 Research scope: Barros-Justo *et al.* [S25] define a comprehensive search string and, even using only four sources, they identify a large number of non-relevant studies. In this case, the ratio of relevant and identified primary studies is 0.7% (22 out of 3070).

On the one hand, Silva and Benitti [S22] define a broader string concerning the keywords used, and even using 14 sources, they identify few studies (56) in comparison with Barros-Justo *et al.* [S25]. However, only three primary studies among the 56 identified (∼5%) are relevant.

Palomares *et al.* [S24], in turn, analyse 22 papers on requirements reuse, of which 11 are specific to requirement patterns. Irshad *et al.* [S26] make use of solely the Google Scholar search engine, but with an extensive search string ('requirements' AND 'reuse') to retrieve studies on requirements reuse, in general. Even so, 14 primary studies for data synthesis are selected.

After analysing the set of primary research reported on those four secondary studies [S22, S24–S26], we identify that:

- There are 44 different primary studies on requirement patterns, but only 4 are cited by more than one of the secondary studies under investigation.
- None of these 44 primary studies is cited simultaneously in these secondary studies.
- About ∼60% of the different primary research (26 of 44) selected date from 2007 to 2016 (year of publication of the most recent primary study).

Facing the numbers presented and the closeness of the research objectives of these secondary research, we conclude that requirement patterns are a non-stagnant research topic, with contributions throughout this decade.

4.1.2 Quality of studies: Regarding the quality of the secondary studies (see Section 2.3.3.), we observe that all secondary studies define criteria for primary research selection (QC1).

When analysing the strategy and scope of research (QC2), Silva and Benitti [S22] perform automatic search only, whereas both Palomares *et al.* [S24] and Barros-Justo *et al.* [S25] adopt a mixed search strategy: both works make automatic and manual search, and the latter also carries out the snowballing technique for primary studies identification. Differently, Irshad *et al.* [S26] execute additional sanity checking procedure to validate the results of their automatic search and snowball sampling in a single source.

Considering the quality assessment of primary research (QC3), Palomares *et al.* [S24] identify and analyse four industrial applications of requirements reuse through patterns. We consider the way how the authors perform the analysis of primary studies as a quality assessment, describing the benefits of using patterns, and the aspects measured (e.g. time saving and the quality of specifications). Irshad *et al.* [S26], in turn, determine how the existing requirements reuse approaches have been evaluated (e.g. as experiments, case studies etc.), and also elicit the outcomes of the evaluations. Besides, they assess the quality of primary empirical studies based on the rigour and relevance criteria.

Still, regarding the criterion QC3, Barros-Justo *et al.* [S25] do not explicitly assess the quality of studies, but a research question concerns the analysis of the research methods used to assess the impact of the use of requirement patterns. Finally, Silva and Benitti [S22] do not mention the quality assessment of primary studies.

About the quality criterion (QC4), related to the detailed description of the primary studies, Palomares *et al.* [S24] and

Table 4 Data extracted from secondary studies: number of sources, search strategy, and number of primary studies analysed

Secondary study	Sources	Search strategy	Primary studies
Palomares *et al.* [S24]	4	A M	11
venue: *empirical software engineering: an international journal*			
objective: map state of the practise of requirements reuse; identify why proposals of requirements reuse are not used in practise			
string: *reuse* AND *requirements*			
Barros-Justo *et al.* [S25]	4	A M S	22
venue: *computer standards and interfaces journal*			
objective: detect patterns reported in scientific research; identify requirements engineering activities impacted by the use of requirement patterns; point out properties of the software development process or the final product affected by the use of requirement patterns			
string: *pattern* AND *engineering* AND (*software* OR *requirement*)			
Irshad *et al.* [S26]	2	A S	14
venue: *information and software technology journal*			
objective: identify different approaches of requirements reuse; discover evaluation methods and the respective results for requirements reuse approaches; and the relevance and rigour of the empirical studies of requirements reuse approaches			
string: *requirements* AND *reuse*			
Silva and Benitti [S22]	14	A	3
venue: *workshop on requirements engineering*			
objective: map current standards for writing software requirements			
string: (*pattern* OR *catalogue* OR *category*) AND ('*software requirement*' OR '*requirements engineering*' OR '*requirements elicitation*')			

Barros-Justo *et al.* [S25] describe in detail both included and excluded studies. Irshad *et al.* [S26] superficially describe each primary research, grouping it into categories defined as a result of their analysis. In turn, Silva and Benitti [S22] do not individually detail any primary research, but do only a brief description of it.

Therefore, only Palomares *et al.* [S24] meet all the quality requirements defined herein, obtaining the maximum quality score as indicated in Table 3.

4.2 About the research question 2

To answer the question 'What are the most searched topics regarding requirement patterns?', we present our considerations next.

4.2.1 Requirement patterns format: Both Palomares *et al.* [S24] and Irshad *et al.* [S26] conclude that most of the requirement patterns identified in their studies are written using natural language, use cases, or other diagrams.

Silva and Benitti [S22] report that the three primary studies analysed represent requirement patterns as predefined forms (or templates). In turn, Barros-Justo *et al.* [S25] do not detail primary research with the analysis of the format of requirement patterns.

4.2.2 Requirement patterns availability: Most of the proposals of requirement patterns are available as a repository or catalogue of patterns, as stated by all secondary research under analysis. Palomares *et al.* [S24] also report that some primary studies provide details of how to build a repository of requirement patterns, and of how to classify and identify patterns to facilitate reuse. Barros-Justo *et al.* [S25], in turn, report the existence of more than 300 software patterns grouped in catalogues.

4.2.3 Requirement patterns scope: All secondary studies also state that most of the proposed requirement patterns are domain independent, though some primary researches develop patterns for specific domains.

Concerning the four secondary studies under analysis, only Silva and Benitti's work [S22] does not report primary research proposing requirement patterns for specific domains such as embedded systems, real-time systems, and non-functional security requirements. Besides, Irshad *et al.* [S26] give particular focus on requirements reuse for software product lines.

4.2.4 Requirement patterns purpose: Most of the primary studies reported in Palomares *et al.* [S24] and Silva and Benitti [S22] focus on requirements elicitation and specification.

Despite reaching this same conclusion, Barros-Justo *et al.* [S25] also describe primary studies focusing on requirements validation and requirements management. Irshad *et al.* [S26], in turn, do not make any analysis of the requirements-related activities benefited from the use of requirement patterns.

4.3 About the research question 3

To answer the question 'What are the current gaps of research on requirement patterns?', we present our analysis next.

4.3.1 Difficulty of reuse: Palomares *et al.* [S24] identify the difficulty of reusing requirements in business practise since it seems that only requirements engineers of large companies with well-established processes are partially capable of reuse requirements artefacts. The main difficulty lies in the fact that professionals do know neither requirement patterns, nor how to use them. On the basis of this difficulty of requirement patterns reuse, those authors propose five levels of reuse, each for a company profile:

(i) For companies that do not employ any reuse practise.
(ii) For companies aware of the benefits of reuse, but who have not implemented it in their development processes.
(iii) For companies that use both 'copy and paste' techniques and natural language during the requirements specification activity.
(iv) For companies that make use of a catalogue of requirement patterns.
(v) For companies that own a well-established process of management and customisation of requirements catalogue.

Barros-Justo *et al.* [S25] confirm the assertion that the use of requirement patterns in practise is unusual, which is justified by the low scores of selected primary studies reporting the use of requirement patterns.

4.3.2 Lack of validation: Palomares *et al.* [S24] report that few proposals for reusing requirement patterns have been evaluated or validated in enterprise settings. From their exploratory study, they conclude that engineers demand more evidence about the benefits of the de facto use of requirement patterns.

Irshad *et al.* [S26] conclude that approximately two-thirds of the requirements reuse approaches including requirement patterns are not validated in the industry setting. Also, they argue that this lack of validation has no apparent reason in the primary studies analysed. However, the same authors note that textual artefacts have more validation evidence in the industry, possibly due to its ease of use.

Barros-Justo *et al.* [S25] identify the difficulty of evaluating the impact of the use of requirement patterns due to the lack of objective metrics such as the time of writing or editing requirements. These authors also point out a research gap in metrics to evaluate the impact of patterns on the different activities of requirements engineering.

4.3.3 Enabling the use of requirement patterns: Both Silva and Benitti [S22] and Palomares *et al.* [S24] observe the need for tool support to promote further the adoption of requirement patterns. Silva and Benitti [S22] also report that their primary studies analysed mention proposals of tools, but in preliminary, poorly consolidated versions.

Palomares *et al.* [S24], in turn, reach this same observation based on the opinions of 71 professionals who answered an online questionnaire. Also regarding this questionnaire responses, restrictions on the use of requirement patterns might be related to two aspects: the lack of methodologies integrating requirement patterns to companies' requirements processes and the difficulty of access to the existing catalogues of requirement patterns.

4.3.4 Benefits from the use of requirement patterns: Among the four secondary studies selected, only Barros-Justo *et al.* [S25] analyse in detail the reports of the impact of the use of requirement patterns in requirements engineering activities. They conclude that the elicitation and specification activities are the ones that benefit most from the use of requirement patterns, and the positive impacts are on quality, productivity, reuse, design time, and support to identify problematic requirements.

In turn, Irshad *et al.* [S26] claim that the main benefits from requirements reuse and, more specifically, from requirement patterns reuse, are the time savings and resource spending in creating new artefacts. However, they also report that only two primary studies discuss these benefits: most of the primary studies discuss details of modelling, execution, and outcomes of requirements reuse approaches.

4.3.5 Software development process: Once elicited, analysed, specified, and validated, software requirements are input data for the definition of several artefacts produced in other phases of the software development process including macro and microarchitectural designs, user interface design, test cases, maintenance strategies, among others.

For this reason, we would like to have analysed the impact of the use of requirement patterns on other phases of the software life cycle (design, construction, testing, and maintenance) as well as the traceability between requirements and artefacts produced in other development phases (e.g. test cases). However, none of the

four secondary studies found in this tertiary study brings up this issue, which arises as a research gap on requirement patterns.

4.4 Data synthesis overview

Results of this synthesis activity are compiled in Table 5. Owing to space reasons, we abbreviate some words as follows: A for automatic, M for manual, and S for snowballing (search strategy); NL for natural language and UC for use case (format); E for elicitation, Sp for specification, V for validation, and M for management (purpose); and SRP for software requirement pattern.

5 Threats to validity

One of the problems faced in systematic studies is to find all relevant researches on a particular theme. To address this issue, the tertiary study protocol includes automatic search in five relevant sources for the software engineering area. A previous pilot search activity also contributes to the refinement of the study protocol quality. Besides, backward snowballing on the four secondary studies selected mitigates the likelihood of missing relevant research. This study selection strategy follows quality criteria (see QC 2 in Section 2.3.3) defined in state-of-the-art systematic literature studies such as [13, 15, 20]. Note that grey literature searching is not part of our protocol because we assume that we find good quality studies on requirement patterns in journals or conferences indexed by the sources chosen.

Still, regarding the extent of search results, the popular Springer Link digital library is not chosen because it does not support searches over study abstracts, but only over study title or full text. However, the Scopus engine mitigates the likely negative impacts on search coverage because it indexes the Springer Link database (e.g. studies [S17], [S23], and S24]).

Three researchers conceived this tertiary study protocol: the definition of research questions, search strategy, information sources, search string, IC and EC, and QC. Researcher A is the team leader with vast expertise in software engineering. Researcher B, an expert in requirements engineering, conducts the identification, extraction, and synthesis of secondary studies relevant following the search strategy and the mentioned criteria (including the pilot search). Researcher C, also an expert in software engineering, verifies all the results of such phases to mitigate the possibility of biases throughout the process. In the case of divergences, both researchers B and C resolve the conflicts together.

Another point to highlight is the number of relevant secondary studies included for data extraction and synthesis in this study. Despite the small number of secondary studies included, the whole information extracted from these is complementary, and it also gives evidence on gaps in requirement patterns research as well as a preliminary research agenda to handle them. A similar analysis over solely primary studies may not reveal these gaps since each primary research presents a specific view of the requirement patterns subject.

Even though the reproducibility of systematic studies is extremely difficult [23], the documentation of this tertiary study protocol is in enough detail for replication and audit purposes. For instance, the reporting of the protocol includes not only the list of studies found (see the Appendix) as a result of the search strategy, but also the list of papers duplicate and rejected after employing the selection criteria (see Tables 1 and 2). Thus, the transparency of the study protocol also allows access to the papers rejected since these can be useful for readers' purposes.

Finally, the quality of the secondary studies included was also considered according to quality criteria widely accepted in the literatures [13, 15, 20], what guarantees reliability to the evidence and research agenda reported.

6 Research agenda

From the analyses of [S22, S24–S26], we find evidence of a gap between the states of the art and practise regarding the effective use of requirement patterns. Existing research on requirement patterns still does not adequately address the needs of requirements engineers in particular, and software engineers, in general.

As a contribution, we outline a preliminary, but well-founded, research agenda to lessen this gap, containing studies that:

(i) Demonstrate in industry software projects the benefits of using requirement patterns in other phases of the software development process – none of the secondary studies analysed explicitly identified this gap.
(ii) Investigate, in a complementary way to the previous action, the traceability between requirements expressed as patterns and the artefacts produced in other development phases – this is other likely research not reported in any of the secondary studies analysed.
(iii) Show evidence of the effective use of requirement patterns particularly in the requirements engineering process of industry software projects.
(iv) Establish objective metrics that help software engineers, in general, and requirements engineers in particular, measure the impact of the use of requirement patterns as described in items (i)–(iii).
(v) Enable the joint use of existing and well-established requirement patterns and methodologies in industry software projects such as agile approaches.

Table 5 Tertiary study synthesis overview

	[S22]	[S24]	[S25]	[S26]
RQ1: *what is the state of the art in requirement patterns?*				
scope	*search strategy*: A *sources*: 14 *primary studies*: 3	*search strategy*: A M *sources*: 4 *primary studies*: 11	*search strategy*: A M S *sources*: 4 *primary studies*: 22	*search strategy*: A S *sources*: 2 *primary studies*: 14
quality	*score*: 0.37	*score*: 1	*score*: 0.87	*score*: 0.62
RQ2: *what are the most searched topics regarding requirement patterns?*				
format	predefined forms	NL, UC, or other diagrams		NL, UC, or other diagrams
availability	repository or catalogue	repository or catalogue	repository or catalogue	repository or catalogue
scope	domain independent	domain independent	domain independent	software product line
purpose	E Sp	E Sp	E Sp V M	
RQ3: *What are the current gaps in research on requirement patterns?*				
difficulty		unfamiliarity with SRP	unfamiliarity with SRP	
validation		few validations in industry	need for metrics	few validations in industry
enabling SRP	tool support	tool support		
benefits			quality of specification, reuse support, less design time, higher productivity, problematical requirements resolution assistance	time and resources saving
process	none of these secondary studies analyses the impact of SRP usage in the software development life cycle			

(vi) Build tools that effectively support practitioners' practises in the use of requirement patterns.
(vii) Disseminate current and future catalogues of requirement patterns in a systematised manner, for example, with classification by specific domains or by use of versioning mechanisms.

These lines of action aim to approximate academics and requirements engineers so that both can improve their understanding of the impact of using requirement patterns in industry software projects, considering related processes and activities, quality, organisational culture, to name a few.

7 Conclusions

The consolidation of secondary studies on a particular research topic justifies the conception of a tertiary study. In software engineering, however, there are few tertiary studies such as the studies [15, 24], both with emphasis on secondary studies covering general subjects of software engineering.

There are other tertiary studies in the literature, each with a different focus such as the process of conducting systematic reviews in software engineering [25] and the quality assessment of systematic reviews in software engineering [26]. There is also a tertiary study in software product lines [27], but it does not give attention to secondary research on requirement patterns.

To the best of authors' knowledge, a novelty in this paper is that no tertiary study on the states of the art and practise of requirement patterns exists. On the basis of the data synthesis performed in this tertiary study, requirement patterns are a research topic with many contributions over the last decade.

We advocate the use of our tertiary study protocol as guidance for conducting new tertiary research on requirement patterns. Enough information is available to replicate and augment this tertiary study such as:

- The pilot search for protocol verification and validation.
- The alignment between objective and research questions.
- Arguments for the choice of the search string and sources of studies.
- The use of IC and EC.
- The application of the snowballing technique.
- The quality assessment of secondary studies.
- The data extraction and the synthesis of the responses to the research questions.

Finally, we also contribute to state of the art with a research agenda to match researchers efforts to the software industry needs concerning requirement patterns usage in practise.

8 Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The authors thank the anonymous referees for their valuable comments and helpful suggestions. Renato Bulcão-Neto is grateful for the scholarship granted by the CAPES/FAPEG (88887.305511/2018-00) in the context of the postdoctoral internship held at the Department of Computation and Mathematics, FFCLRP, University of São Paulo.

9 References

[1] Tockey, S.: 'Insanity, hiring, and the software industry', *Computer (Long Beach Calif.)*, 2015, **48**, (11), pp. 96–101
[2] Fernández, D.M., Wagner, S., Kalinowski, M., *et al.*: 'Naming the pain in requirements engineering: contemporary problems, causes, and effects in practice', *Empir. Softw. Eng.*, 2017, **22**, (5), pp. 2298–2338
[3] Chernak, Y.: 'Requirements reuse: the state of the practice'. 2012 IEEE Int. Conf. Software Science, Technology and Engineering, Herzlia, Israel, 2012, pp. 46–53
[4] Wiegers, K., Beatty, J.: '*Software requirements*' (Microsoft Press, Redmond, WA, USA, 2013, 3rd edn.)
[5] Withall, S.: '*Software requirement patterns*' (Microsoft Press, Redmond, WA, USA, 2007, 1st edn.)
[6] Kumar, K., Saravanaguru, R.K.: 'Context aware requirement patterns (CaRePa) methodology and its evaluation', *Far East J. Electron. Commun.*, 2016, **16**, (1), pp. 101–117
[7] Schweiger, A.: 'Applying software patterns to requirements engineering for avionics systems'. 2013 IEEE Int. Systems Conf., Orlando, FL, USA, April 2013, pp. 25–30
[8] Palomares, C., Quer, C., Franch, X.: 'Requirements reuse and requirement patterns: a state of the practice survey', *Empir. Softw. Eng.*, 2017, **22**, (6), pp. 2719–2762
[9] Irshad, M., Petersen, K., Poulding, S.: 'A systematic literature review of software requirements reuse approaches', *Inf. Softw. Technol.*, 2018, **93**, (C), pp. 223–245
[10] Vasconcellos, F.J.S., Landre, G.B., da Cunha, J.A.O.G., *et al.*: 'Approaches to strategic alignment of software process improvement: a systematic literature review', *J. Syst. Softw.*, 2017, **123**, pp. 45–63
[11] Vallon, R., da Silva Estácio, B.J., Prikladnicki, R., *et al.*: 'Systematic literature review on agile practices in global software development', *Inf. Softw. Technol.*, 2018, **96**, pp. 161–180
[12] Dikici, A., Türetken, O., Demirörs, O.: 'Factors influencing the understandability of process models: a systematic literature review', *Inf. Softw. Technol.*, 2018, **93**, pp. 112–129
[13] Kitchenham, B., Charters, S.: 'Guidelines for performing systematic literature reviews in software engineering'. Durham, UK, 2007. EBSE 2007-001
[14] Kitchenham, B., Pretorius, R., Budgen, D., *et al.*: 'Systematic literature reviews in software engineering – a tertiary study', *Inf. Softw. Technol.*, 2010, **52**, (8), pp. 792–805
[15] Cruzes, D., Dybå, T.: 'Research synthesis in software engineering: a tertiary study', *Inf. Softw. Technol.*, 2011, **53**, (5), pp. 440–455
[16] Fabbri, S., Silva, C., Hernandes, E.M., *et al.*: 'Improvements in the start tool to better support the systematic review process'. Proc. 20th Int. Conf. Evaluation and Assessment in Software Engineering, Limerick, Ireland, 2016, pp. 21:1–21:5
[17] Fabbri, S.C.P.F., Felizardo, K.R., Ferrari, F.C., *et al.*: 'Externalising tacit knowledge of the systematic review process', *IET Softw.*, 2013, **7**, (6), pp. 298–307
[18] Zhang, H., Babar, M.A., Tell, P.: 'Identifying relevant studies in software engineering', *Inf. Softw. Technol.*, 2011, **53**, (6), pp. 625–637
[19] Wohlin, C.: 'Guidelines for snowballing in systematic literature studies and a replication in software engineering'. 18th Int. Conf. Evaluation and Assessment in Software Engineering, London, England, 2014, pp. 38:1–38:10
[20] Centre for Reviews and Dissemination – University of York, *Effectiveness Matters*, 2002, **6**, (2), pp. 1–4
[21] Silva, R.C., Benitti, F.B.V.: 'Writing standards requirements: a systematic literature mapping', Proc. Workshop on Requirements Engineering, Rio de Janeiro, RJ, Brazil, April 2011, pp. 259–272
[22] Barros-Justo, J.L., Benitti, F.B.V., Leal, A.C.: 'Software patterns and requirements engineering activities in real-world settings: a systematic mapping study', *Comput. Stand. Interfaces*, 2018, **58**, pp. 23–42
[23] Kuhrmann, M., Fernández, D.M., Daneva, M.: 'On the pragmatic design of literature studies in software engineering: an experience-based guideline', *Empir. Softw. Eng.*, 2017, **22**, (6), pp. 2852–2891
[24] Kitchenham, B., Pearl Brereton, O., Budgen, D., *et al.*: 'Systematic literature reviews in software engineering – a systematic literature review', *Inf. Softw. Technol.*, 2009, **51**, (1), pp. 7–15
[25] Imtiaz, S., Bano, M., Ikram, N., *et al.*: 'A tertiary study: experiences of conducting systematic literature reviews in software engineering'. 17th Int. Conf. Evaluation and Assessment in Software Engineering, Porto de Galinhas, Brazil, 2013, pp. 177–182
[26] Zhou, Y., Zhang, H., Huang, X., *et al.*: 'Quality assessment of systematic reviews in software engineering: a tertiary study'. Proc. 19th Int. Conf. Evaluation and Assessment in Software Engineering, Nanjing, China, 2015, pp. 14:1–14:14
[27] Marimuthu, C., Chandrasekaran, K.: 'Systematic studies in software product lines: a tertiary study'. Proc. 21st Int. Systems and Software Product Line Conf., Sevilla, Spain: ACM, September 2017, pp. 143–152

10 Appendix

Here, we include the list of studies identified (except duplicates) in the automatic search performed on 8 March 2018. As search engines capabilities influence each search string used, some information sources require variations of terms 'requirement pattern' and 'requirement template'. However, this type of adaptation does not prejudice the replication of this tertiary study protocol since we associate each search string with the corresponding information source.

[S1] Franch X.: 'Software requirements patterns – a state of the art and the practice'. Proc. 37th IEEE/ACM Int. Conf. Software Engineering, Florence, Italy, May 2015, pp. 943–944

[S2] Smith J.E. (Ed.): Proc. 11th IASTED Int. Conf. Software Engineering and Applications, Cambridge, MA, 2007

[S3] Vian K., Johansen R.: 'Knowledge synthesis and computer-based communication systems: changing behaviours and concepts', in (Eds.): '*Knowledge management tools*' (Butterworth-Heinemann, 1997), pp. 187–208

[S4] Franch X., Berry D.M., Herrmann A., *et al.* (Eds.): 'Joint workshops on 20th international conference on requirements engineering: foundation for software quality', CEUR-WS, 2014, Essen, Germany

[S5] Kohlmeier M.: 'Minerals and trace elements', in Kohlmeier M. (Ed.): '*Nutrient metabolism*' (Academic Press, 2015, 2nd edn.), pp. 673–807

[S6] Veiga-Gil L., López-Olaondo L., Pueyo J., *et al.*: 'Low doses of haloperidol combined with ondansetron are not effective for prophylaxis of postoperative nausea and vomiting in susceptible patients', *Cirugía Española*, 2015, **93**, (2), pp. 110–116

[S7] Zheng Z., Wang Y., Wang S., *et al.*: 'Research on tomato water requirement with drip irrigation under plastic mulch in greenhouse'. Proc. Int. Conf. Agro-Geoinformatics, Tianjin, China, July 2016, pp. 1–5

[S8] Bakar N.H., Kasirun Z.M., Salleh N.: 'Feature extraction approaches from natural language requirements for reuse in software product lines: a systematic literature review', *J. Syst. Softw.*, 2015, **106**, pp. 132–149

[S9] Daneva M.: 'Striving for balance: a look at gameplay requirements of massively multiplayer online role-playing games', *J. Syst. Softw.*, 2017, **134**, pp. 54–75

[S10] Nicolás J., Álvarez J.A.T.: 'On the generation of requirements specifications from software engineering models: a systematic literature review', *Inf. Softw. Technol.*, 2009, **51**, (9), pp. 1291–1307

[S11] Riaz M., Breaux T., Williams L.: 'How have we evaluated software pattern application? A systematic mapping study of research design practices', *Inf. Softw. Technol.*, 2015, **65**, pp. 14–38

[S12] Weir C.R., Rubin M.A., Nebeker J., *et al.*: 'Modelling the mind: How do we design effective decision-support?', *J. Biomed. Inf.*, 2017, **71**, pp. S1–S76

[S13] Bürger J., Strüber D., Gärtner S., *et al.*: 'A framework for semi-automated co-evolution of security knowledge and system models', *J. Syst. Softw.*, 2018, **139**, pp. 142–160

[S14] Hoffmann A., Hoffmann H., Sollner M.: 'Fostering initial trust in applications – developing and evaluating requirement patterns for application websites'. Proc. European Conf. Information Systems, Utrecht, The Netherlands, June 2013, pp. 1–13

[S15] Kumar K., Saravanaguru R.A.K.: 'Context aware requirement patterns (CaRePa) methodology and its evaluation', *J. Teknol.*, 2016, 78, (6), pp. 101–117

[S16] Kim D.-K., France R.B., Ghosh S., *et al.*: 'A role-based metamodeling approach to specifying design patterns'. Proc. Int. Computer Software and Applications Conf., Dallas, TX, USA, November 2003, pp. 452–457

[S17] Li Y., Bruegge B., Stähler S., *et al.*: 'Requirements engineering for computational seismology software', in Bader M., Bungartz H.-J., Weinzierl T. (Eds.): '*Advanced computing. Lecture notes in computational science and engineering*' (Springer, Berlin Heidelberg, 2013), pp. 157–175

[S18] Sultan M., Miranskyy A.V.: 'Ordering interrogative questions for effective requirements engineering: the W6H pattern'. Proc. IEEE Int. Workshop on Requirements Patterns, Ottawa, ON, Canada, August 25, pp. 1–8

[S19] Zhu F.: 'A requirement verification framework for real-time embedded systems'. PhD thesis, University of Minnesota, 2002

[S20] Mahendra P., Ghazarian A.: 'Patterns in the requirements engineering: a survey and analysis study', *WSEAS Trans. Inf. Sci. Appl.*, 2014, **11**, pp. 214–230

[S21] Schweiger A.: 'Applying software patterns to requirements engineering for avionics systems'. Proc. IEEE Int. Systems Conf., Orlando, FL, USA, April 2013, pp. 25–30

[S22] Silva R.C., Benitti F.B.V.: 'Writing standards requirements: a systematic literature mapping'. Proc. Workshop on Requirements Engineering, Rio de Janeiro, RJ, Brazil, April 2011, pp. 259–272

[S23] Palomares C., Franch X., Quer C.: 'Requirements reuse and patterns: a survey', in Salinesi C., van de Weerd I. (Eds.): '*Requirements engineering: foundation for software quality*'. Lecture notes in computer science (Springer, Cham, 2014), pp. 301–308

[S24] Palomares C., Quer C., Franch X.: 'Requirements reuse and requirement patterns: a state of the practice survey', *Empir. Softw. Eng.*, 2017, **22**, (6), pp. 2719–2762

[S25] Barros-Justo J.L., Benitti F.B.V., Leal A.C.: 'Software patterns and requirements engineering activities in real-world settings: a systematic mapping study', *Comput. Stand. Interfaces*, 2018, **58**, pp. 23–42

[S26] Irshad M., Petersen K., Poulding S.: 'A systematic literature review of software requirements reuse approaches', *Inf. Softw. Technol.*, 2018, **93**, pp. 223–245

Capítulo 3

UM MAPEAMENTO SISTEMÁTICO SOBRE PADRÕES DE REQUISITOS NO CICLO DE VIDA DO SOFTWARE

Artigo publicado na Conferencia Iberoamericana de Software Engineering - CIbSE 2019.

KUDO, T. N.; BULCÃO-NETO, R. F.; MACEDO, A. A.; VINCENZI, A. M. R. Padrão de requisitos no ciclo de vida de software: Um mapeamento sistemático. In: Proceedings of the Conference: Congresso Ibero Americano em Engenharia de Software, CIbSE 2019, p. 420–433, Havana, Cuba. 2019. (*Prêmio de 3º melhor artigo da trilha de Engenharia de Requisitos)*.

Padrão de Requisitos no Ciclo de Vida de Software: Um Mapeamento Sistemático

Taciana N. Kudo[1,3], Renato F. Bulcão-Neto[1], Alessandra A. Macedo[2], and Auri M. R. Vincenzi[3]

[1] Instituto de Informática, Universidade Federal de Goiás, Goiânia-GO, Brasil
{taciana,renato}@inf.ufg.br
[2] Depto. de Computação e Matemática, FFCLRP-USP, Ribeirão Preto-SP, Brasil
ale.alaniz@usp.br
[3] Depto. de Computação, UFSCAR, São Carlos-SP, Brasil
auri@dc.ufscar.br

Resumo Pesquisas demonstram que a prática de reúso por meio de padrões de requisitos é uma alternativa eficaz para tratar problemas de qualidade de especificação, com o benefício adicional de economia de tempo. Acredita-se que, em função das interações existentes entre a engenharia de requisitos e demais fases do ciclo de vida de software, esses benefícios se estenderão para todo o processo de desenvolvimento. Este artigo descreve um estudo secundário que identifica e analisa pesquisas que evidenciam tais benefícios com o uso de padrões de requisitos nas fases de projeto, construção, teste e manutenção. Realizou-se um mapeamento sistemático, cujo protocolo inclui busca automática em 4 bases de dados e a aplicação de critérios de inclusão e exclusão sobre 218 estudos, dos quais selecionaram-se 9 estudos primários para análise e síntese. Identificou-se uma carência de estudos relacionados ao objeto desta pesquisa, sendo a maioria relacionada a projeto de software, e nenhum a manutenção. Além disso, apenas um dos estudos analisados caracteriza-se como pesquisa de validação com estudo de caso, enquanto que os demais são propostas de solução, sem avaliação empírica, ou prática. Portanto, existe um campo aberto para pesquisas que explorem e comprovem, com avaliações empíricas e uso na prática, os benefícios de reúso com padrões de requisitos no ciclo de vida de software.

Keywords: Padrão de requisito · ciclo de vida de software · mapeamento sistemático

1 Introdução

Embora seja consenso que a Engenharia de Requisitos é uma atividade de extrema importância, na qual as funcionalidades e restrições do software devem ser bem identificadas e compreendidas, uma alta porcentagem de projetos de software não cumpre prazos e orçamento planejados, devido à documentação incompleta, com requisitos mal interpretados, conflitantes ou omitidos [21,17].

O reúso por meio de padrão de requisitos é uma abordagem que tem sido bastante explorada com o objetivo de melhorar a qualidade da documentação de requisitos de software [7,17]. Um padrão de requisito é uma abstração que reúne comportamentos e serviços de aplicações com características semelhantes, os quais podem ser replicados em documentações futuras. Um padrão de requisito serve também como um modelo para especificação de um novo requisito [24].

Por exemplo, para escrever um requisito de autenticação de usuário, é possível utilizar um padrão de requisito para esse fim, fazendo as devidas adaptações no requisito, se necessário. Na literatura são encontradas várias propostas de padrões de requisitos, por exemplo, para sistemas embarcados [14], sistemas gerenciadores de conteúdo [18] e sistemas de computação em nuvem [3]. Dentre os benefícios obtidos com a adoção de padrões de requisitos, têm-se: maior eficiência na elicitação, pois os requisitos não são identificados do zero; melhoria na qualidade e consistência no documento de especificação; e melhoria no gerenciamento dos requisitos [24].

Devido à interação inerente entre a engenharia de requisitos e as demais fases do ciclo de vida de software, pressupõe-se que os benefícios advindos do uso de padrões de requisitos podem alcançar as outras atividades de desenvolvimento. Embora existam estudos secundários sobre engenharia de software [19], engenharia de requisitos [15] e padrões de requisitos [7,2], não existem evidências em estudos dessa natureza que analisem o uso de padrões de requisitos nas demais fases do processo de desenvolvimento de software. Em suma, os estudos secundários existentes restringem-se a analisar a adoção de padrões de requisitos particularmente na fase de engenharia de requisitos.

Sendo assim, o objetivo deste artigo consiste em apresentar um estudo secundário na forma de um mapeamento sistemático, que identifica e analisa estudos primários que evidenciam o uso de padrões de requisitos nas fases de projeto, construção, teste e manutenção de software[4].

A metodologia definida apóia-se em trabalhos clássicos de estudos secundários em Engenharia de Software [12,19,8] com suporte da ferramenta *StArt*[5] para o planejamento e a condução do protocolo do mapeamento sistemático. Buscas automáticas foram realizadas em 4 bases de dados [9] e critérios de inclusão e exclusão foram aplicados para identificar os estudos relevantes.

Como resultado da aplicação do protocolo, foram identificados e analisados 9 estudos relevantes, classificados segundo o tipo de pesquisa realizado e o tipo de contribuição do padrão de requisito nas fases citadas, resultando no mapeamento de lacunas relacionadas a padrões de requisitos no ciclo de vida de software.

Este artigo está organizado como segue: a Seção 2 detalha o protocolo do mapeamento sistemático; a Seção 3 reporta a extração de dados dos estudos relevantes. As respostas às questões de pesquisa deste estudo e as lacunas de pesquisa estão sintetizadas na Seção 4. Por fim, a Seção 5 expõe as ameaças à validade deste estudo sistemático, bem como nossas considerações finais.

[4] Neste artigo, adotar-se-á a terminologia das fases do ciclo de vida de software proposta no SWEBOK (*Software Engineering Body of Knowledge*) [4].

[5] Disponível para download em `http://lapes.dc.ufscar.br/tools/start_tool`.

2 Protocolo do Mapeamento Sistemático

Um processo de estudo sistemático pode ser dividido em três fases principais [9]: planejamento, condução e publicação dos resultados. No planejamento, define-se um protocolo de modo que o estudo possa ser replicado futuramente. São integrantes desse protocolo: questões de pesquisa, estratégia de busca, *string* de busca, fontes de pesquisa e critérios de inclusão e exclusão dos estudos.

Na fase de condução, faz-se a identificação e seleção dos estudos relevantes, segundo os critérios definidos no protocolo. Desses estudos são extraídos dados, a partir dos quais faz-se uma síntese para responder às questões de pesquisa do protocolo. O mapeamento sistemático aqui apresentado seguiu essas fases.

2.1 Questões de Pesquisa e Palavras-chave

O principal objetivo deste mapeamento sistemático é identificar propostas de uso de padrões de requisitos e o seu impacto nas demais fases do ciclo de vida de software, além da engenharia de requisitos. Para atender a esse objetivo, as questões de pesquisa (**QP**) que se deseja responder são:

QP1. Em quais fases do ciclo de vida de software são usados padrões de requisitos: projeto, construção, teste e/ou manutenção?
QP2. Há evidências de uso prático de padrões de requisitos nessas fases do ciclo de vida de software?
QP3. Há benefícios relatados do uso de padrão de requisitos nessas fases? Se sim, quais métricas são utilizadas para medir esses benefícios?

Para responder adequadamente às questões de pesquisa, é necessário seguir uma estratégia de busca apropriada [9]. Para fundamentar a definição de termos padronizados da Engenharia de Software, os termos de pesquisa foram definido de acordo com o SEVOCAB[6]. Como resultado, chegou-se ao seguinte conjunto de palavras-chave em inglês: *requirement pattern, development process, software development, life cycle, design, construction, coding, implementation, test, integration* e *maintenance*.

2.2 Estratégia de Busca

Com base nas palavras-chave, e em busca de equilíbrio entre abrangência[7] e relevância dos resultados, foi definida a seguinte *string* de busca:

(“*requirement pattern*” OR “*requirement patterns*” OR
“*requirements pattern*” OR “*requirements patterns*”)
AND ((“*software development*” OR “*development process*”) OR
(“*life cycle*” OR *design* OR *construction* OR *coding* OR
implementation OR *test* OR *integration* OR *maintenance*))

[6] *Software and Systems Engineering Vocabulary* é uma iniciativa da ISO/IEEE de padronizar os termos utilizados na área de Engenharia de Software [11]

[7] Variações em inglês do termo “padrão de requisito” foram usadas em função das capacidades das máquinas de busca das fontes de pesquisa escolhidas.

Foram escolhidas 4 fontes de pesquisa relevantes (*ACM DL*, *Engineering Village*, *IEEE Xplorer* e *Scopus*) sobre as quais foram realizadas buscas sobre os metadados dos estudos. Como resultado, foram identificados 218 estudos[8], assim distribuídos: *ACM Digital Library* (24), *Engineering Village* (100), *IEEE Xplorer* (23) e *Scopus* (71). No caso da *ACM DL*, a busca foi expandida para *The ACM Guide to Computing Literature* por incluir uma coleção maior de artigos.

2.3 Identificação e Seleção dos Estudos Primários

Esta seção descreve o método utilizado para selecionar, dentre os 218 estudos identificados, quais são relevantes para responder às questões de pesquisa deste mapeamento sistemático. Um conjunto de 7 (sete) critérios de exclusão foi definido e aplicado a esses estudos. Os critérios de exclusão (CE) são:

CE1 - Não é um estudo primário.
CE2 - Não é um artigo (p.ex. prefácio/sumário de periódicos/anais de eventos).
CE3 - Não é relacionado com padrões de requisitos de software.
CE4 - Aborda padrões de requisitos apenas na fase de engenharia de requisitos.
CE5 - O texto completo do estudo não está em inglês.
CE6 - Não foi possível acessar o texto completo do estudo.
CE7 - É uma versão preliminar ou reduzida de outro estudo (não adiciona informação relevante).

Quando um estudo se encaixa em pelo menos um critério de exclusão, ele está automaticamente excluído do mapeamento. Caso contrário, o estudo deverá ser categorizado em um critério de inclusão (CI):

CI1 - Aborda padrões de requisitos na fase de projeto de software.
CI2 - Aborda padrões de requisitos na fase de construção de software.
CI3 - Aborda padrões de requisitos na fase de teste de software.
CI4 - Aborda padrões de requisitos na fase de manutenção de software.

O processo completo de seleção dos estudos primários pode ser visualizado na Figura 1, na qual são descritas também as quantidades de estudos selecionados em cada uma das atividades da fase de condução deste mapeamento sistemático.

Após a atividade de busca sobre as fontes de pesquisa, foi realizada a identificação e eliminação de 111 estudos duplicados. Em seguida, iniciou-se a leitura do título, resumo e palavras-chave de cada um dos 117 estudos restantes, sobre os quais foram aplicados os critérios de exclusão e inclusão. Como resultado, foram selecionados 39 estudos potencialmente relevantes, pois esta seleção baseou-se apenas na leitura e interpretação do conteúdo dos respectivos metadados.

Na atividade de extração dos dados, descrita na próxima seção, foi realizada a leitura na íntegra de cada um desses 39 estudos e constatou-se que apenas 9 deles realmente investigam o uso de padrões de requisitos em projeto, construção, teste ou manutenção de software.

[8] A atividade de busca foi realizada no período de 24 de abril a 5 de maio de 2018.

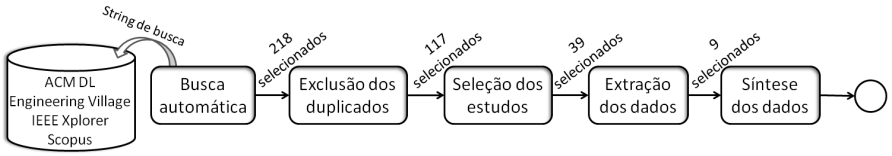

Figura 1. Atividades da fase de condução do Mapeamento Sistemático.

A lista a seguir consiste dos 9 estudos relevantes para extração e síntese (vide Seções 3 e 4), segundo o protocolo definido para este mapeamento sistemático. Esses estudos serão identificados ao longo deste artigo como A1 a A9:

A1 - Adaptive requirement-driven architecture for integrated healthcare systems [25]
A2 - Analysing security requirements patterns based on problems decomposition and composition [23]
A3 - An architectural framework of the integrated transportation information service system [5]
A4 - Application of ontologies in identifying requirements patterns in use cases [6]
A5 - Effective security impact analysis with patterns for software enhancement [16]
A6 - From requirement to design patterns for ubiquitous computing applications [13]
A7 - Modeling design patterns with description logics: A case study [1]
A8 - Mutation patterns for temporal requirements of reactive systems [22]
A9 - SACS: A pattern language for Safe Adaptive Control Software [10]

Quanto à aplicação dos critérios de exclusão, dos 117 artigos não-duplicados, 108 foram excluídos da seguinte maneira: 48% dos artigos (52) foram excluídos por abordarem padrões de requisitos apenas na engenharia de requisitos (**CE4**); 31,5% deles (34) por tratarem de padrões de requisitos, porém não relacionados a software (**CE3**); e os 20,5% restantes (22) pelos demais critérios.

3 Extração dos Dados

Esta seção descreve o processo de extração de dados baseado na leitura completa dos 9 artigos relevantes (A1 a A9) deste estudo. Apresenta-se uma análise comparativa dos tipos de contribuição, bem como o seguinte conteúdo extraído de cada um desses artigos:

1. o tipo de pesquisa apresentado no estudo;
2. o tipo de requisito para o qual o padrão de requisito é proposto;
3. a fase do ciclo de vida a qual o padrão de requisito está relacionado; e
4. o tipo de contribuição apresentada no estudo.

Dado o primeiro critério, os 9 estudos foram classificados pelo tipo de pesquisa desenvolvida, usando para este critério a definição de Petersen et al. [20], em que um conjunto de condições atendidas por uma pesquisa conduzem a uma classificação da mesma. Por exemplo, um artigo de opinião reporta o ponto

de vista do autor sobre um assunto, porém sem nenhuma descrição de uso na prática, nem avaliação empírica, relato de experiência do autor, ou proposta de arcabouço conceitual ou de uma nova solução.

Conforme apresenta a Tabela 1, a maioria dos artigos (8 de 9) enquadra-se, pelos critérios de Petersen et al. [20], como "proposta de solução", pois não apresentam avaliação empírica da solução proposta em suas pesquisas. Dessas 8 propostas de solução, 3 apresentam uma prova de conceito informal como validação, sem descrever um estudo de caso, nem um experimento; os outros 5 não validam a solução proposta. Por fim, dentre os 9 artigos relevantes, apenas um foi classificado como "pesquisa de validação" (A7), por apresentar um estudo de caso, embora realizado em ambiente acadêmico.

Tabela 1. Tipos de pesquisa e validação dos artigos relevantes.

Tipo da pesquisa	**Tipo de Validação**
Proposta de solução	Prova de conceito informal: A2, A5, A9
	Sem validação: A1, A3, A4, A6, A8
Pesquisa de validação	Estudo de caso: A7

Em seguida, foi analisado se havia algum tipo específico de requisito tratado pelos padrões de requisitos propostos. Dos 9 artigos relevantes, 4 deles definem padrão para requisitos de adaptabilidade, 4 para requisitos de segurança e 1 define padrão para requisitos de propósito geral. A Tabela 2 mostra os artigos de acordo com os tipos de requisitos.

Tabela 2. Tipo de requisito considerado no padrão proposto.

Tipo do requisito	**Artigos**
Requisito de adaptabilidade	A1, A3, A6, A8
Requisito de segurança	A2, A5, A7, A9
Requisito de propósito geral	A4

Foi realizada uma análise comparativa detalhada entre as contribuições propostas em cada um dos trabalhos, a partir da qual pode-se notar algumas similaridades apresentadas a seguir.

Os artigos A1 e A3 propõem uma *arquitetura conceitual* semelhante para sistemas desenvolvidos a partir de padrões de requisitos. A comparação realizada entre essas propostas está representada na Figura 2: as linhas tracejadas A, B, C e D demonstram os pontos em comum entre as arquiteturas propostas por A1 (à esquerda) e A3 (à direita). Nas duas arquiteturas, a camada de requisitos (A) identifica, analisa e modela os possíveis requisitos como padrões de requisitos de

usuário (URP). A camada de serviço (B) interage com a camada de requisitos e fornece serviços para satisfazer o URP. O mecanismo de segurança e compartilhamento de informação (C) garante um processo de troca de informação confiável entre os sistemas do mesmo domínio. A base de conhecimento (D) agrupa padrões, normas e ontologias do domínio do sistema. A motivação de ambas as pesquisas é a necessidade de compartilhamento de informações entre sistemas de um mesmo domínio: A1, para sistemas da área médica, e A3, para sistemas de transporte.

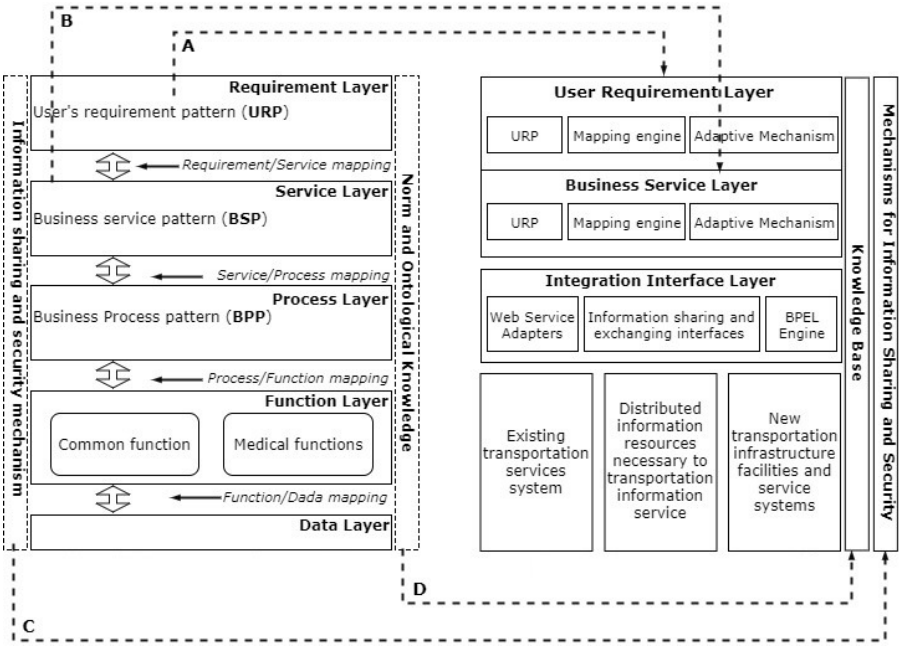

Figura 2. Análise comparativa entre as arquiteturas baseadas em padrões de requisitos propostas em A1 (à esquerda) e A3 (à direita).

Com relação ao objeto de estudo deste mapeamento sistemático, observa-se que os trabalhos apresentados em A1 e A3 estão diretamente relacionados à fase de *projeto de software*, pois os padrões de requisitos do usuário (URP) da camada de requisitos servem de base para a seleção eficiente dos serviços (BSP) na camada de serviços. Os URPs são os principais elementos na camada de requisitos que correspondem aos requisitos do usuário e orientam o funcionamento de todo o sistema.

Semelhanças também foram identificadas nos artigos A2 e A5 sobre como *representar requisitos de segurança na forma de padrão de requisito*. Foi possível observar que ambas as pesquisas tratam os mesmos conceitos de segurança (i.e.

contexto, ativos e ameaças) como padrões de requisitos, e as medidas de proteção são tratadas nos padrões de projeto. As semelhanças identificadas entre os estudos de A2 (à esquerda) e A5 (à direita) estão ilustradas na Figura 3 como linhas tracejadas. Os passos descritos na abordagem do estudo A2 — a identificação de interessados e objetivos, de ativos essenciais de informação, e de fontes de ameaças usando padrões — correspondem, respectivamente, aos seguintes itens do padrão de requisito de segurança do estudo A5: a definição do formato do padrão (contexto, problema, solução e estrutura), dos ativos, e das ameaças. A atividade "adicionar medidas de proteção no projeto do sistema" do estudo A2 é tratada como padrão de projeto de segurança no estudo A5.

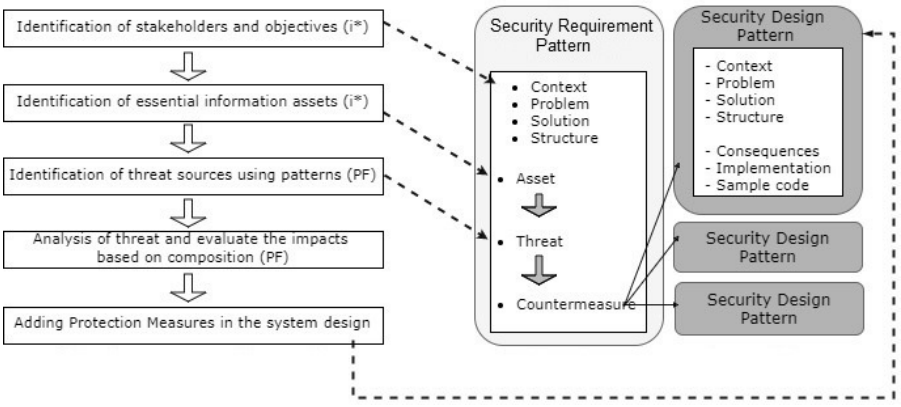

Figura 3. Análise comparativa entre as abordagens de segurança apresentadas em A2 (à esquerda) e A5 (à direita).

Conclui-se que os trabalhos A2 e A5 também estão relacionados com a fase de *projeto de software*, uma vez que ambos definem padrões de requisitos de segurança, de forma a impactar diretamente na definição de medidas de proteção por meio de padrões de projeto.

Foi também identificado que os artigos A8 e A9 apresentam propostas relacionadas ao formato de representação de padrões de requisitos. No artigo A8, associa-se uma fórmula em linguagem temporal linear para cada requisito escrito em linguagem natural, utilizando mutações para amenizar os problemas que podem surgir nessa associação. Para cada tipo de padrão de requisito, as falhas potenciais são identificadas e mutações apropriadas são introduzidas. As fórmulas que acompanham os mutantes podem ser usadas para a geração de testes, a adequação de conjuntos de testes, ou a construção automática de monitores para a verificação em tempo de execução do comportamento do sistema. Sendo assim, as mutações incluídas na transformação dos padrões de requisitos contribuem para a *fase de teste* no ciclo de vida de software. Já no artigo A9 é apresentada uma proposta de padrão de software que integra três tipos de padrão

(de requisito, projeto e segurança), o qual nomeia como padrão composto. Baseado na ideia de *problem frames*, o padrão composto utiliza parâmetros extraídos do seu padrão de requisito. Cada conjunto de funções do padrão de requisito corresponde a soluções no padrão de projeto e a elementos contextuais no padrão de segurança. Portanto, essa proposta está também relacionada à *fase de projeto* de software.

Os artigos A4 e A7 utilizam o formalismo da lógica de descrição (LD) de ontologias para representar padrões de requisitos. No estudo A4, requisitos escritos em LD permitem a geração automática de código fonte; portanto, refere-se à *fase de construção* de software. Na proposta A7, foi implementado um mecanismo que, dado um padrão de requisito de segurança descrito em LD, consegue encontrar uma solução correspondente na forma de padrão de projeto. Daí também relaciona-se à *fase de projeto* de software.

Por fim, o estudo A6 tem como objetivo mapear a dependência entre padrões de projeto e padrões de requisitos de aplicações de computação ubíqua, cuja característica recorrente é a adaptabilidade. Uma abordagem que integra esses padrões tem como objetivo transpor as lacunas da fase inicial do desenvolvimento de software, em que os requisitos recorrentes exigem soluções semelhantes. Em suma, a principal contribuição do artigo A6 é para a fase de *projeto de software*.

As informações extraídas e a análise comparativa realizada na fase de extração desse mapeamento estão sintetizadas na Tabela 3 de acordo com 4 tipos de contribuições identificadas nos artigos: arquitetura conceitual para sistemas que usam padrão de requisito; processo para descoberta e uso de padrão de requisito; definição de formato de representação de padrão de requisito; e proposta de catálogo de padrões de requisitos.

Tabela 3. Informações extraídas dos artigos relevantes.

Tipo de contribuição do estudo	**Fase do ciclo de vida**	**Tipo de requisito**	**Artigos**
Arquitetura conceitual baseada em padrão de requisito	Projeto	Adaptabilidade	A1, A3
Formato de representação de padrão de requisito	Projeto	Segurança	A2, A5, A9
	Teste	Adaptabilidade	A8
Processo para descoberta e uso de padrão de requisito	Projeto	Segurança	A7
	Construção	Propósito Geral	A4
Catálogo de padrão de requisito	Projeto	Adaptabilidade	A6

4 Síntese dos dados

Esta seção apresenta uma síntese dos dados obtidos na fase de extração com o objetivo de responder às questões de pesquisa deste estudo secundário.

4.1 Sobre a Questão de Pesquisa 1

Em resposta à questão de pesquisa "*Em quais fases do ciclo de vida de software são usados padrões de requisitos: projeto, construção, teste e/ou manutenção?*", 7 estudos utilizam padrões de requisitos na fase de projeto de software, 1 na construção, 1 em testes de software e não houve nenhum para manutenção.

Dentre os 7 estudos que abordam padrões de requisitos em projeto de software (A1–A3, A5–A7 e A9), não existem autores de mais de um estudo, logo não se trata de uma iniciativa de um ou mais grupos de pesquisa. Duas hipóteses podem ser pensadas para a alta concentração dos estudos relacionados à fase de projeto: primeiro é o fato desta ser subsequente à engenharia de requisitos no processo de desenvolvimento tradicional, segundo a disseminação crescente do uso de padrões de projeto no desenvolvimento de software.

Além disso, embora tenham sido identificados 9 estudos que relacionam padrões de requisitos a alguma fase do ciclo de vida, 52 artigos foram excluídos na atividade de seleção pelo critério **CE4**, i.e. tratavam de padrão de requisito apenas na engenharia de requisitos. Essa diferença na quantidade de pesquisas relacionadas à engenharia de requisitos e aquelas que expandem padrões de requisitos para as demais fases do ciclo de vida deixa evidente uma oportunidade de pesquisa sobre padrão de requisito nas fases de construção, teste e manutenção.

Outra evidência é a inexistência de pesquisas sobre o uso de padrão de requisito que permeie o processo de desenvolvimento ao longo do ciclo de vida, i.e. partindo da engenharia de requisitos, passando pelo projeto, construção, teste, e sendo refinado na manutenção de software. Um desafio de estudo é pensar em como propor o uso de padrões de requisitos de forma que estes impactem na melhoria do processo de desenvolvimento como um todo, além dos conhecidos benefícios de qualidade de documentação e de economia de tempo.

4.2 Sobre a Questão de Pesquisa 2

Com relação à questão de pesquisa: "*Há evidências de uso prático de padrões de requisitos nessas fases do ciclo de vida de software?*", em nenhum dos artigos foi reportada evidência do uso de padrão de requisitos em ambiente de produção de software. Um total de 8 dos 9 artigos encontra-se no estágio de propostas de solução, sem validação, e apenas 1 artigo (A7) é validado com estudo de caso. Esta análise sugere que futuros trabalhos no tema em questão atenham-se cada vez mais à aplicação prática de padrão de requisito na indústria de software em todo o ciclo de desenvolvimento.

4.3 Sobre a Questão de Pesquisa 3

Referente à questão de pesquisa "*Há benefícios relatados do uso de padrão de requisitos nessas fases? Se sim, quais métricas são utilizadas para medir esses benefícios?*", identificou-se que nenhum dos artigos relatou explicitamente qualquer benefício oriundo da prática de padrão de requisito.

Essa ausência de preocupação com métricas para analisar os benefícios do uso de padrões de requisitos se deve ao fato da maioria dos artigos estar no nível de propostas, sem uso prático dos mesmos.

4.4 Discussão

Ao interligar as informações da fase do ciclo de vida, do tipo de padrão de requisito proposto e o tipo de pesquisa de cada artigo, foi possível elaborar o gráfico de bolhas da Figura 4. Ao analisar o gráfico, observa-se que 4 estudos (A2, A5, A7 e A9) propõem padrão de requisito de *segurança* com uso na fase de *projeto*. Supomos que usar padrão de requisito para segurança deve-se ao fato desta característica ser recorrente em muitos sistemas, contando também com o apoio de padrões internacionais, como a família de padrões ISO/IEC 27000. Entretanto, esses estudos com segurança ainda carecem de maior validação com avaliações empíricas e uso na indústria de software.

Dos 9 estudos relevantes, 4 estudos (A1, A3, A6 e A8) utilizam padrão de requisito para *adaptabilidade*, sendo o último na fase de *testes* e os demais na fase de *projeto*. Além disso, nenhum desses 4 estudos apresenta validação de seus padrões, i.e., caracterizam-se como pesquisas de proposta conceitual de solução. Por fim, o estudo A4 é voltado para padrão de requisito de *propósito geral* com uso na fase de *construção* de software, mas também não apresentou validação.

Considerando ainda a Figura 4, tão importante quanto mapear as propostas de pesquisa é analisar as lacunas existentes:

1. existe uma carência geral de pesquisas de adoção de padrão de requisito em outras fases do ciclo de vida (9 estudos), excetuando a engenharia de requisitos (52 estudos).
2. ao analisar o lado esquerdo da figura, conclui-se que requisitos não funcionais (adaptabilidade e segurança) são os mais estudados quando do uso de padrão de requisito para as fases de projeto e testes. Porém, outros tipos de requisitos não funcionais podem ser explorados com padrão de requisito envolvendo outras fases do ciclo de vida como, por exemplo, aspectos de usabilidade com a geração automática de código e de casos de teste.
3. ao considerar o lado direito da figura, identificamos uma lacuna quanto à aplicação, na indústria de software, dos resultados das pesquisas de padrões de requisitos nas fases de projeto, construção, testes e manutenção — quase todas as propostas encontram-se em estágio de prova de conceito.

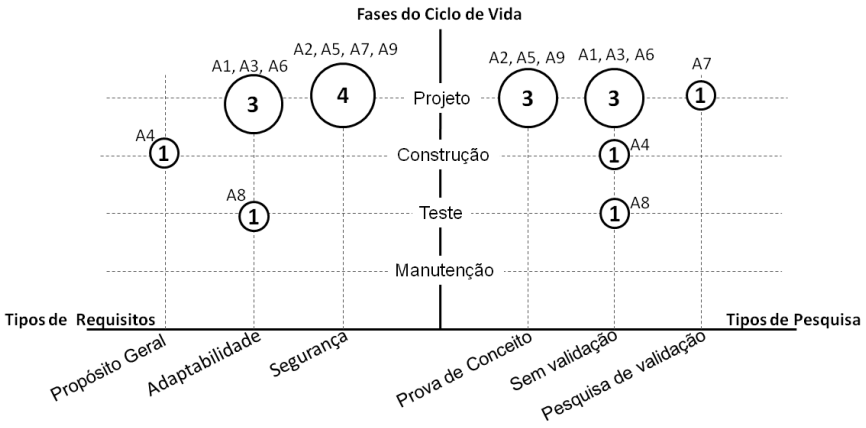

Figura 4. Mapeamento dos tipos de requisitos e de pesquisas sobre padrões de requisitos mapeados nas fases do ciclo de vida de software.

5 Considerações Finais

Os maiores problemas enfrentados em estudos sistemáticos são encontrar todos os estudos relevantes sobre um tema e, a partir dos estudos relevantes, selecionar evidências de qualidade. Por isso, ameaças foram mitigadas com 2 ações ao longo do planejamento e da condução deste mapeamento sistemático.

Primeiro, adotou-se uma estratégia de busca automática que combina fontes de informação consideradas relevantes para a Engenharia de Software com termos da *string* de busca baseados no vocabulário padrão SEVOCAB. Busca na literatura cinzenta não fez parte do protocolo, mais especificamente teses e relatórios técnicos, pois supõe-se que esse tipo de literatura, quando de boa qualidade, está publicada como artigo em periódicos ou conferências.

Segundo, três pesquisadores elaboraram o protocolo deste mapeamento sistemático e na condução do mapeamento, cada pesquisador exerceu seu papel: o pesquisador A, especialista em Engenharia de Requisitos, realizou a identificação e a extração de dados dos estudos relevantes; o pesquisador B, especialista em Engenharia de Software, verificou os resultados da fase de extração para mitigar a possibilidade de vieses ao longo do processo; e o pesquisador C, líder da equipe e com vasta experiência em Engenharia de Software, juntamente com A e B, realizou a análise, síntese e escrita dos resultados. No caso de divergências, A, B e C resolveram os conflitos em conjunto.

Apesar dos resultados obtidos neste mapeamento sistemático, de poucos estudos sobre o tema e com baixo índice de validação das propostas, os autores reforçam a reconhecida importância de padrões de requisitos no meio acadêmico [17,2]. Para impulsionar o desenvolvimento das pesquisas em padrões de requisitos no processo de desenvolvimento de software, sugerimos que a comunidade acadêmica aproxime-se da indústria de software para identificar as

reais expectativas do setor. A comunidade deve também estabelecer métricas que evidenciem as vantagens do uso de padrões de requisitos em todas as fases do ciclo de vida como, por exemplo, a redução no tempo de projeto e na geração automática de código-fonte, a execução padronizada de testes e a melhoria na qualidade das especificações de software. Complementar a essas ações, a comunidade deve propor novas ferramentas de suporte ao desenvolvimento de aplicações com o uso de padrões de requisitos. Por fim, os pesquisadores devem elaborar metodologias de desenvolvimento que demonstrem como usar padrões de requisitos ao longo das fases do ciclo de vida de forma integrada às demais ações propostas.

Agradecimentos

Taciana Kudo agradece o auxílio financeiro concedido pelo Programa de Apoio à Pós-graduação (PROAP/CAPES). Renato Bulcão-Neto agradece a bolsa concedida pela CAPES/FAPEG (proc. n. 88887.305511/2018-00), vinculada ao estágio pós-doutoral realizado no Depto. de Computação e Matemática da FFCLRP-USP. Alessandra Macedo agradece o apoio financeiro da FAPESP (proc. n. 16/13206-4) e do CNPq (proc. n. 302031/2016-2 e 442533/2016-0).

Referências

1. Asnar, Y., Paja, E., Mylopoulos, J.: Modeling design patterns with description logics: A case study. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 6741 LNCS, pp. 169 – 183. London, United kingdom (2011)
2. Barros-Justo, J.L., Benitti, F.B.V., Leal, A.C.: Software patterns and requirements engineering activities in real-world settings:a systematic mapping study. Comp. Standards & Interfaces **58**, 23–42 (2018)
3. Beckers, K., Côté, I., Goeke, L.: 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 (2014)
4. Bourque, P., Fairley, R.E. (eds.): SWEBOK: Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, Los Alamitos, CA, version 3.0 edn. (2014)
5. Chang, F., Gan, R.: An architectural framework of the integrated transportation information service system. In: 2009 IEEE International Conference on Grey Systems and Intelligent Services, GSIS 2009. pp. 1342 – 1346. Nanjing, China (2009)
6. Couto, R., Ribeiro, A.N., Campos, J.C.: Application of ontologies in identifying requirements patterns in use cases. In: Electronic Proceedings in Theoretical Computer Science, EPTCS. vol. 147, pp. 62 – 76. Grenoble, France (2014)
7. Da Silva, R., Benitti, F.: Standards writing requirements: A mapping systematic literature [Padrões de escrita de requisitos: Um mapeamento sistemático da literatura]. In: 14th Ibero-American Conference on Software Engineering and 14th Workshop on Requirements Engineering, CIbSE 2011. pp. 259–270 (2011)
8. Fabbri, S.C.P.F., Felizardo, K.R., Ferrari, F.C., Hernandes, E.C.M., Octaviano, F.R., Nakagawa, E.Y., Maldonado, J.C.: Externalising tacit knowledge of the systematic review process. IET Software **7**(6), 298–307 (2013)

9. Felizardo, K.R., Nakagawa, E.Y., Fabbri, S.C.P.F., Ferrari, F.C.: Revisão sistemática da literatura em Engenharia de Software: Teoria e Prática. Elsevier, first edn. (2017)
10. Hauge, A.A., Stølen, K.: SACS: A pattern language for safe adaptive control software. In: Proceedings of the 18th Conference on Pattern Languages of Programs. pp. 7:1–7:22. PLoP '11, ACM, New York, NY, USA (2011)
11. ISO/IEC/IEEE: IEEE software and systems engineering vocabulary. IEEE Computer Society pp. 1 – 437 (2016)
12. Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software engineering - a systematic literature review. Inf. Softw. Technol. **51**(1), 7–15 (Jan 2009)
13. Knote, R., Baraki, H., Söllner, M., Geihs, K., Leimeister, J.M.: From requirement to design patterns for ubiquitous computing applications. In: Proceedings of the 21st European Conference on Pattern Languages of Programs (Jul 2016)
14. Konrad, S., Cheng, B.H.C.: Requirements patterns for embedded systems. In: Proceedings IEEE Joint International Conference on Requirements Engineering. pp. 127–136 (2002)
15. Nicolas, J., Toval, A.: On the generation of requirements specifications from software engineering models: A systematic literature review. Inf. Softw. Technol. **51**(9), 1291–1307 (Sep 2009)
16. Okubo, T., Kaiya, H., Yoshioka, N.: Effective security impact analysis with patterns for software enhancement. In: 2011 Sixth International Conference on Availability, Reliability and Security. pp. 527–534 (Aug 2011)
17. Palomares, C., Quer, C., Franch, X.: Requirements reuse and requirement patterns: a state of the practice survey. Empirical Software Engineering **22**(6), 2719–2762 (2017)
18. Palomares, C., Quer, C., Franch, X., Renault, S., Guerlain, C.: 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, Coimbra, Portugal, March 18-22, 2013. pp. 1260–1265 (2013)
19. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. pp. 68–77. EASE'08, BCS Learning & Development Ltd., Swindon, UK (2008)
20. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology **64**, 1–18 (2015)
21. Tockey, S.: Insanity, hiring, and the software industry. Computer **48**(11), 96–101 (Nov 2015)
22. Trakhtenbrot, M.: Mutation patterns for temporal requirements of reactive systems. In: Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2017. pp. 116–121 (2017)
23. Wen, Y., Zhao, H., Liu, L.: Analysing security requirements patterns based on problems decomposition and composition. In: 2011 1st International Workshop on Requirements Patterns, RePa'11. pp. 11 – 20. Trento, Italy (2011)
24. Withall, S.: Software Requirement Patterns. Best practices, Microsoft Press, Redmond, Washington (2007)
25. Yang, H., Liu, K., Li, W.: Adaptive requirement-driven architecture for integrated healthcare systems. Journal of Computers **5**(2) (2010)

Capítulo 4

UMA EXTENSÃO DO MAPEAMENTO SISTEMÁTICO SOBRE PADRÕES DE REQUISITOS NO CICLO DE VIDA DO SOFTWARE

Artigo publicado no Journal of Software Engineering Research and Development.

KUDO, T. N.; BULCÃO-NETO, R. F.; MACEDO, A. A.; VINCENZI, A. M. R. A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle. Journal of Software Engineering Research and Development, v. 7, p. 9:1-9:11, 2019.

Journal of Software Engineering Research and Development, 2019, 6:1, doi: 10.5753/jserd.2019.xxx

A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle

Taciana N. Kudo [**DC-UFSCar, São Carlos-SP, Brazil** | ***taciana@dc.ufscar.br***]
Renato F. Bulcão-Neto [**INF-UFG, Goiânia-GO, Brazil** | ***rbulcao@ufg.br***]
Alessandra A. Macedo [**FFCLRP-USP, Ribeirão Preto-SP, Brazil** | ***ale.alaniz@usp.br***]
Auri M. R. Vincenzi [**DC-UFSCar, São Carlos-SP, Brazil** | ***auri@dc.ufscar.br***]

Abstract In the past few years, the literature has shown that the practice of reuse through requirement patterns is an effective alternative to address specification quality issues, with the additional benefit of time savings. Due to the interactions between requirements engineering and other phases of the software development life cycle (SDLC), these benefits may extend to the entire development process. This paper describes a revisited systematic literature mapping (SLM) that identifies and analyzes research that demonstrates those benefits from the use of requirement patterns for software design, construction, testing, and maintenance. In this extended version, the SLM protocol includes automatic search over two additional sources of information and the application of the snowballing technique, resulting in ten primary studies for analysis and synthesis. In spite of this new version of the SLM protocol, results still point out a small number of studies on requirement patterns at the SDLC (excluding requirements engineering). Results indicate that there is yet an open field for research that demonstrates, through empirical evaluation and usage in practice, the pertinence of requirement patterns at software design, construction, testing, and maintenance.

Keywords: *Requirement pattern, Software development life cycle, Systematic literature mapping*

1 Introduction

Requirements engineering is a critical development phase in which software functionalities and constraints must be well identified and understood. However, a high percentage of software projects do not meet deadlines and budget due to incomplete, misinterpreted, conflicting, or omitted requirements (Tockey, 2015; Palomares et al., 2017).

To deal with this issue of quality of requirements specifications, software requirement patterns (SRP) have been given special attention in the recent years (Palomares et al., 2017; Irshad et al., 2018). An SRP is an abstraction that groups both behaviors and services of applications with similar characteristics. It works as a template for new requirements specification, and it can also be replicated in future requirements documentation (Withall, 2007). For instance, to write a user authentication functional requirement, one can use an SRP for this purpose and make appropriate adaptations to the requirement, if necessary.

Several proposals for SRPs are found in the literature such as for embedded (Konrad and Cheng, 2002), content management (Palomares et al., 2013), and cloud computing systems (Beckers et al., 2014). Among the benefits obtained with the adoption of SRPs are: (i) greater efficiency in requirements elicitation since these are not identified from scratch; (ii) quality and consistency improvement in the requirements specification document; and (iii) improved requirements management (Withall, 2007).

Because of the inherent interaction between requirements engineering and other phases of the software development life cycle (SDLC), it is assumed that the benefits of using SRPs can reach other development activities. Although there are secondary studies on software engineering (Kitchenham and Brereton, 2013), requirements engineering (Curcio et al., 2018), and requirement patterns (Barros-Justo et al., 2018), there is no evidence of secondary studies that analyze the use of SRPs at other SDLC phases. In short, existing secondary studies are restricted to analyzing the adoption of SRPs exclusively in the requirements engineering phase.

In recent work, we performed a systematic literature mapping (SLM) that identifies and analyses primary studies that put in evidence the usage of SRPs at the software design, construction, testing, and maintenance phases (Kudo et al., 2019)[1]. The underlying protocol included automatic search over four sources of information, and the definition and application of inclusion and exclusion criteria over 117 non-duplicate studies found. Only nine primary studies were considered relevant, given the research aim (Kudo et al., 2019).

Main results indicated that most of the relevant studies apply SRPs in software design, but none in software maintenance. Moreover, only one study was featured as validation research, while the remaining studies were solution proposals. From these results, we concluded that the benefits from the SRPs usage in practice at other SDLC phases are still in its early stages.

In this paper, we revisit the SLM described in Kudo et al. (2019) and improve the identification and selection methods of primary studies. Besides the inclusion of two additional sources of information in the automatic search process, we also perform the snowballing technique (Wohlin, 2014) that identifies relevant studies through the scanning of the list of bibliographic references or citations of a paper.

The inclusion of two sources of studies resulted in 32 extra, non-duplicate papers, from which one novel relevant study arose. Considering the 9 relevant primary studies found in our previous work, we obtained a ten-primary-study group in this research. To check whether other essential studies on

[1] We adopt the terminology of the Software Engineering Body of Knowledge (SWEBOK) for the SDLC phases (Bourque and Fairley, 2014).

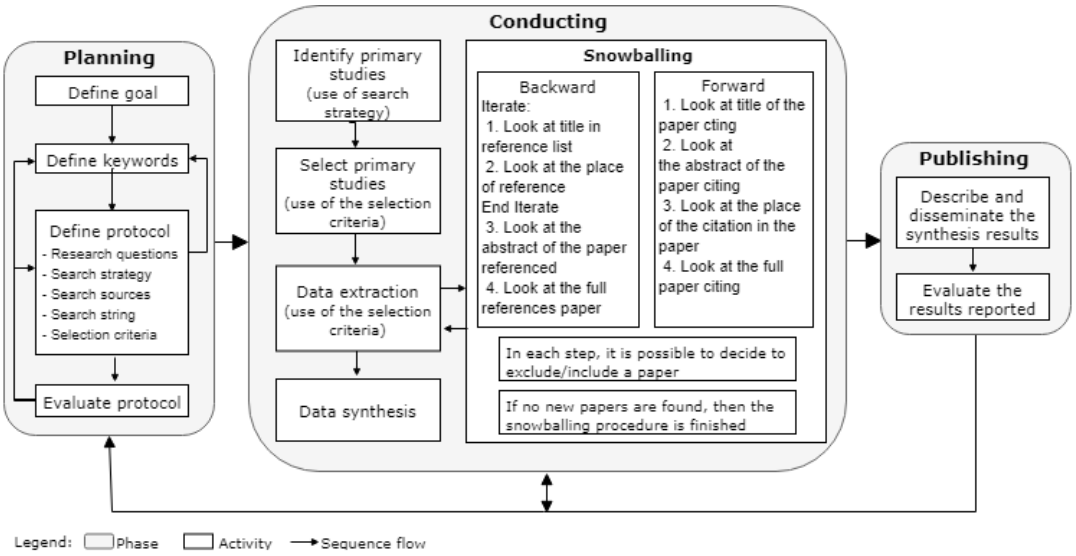

Figure 1. Phases and activities of this SLM, adapted from (Fabbri et al., 2013; Wohlin, 2014)

this research exist, we also analyzed the list of bibliographic references as well as the citing papers of each one of these 10 studies. The snowballing technique resulted in 202 non-duplicate papers from which none was assessed as relevant after the re-application of inclusion and exclusion criteria.

We read the full text of 10 studies towards the extraction of answers to the SLM research questions. Finally, we synthesized in a bubble graph a map with the remarkable characteristics of this ten-study-group. In comparison with our previous work, results continue to point out a lack of research on SRPs for the software design, construction, testing, and maintenance phases.

The organization of this paper is as follows. Section 2 details the protocol of this SLM. Section 3 reports the extraction of data from the relevant studies. The answers to the research questions in this study and the research gaps are summarized in Section 4. Finally, Section 5 describes the validation threats of this SLM, whereas Section 6 presents our final remarks.

2 The Systematic Mapping Protocol

In general, a systematic study process can be divided into three distinct phases (Fabbri et al., 2013): planning, conduction, and publishing of results. First, a protocol is planned in such a way one can reproduce it later. This systematic mapping protocol includes the definition of the main goal, research questions, search strategy, search string, sources of studies, and inclusion and exclusion criteria.

In the conduction phase, studies gathered from search engines and bibliographic databases are identified and selected using the inclusion and exclusion criteria previously defined. A set of useful information is extracted from these selected studies that, in turn, can be still excluded from the SLM. The snowballing approach is performed over these included papers by firstly checking their references list. Decisions about inclusion and exclusion of the studies from this backward analysis are also based on the previous reading of the paper's title and abstract. This same process is also carried out with the citation list of the same papers examined in the data extraction step. Forward and backward analyses finish when no new study is included. Following the SLM goal, the studies remaining constitute the set of relevant papers from which answers for the research questions of the protocol are analyzed and synthesized.

In the publishing phase, the entire protocol and the results of each previous stage are documented as scientific papers or technical reports. The SLM presented in this paper is an extension of the Kudo et al. (2019)'s work and follows those three phases, as depicted in Figure 1.

2.1 Research questions and keywords

The main goal of this SLM is to identify studies that explore the benefits of requirement patterns for every SDLC phase, except for the requirements engineering process. Based on this goal, the set of research questions (**RQ**) that the SLM should answer and the respect justifications are presented next:

RQ1. At what SDLC phases are requirement patterns used: design, construction, testing and/or maintenance?

This question is essential to find out if there is research on requirement patterns covering other SDLC phases, beyond requirements engineering.

RQ2. Is there evidence of requirement patterns usage in practice at those SDLC phases?

This question is relevant to discover empirical evidence on requirement patterns usage at other SDLC phases, i.e., not only solution proposals.

RQ3. Are there reported benefits of using requirement patterns at those phases? If so, what metrics are used to measure these benefits?

This question is also important to find out if the benefits of requirement patterns (e.g., development time savings, better quality specifications, etc.) have been exploited at other

SDLC phases. If so, we want to know how these benefits have been measured.

To support the definition of standardized terms in Software Engineering, the search terms are borrowed from the SEVOCAB (Software and Systems Engineering Vocabulary), which is an ISO/IEEE initiative to standardize the terms used in Software Engineering (ISO/IEC/IEEE, 2017). The following is the set of keywords used for the definition of the search string: *requirement pattern*, *development process*, *software development*, *life cycle*, *design*, *construction*, *coding*, *implementation*, *test*, *integration*, and *maintenance*.

A search strategy should find relevant studies to answer the research questions. Next, we present the search strategy performed in this SLM that includes automatic search and the snowballing technique.

2.2 Automatic search

After evaluating the trade-off between coverage and relevance of the search results in a pilot search, we opted for the following combination of keywords[2] as search string:

("*requirement pattern*" OR "*requirement patterns*" OR "*requirements pattern*" OR "*requirements patterns*")
AND (("*software development*" OR "*development process*")
OR
("*life cycle*" OR *design* OR *construction* OR *coding* OR *implementation* OR *test* OR *integration* OR *maintenance*))

Besides *ACM DL*[3], *Engineering Village*, *IEEE Xplorer*, and *Scopus*, we also performed searches at the *ScienceDirect* and the *Web of Science* websites. Similarly, we did searches based on studies metadata, at least over the abstracts because of their richer content.

Table 1 describes in detail the number of studies returned per source of studies, both in the original search[4] (Kudo et al., 2019) and in this revisited version[5]. Therefore, 85 studies were identified (including duplicate papers) after the inclusion of two new bibliographic databases (*ScienceDirect* and *Web of Science*) and the update of search results over the four initial sources of studies.

Table 1. Number of studies returned per source.

Source	Original	Extension	Difference
ACM DL	24	26	2
Engineering Village	100	106	6
IEEE Xplorer	23	25	2
Scopus	71	76	5
ScienceDirect	-	9	9
Web of Science	-	61	61
Total	**218**	**303**	**85**

[2]Plural variations of the term "requirement pattern" are necessary due to the capabilities of the search engines of each source of studies.

[3]We chose the *The ACM Guide to Computing Literature* because it is a most comprehensive bibliographic database on Computing, including the full-text collection of all ACM publications.

[4]Search carried out from April 24 to May 5, 2018.

[5]Additional search performed on June 3 and 4, 2019.

2.3 Selection of primary studies

This section describes the selection method of relevant studies to answer the research questions of this SLM. The same original selection criteria were applied to the 303 papers returned by the automatic search process. The exclusion criteria (EC) are:

EC1 - It is not a primary study.
EC2 - It is not a paper (e.g., preface or summary of journals or conference proceedings).
EC3 - The research is not about SRP.
EC4 - The research addresses SRP in requirements engineering only.
EC5 - The full study text is not in English..
EC6 - The full study text is not accessible.
EC7 - It is a preliminary or short version of another study.

A paper is removed from this SLM whenever it meets at least one of the exclusion criteria (EC) presented; otherwise, the study is categorized based on the following inclusion criteria (IC):

IC1 - It addresses SRP in software design.
IC2 - It addresses SRP in software construction.
IC3 - It addresses SRP in software testing.
IC4 - It addresses SRP in software maintenance.

Figure 2 depicts the entire selection process with the respective number of primary studies chosen and removed in each activity of the conduction phase.

After the automatic search process, 155 duplicate papers are identified and removed (from the 303 studies group) with the support of the StArt tool (Fabbri et al., 2016). Next, we proceeded with reading of the title, summary, and keywords of each of the 148, upon which we applied the exclusion and inclusion criteria. As a result, we selected 41 possibly relevant studies because this selection relies on the reading and interpretation of papers' metadata only.

In the data extraction activity, we read the full text of these 41 studies from which we excluded 31 papers by the EC4 criterion, i.e., their research focus is on SRP in the requirements engineering phase. We describe the process of data extraction of the 10 studies remaining in Section 3. These studies will be identified throughout this paper as S1 to S10 as follows:

S1 - Adaptive requirement-driven architecture for integrated healthcare systems (Yang et al., 2010)
S2 - Analysing security requirements patterns based on problems decomposition and composition (Wen et al., 2011)
S3 - An architectural framework of the integrated transportation information service system (Chang and Gan, 2009)
S4 - Application of ontologies in identifying requirements patterns in use cases (Couto et al., 2014)
S5 - Effective security impact analysis with patterns for software enhancement (Okubo et al., 2011)
S6 - From requirement to design patterns for ubiquitous computing applications (Knote et al., 2016)
S7 - Modeling design patterns with description logics: A case study (Asnar et al., 2011)
S8 - Mutation patterns for temporal requirements of reactive systems (Trakhtenbrot, 2017)

Table 2. The total number of studies removed per exclusion criteria throughout the conducting phase.

Activity	EC1	EC2	EC3	EC4	EC5	EC6	EC7	Total
Automatic search	7	13	40	47	0	0	0	**107**
Data extraction	0	1	10	15	0	2	3	**31**
Snowballing	2	0	186	14	0	0	0	**202**
Total	**9**	**14**	**236**	**76**	**0**	**2**	**3**	**340**

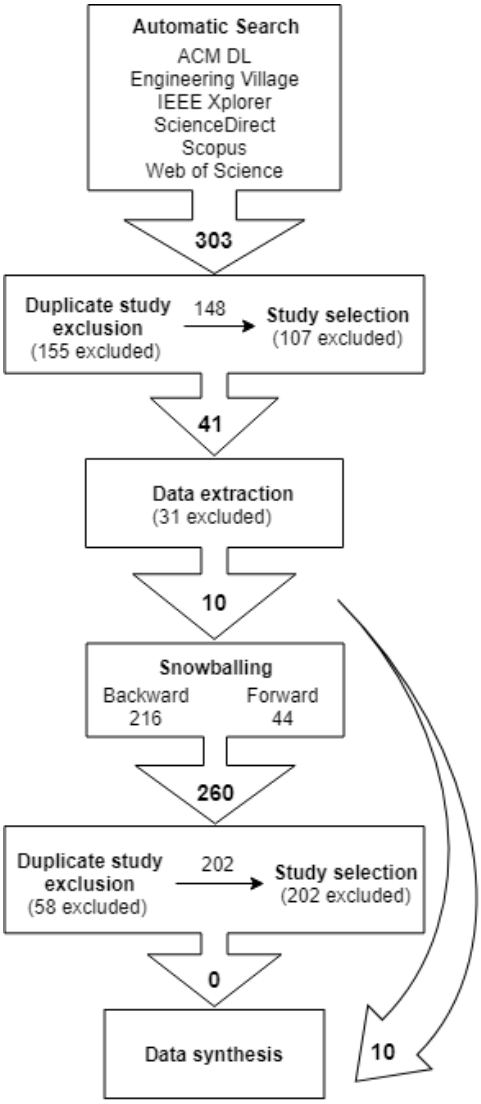

Figure 2. A detailed view of the conduction phase: automatic search, duplicate study exclusion, study selection, data extraction, snowballing, and data synthesis.

S9 - SACS: A pattern language for Safe Adaptive Control Software (Hauge and Stølen, 2011)
S10 - Re-engineering legacy Web applications into RIAs by aligning modernization requirements, patterns and RIA features (Conejero et al., 2013)

2.4 Snowballing

Besides automatic search, our search strategy includes snowballing as an attempt of obtaining other relevant studies using the papers S1 to S10 as input.

Regarding backward snowballing, we collected the reference list of each paper from the Scopus database, resulting in 216 documents whose metadata (title, abstract, and keywords) we stored into the StArt tool. After the removal of 49 duplicate studies, we read the metadata of the 167 documents remaining to decide for the exclusion or the tentative inclusion of a paper for further analysis. As no new paper was found in the first round of backward snowballing, we finished this analysis earlier.

In sequence, we searched the citation list of S1 to S10 from the Scopus website, resulting in 44 papers also registered into the StArt tool. Similarly, no new paper was retrieved in the first round of this forward snowballing step, resulting from the removal of 9 duplicate studies and the reading of the metadata of the 35 documents remaining.

Both snowballing procedures end up the process of selection of relevant studies of this SLM. Figure 2 depicts the total number of studies identified (260), excluded (58), and selected (0) from the overall snowballing process. As a result, the data extraction and synthesis activities include only the studies S1 to S10 previously presented.

Finally, Table 2 summarizes the studies removal process in the conduction phase. Most of the papers removed in the automatic search (87 of 107) are due to the EC3 and EC4 criteria, i.e., they do not address SRP, or they do it in the requirements engineering phase only, respectively. Studies were excluded at a similar rate (25 of 31) in data extraction activity. These exclusion rates around 80% are expected because of the trade-off analysis between coverage and relevance of the search string.

Differently, most of the studies removed during both snowballing procedures (186 of 202) are because of the EC3 criterion. Two related reasons explain this 92% exclusion rate: first, in general, the size of the reference list of a paper is far more extensive than the number of studies citing that paper; second, the papers in a reference list often address other research topics. Besides, only 7% of the studies referenced by or citing them represent research on SRPs (14 of 202). Even so, none of these explores SRPs at other stages of SDLC other than requirements engineering.

3 Data Extraction

This section describes the data extraction process from the full-text-reading of the 10 relevant studies (S1 to S10) of this SLM. Besides presenting a comparative analysis of the contribution types of each paper, we also extract:

1. the type of research carried out;
2. the type of requirement addressed by SRP;
3. the SDLC phase supported by SRP; and
4. the contribution type presented.

Table 3. Types of research and validation of relevant studies.

Type of research	Type of validation
Solution proposal	Proof of concept: S2 S5 S9
	No validation: S1 S3 S4 S6 S8
Validation research	Case study: S7
	Experiment: S10

Regarding the first item, we classified the ten-study-group using Petersen et al. (2015)'s criteria in which a set of conditions determine the type of research developed. For instance, opinion research solely reports the author's point of view about a subject. In this case, there is no usage in practice, empirical evaluation, author's experience report, or proposal of a conceptual framework or a novel solution.

Table 3 shows that, according to Petersen et al. (2015)'s taxonomy, most of the studies (8 of 10) is a solution proposal because there is no empirical evaluation: three studies are validated by a free proof of concept, whereas the five remaining do not even confirm their proposals. Furthermore, only two of ten studies are validation research: S7 presents a case study, and S10 describes an experiment with controlled conditions.

Next, we analyzed the particular type of software requirement covered by SRP, as presented in Table 4. Four of the relevant studies define SRP for the adaptability requirement and another four papers for the security one. The proposals of SRP described in the two studies remaining do not address a specific type of software requirement.

Table 4. Type of requirement covered by an SRP.

Type of requirement	Studies
Adaptability	S1 S3 S6 S8
Security	S2 S5 S7 S9
General purpose	S4 S10

Next, we describe a detailed comparative analysis of the contributions proposed in S1 to S10, from which we perceived some similarities.

Studies S1 and S3 propose a similar conceptual architecture for systems developed from SRPs as illustrated in Figure 3. The dashed lines A, B, C, and D show the similarities between the architectures proposed in S1 (left-hand side) and S3 (right-hand side). The requirements layer (A) identifies, analyzes, and models requirements as user requirement patterns (URP). The service layer (B) interacts with the requirements layer and provides services to satisfy the URP. The security and information sharing mechanism (C) establishes a process of reliable information exchange between systems of the same domain. The knowledge base (D) combines standards, norms, and ontologies of the system domain. The motivation of both research efforts is the need to share information between systems of the same area: medical systems (in S1) and transport systems (in S3).

Regarding S1 and S3 again, these studies make use of SRP to support the software design phase. In both studies, a URP in the requirements layer leads to the efficient selection of services in the service layers. A URP is a crucial element not only because it represents user requirements but also due to the fact it guides the operation of the entire system.

We also observed commonalities on how S2 and S5 represent security requirements as an SRP, as depicted in Figure 4. Both studies specify security requirement patterns with similar structure and security concepts (context, assets, and threats) as well as protection measures as design patterns.

Illustrated as dashed lines in Figure 4, the steps outlined in S2 (left-hand side) — the identification of stakeholders and objectives, essential information assets, and threat sources using standards — match with the following items of the security requirement pattern in S5 (right-hand side), respectively: the pattern definition format (context, problem, solution, and structure), asset, and threat. Finally, the step "adding protection measures in the system design" in S2 matches with the countermeasure concept described as security design patterns in S5.

From this analysis, we concluded that S2 and S5 also make use of SRP to benefit the software design phase because they define security requirement patterns and relate them to design-pattern-based protection measures.

As a result of the analysis of S8 and S9, we identified that both studies present proposals of requirement patterns representation format. In S8, each natural language-written requirement binds to a linear-temporal-language-written formula, in which mutations soften the likely issues in this association. Each type of requirement pattern attaches its potential failures and the respective appropriate variations. The formulas associated with mutants have multiple purposes such as tests generation, the adequacy of test sets, or the automatic construction of monitors for the system's behavior verification at run-time. Thus, the mutations included in the transformation of the requirement patterns contribute to the software testing phase.

In the case of S9, a composite pattern integrates three types of software patterns (i.e., requirement, design, and security). Based on the problem frames theory, this composite pattern uses parameters extracted from an inner requirement pattern, from which a set of functions correspond both to solutions in a design pattern and contextual elements in a security pattern. Thus, this applicability of SRP is at software design.

Both the studies S4 and S7 model requirement patterns using ontologies based on formal description logic. As ontology-based SRPs allow the automatic generation of source code in S4, this SRP contribution is to the software construction phase. In study S7, authors implement a mechanism that automatically binds an ontology-based security requirement pattern to a corresponding design pattern solution. Thus, the SRP main contribution in S7 is for the design phase of the SDLC.

In the context of ubiquitous computing (ubicomp) applications, S6 aims to map dependencies between design patterns and requirement patterns. This software pattern-integration approach bridges the gaps of the early software development phase, where recurring requirements demand similar design solutions, such as the case of the adaptability requirement for ubicomp applications. Consequently, the main contribution of S6 is for the software design phase.

Regarding the study S10, it presents a systematic process to modernize legacy Web applications into Rich Internet Applications (RIA). The core of that process is a set of traceability matrices that relate modernization requirements, RIA

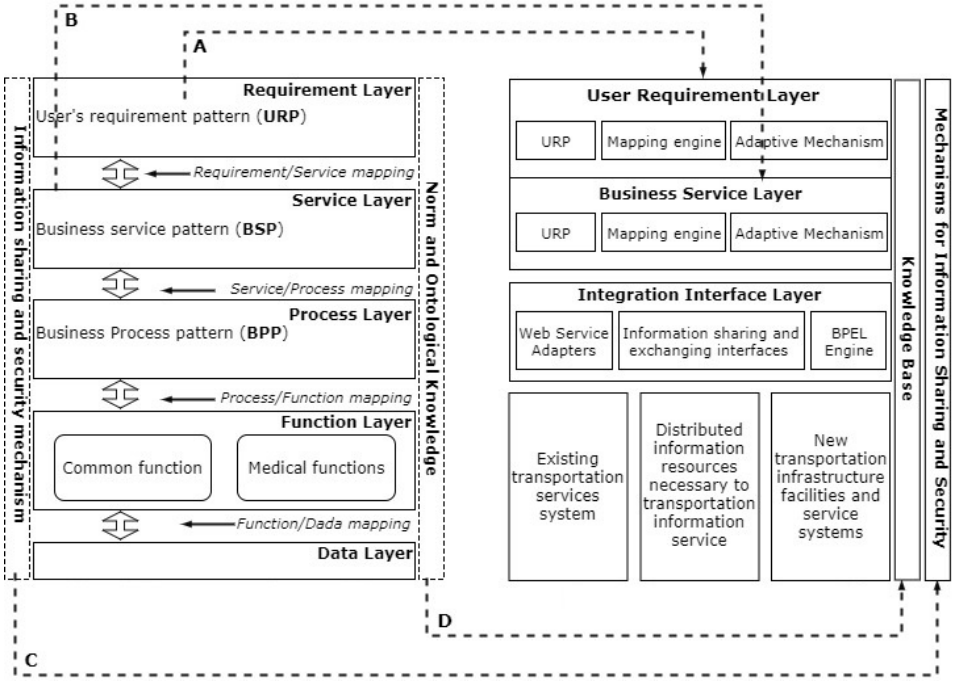

Figure 3. A comparative analysis of the SRP-based conceptual architectures discussed in S1 (left-hand side) and S3 (right-hand side).

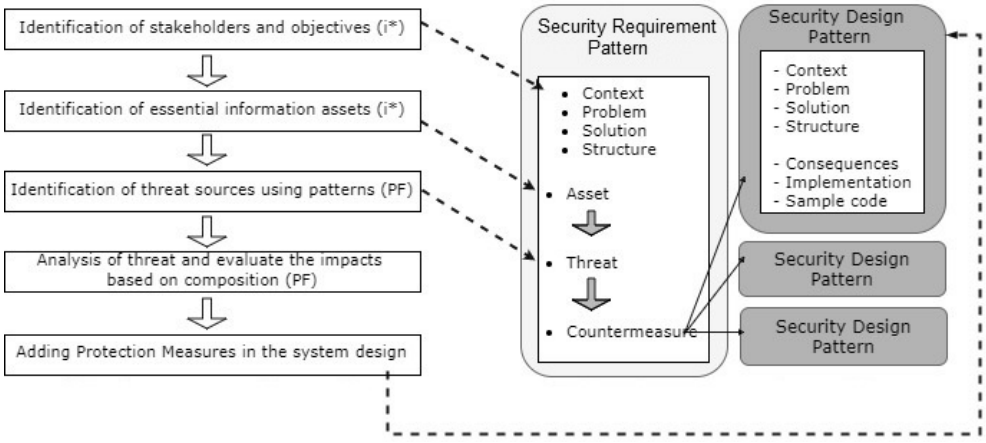

Figure 4. A comparative analysis of the SRP-based security approaches discussed in S2 (left-hand side) and S5 (right-hand side).

features, and patterns. A final traceability matrix suggests the most suitable RIA patterns for each new requirement based on the values of two different metrics: the degree of requirement realization (DRR) and the degree of pattern realization (DPR). Once selected, the RIA patterns are weaved into the legacy models so that those pattern-based RIA functionalities are incorporated into the system. The reusability of RIA patterns is very clear because the patterns traceability matrix — and other artifacts — is built once and used in any modernization process that, in turn, takes a lesser design time. Thus, in this approach, SRPs cover the gap between requirements elicitation and architectural design along the RIA development process.

Finally, Table 5 summarizes the analysis of the ten-study-group by the types of contributions identified: conceptual architectures for SRP-based systems; processes for discovery and use; representation formats; and catalogs of SRP.

4 Data Synthesis

This section presents a synthesis of the data extracted from the relevant studies to answer the research questions.

4.1 About the research question 1

To answer the research question "*At what SDLC phases are requirement patterns used: design, construction, testing and/or maintenance?*", eight studies use SRPs at the design phase, one at construction, one at software testing, and none at software maintenance.

Among the eight studies that address SRPs at software design (S1 to S3, S5 to S7, S9, and S10), there are no repeating authors, neither the convergence of studies to one or more research groups. Two hypotheses can explain the high concentration of studies related to the design phase: the fact that it is after requirements engineering as well as the increasing usage of design patterns in software development.

A significant difference between the number of relevant

Table 5. Data extraction from the 10 relevant studies.

Type of contribution	SDLC phase	Type of requirement	Studies
Conceptual architectures for SRP-based systems	Design	Adaptability	S1 S3
Representation formats for SRP	Design	Security	S2 S5 S9
	Testing	Adaptability	S8
Processes for discovery and use of SRP	Design	Security	S7
	Design	General purpose	S10
	Construction	General purpose	S4
Catalog of SRP	Design	Adaptability	S6

studies (10) and the number of papers excluded (77) is because these investigate SRPs exclusively for requirements engineering. This unbalance makes it clear that there is still an open field for research on the benefits of SRPs for the other SDLC phases such as testing and (1) maintenance (0).

As a consequence, another evidence is the lack of research on the use of SRPs along the entire SDLC, from requirements engineering to software maintenance. An example of a challenging study could be the evaluation of the improvements for the SDLC resulting from the adoption of SRPs, beyond the well-known benefits of time savings and better quality specifications.

4.2 About the research question 2

Regarding the research question "*Is there evidence of requirement patterns usage in practice at those SDLC phases?*", there is no study that reports evidence of SRPs usage in the software industry. Eight of the ten-relevant-studies are solution proposals with no validation, and only two papers (S7 and S10) are validation research. This analysis suggests that future work should be more focused on the use of SRPs along the SDLC in the software industry.

4.3 About the research question 3

To answer the research question "*Are there reported benefits of using requirement patterns at those phases? If so, what metrics are used to measure these benefits?*", S10 is the only study that defines SRP-related metrics. We believe that this lack of concern with metrics is because most articles are solution proposals, thus without use in practice.

In S10, the metrics DRR (degree of requirement realization) and DPR (degree of pattern realization) select candidate RIA patterns in the process of re-engineering of legacy web applications. A value of 1 in DRR indicates that a pattern fully supports all the RIA features demanded by the requirement, whereas a value of 0 means that the requirement and the pattern do not share any feature. Similarly, a value of 1 in DPR denotes that the requirement demands all the RIA features supported by the pattern, whereas a value close to 0 implies that the requirement needs an insignificant amount of the RIA features supported by the pattern.

The experiment results in S10 show that, in the worst case, more than half of the patterns would have been automatically suggested by the authors' method. Furthermore, the synchronization patterns indicated by the approach and those used by developers are the same in all systems tested in the experiment. Both results allow concluding that SRPs usage in S10 implies significant development time savings.

4.4 Discussion

Figure 5 illustrates a bubble graph that synthesizes the information we extracted and analyzed from each relevant paper.

Observe that four studies (S2, S5, S7, and S9) propose security requirement patterns with contributions to the software design phase. We conclude that this is because security is a recurrent requirement of many software systems, besides the support of well-established international standards (ISO/IEC, 2018). However, these studies mentioned above still require more significant validation with empirical assessments and use in the software industry.

Another four studies (S1, S3, S6, and S8) explore SRP for the adaptability nonfunctional requirement: one in software testing (S8), and the others in software design. Besides, none of these studies presents any validation of the proposal. S4, in turn, investigates SRPs for general purpose requirements used in software construction, but also with no validation.

Still regarding Figure 5, as important as mapping the research endeavors is the analysis of the existing gaps:

1. there is a general lack of investigation on the adoption of SRPs at other SDLC stages (10), while many research endeavors still focus on requirements engineering (77);
2. adaptability and security are the most addressed nonfunctional requirements as SRPs at the software design and testing phases, from the analysis of the left-hand side of the bubble graph. However, other types of nonfunctional requirements can be specified as SRP at different SDLC phases, e.g., usability aspects with automated support for code and test cases generation.
3. the application of research results on SRPs in the software industry (right-hand side of the figure); except for the studies S7 and S10, the remaining are in the proof of concept level.

5 Threats to Validity

The major problems in systematic studies are to find all relevant research on a topic and, from these, to select evidence of quality. For these reasons, three procedures were carried out throughout the planning and the conduction phases to reduce the threats to the validity of this SLM.

First, we performed an automatic search strategy that combines six relevant sources of studies with search string

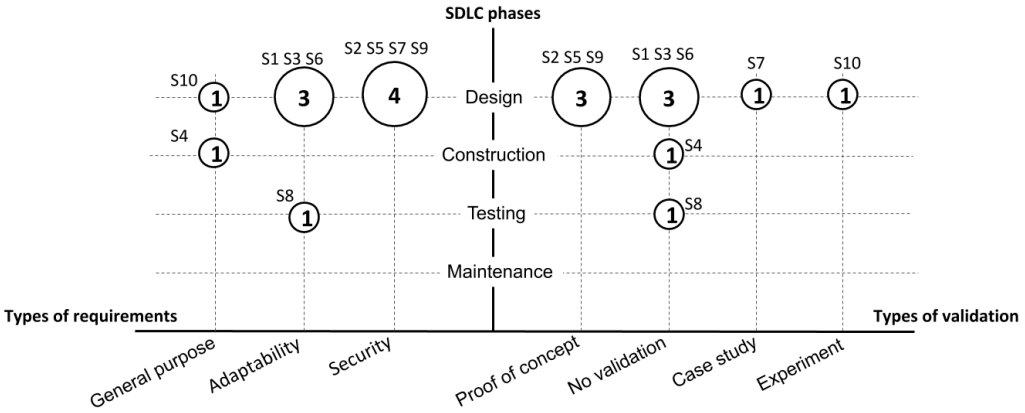

Figure 5. Mapping of the types of requirements and validation on SRPs for software design, construction, testing, and maintenance.

terms based on the SEVOCAB standard vocabulary. Besides, search in the gray literature is not part of the protocol (e.g., dissertations, theses, and technical reports) because we assume that good quality research is mostly published in journals or conferences.

Secondly, we were aware that searches could be extended to two additional important sources of research, i.e., *ScienceDirect* and *Web of Science*. Surprisingly, the number of relevant studies resulting from the automatic search increased only from 9 to 10 (the study S10 retrieved from *Web of Science*), even introducing those two new sources. As a means of retrieving a higher number of papers, we extended the search strategy again by performing the snowballing technique over those ten relevant studies. In spite of this, this hybrid search strategy included no new research.

Thirdly, but not less important, to mitigate the possibility of biases of this research, three researchers participated in the planning and conduction phases of this SLM as follows:

A: with 14 years of experience in Requirements Engineering, she performed the protocol planning, the selection, and the extraction and synthesis of data from the studies remaining;

B: with 13 years of experience in Software Engineering, he also performed the protocol planning, but his most significant contribution was on the verification of the results of the selection, extraction, and synthesis activities.

C: the team leader has more than 20 years of experience in Software Engineering, he helped the synthesis and writing of the results. In the case of divergences, A, B, and C solved conflicts together.

6 Final remarks

In the past few years, the literature has demonstrated the positive impacts of software requirement patterns on requirements specification quality, team productivity, elicitation and specification costs, among others (Barros-Justo et al., 2018; Irshad et al., 2018)

This paper presents a revisited version of a recent work (Kudo et al., 2019) that investigates if those benefits from SRPs usage have also been studied for the software design, construction, testing, and maintenance. Here, we expand the scope of the search strategy with two additional and pertinent sources of studies and the application of the snowballing technique.

In spite of this further workload in the search strategy, we obtained only one new relevant paper (S10) in comparison with our previous SLM. Nevertheless, we are confident that our results are valuable not only for new secondary studies on this same subject but also as a basis for future primary research.

To promote further research on SRPs in the whole software development process, we continue suggesting that the academic community should approach the software industry as a means of identifying its exact expectations. Researchers should also establish more metrics that corroborate the advantages of SRPs usage, such as reduced design time, automatic source code generation, standardized testing, and improvement in the quality of specifications in general.

At last, we also conclude that the concrete results of the SRPs usage in practice can be better experienced through two more lines of action: SRP-based innovative development tools, and the enhancement of the current development methodologies that could integrate SRPs along the SDLC.

As future work, we plan the inclusion of the term "analysis pattern" (and its variants) in the search string of this systematic mapping to augment the group of relevant studies. The main reason is that analysis patterns and requirements patterns are complementary approaches in such a way that the former can be transformed into the latter to migrate to the implementation details level (Pantoquilho et al., 2003).

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Renato Bulcão-Neto is grateful for the scholarship granted by CAPES/FAPEG (88887.305511/2018-00), in the context of the postdoctoral internship held at the Dept. of Computation and Mathematics of FFCLRP-USP. Alessandra Macedo is grateful for the financial support of FAPESP (16/13206-4) and CNPq (302031/2016-2 and 442533/2016-0).

References

Asnar, Y., Paja, E., and Mylopoulos, J. (2011). Modeling design patterns with description logics: A case study. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6741 LNCS, pages 169 – 183, London, United kingdom.

Barros-Justo, J. L., Benitti, F. B. V., and Leal, A. C. (2018). Software patterns and requirements engineering activities in real-world settings:a systematic mapping study. *Comp. Standards & Interfaces*, 58:23–42.

Beckers, K., Côté, I., and 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*, pages 337–342.

Bourque, P. and Fairley, R. E., editors (2014). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition.

Chang, F. and Gan, R. (2009). An architectural framework of the integrated transportation information service system. In *2009 IEEE International Conference on Grey Systems and Intelligent Services, GSIS 2009*, pages 1342 – 1346, Nanjing, China.

Conejero, J. M., Rodríguez-Echeverría, R., Sánchez-Figueroa, F., Linaje, M., Preciado, J. C., and Clemente, P. J. (2013). Re-engineering legacy web applications into rias by aligning modernization requirements, patterns and ria features. *Journal of Systems and Software*, 86(12):2981 – 2994.

Couto, R., Ribeiro, A. N., and Campos, J. C. (2014). Application of ontologies in identifying requirements patterns in use cases. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*, volume 147, pages 62 – 76, Grenoble, France.

Curcio, K., Navarro, T., Malucelli, A., and Reinehr, S. (2018). Requirements engineering: A systematic mapping study in agile software development. *Journal of Systems and Software*, 139:32 – 50.

Fabbri, S., Silva, C., Hernandes, E. M., Octaviano, F., Thommazo, A. D., and Belgamo, A. (2016). Improvements in the start tool to better support the systematic review process. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE 2016, Limerick, Ireland, June 01 - 03, 2016*, pages 21:1–21:5.

Fabbri, S. C. P. F., Felizardo, K. R., Ferrari, F. C., Hernandes, E. C. M., Octaviano, F. R., Nakagawa, E. Y., and Maldonado, J. C. (2013). Externalising tacit knowledge of the systematic review process. *IET Software*, 7(6):298–307.

Hauge, A. A. and Stølen, K. (2011). SACS: A pattern language for safe adaptive control software. In *Proceedings of the 18th Conference on Pattern Languages of Programs*, PLoP '11, pages 7:1–7:22, New York, NY, USA. ACM.

Irshad, M., Petersen, K., and Poulding, S. (2018). A systematic literature review of software requirements reuse approaches. *Inf. Softw. Technol.*, 93(C):223–245.

ISO/IEC (2018). ISO/IEC 27000:2018 Information technology – Security techniques – Information security management systems – Overview and vocabulary.

ISO/IEC/IEEE (2017). ISO/IEC/IEEE 24765:2017 Systems and software engineering – Vocabulary.

Kitchenham, B. A. and Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information & Software Technology*, 55(12):2049–2075.

Knote, R., Baraki, H., Söllner, M., Geihs, K., and Leimeister, J. M. (2016). From requirement to design patterns for ubiquitous computing applications. In *Proceedings of the 21st European Conference on Pattern Languages of Programs*.

Konrad, S. and Cheng, B. H. C. (2002). Requirements patterns for embedded systems. In *Proceedings IEEE Joint International Conference on Requirements Engineering*, pages 127–136.

Kudo, T. N., Bulcão-Neto, R. F., Macedo, A. A., and Vincenzi, A. M. (2019). Padrão de requisitos no ciclo de vida de software: Um mapeamento sistemático. In *22th Ibero-American Conference on Software Engineering (CIBSE)*, pages 1–14.

Okubo, T., Kaiya, H., and Yoshioka, N. (2011). Effective security impact analysis with patterns for software enhancement. In *2011 Sixth International Conference on Availability, Reliability and Security*, pages 527–534.

Palomares, C., Quer, C., and Franch, X. (2017). Requirements reuse and requirement patterns: a state of the practice survey. *Empirical Software Engineering*, 22(6):2719–2762.

Palomares, C., Quer, C., Franch, X., Renault, S., and 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, Coimbra, Portugal, March 18-22, 2013*, pages 1260–1265.

Pantoquilho, M., Raminhos, R., and Araújo, J. (2003). Analysis patterns specifications: Filling the gaps. In *Viking Plop*, Bergen, Norway.

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

Tockey, S. (2015). Insanity, hiring, and the software industry. *Computer*, 48(11):96–101.

Trakhtenbrot, M. (2017). Mutation patterns for temporal requirements of reactive systems. In *Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2017*, pages 116–121.

Wen, Y., Zhao, H., and Liu, L. (2011). Analysing security requirements patterns based on problems decomposition and composition. In *2011 1st International Workshop on Requirements Patterns, RePa'11*, pages 11 – 20, Trento, Italy.

Withall, S. (2007). *Software Requirement Patterns*. Best practices. Microsoft Press, Redmond, Washington.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineer-

ing. In *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014*, pages 38:1–38:10.

Yang, H., Liu, K., and Li, W. (2010). Adaptive requirement-driven architecture for integrated healthcare systems. *Journal of Computers*, 5(2).

Capítulo 5

UM METAMODELO PARA RELACIONAR PADRÕES DE REQUISITOS E PADRÕES DE TESTES

Artigo publicado no Simpósio Brasileiro de Engenharia de Software - SBES 2019.

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. A conceptual metamodel to bridging requirement patterns to test patterns. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering, SBES 2019, New York, NY, USA: ACM, p. 155–160, 2019.

A Conceptual Metamodel to Bridging Requirement Patterns to Test Patterns

Taciana Novo Kudo*
Departmento de Computação
Universidade Federal de São Carlos
São Carlos, São Paulo, Brazil
taciana@dc.ufscar.br

Renato F. Bulcão-Neto
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Goiás, Brazil
rbulcao@ufg.br

Auri M. R. Vincenzi
Departmento de Computação
Universidade Federal de São Carlos
São Carlos, São Paulo, Brazil
auri@dc.ufscar.br

ABSTRACT

Requirement patterns represent an abstraction of an application's behaviors and services that, in turn, may be replicated in similar applications. However, there has been a lack of efforts exploiting the benefits of requirement patterns in other phases of the software development life cycle, besides the requirements engineering itself. To address this gap, we propose the Software Pattern MetaModel (*SoPaMM*) that bridges requirement patterns to groups of scenarios with similar behaviors in the form of test patterns. *SoPaMM* allows the description of the behavior of a requirement pattern through a time executable and easy-to-use language aiming at the automatic generation of test patterns. Using *SoPaMM*, we model and implement a behavior-driven functional requirement pattern for a web-based user authentication application. Our preliminary results point out that a requirement pattern can be an executable specification capable of generating automated tests.

CCS CONCEPTS

• **Software and its engineering** → **Requirements analysis**; **Reusability**; *Acceptance testing*; • **Computing methodologies** → *Modeling methodologies.*

KEYWORDS

requirement pattern, test pattern, reuse, behavior, metamodeling

ACM Reference Format:
Taciana Novo Kudo, Renato F. Bulcão-Neto, and Auri M. R. Vincenzi. 2019. A Conceptual Metamodel to Bridging Requirement Patterns to Test Patterns. In *XXXIII Brazilian Symposium on Software Engineering (SBES 2019), September 23–27, 2019, Salvador, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3350768.3351300

1 INTRODUCTION

There is a consensus among researchers and software industry professionals about the value of requirements engineering and the critical vulnerability of software projects whenever requirements-related activities are poorly executed. Most software projects extrapolate budget and delivery times due to incorrect, omitted, misinterpreted, or conflicting requirements [3, 21].

In the last decade, the practice of requirements reuse [2, 7] has been a feasible alternative to mitigate those issues, assisting requirements engineers to produce better quality specifications. A fairly discussed reuse approach is the requirement pattern (RP) concept, which is an abstraction that aggregates behaviors and services observed in multiple applications that can be reused in similar software applications [23]. Several studies demonstrate that RPs can promote the completeness of requirement specifications and the increase of the productivity of the development team, among other benefits [3, 4, 8, 14, 23].

However, there have been few research efforts on the applicability of RPs into other phases of the software development life cycle (SDLC), disregarding the overall impact of requirements engineering as reported in a previous work [9]. As an example of the influence of requirements activities in the SDLC, the popular V-model [18] associates a user acceptance testing phase for the requirement analysis phase to determine whether a software system satisfies the requirements specified.

As another type of software pattern, a test pattern (TP) describes generic solutions to test common recurrent behaviors [12]. TPs help a tester understand the context of a testing practice as well as decision making among alternative patterns [11]. Repetitive, alike, and high-value test practices are likely candidates to be documented as TPs.

Therefore, by considering both the intrinsic relation between requirements engineering and testing and the lack of investigations into the advantages of RPs at other phases of the SDLC [9], we suggest that an RP can be related to a TP as well as be tested more assertively if test cases are documented into that TP.

This paper presents a conceptual metamodel called *Software Pattern MetaModel* (*SoPaMM*), which defines how requirement and test patterns can be written, organized, related, and classified. For this, *SoPaMM* reuses concepts and practices of the Behavior-Driven Development (BDD) agile methodology [1, 15] and the design pattern called Page Object Model (POM) [10, 20].

We show the applicability of *SoPaMM* through the modeling and implementation of a functional requirement pattern for user authentication into a web application integrated with automatically generated user acceptance test cases. Our preliminary results demonstrate the advantages of using *SoPaMM*: a comprehensive and structured way of combining, extending and reusing RPs and

*Currently, Mrs. Kudo is a Ph.D. candidate at the Universidade Federal de São Carlos and also works as an assistant professor at the Universidade Federal de Goiás.

SBES 2019, September 23–27, 2019, Salvador, Brazil

ACM ISBN 978-1-4503-7651-8/19/09…$15.00
https://doi.org/10.1145/3350768.3351300

TPs, the automated testing of RPs, and the decoupling of user interface elements from application test cases.

The structure of this paper is organized as follows: Section 2 discusses related work; Section 3 describes the conceptual metamodel proposed; Section 4 presents an instance model of *SoPaMM* implemented into a web application; and Section 5 brings our concluding remarks and future work.

2 RELATED WORK

Based on a recent previous work [9], there is little research on the integration of RPs with artifacts produced in other SDLC phases. For this reason, this section describes research endeavors that combine metamodeling and at least one of the types of software patterns covered in this paper.

Some studies propose modeling of RPs using metamodels, but with no integration with other SDLC artifacts [4, 24]. As one of the pioneering efforts on metamodeling of RPs, Franch et al. [4] define the structure of an RP, the possible relationships among RPs, and classification criteria for grouping them. The main idea behind the use of 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.

Another research on metamodeling and RPs, Badamasi et al. [24] present a metamodel that represents RPs with variability modeling and traceability of software artifacts. The main goal is to improve the systematic reuse of RPs by integrating concepts of software product line and model-driven engineering.

To the best of authors' knowledge, there is no work proposing the combined use of the metamodeling theory and TPs. The proposals of use of TPs range from unit testing, integration testing, and user interface testing [11, 12, 17]. Coelho et al. [17] define a TP for unit and integration testing of layered information systems. The main objective is that each test class focuses on testing the specific features implemented by each layer in an application. Meszaros [11], in turn, describes sixty-eight test patterns to facilitate how to write, understand, and maintain unit testing. The goal is to allow unit tests using TPs more robust and repeatable, and thus more cost-effective. Finally, Moreira and Paiva [12] develop six types of graphical user interface test pattern, aiming at elaborating generic test strategies for recurrent behaviors applicable over different applications. Those authors also develop a domain-specific language that supports the representation of such test patterns.

Despite not using metamodeling with patterns as other works do, we have found one study that integrates RPs with mutation tests [22]. Natural language-based requirement descriptions generate mutants, and every mutant is associated with a linear temporal logic formula that conveys its semantics. For each type of RP, the relevant potential faults are identified, and appropriate mutations are introduced. Formulas of mutants may have several purposes, such as test generation, and analysis of test case sets adequacy.

In comparison with those mentioned works above, the novelty here is the integration of RPs and TPs into a metamodel, where BDD concepts (feature, scenario, example) describe the behavior of an RP. This structured description of an RP can produce an executable and, at the same time, an easy-to-understand specification that enables the generation of TPs that can be automated.

3 THE SOPAMM CONCEPTUAL METAMODEL

The Software Pattern Metamodel (*SoPaMM*) describes how RPs and TPs are to be specified, related, stored, and classified. The underlying idea to *SoPaMM* is not only the reuse of an RP to be part of a software requirements specification, but also the further reuse of one or more TPs associated.

The *SoPaMM* metamodel borrows the MetaObject Facility's (MOF) metamodeling architecture [13] to make it easier the interoperability of the proposed model. The OMG's MOF defines a four-layered architecture that constitutes the foundation of the *SoPaMM* proposal, in which lower layer models are instances of immediately upper layer models, as illustrated in Figure 1.

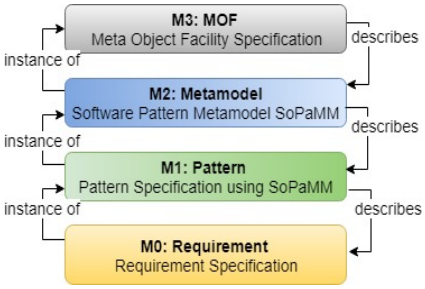

Figure 1: The integration between OMG's MOF and *SoPaMM*.

The MOF specification is in M3-layer and serves as a reference for the description of the *SoPaMM* metamodel located in M2-layer; hence, it is an instance of OMG's MOF. The M1-layer, in turn, describes requirement patterns and test patterns instanced from the *SoPaMM* metamodel. Finally, the M0-layer represents the description of a real-world requirement specification document, whose contents are instances of the M1-layer patterns.

Figure 2 depicts a UML class diagram of the *SoPAMM* metamodel, i.e., this corresponds to the M2-layer in Figure 1. Each *SoPaMM* concept — classes, attributes, and associations — is described next.

3.1 SoftwarePattern and SoftwarePatternBag

As its name suggests, the class *SoftwarePattern* (SP) represents the set of software patterns that can be supported by the *SoPaMM* metamodel. Software patterns research efforts and standardized specifications [4, 6, 8, 12, 23] have influenced the definition of the class SP. The definition of the class SP attributes is as follows:

- *Name*: the software pattern name (e.g., *LoginPasswordAuthenticationFRP*).
- *Author*: the software pattern author's name (e.g., John Doe).
- *Version*: the software pattern version date and number (e.g., version 1.0 of 2019/03/28).
- *Goal*: it describes what the software pattern is for (e.g., a functional requirement pattern for user authentication through login and password).
- *Source*: the source of information for the creation of the software pattern such as stakeholders' and similar software systems' names, and the identification of a physical or digital document, a standard specification, or law. For instance, consider the URL of the Brazilian Certification Manual for Electronic Health Record Systems (SBIS-CFM-2016-v4-2).

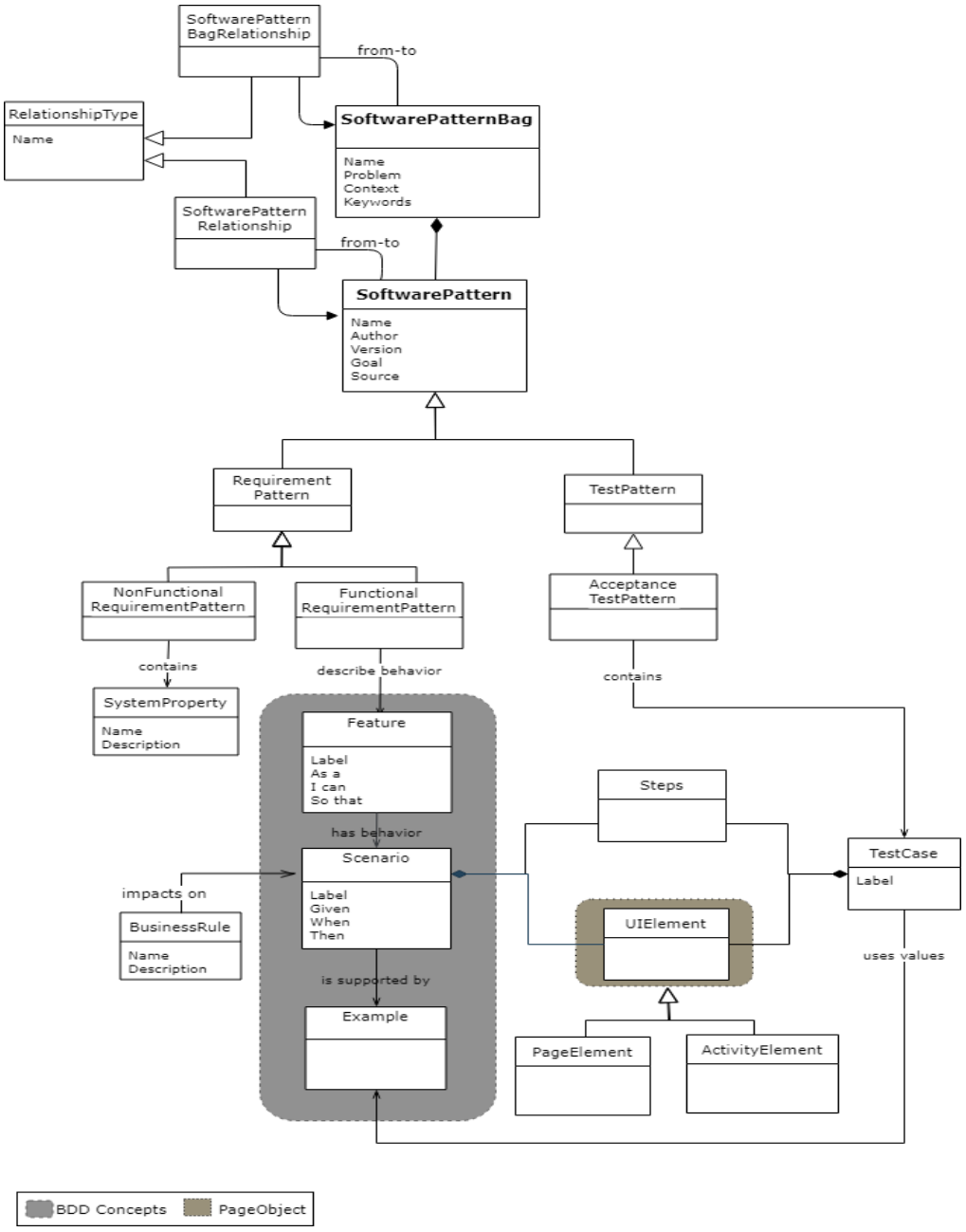

Figure 2: An overview of the *SoPaMM* metamodel.

In turn, the class *SoftwarePatternBag* (SPB) is a composition of one or more SPs. The goal is to allow not only the composition of multiple SPs but also of different SP types such as requirement patterns and test patterns. As currently modeled, the *SoPaMM* metamodel is extensible in terms of new types of SPs, e.g., by extending the superclass *SoftwarePattern* to include the class *Design Pattern*, with minimal impact on the structure defined. The definition of the class SPB attributes is as follows [5, 16]:

- *Name*: a sequence of characters that smoothly conveys the SPB contents to mitigate misinterpretation (e.g., *LoginPasswordAuthenticationSPB*).
- *Problem*: a statement of the problem the SPB addresses (e.g., only identified and authenticated users can get access to system data and functionalities).

- *Context*: the situation in which the problem occurs (e.g., a user must be able to access the system through login and password authentication).
- *Keywords*: terms for indexing and search purposes (e.g., authentication, login, and password).

3.2 Relationships of SP and SPB

An SPB may be related to other entities of the same type (e.g., an SPB is a *composition of* other SPBs, or *depends on* a particular SPB). Similarly, an SP may also be related to other SPs (e.g., an SP *depends on* another SP, or *uses* information from another SP).

Although the literature has defined relationships types among software requirement patterns such as *extends*, *has*, and *uses* [4, 23], a higher diversity of types may exist according to applications' context. For this reason, both relationships of SPBs and SPs are not predefined in *SoPaMM*, aiming thus metamodel flexibility. The classes *SoftwarePatternBagRelationship* and *SoftwarePatternRelationship* allow this flexibility through the superclass *RelationshipType*.

3.3 Requirement Pattern Definitions

Back to the definition of the class *SoftwarePattern* (SP), this specializes in two classes until this moment: the classes *RequirementPattern* (RP) and *TestPattern* (TP). The class RP, in turn, specializes in the classes *NonFunctionalRequirementPattern* (NFRP) and *FunctionalRequirementPattern* (FRP). As subclasses of SP, the classes RP, NFRP, FRP, and TP share the same SP attributes and differ each other regarding their relationships with the other *SoPaMM* classes.

Nonfunctional requirement patterns (NFRP) relate to software system properties (the class *SystemProperty*) described by textual attributes (i.e., name and description). For instance, user credentials must be ever validated by an authentication server, forbidding user authentication on the client-side.

Different from related works, the modeling of functional requirement patterns (FRP) in *SoPaMM* is influenced by concepts of the Behavior-Driven Development (BDD) methodology [1]. The class *Feature* describes FRPs' behaviors through a high-level and straightforward user story syntax (As <*stakeholder*>, I can <*what?*> so that <*why?*>).

The class *Feature* also groups related *Scenario*s supported by one or several *Example*s. Scenarios make use of the Gherkin language syntax [19] (Given <*an initial context*> When <*an event occurs*> Then <*an expected outcome*>), which allows a concise description of multiple examples with different values for a scenario. The concepts *Feature*, *Scenario*, and *Example* give not only structure and meaning to FRPs' behavior but also automate the respective test specifications. *SoPaMM* also allows the description of business rules that impact on an FRP's behavior.

3.4 TestPattern Definitions

Until this moment, the *TestPattern* (TP) superclass specializes in the class *AcceptanceTestPattern* (ATP), which contains one or more elements of *TestCase* which, in turn, use input data of the class *Example* of a scenario.

The way we integrate FRPs and ATPs is through the modeling of each FRP scenario as a composition of steps and *UIElement* object types. Steps are testable abstractions of the steps (Given, When, and Then), which are characteristic of BDD scenarios, while *UIElement* represents user interface elements. Besides, a *TestCase* related to an ATP is also described as a composition of steps and user interface elements, and makes use of data values of the class *Example* for running tests.

The class *UIElement* demonstrates the concept of the *Page Object* design pattern [10, 20], which reduces the coupling between user interface elements and test cases. This way, we promote reusability and also facilitate test cases maintenance. The *UIElement* can be of two types: *PageElement* for web applications, or *ActivityElement* for mobile applications.

4 PROOF OF CONCEPT

The first step of this proof of concept includes the development of a *SoPaMM*-based M1 model for user authentication through login and password, which is a recurrent functional requirement in most software systems. The structure of the *UserAuthLoginPassword:SPB* (*SoftwarePatternBag*) in Figure 3 is described next.

- a composition of one *FunctionalRequirementPattern* (FRP) and one *AcceptanceTestPattern* (ATP);
- the *UserAuthLoginPassword:FRP* includes one feature *LoginPassword:Feature* that, in turn, has two possible behaviors modeled as scenarios, i.e., *SuccessfulLogin:Scenario* and *UnsuccessfulLogin:Scenario*;
- the classes *SuccessfulLogin:Example* and *UnsuccessfulLogin:Example* encapsulate data fields and test data to support the scenarios above, respectively; and
- the *UserAuthLoginPassword:ATP* contains two *TestCases* (i.e., *SuccessfulLogin:TestCase* and *UnsuccessfulLogin:TestCase*) that, in turn, reuse the same classes *Example* previously defined for the FRP.

Notice that the integration between the FRP and the ATP occurs in the composition of steps and page elements, whose modeling decision is shared between scenarios and respective test cases. Besides, the reuse of the classes *Example* between scenarios and test cases co-related strengthens this FRP-ATP integration.

For instance, both the classes *UnsuccessfulLogin:Scenario* and *UnsuccessfulLogin:TestCase* reuse the same classes *UnsuccessfulLogin:Steps*, *UnsuccessfulLogin:PageElement*, and *UnsuccessfulLogin:Example*. The same occurs with the class *SuccessfulLogin:Scenario*. This way, test cases are pre-defined in a requirement pattern and are up for running when the requirement is implemented. Therefore, the more the examples support the requirement, the higher the number of acceptance test cases and, consequently, the wider the coverage test in the implementation.

The second step of this proof of concept is the implementation[1] of a web-based user authentication application supported by the M1 model in Figure 3. The main goal is to demonstrate the automation of acceptance tests associated with functional requirement patterns.

The starting point is the implementation of the *Login.feature* file: it contains a user story description of the feature of the *UserAuthLoginPassword:FRP*, a Gherkin-based definition of the particular scenarios for successful and unsuccessful login behaviors, and test input data (or examples) defined in the M1 model are used for each corresponding scenario.

[1] Apache Maven, the JUnit framework, the Selenium WebDriver, among others.

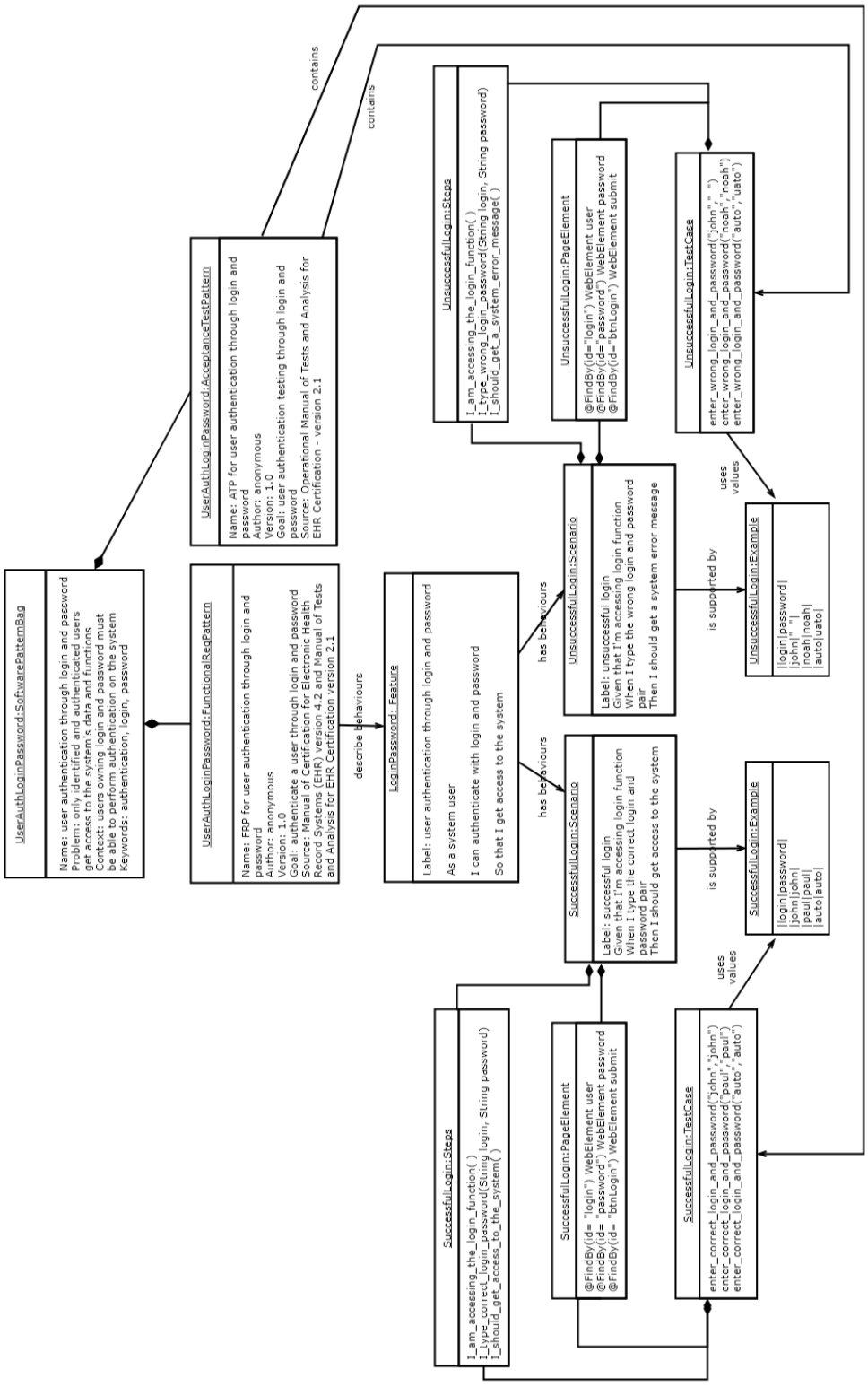

Figure 3: An example of an M1 model for the *SoPaMM* metamodel.

The class *LoginSteps* implements the steps (Given, When, Then) of each scenario (successful and unsuccessful login), whereas the class *LoginElementMap* represents the page elements for user authentication (i.e., text fields for login and password, and the submit button). The key point is the class *LoginPage* that extends *LoginElementMap* and associates the page elements with the steps of each scenario and the respective examples defined in *Login.feature*.

As a result, 18 steps were successfully tested for the user authentication FRP: 9 for each scenario (i.e., (un)successful login). We are aware that there is a tradeoff between test automation and the burden on the specification of RP with behavior (feature, scenario, and example). However, we advocate that test automation of RP with behavior may facilitate analysts', developers', and testers' tasks as well as the traceability between requirements and test cases.

5 FINAL REMARKS

Current trends in requirements engineering combine agility in the development process, the need for addressing emergent requirements, collaborative and test-oriented development, among others. To achieve these needs, requirements activities (e.g., acquisition and specification) have been changing in the past few years.

In that context, we propose the *SoPaMM* metamodel as a structured way of integrating requirement patterns and test patterns through concepts inherited from the behavior-driven development methodology. We describe a proof of concept in which we built a *SoPaMM*-based integrated model of RP and TP for user identification and authentication through login and password. In sequence, we implemented a simple web application with user access control to demonstrate the use in practice of that instance of *SoPaMM*. As a result, acceptance tests are automatically executed once the user identification and authentication is implemented.

The potential benefits of *SoPaMM* are as follows:

- verifiability – an RP is an executable specification capable of generating automated tests;
- ease of communication – the specification of an RP makes use of well-known and easy-to-use languages (user story and Gherkin syntax) for several stakeholders;
- extensibility – the metamodel has extensible points for the definition of new types of relationships between RPs and TPs; the same applies to the support for other types of software patterns (e.g., unit testing patterns);
- separation of concerns – test cases and UI elements are modeled apart to ease the reuse and maintenance of UI test cases.

Currently, we have been working on two lines of investigation: (i) the creation of a catalog of behavior-driven RPs from a set of requirements used in the certification process of electronic health record (EHR) systems in Brazil; and (ii) the development of a software tool to facilitate the use of *SoPaMM*-based patterns in real-world software projects.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Renato Bulcão-Neto is grateful for the scholarship granted by CAPES/FAPEG (88887.305511/2018-00), as postdoctoral intern at the Dept. of Computing and Mathematics, FFCLRP-USP.

REFERENCES

[1] David Chelimsky, Dave Astels, Bryan Helmkamp, Dan North, Zach Dennis, and Aslak Hellesoy. 2010. *The RSpec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends* (1st ed.). Pragmatic Bookshelf, Raleigh, NC.

[2] Yuri Chernak. 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, 46–53.

[3] Xavier Franch. 2015. Software Requirements Patterns: A State of the Art and the Practice. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 943–944.

[4] Xavier Franch, Cristina Palomares, Carme Quer, Samuel Renault, and Francois De Lazzer. 2010. A Metamodel for Software Requirement Patterns. In *Requirements Engineering: Foundation for Software Quality*, Roel Wieringa and Anne Persson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 85–90.

[5] Cecilia Haskins. 2003. Using Patterns to Share Best Results - A proposal to codify the SEBOK. *INCOSE International Symposium* 13, 1 (2003), 15–23.

[6] IEEE. 2018. ISO/IEC/IEEE 29148:2018 International Standard - Systems and software engineering - Life cycle processes - Requirements engineering. *ISO/IEC/IEEE 29148:2018(E)* 2 (2018), 1–92.

[7] Mohsin Irshad, Kai Petersen, and Simon Poulding. 2018. A Systematic Literature Review of Software Requirements Reuse Approaches. *Inf. Softw. Technol.* 93, C (Jan. 2018), 223–245.

[8] Sascha Konrad and Betty H.C. Cheng. 2002. Requirements patterns for embedded systems. In *Proceedings IEEE Joint International Conference on Requirements Engineering*. IEEE, Essen, Germany, 127–136.

[9] Taciana N. Kudo, Renato F. Bulcão-Neto, Alessandra A. Macedo, and Auri M.R. Vincenzi. 2019. Padrão de Requisitos no Ciclo de Vida de Software: Um Mapeamento Sistemático. In *XXII Ibero-American Conference on Software Engineering, CIbSE 2019*. Curran Associates, Inc., New York, USA, 420–433.

[10] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. 2013. Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study.. In *ICST Workshops*. IEEE Computer Society, Washington, DC, USA, 108–113.

[11] Gerard Meszaros. 2006. *XUnit Test Patterns: Refactoring Test Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[12] Rodrigo M. L. M. Moreira and Ana C. R. Paiva. 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*. IEEE, Lisbon, Portugal, 126–135.

[13] OMG. 2002. Meta Object Facility (MOF) Specification, Version 1.4. *Object Management Group, Inc.* (2002).

[14] Cristina Palomares, Carme Quer, and Xavier Franch. 2017. Requirements reuse and requirement patterns: a state of the practice survey. *Empirical Software Engineering* 22, 6 (2017), 2719–2762.

[15] Lauriane Pereira, Helen Sharp, Cleidson de Souza, Gabriel Oliveira, Sabrina Marczak, and Ricardo Bastos. 2018. Behavior-driven Development Benefits and Challenges: Reports from an Industrial Study. In *Proceedings of the 19th International Conference on Agile Software Development: Companion (XP '18)*. ACM, New York, NY, USA, Article 42, 4 pages.

[16] Linda Rising. 1999. Patterns: A way to reuse expertise. *IEEE Communications Magazine* 37, 4 (April 1999), 34–36.

[17] Arndt von Staa Roberta Coelho, Uirá Kulesza and Carlos Lucena. 2005. The Layered Information System Test Pattern. In *Fifth Latin American Conference on Patterns Languages of Programming (SugarLoafPlop05)*. Campos do Jordão - Brasil, 1–16.

[18] Paul Rook. 1986. Controlling software projects. *Software Engineering Journal* 1 (02 1986), 7.

[19] John Ferguson Smart. 2014. *BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle* (first ed.). Manning Publications.

[20] Andrea Stocco, Maurizio Leotta, Filippo Ricca, and Paolo Tonella. 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)*. IEEE Press, Piscataway, NJ, USA, 70–74.

[21] Steve Tockey. 2015. Insanity, Hiring, and the Software Industry. *Computer* 48 (11 2015), 96–101.

[22] Mark Trakhtenbrot. 2017. Mutation Patterns for Temporal Requirements of Reactive Systems. In *Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2017*. IEEE, 116–121.

[23] Stephen Withall. 2007. *Software Requirement Patterns*. Microsoft Press, Redmond, Washington.

[24] Badamasi Ya'u, Azlin Nordin, and Norsaremah Salleh. 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*. ICT4M, Jakarta, Indonesia, 188–193.

Capítulo 6

UMA FERRAMENTA PARA CONSTRUÇÃO DE CATÁLOGOS DE PADRÕES DE REQUISITOS COM COMPORTAMENTO

Artigo publicado no Workshop de Engenharia de Requisitos - WER 2020.

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Uma Ferramenta para Construção de Catálogos de Padrões de Requisitos com Comportamento In: Anais do Workshop em Engenharia de Requisitos, WER 2020, p. 1-14, São José dos Campos, SP, Brasil, August. 2020.

Uma Ferramenta para Construção de Catálogos de Padrões de Requisitos com Comportamento

Taciana N. Kudo[1,2], Renato F. Bulcão-Neto[1], and Auri M. R. Vincenzi[2]

[1] Instituto de Informática, Universidade Federal de Goiás, Goiânia-GO, Brazil
{taciana,rbulcao}@ufg.br
[2] Departamento de Computação, UFSCAR, São Carlos-SP, Brazil
auri@dc.ufscar.br

Resumo Um padrão de requisitos de software (PRS) reúne comportamentos e serviços de aplicativos com características semelhantes. Apesar dos benefícios obtidos com a adoção de PRS em projetos de software, há uma carência de pesquisas sobre PRS em outras fases do desenvolvimento, além da Engenharia de Requisitos. Com base em descobertas na literatura, o uso prático de PRS pode ser melhor experimentado pelo uso conjunto de metodologias de desenvolvimento bem estabelecidas, ferramentas de software orientadas a PRS e catálogos de PRS de modo sistematizado. Nesse sentido, o metamodelo *Software Pattern MetaModel* (SoPaMM) permite relacionar um PRS com outros tipos de padrão de software e incorpora comportamentos sob a influência da metodologia ágil *Behavior-Driven Development* (BDD). Neste artigo, propõe-se a ferramenta *Terminal Model Editor* (TMEd) para apoiar a construção de modelos terminais para domínios específicos, usando o metamodelo SoPaMM como modelo de referência. Um exemplo de uso da TMEd é apresentado com a elaboração de um catálogo de padrões de requisitos legais para a certificação de Sistemas de Registro Eletrônico de Saúde (S-RES) no Brasil. Espera-se que os esforços com a ferramenta TMEd e o catálogo de padrões de requisitos para S-RES baseados no metamodelo SoPaMM ajudem a comunidade a melhor compreender o impacto geral do uso de PRS no ciclo de vida de software, não limitando-se à Engenharia de Requisitos.

Keywords: Padrão de requisito de software · BDD · Ferramenta · Catálogo

1 Introdução

À medida que uma empresa de desenvolvimento de software produz especificações de requisitos, é natural que uma porção desses requisitos seja específica de cada software, enquanto que uma parcela significativa dos requisitos se repita ao longo do tempo [22]. Ou seja, nem todos os requisitos que definem um software lhe são específicos, e reusar o conhecimento adquirido em projetos anteriores é uma estratégia adequada para melhorar a qualidade dos requisitos e a eficiência do processo de Engenharia de Requisitos [16].

Uma das abordagens voltadas ao reúso de requisitos é a de Padrão de Requisitos de Software (PRS), uma abstração que agrega comportamentos e serviços comuns a vários sistemas e que pode ser reutilizada em software similar [22]. A literatura reporta propostas de PRS para diversos domínios de aplicação, como sistemas embarcados [6], de gerenciamento de conteúdo [17] e de computação em nuvem [2]. Das experiências existentes do uso de PRS em projetos de software, identifica-se uma série de benefícios que afetam as atividades de coleta e especificação de requisitos, como a economia de tempo e a melhoria na qualidade dos requisitos quanto à completude, uniformidade, consistência e clareza [20,21,22].

Entretanto, estudos secundários [8,9] destacam a carência de relatos dessas vantagens em outras fases do ciclo de vida de software (CVS), vantagens reportadas em 76 diferentes estudos sobre PRS na Engenharia de Requisitos. Esses mesmos estudos secundários reportam apenas oito pesquisas sobre PRS em *design* de software, uma em construção, uma em teste, e nenhuma em manutenção, apesar da intrínseca relação entre requisitos e artefatos produzidos nessas fases.

Nesse contexto, o metamodelo *Software Pattern MetaModel* (SoPaMM) [10] é proposto de modo a relacionar PRS com padrões voltados às demais fases do CVS. Para cobrir a lacuna de uso de PRS na fase de teste [8,9], o metamodelo SoPaMM permite relacionar PRS com padrões de teste de software (PTS), uma abstração de práticas de teste repetitivas, semelhantes e de alto valor [12].

Contudo, uma abordagem de metamodelo em si não é suficiente. Evidências encontradas em [7] apontam que, para promover os estados da arte e da prática de PRS, as abordagens de PRS devem combinar:

(a) metodologias de desenvolvimento estabelecidas;
(b) ferramentas de software; e
(c) catálogos (ou conjuntos) de PRS de modo padronizado.

Em resposta ao item (a), o metamodelo SoPaMM relaciona PRS a PTS, representa um PRS com comportamentos como descritos na metodologia *Behavior-Driven Development* (BDD) [4] e possibilita transformar esse PRS em uma especificação executável capaz de gerar testes automatizados [10].

Em relação ao item (b), propõe-se neste artigo a ferramenta *Terminal Model Editor* (TMEd) para apoiar a construção de modelos terminais (instâncias) do metamodelo SoPaMM para domínios específicos, segundo o padrão *Meta Object Facility* (MOF) [13].

Quanto ao item (c), a ferramenta TMEd é utilizada para a elaboração de um catálogo de PRS baseados em comportamento com base em requisitos legais definidos e utilizados pela Sociedade Brasileira de Informática em Saúde (SBIS) no processo de certificação de Sistemas de Registro Eletrônico de Saúde (S-RES) [18]. A ferramenta TMEd é, até o presente momento, a única a permitir a criação de catálogos de PRS baseados em comportamento.

Este artigo está assim organizado: a Seção 2 descreve e analisa trabalhos relacionados; a Seção 3 apresenta fundamentos teóricos da pesquisa; a Seção 4 detalha o desenvolvimento e um exemplo de uso da ferramenta TMEd; e a Seção 5 traz nossas considerações finais e propostas de trabalhos futuros.

2 Trabalhos Relacionados

Esta seção descreve ferramentas de gerenciamento de catálogos de PRS, que organizam PRS em catálogos e permitem a criação, edição, exclusão e relacionamento de PRS, assim como a ferramenta TMEd proposta.

PABRE-Man [14] é uma ferramenta *desktop* que se conecta a uma base de catálogos de PRS [15] que seguem o modelo de referência PABRE definido pelos seus autores [5]. As principais funcionalidades da PABRE-Man são o gerenciamento e a busca por PRS e a exportação e impressão de um catálogo de padrões PABRE. Embora a ferramenta suporte PRS de domínios de aplicação variados, os PRS são relativos apenas a requisitos não funcionais.

Já a ferramenta proposta em [1] é específica para o domínio de Sistemas de Informação e os PRS manipulados representam tanto requisitos funcionais quanto não funcionais. Três módulos principais da ferramenta fornecem apoio à manutenção de usuário, cliente e projeto de software, especificação e gestão de PRS e instanciação de PRS. O módulo de gestão de PRS dessa ferramenta é o que mais se assemelha à proposta da ferramenta TMEd, enquanto que o módulo de instanciação de PRS apoia a elaboração de documentos de especificação de requisitos a partir do reúso de PRS.

A ferramenta *Requirement Pattern Editor* (*RP Editor*) [3] gerencia catálogos de PRS específicos de segurança. Assim como a TMEd, sua implementação é baseada na plataforma *Eclipse Modeling Framework* (EMF), mas apresenta suporte para edição gráfica de PRS via *Graphical Editing Framework* (GEF) e *Graphical Modeling Framework* (GMF). A ferramenta *RP Editor* permite exibir, criar, modificar e excluir PRS de segurança. Complementar a essa ferramenta, existe a *Instantiated Requirement Pattern Editor* (*InstRP Editor*) [3], que instancia os PRS de segurança para a produção de documentos de especificação de requisitos.

Comparativamente, a ferramenta TMEd possui menos funcionalidades que as apresentadas, como o fato de não apoiar a elaboração de instâncias de PRS [1,3] ou gerenciar perfis de usuários [1]. Por outro lado, os PRS gerenciados pela TMEd são independentes de domínio de aplicação e são relativos a requisitos funcionais e não funcionais. Mais importante ainda, a ferramenta TMEd é a única, até o presente momento, a permitir a criação de catálogos de PRS com comportamento, combinando padrões de requisitos e de testes.

3 Fundamentação Teórica

Esta seção discorre sobre dois pilares da proposta da ferramenta TMEd: a metodologia ágil BDD e o metamodelo SoPaMM, desenvolvido pelos autores.

3.1 *Behavior-Driven Development* (BDD)

BDD é uma abordagem de desenvolvimento ágil que promove a colaboração e o entendimento comum entre equipes técnica e de negócios em relação ao comportamento esperado do software a ser desenvolvido [4]. A comunicação entre essas

equipes ocorre por meio da linguagem *Gherkin*, que descreve cenários usando a sintaxe *Given-When-Then*, em que: *Given* define as condições para executar o cenário; *When* descreve os passos do cenário; e *Then* especifica os comportamentos esperados com a execução do cenário.

Os comportamentos derivam-se dos objetivos de negócio decompostos em *features*, cada qual associada a uma ou mais histórias de usuário (AS <interessado>, I CAN <o quê?>, SO THAT <por quê?>). Cada história de usuário pode ter um ou mais cenários que descrevem comportamentos esperados da aplicação. A seguir, apresenta-se uma descrição resumida de uma *feature* de autenticação de usuário por *login* e senha, na forma de história de usuário, com um comportamento de sucesso esperado descrito como cenário em *Gherkin*. Detalhes adicionais sobre a abordagem BDD podem ser obtidos em [4,19].

```
Feature: autenticação de usuário com nome de usuário e senha
  AS usuário do sistema
  I CAN autenticar com nome de usuário e senha
  SO THAT eu possa ter acesso ao sistema

  Scenario: usuário autenticado com sucesso
    GIVEN que estou executando a função de login
    WHEN eu digito o nome de usuário e a senha corretos
    THEN eu tenho acesso ao sistema
```

3.2 *Software Pattern Metamodel* (SoPaMM)

O metamodelo SoPaMM permite a reutilização de padrões de requisitos de software com o benefício adicional destes apresentarem comportamentos baseados em conceitos e práticas da abordagem BDD. O objetivo principal com o uso do SoPaMM é construir especificações de requisitos e de testes de melhor qualidade, em um tempo menor, além da geração de testes automatizados [10].

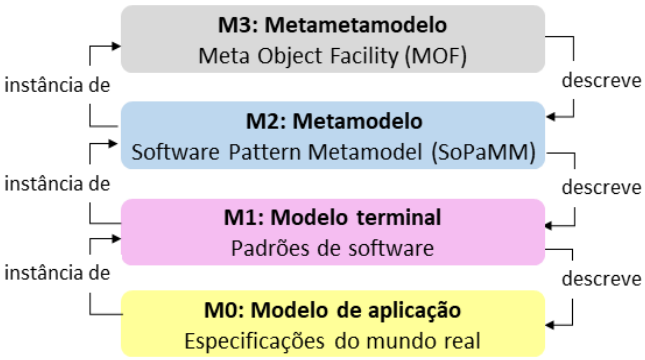

Figura 1. Relacionamentos entre MOF, SoPaMM e modelos terminais e de aplicação.

A construção do metamodelo SoPaMM segue a arquitetura de metamodelagem do *Meta Object Facility* (MOF) [13], tal como mostra a Figura 1. Nessa arquitetura, os modelos são refinados progressivamente e organizados em três categorias [11]: metametamodelos, metamodelos e modelos terminais. Na camada M3, estão os conceitos do MOF, que atuam como um metametamodelo. Na camada M2, encontra-se o metamodelo SoPaMM, que tem como modelo de referência o metametamodelo MOF. Na camada M1, encontram-se instâncias do metamodelo SoPaMM, ou seja, modelos terminais que representam padrões de requisitos e de testes, por exemplo. Por fim, na camada M0, está o modelo de aplicação, que descreve as especificações de aplicações do mundo real, e segue como modelo de referência o modelo terminal da camada M1.

No contexto desta pesquisa, o foco é a construção de modelos terminais a partir da estrutura do metamodelo SoPaMM. Os modelos terminais reúnem padrões de requisitos e de testes conforme o esquema do SoPaMM, cujos principais conceitos aparecem em destaque na Figura 2: *Catalogue*, elemento que reúne todas as definições do modelo terminal; *SoftwarePatternBag*, que agrupa padrões de software; *SoftwarePattern* que pode ser especializado em *RequirementPattern* e *TestPattern*, representados por PRS e PTS, respectivamente; e os relacionamentos[3] entre diferentes *SoftwarePatternBag* ou *SoftwarePattern*.

A classe *FunctionalRequirementPattern*, especializada da classe *RequirementPattern*, corresponde aos padrões de requisitos funcionais e são detalhados conforme os conceitos do BDD. Um padrão de requisito funcional é constituído por elementos da classe *Feature*, cujos comportamentos são descritos por elementos da classe *Scenario* que, por sua vez, são apoiados por dados de teste na classe *Example*. Estes e outros conceitos do metamodelo SoPaMM [10] apoiam a definição da ferramenta TMEd para a construção de instâncias desse metamodelo.

4 *Terminal Model Editor* (TMEd)

Esta seção descreve a ferramenta TMEd sob a ótica de seus requisitos fundamentais, detalhes de implementação, operação de principais funcionalidades e um exemplo de uso. Em linhas gerais,

- os padrões de software (ou modelos terminais) construídos a partir da TMEd seguem o esquema do metamodelo SoPaMM (vide níveis M1 e M2 na Figura 1, respectivamente);
- a saída da ferramenta TMEd é generalizada como padrões de software porque o metamodelo SoPaMM permite representar outros tipos de padrões de software, que não unicamente PRS; e
- a ferramenta apoia os papéis de analistas de requisitos e de testes, pois a versão atual do SoPaMM permite especificar PRS com comportamentos e ainda relacioná-los a PTS e elementos de interface com usuário para fins de teste de aceitação.

[3] Um exemplo de tipo de relacionamento é o *Refers to*, que pode permitir que um padrão de software refira-se a outro que contém informações adicionais sobre um tópico específico [22].

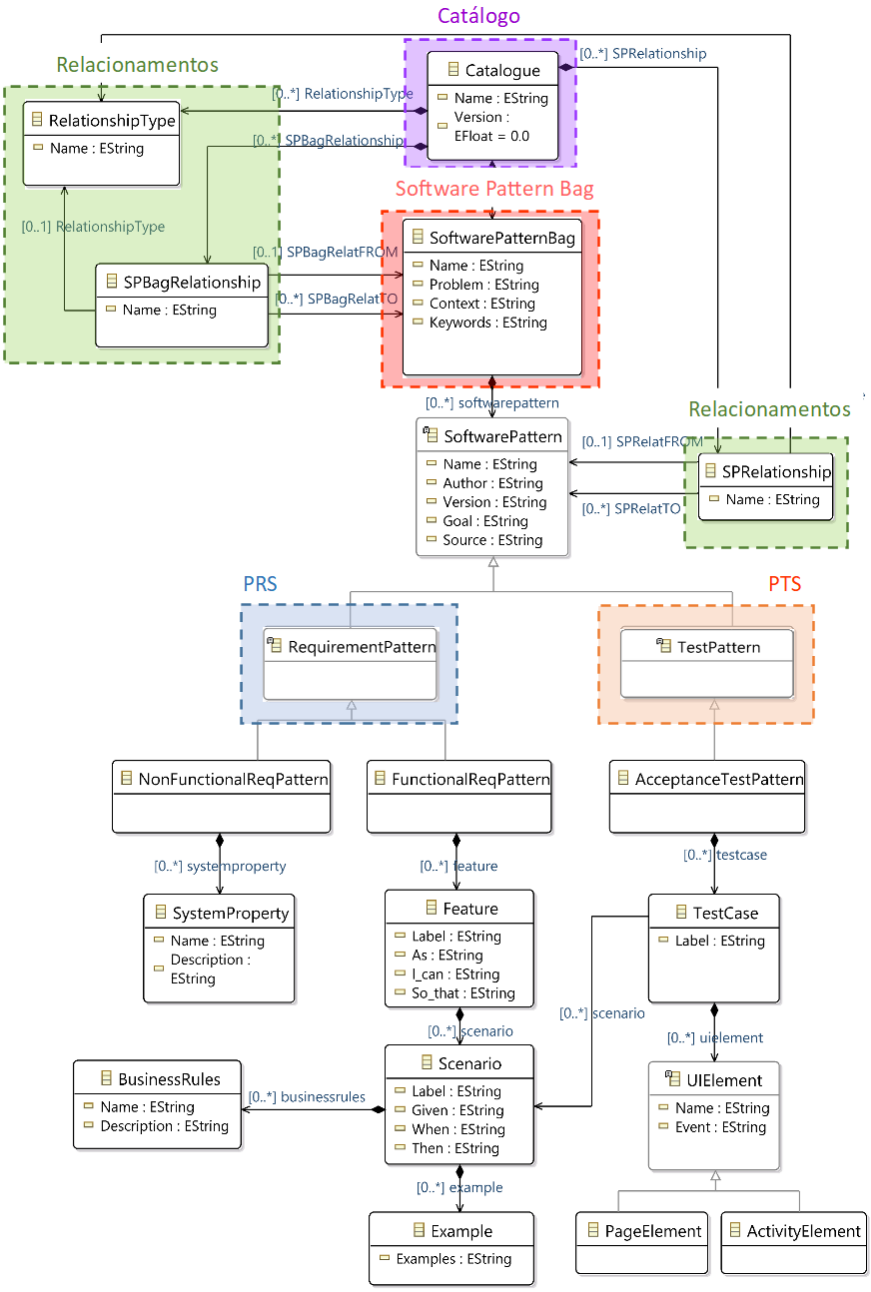

Figura 2. Visão geral do metamodelo SoPaMM, destacando principais conceitos [10].

4.1 Requisitos

Em geral, ferramentas que fornecem apoio à construção de PRS utilizam algum modelo ou *template* de referência. Neste caso, o requisito essencial da TMEd é implementar o esquema do metamodelo SoPaMM, para que o usuário possa explorar a construção de modelos terminais que incluem PRS e PTS.

Derivados desse requisito-base e influenciados por experiências relatadas em [5], seguem os principais requisitos da ferramenta TMEd:

1. gerenciar metadados de catálogos de padrões de software;
2. gerenciar tipos de relacionamentos;
3. gerenciar *bags* de padrões em um catálogo;
4. definir relacionamentos entre *bags* de padrões;
5. definir relacionamentos entre padrões de software;
6. gerenciar padrões de requisitos não-funcionais de uma *bag* (e suas *system properties*);
7. gerenciar padrões de requisitos funcionais de uma *bag* (composto de *features*, *scenarios* e *examples*);
8. gerenciar padrões de testes de aceitação de uma *bag* (composto de casos de teste e elementos de interface com usuário); e
9. validar a estrutura do catálogo com a gramática do metamodelo SoPaMM.

4.2 Desenvolvimento

A ferramenta TMEd foi desenvolvida utilizando a plataforma *Eclipse* e o *plug-in Eclipse Modeling Framework*[4] (EMF), cujo arcabouço consiste de três elementos:

- *EMF Ecore*, que descreve e oferece suporte para modelos, incluindo notificação de alterações, persistência com serialização XMI e uma API para manipular objetos EMF;
- *EMF.Edit*, que fornece classes para criar editores para modelos EMF e permite que os modelos sejam exibidos usando visualizadores de área de trabalho padrão;
- *EMF.Codegen*, que inclui uma GUI na qual as opções de geração dos editores são especificadas e os geradores são chamados.

O metamodelo SoPaMM foi representado em formatos nativos do EMF: o *ecore* e o *genmodel*. A Figura 3 ilustra, à sua esquerda, o arquivo *soPaMM.ecore* que armazena os atributos e relacionamentos do modelo. No lado direito dessa figura, encontra-se o detalhamento do *soPaMM.ecore* com as classes, os atributos e os relacionamentos entre as classes do metamodelo. A Figura 4, por sua vez, apresenta, à sua esquerda, o arquivo *soPaMM.genmodel* que contém as configurações para geração do código do editor de modelos terminais TMEd. No lado direito dessa figura, na aba *Properties*, estão as propriedades do catálogo como, por exemplo, os indicadores para as classes *Editor Plug-in Class* e *Editor Plug-in ID*, utilizadas na criação do editor.

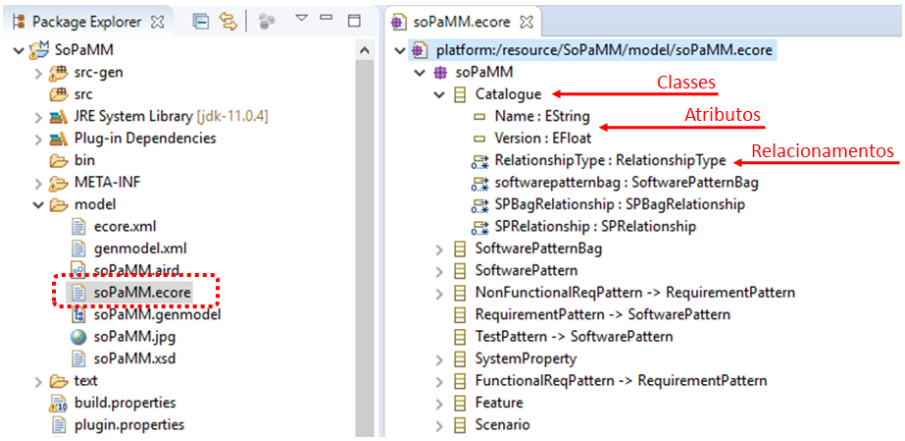

Figura 3. Arquivo de descrição *ecore* do SoPaMM.

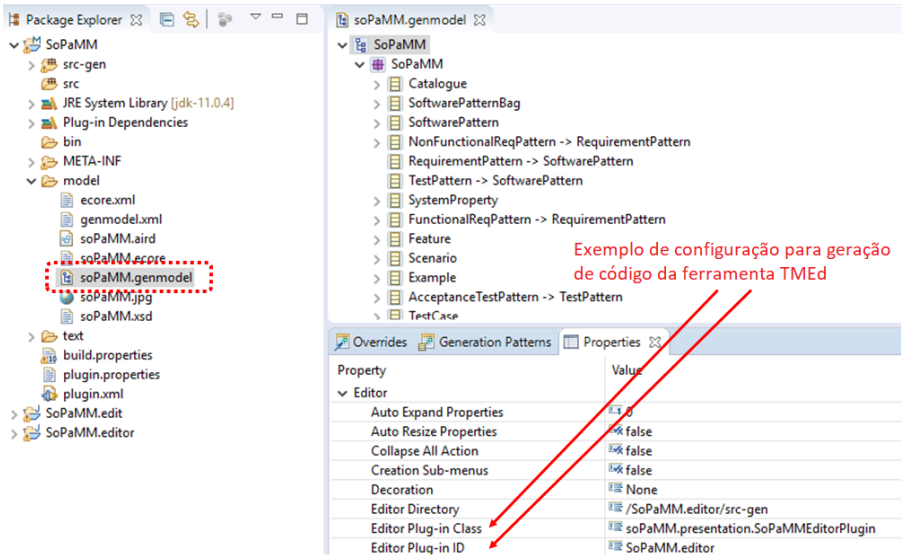

Figura 4. Arquivo de descrição *genmodel* do SoPaMM.

Com os arquivos de descrição *soPaMM.ecore* e *soPaMM.genmodel* construídos, o próximo passo foi a geração dos arquivos da TMEd com o *EMF.Edit*. Neste momento, foram criados os arquivos correspondentes ao *Edit* e ao *Editor*.

[4] Disponível online em https://www.eclipse.org/modeling/emf/.

4.3 Criação de Modelos Terminais

Para utilizar a TMEd, é necessário executar o arquivo *.genmodel* como uma nova aplicação no EMF. Ao realizar essa ação, é aberta uma nova instância do Eclipse com a TMEd e, a partir daí, é possível gerenciar modelos terminais.

Na Figura 5 é apresentada a interface de criação de modelos terminais baseados no SoPaMM. Uma interface do tipo *wizard* permite criar um novo modelo do tipo *SoPaMM Model* na pasta *Example EMF Model Creation Wizard*.

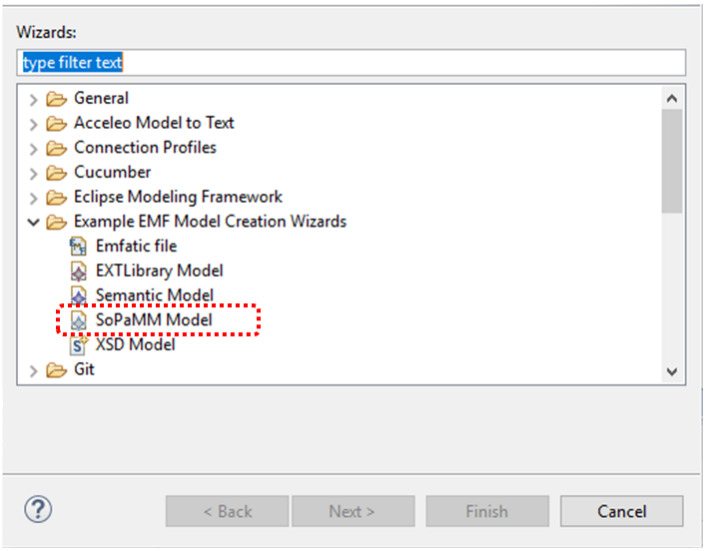

Figura 5. Interface de criação de modelo terminal baseado no SoPaMM.

Em conformidade com a gramática do metamodelo SoPaMM e, atendendo aos requisitos da TMEd, *Catalogue* é a primeira classe gerada em cada novo modelo terminal (requisito 1). A partir desta, é possível então incluir as demais entidades e atributos do modelo terminal.

A interface do TMEd para manipulação de um catálogo é mostrada na Figura 6 e, como exemplo, é apresentado o menu de inclusão de tipos de relacionamentos (*RelationshipType*), *bags* de padrões de software (*SoftwarePatternBag*), relacionamentos entre *bags* de padrões de software (*SoftwarePatternBag*) e relacionamentos entre padrões de software (*SPRelationship*), que correspondem aos requisitos 2 a 5, respectivamente.

Uma vez criada uma *bag* de padrões de software (*SoftwarePatternBag*), pode-se criar as especializações de padrão de software permitidas pelo SoPaMM, neste caso, padrões de requisitos funcionais e não funcionais e padrões de teste de aceitação (requisitos 6 a 8). Todo catálogo de padrões construído na TMEd é exportado para um arquivo XML, segundo o esquema do SoPaMM (requisito 9).

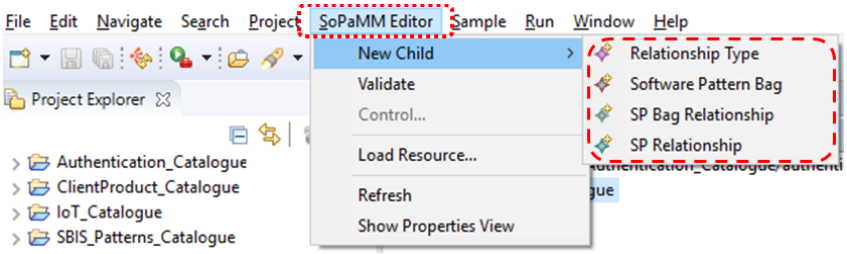

Figura 6. Interface de manipulação de catálogo de PRS.

4.4 Exemplo de Uso

A ferramenta TMEd foi utilizada na construção de um catálogo de PRS baseados em comportamento com base em requisitos legais utilizados na certificação de Sistemas de Registro Eletrônico de Saúde (S-RES) e elaborados pela Sociedade Brasileira de Informática em Saúde (SBIS) [18].

Para a definição desses requisitos, além de realizar uma extensa revisão de experiências e projetos similares de S-RES, a SBIS também utilizou normas ISO e padrões nacionais e internacionais. Com esses requisitos, a SBIS visa a garantir o alinhamento com tendências, bem como a adesão com a legislação nacional.

Os requisitos definidos pela SBIS são divididos em dois Níveis de Garantia de Segurança (NGS). O primeiro nível de certificação (NGS1) determina requisitos obrigatórios para a troca de informação em saúde suplementar, enquanto que o segundo nível (NGS2) permite a substituição de registros de saúde em papel por seus equivalentes eletrônicos. Seguindo o método sistemático de Withall [22], o catálogo elaborado com a TMEd contempla todos os requisitos do NGS1 [18].

A Figura 7 exibe a interface da ferramenta TMEd com as definições do catálogo de padrões para S-RES. No lado esquerdo, tem-se o arquivo *sbisSoPaMM*, que corresponde ao próprio catálogo, e a sua representação equivalente no formato XML. Já à direita, vê-se o conteúdo do catálogo da SBIS, organizado como uma árvore de elementos estruturados segundo a gramática do metamodelo SoPaMM. Por exemplo, o elemento *Catalogue* (A) encontra-se em um nível hierárquico acima das definições de todos os padrões do catálogo.

Considere o conteúdo da *bag* de autenticação *SPB_AuthenticationLoginPassword* (B), composta pelos padrões de requisito funcional e de teste de aceitação *FRP_AuthenticationLoginPassword* (C) e *ATP_AuthenticationLoginPassword* (D), respectivamente. O padrão de requisito descreve a funcionalidade de autenticação com login e senha em S-RES, enquanto que o padrão de teste lista os eventos para a realização dos testes de aceitação dessa funcionalidade. Observe que o padrão de requisito em questão é descrito com os conceitos do BDD, com *feature* (E), *scenario* (F) e *example* (G).

Três cenários estão especificados no padrão de requisito *FRP_AuthenticationLoginPassword*: o de sucesso, *Scenario_SuccessfulLogin* (F), e os de erro, *Scenario_LoginUnsuccessfull_InvalidUserName* e *Scenario_LoginUnsuccessfull_Invalid-*

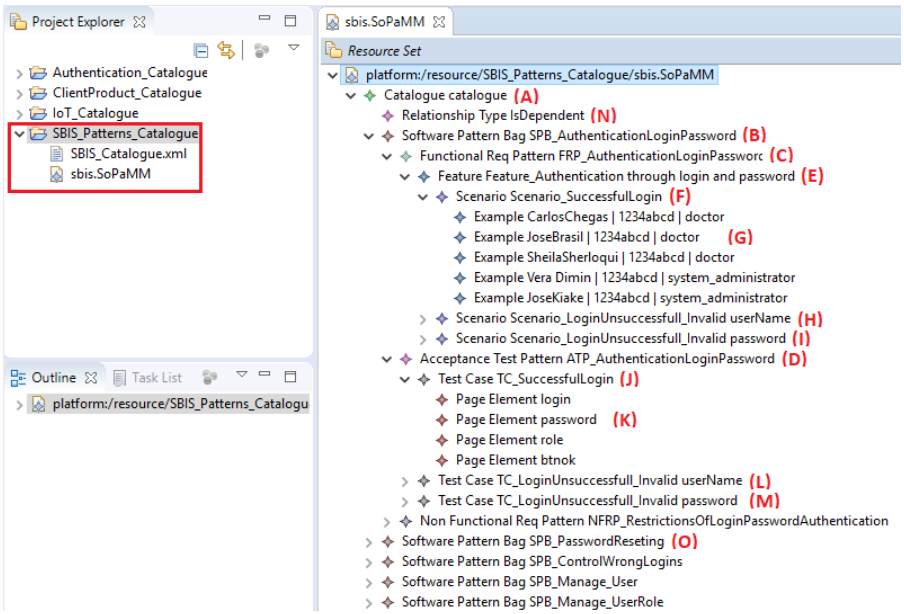

Figura 7. Catálogo de padrões para S-RES na ferramenta TMEd.

Password (H e I, respectivamente). Observe que os elementos do tipo *example* correspondem aos dados utilizados como entrada para os casos de testes, dados estes fornecidos pelo próprio manual de certificação da SBIS.

Já no padrão de teste de aceitação *ATP_AuthenticationLoginPassword* (D) da Figura 7, para cada cenário do padrão de requisito *FRP_AuthenticationLoginPassword* (E) existe um caso de teste (J, L e M) associado com elementos de interface *PageElement* (K). Esses elementos de interface podem ser utilizados para apoiar a automatização de testes de uma aplicação Web ou *mobile*, funcionalidade esta que não é do escopo da ferramenta TMEd.

Além das *bags* de padrões e dos padrões de software em si, a gramática do metamodelo SoPaMM permite a especificação de relacionamentos. No lado direito da Figura 7, tem-se a definição de um relacionamento de dependência, i.e., *RelationshipType IsDependent* (N). Com essa definição, estabeleceu-se uma relação de dependência entre duas *bags* de padrões, no caso, a *bag SPB_PasswordResetting* (O) só pode ser utilizada a partir da *bag SPB_AuthenticationLoginPassword* (B).

Por fim, os autores estão cientes de que a construção de catálogos de PRS com comportamento não é uma tarefa trivial. Para que sejam uma realidade no cotidiano de empresas, os catálogos devem ser elaborados com a ajuda de especialistas de domínio. Neste exemplo de uso da TMEd, o próprio manual de certificação da SBIS serve como compilação dos requisitos legais na visão de especialistas em S-RES, o que serviu de apoio à construção do catálogo realizado por um pesquisador com 15 anos de experiência em Engenharia de Requisitos.

5 Considerações Finais

Apesar das experiências positivas com o uso de PRS na Engenharia de Requisitos e da inerente associação entre requisitos e artefatos produzidos ao longo do ciclo de vida de software, um número reduzido de pesquisas tem investigado a adoção de PRS nas fases de *design*, construção, testes e manutenção.

Voltada para atender às lacunas descritas em [7,8,9], a ferramenta TMEd proposta neste artigo é um dos elementos de nossa abordagem de PRS, ao apoiar a elaboração de catálogos de padrões de requisitos integrados com padrões de teste por meio de especificações de comportamentos segundo as práticas do BDD.

A ferramenta TMEd é resultado de um esforço em andamento, não estando ainda madura para uso em produção, já que lhe falta, por exemplo, uma funcionalidade de gerenciamento de perfis de usuários. Com isso, os autores acreditam que o suporte oferecido pela TMEd seja corroborado por uma avaliação qualitativa junto a profissionais instruídos sobre a gramática do metamodelo SoPaMM.

Espera-se que, em breve, a ferramenta TMEd seja utilizada para produzir catálogos de PRS para outros domínios, além daquele desenvolvido para certificação de S-RES. À medida que novos catálogos forem produzidos, estes deverão ser disponibilizados publicamente, por exemplo, na forma como opera a iniciativa Diaspora[5], para maior disseminação da proposta de PRS com comportamento.

Os catálogos produzidos pela TMEd servirão de insumo para uma ferramenta em desenvolvimento, chamada *behavior-DRivEn Application Model generator* (DREAM). Essa ferramenta apoia a construção de modelos de aplicação usando os catálogos de PRS da TMEd como referência, assim como aquelas descritas em [1,3]. Sendo assim, a ideia é aproximar-se da indústria para investigar, em projetos reais, os benefícios do uso da abordagem de PRS com comportamento proposta. Um exemplo de parceria em andamento é a de membros da SBIS para avaliar o quanto o catálogo de padrões desenvolvido pode auxiliar no processo de certificação de S-RES.

Por fim, espera-se que todos esses esforços – o metamodelo SoPaMM, as ferramentas TMEd e DREAM e os catálogos de PRS com comportamento – ajudem a comunidade a melhor compreender o impacto do uso de PRS além da fase de Engenharia de Requisitos

Agradecimentos

Este estudo foi parcialmente financiado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

1. Barcelos, L.V., Penteado, R.: Especificação de requisitos no domínio de sistemas de informação com o uso de padrões. In: Proceedings of XIX Ibero-American Conference on Software Engineering, CIbSE 2016, Quito, Ecuador, April 27-29, 2016. pp. 338–351 (2016)

[5] Disponível online em `https://wiki.diasporafoundation.org/`.

2. Beckers, K., Côté, I., Goeke, L.: 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. Gyeongju, Republic of Korea (2014)
3. Beckers, K., Heisel, M., Côté, I., Goeke, L., Güler, S.: Structured pattern-based security requirements elicitation for clouds. In: 2013 International Conference on Availability, Reliability and Security. pp. 465–474 (Sep 2013). https://doi.org/10.1109/ARES.2013.61
4. Chelimsky, D., Astels, D., Helmkamp, B., North, D., Dennis, Z., Hellesoy, A.: The RSpec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends. Pragmatic Bookshelf, Raleigh, NC, 1st edn. (2010)
5. Franch, X., Palomares, C., Quer, C., Renault, S., De Lazzer, F.: A metamodel for software requirement patterns. In: Wieringa, R., Persson, A. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 85–90. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
6. Konrad, S., Cheng, B.H.C.: Requirements patterns for embedded systems. In: Proceedings IEEE Joint International Conference on Requirements Engineering. pp. 127–136 (2002)
7. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R.: Requirement patterns: A tertiary study and a research agenda. IET Software **14**(1), 18–26 (2020)
8. Kudo, T.N., Bulcão Neto, R.F., Macedo, A.A., Vincenzi, A.M.R.: Padrão de requisitos no ciclo de vida de software: Um mapeamento sistemático. In: Proceedings of the Conference: CIbSE2019 - Congresso Ibero Americano em Engenharia de Software. pp. 420–433. Havana, Cuba (2019)
9. Kudo, T.N., Bulcão Neto, R.F., Macedo, A.A., Vincenzi, A.M.R.: A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle. Journal of Software Engineering Research and Development **7**, 9:1–9:11 (dec 2019)
10. Kudo, T.N., Bulcão Neto, R.F., Vincenzi, A.M.R.: A conceptual metamodel to bridging requirement patterns to test patterns. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering. pp. 155–160. ACM, New York, NY, USA (2019). https://doi.org/10.1145/3350768.3351300
11. Kurtev, I., Bézivin, J., Jouault, F., Valduriez, P.: Model-based DSL frameworks. In: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications. pp. 602–616. OOPSLA '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1176617.1176632
12. Meszaros, G.: XUnit Test Patterns: Refactoring Test Code. Prentice Hall PTR, Upper Saddle River, NJ, USA (2006)
13. OMG: Meta Object Facility (MOF) Specification, version 1.4. Object Management Group, Inc. (2002)
14. Palomares, C., Quer, C., Franch, X.: Pabre-Man: Management of a requirement patterns catalogue. In: 2011 IEEE 19th International Requirements Engineering Conference. pp. 341–342 (Aug 2011). https://doi.org/10.1109/RE.2011.6051666
15. Palomares, C., Quer, C., Franch, X., Guerlain, C., Renault, S.: A catalogue of non-technical requirement patterns. In: 2012 Second IEEE International Workshop on Requirements Patterns (RePa). pp. 1–6 (Sep 2012). https://doi.org/10.1109/RePa.2012.6359969
16. Palomares, C., Franch, X., Quer, C.: Requirements reuse and patterns: A survey. In: Proceedings of the 20th International Working Conference on Requirements Engineering: Foundation for Software Quality - Volume 8396. pp. 301–308. REFSQ 2014, Springer-Verlag New York, Inc., New York, NY, USA (2014)

17. Palomares, C., Quer, C., Franch, X., Renault, S., Guerlain, C.: 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, Coimbra, Portugal, March 18-22, 2013. pp. 1260–1265 (2013)
18. da Silva, M.L., Junior, L.A.V.: Manual de certificação para sistemas de registro eletrônico em Saúde. Sociedade Brasileira de Informática em Saúde, 4.3 edn. (mar 2019)
19. Smart, J.: BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle. Manning Publications (2014), https://books.google.com.br/books?id=2BGxngEACAAJ
20. Supakkul, S., Hill, T., Chung, L., Tun, T.T., do Prado Leite, J.C.S.: An NFR pattern approach to dealing with NFRs. In: 2010 18th IEEE International Requirements Engineering Conference. pp. 179–188 (2010)
21. Wiegers, K.E., Beatty, J.: Software Requirements 3. Microsoft Press, Redmond, WA, USA (2013)
22. Withall, S.: Software Requirement Patterns. Best practices, Microsoft Press, Redmond, Washington (2007)

Capítulo 7

UM ARCABOUÇO PARA AVALIAÇÃO DE METAMODELOS DE SOFTWARE

Artigo publicado no Simpósio Brasileiro de Engenharia de Software - SBES 2020.

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Toward a Metamodel Quality Evaluation Framework: Requirements, Model, Measures, and Process. In: Proceedings of the XXXIV Brazilian Symposium on Software Engineering, SBES 2020, Rio Grande do Norte, Brazil, October. 2020.

Toward a Metamodel Quality Evaluation Framework: Requirements, Model, Measures, and Process

Taciana Novo Kudo*
Departmento de Computação
Universidade Federal de São Carlos
São Carlos, São Paulo, Brazil
taciana@ufscar.br

Renato F. Bulcão-Neto
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Goiás, Brazil
rbulcao@ufg.br

Auri M. R. Vincenzi
Departmento de Computação
Universidade Federal de São Carlos
São Carlos, São Paulo, Brazil
auri@ufscar.br

ABSTRACT

The quality of metamodel considerably affects the models and transformations that conform to it. Despite that, there is still little discussion about a comprehensive form to evaluate the quality of metamodels and its consequences in model-driven development processes. This paper proposes a metamodel quality evaluation framework called MQuaRE (Metamodel Quality Requirements and Evaluation). MQuaRE comprises metamodel quality requirements and measures, a quality model, and an evaluation process, with the evident influence of international standards for software product quality, such as ISO/IEC 25000 series. We present a simple use case of MQuaRE describing how requirements, measures, and the quality model should be used during the evaluation process of a metamodel for software patterns. Among other benefits, MQuaRE can help determine final metamodel quality, decide on the acceptance of a metamodel, and also assess the positive and negative aspects of a metamodel, contributing to its quality evolution.

CCS CONCEPTS

• **Software and its engineering** → **Model-driven software engineering**; • **General and reference** → *Evaluation.*

KEYWORDS

Model-driven software engineering, metamodeling, quality, requirement, model, measure, process, evaluation

ACM Reference Format:
Taciana Novo Kudo, Renato F. Bulcão-Neto, and Auri M. R. Vincenzi. 2020. Toward a Metamodel Quality Evaluation Framework: Requirements, Model, Measures, and Process. In *34th Brazilian Symposium on Software Engineering (SBES '20), October 21–23, 2020, Natal, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3422392.3422461

*Currently, Mrs. Kudo is a Ph.D. candidate at the Universidade Federal de São Carlos and also works as an assistant professor at the Universidade Federal de Goiás.

SBES '20, October 21–23, 2020, Natal, Brazil

ACM ISBN 978-1-4503-8753-8/20/09…$15.00
https://doi.org/10.1145/3422392.3422461

1 INTRODUCTION

Models are the primary artifacts of model-driven software engineering (MDSD) [1], and a terminal model is a representation that conforms to a given software metamodel [10, 13]. As the quality of a software metamodel directly impacts the quality of terminal models, software metamodel quality is an essential aspect of MDSD.

However, the literature reports a few proposals for metamodel quality evaluation, but most lack a general solution for the quality issue. Some efforts focus on quality measures [8], a quality evaluation model [12], or a quality evaluation model with structural measures borrowed from OO design [9, 11, 14]. Thus, we support there is a need for a more thorough solution for metamodel quality evaluation, with potential benefits to MDSD in general.

In this paper, we propose a metamodel quality evaluation framework called MQuaRE (Metamodel Quality Requirements and Evaluation). MQuaRE is an integrated framework composed of metamodel quality requirements and measures, a metamodel quality model, and an evaluation process, with a great contribution of the ISO/IEC 25000 series [4] for software product quality evaluation.

MQuaRe points out the relevance of quality requirements for metamodel evaluation. The quality model includes quality characteristics and sub-characteristics of metamodels. MQuaRe also provides general measures for metamodel quality. Finally, MQuaRE's evaluation process arranges requirements, model, and measures with activities, tasks, input and output artifacts, and users' roles. We also present a simple use case of the evaluation of the quality of a metamodel for software patterns [6].

This paper is organized as follows: Section 2 outlines related work; Section 3 describes the MQuaRE framework; Section 4 presents a use case of MQuaRE; and Section 5 brings our conclusions and future directions.

2 RELATED WORK

This section presents related work on metamodel quality evaluation.

Ma et al. [8] define a method and quality measures to assess UML-based metamodels. The proposal can identify and characterize the stability and design quality of metamodels, influenced mainly by the OO paradigm.

Strahonja [12] defines a quality evaluation model for workflow metamodels based on nine categories: domain application, origins, concepts and constructs, modeling language and notation, cohesion, openness, usability, maintainability, and pragmatic aspects. The use of the proposed model relies on the individual capabilities and competencies of the metamodel evaluators. This subjective aspect

of Strahonja's work is mainly due to the non-existence of measures associated with the quality categories.

Williams et al. [14] propose measures based on class diagram size (e.g., no. of classes), whereas Rocco and others [11] correlate metamodel measures, such as the no. of classes and inheritance.

Ma et al. [9] define an evaluation model composed of quality attributes, quality properties characterizing each attribute, and measures assigning a value to each property. Their model includes five quality attributes, each one with its respective properties. Besides, measures are defined based on quality measures for OO models.

In turn, MQuaRE is a comprehensive metamodel evaluation framework comprised of a process that combines a quality model, requirements, and measures. To the best of our knowledge, none of the previous work congregates MQuaRE's features.

3 THE MQUARE FRAMEWORK

This section details how MQuaRE components are arranged toward metamodel quality evaluation, as seen in Figure 1.

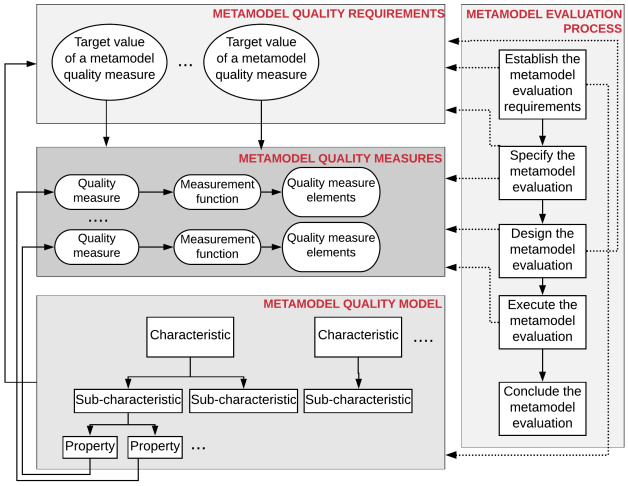

Figure 1: An overview of MQuaRE: evaluation process and quality requirements, measures, and model.

3.1 Metamodel Quality Requirements

Quality requirements specification plays a crucial role in the metamodel evaluation process. Should quality requirements are not stated clearly, the same metamodel may be interpreted and evaluated variously by different people. As a result, one achieves an inconsistent metamodel evaluation.

Metamodel quality requirements (MQR) may comprise multiples aspects of a metamodel, e.g., whether it is easy to use and maintain or compliant to specific standards, if applicable. MQR can be categorised through the MQuaRE's quality model we present in Section 3.2.

3.1.1 Pre-conditions for Quality Requirements. A quality model drives the documentation of metamodel quality requirements. Despite that, we recommend the following pre-conditions for MQR:

- MQR shall be uniquely identified and following the objective of the metamodel evaluation;
- MQR shall be associated with quality sub-characteristics, as defined in MQuaRE's quality model;
- MQR shall be specified in terms of a quality measure and a target value, which is the acceptable value for fulfilling a particular MQR;
- An acceptable tolerance value for the target value of a particular MQR shall be documented;
- Specific concepts and terms used in the metamodel should be used to avoid misunderstandings of the MQR;
- MQR shall be validated and approved by an evaluation requester.

The current version of MQuaRE provides 19 MQRs that meet those pre-conditions and can be reused by metamodel users.

3.1.2 Quality Requirements Verification. Defining the MQR is essential to avoid inconsistencies in the metamodel evaluation. Here is a list of recommendations to ensure the quality of MQR:

- MQR shall be verifiable, reviewed, and approved;
- Evaluation tools, techniques, or other resources (e.g., effort or time) required for verification shall be documented;
- Identified conflicts between MQR or between MQR and metamodel concepts shall be documented;
- The stakeholders' identities shall be documented.

Examples of MQR for the quality evaluation of a software pattern metamodel are shown in Section 4. Next, we present the MQuaRE's quality model, which works as guidance for the definition of MQR.

3.2 Metamodel Quality Model

The quality of a metamodel is the degree to which it provides value to a modeling activity. These stated needs are represented in the MQuaRE by a quality model that categorizes metamodel quality into *characteristics*, which in some cases, subdivide into *sub-characteristics*, as depicted in Figure 1. This hierarchical decomposition provides a convenient breakdown of metamodel quality.

The measurable quality-related properties of a metamodel are called *quality properties*. It is necessary to identify a collection of properties that cover characteristics or sub-characteristics, obtain quality measures for each, and combine them to achieve a derived quality measure corresponding to the quality characteristic or sub-characteristic. Thus, the quality model allows the categorization of metamodel quality requirements.

The quality model we propose revises ISO/IEC 25010:2011 [4], 9126-1:2001 [2] and related research [8, 9, 11, 12], and incorporates some quality characteristics and sub-characteristics with some amendments. Five characteristics form the MQuaRE's quality model as depicted in Figure 2: *Compliance*, *Conceptual Suitability*, *Usability*, *Maintainability*, and *Portability*. These characteristics may work as a checklist for ensuring comprehensive coverage of metamodel quality. Next, we overview the characteristics and sub-characteristics present in the MQuaRE's quality model.

3.2.1 Compliance. The degree to which the metamodel must comply with items such as widely accepted and sound theories, regulations, standards, and conventions. This characteristic includes conceptual compliance sub-characteristic.

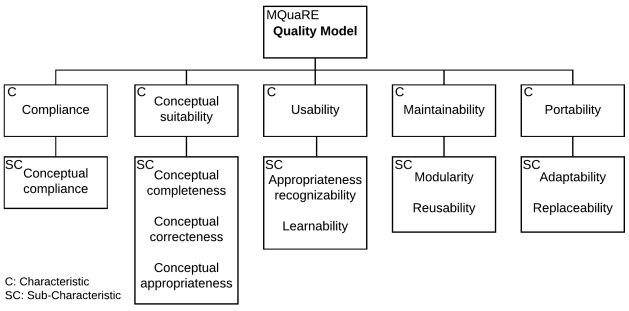

Figure 2: The MQuaRE's quality model with quality characteristics and sub-characteristics.

Conceptual compliance: the degree to which the metamodel to comply with such items as widely-accepted and sound theories, regulations, standards, and conventions concerning its conceptual foundation.

3.2.2 Conceptual Suitability. The degree to which the metamodel satisfies requirements when used under specified conditions. This characteristic subdivides into conceptual completeness, conceptual correctness, and conceptual appropriateness sub-characteristics.

Conceptual completeness: the degree to which the set of the metamodel concepts covers all the specified requirements.

Conceptual correctness: the degree to which the metamodel provides the correct modeling results with the needed degree of precision.

Conceptual appropriateness: the degree to which the metamodel facilitates the accomplishment of modeling tasks, and for determining their adequacy for performing these tasks.

3.2.3 Usability. The degree to which the metamodel can be used to achieve specified goals in a particular application domain. This characteristic includes the appropriateness recognizability and learnability sub-characteristics.

Appropriateness recognizability: the degree to which users can recognize whether the metamodel is appropriate for their needs or not.

Learnability: the degree to which the metamodel can be used by declared users to achieve specified goals of learning in a given context of use.

3.2.4 Maintainability. The degree of effectiveness and efficiency with which the metamodel can be modified by the intended maintainers. This characteristic includes modularity, reusability, and modifiability sub-characteristics.

Modularity: the degree to which the metamodel is composed of discrete concepts in such a way that a change of one concept has minimal impact on other concepts.

Reusability: the degree to which an asset can be used in more than one metamodel or in building other assets.

Modifiability: the degree to which the metamodel can be effectively and efficiently modified without introducing inconsistencies or degrading existing metamodel quality.

3.2.5 Portability. The degree of effectiveness and efficiency with which the metamodel can be transferred from one application domain to another. This characteristic includes adaptability and replaceability sub-characteristics.

Adaptability: the degree to which the metamodel can effectively and efficiently be adapted for different application domains.

Replaceability: the degree to which the metamodel can replace another specified metamodel for the same purpose in the same application domain.

The MQuaRE's quality model provides a basis for quantifying metamodel quality requirements through metamodel quality measures that we present next.

3.3 Metamodel Quality Measures

The quality characteristics and sub-characteristics can be quantified by applying *measurement functions*. A measurement function is an algorithm used to combine quality measure elements. The result of applying a measurement function is called a *quality measure*. In this way, quality measures are quantifications of the quality characteristics and sub-characteristics.

The current version of MQuaRE includes 23 quality measures bound to the quality model as follows: *Compliance* (2), *Conceptual Suitability* (4), *Usability* (5), *Maintainability* (8), and *Portability* (4). Every quality measure is described by an identification code, the measure name, a description of the information provided by the measure, and a measurement function.

Our quality measures are the result of an analysis of related work [8, 9, 11, 12] and the ISO/IEC 25023:2016 [5] and ISO/IEC 9126-3:2003 [3] standards. It is also noteworthy that the metamodel quality measures depend on the purpose of the evaluation, the selected quality characteristics, and the possibility to apply the measurements.

3.4 Quality Evaluation Process

The MQuaRE's process model assumes that the evaluation founds on the MQuaRE's requirements, making clear the objectives and criteria of assessment. Besides, the MQuaRE's quality model and measures should also be considered in the evaluation process.

Figure 3 depicts a BPMN-based representation for the MQuaRE's evaluation process with activities, user roles, and input and output artifacts. Activities and the respective tasks are detailed next.

3.4.1 Establish the evaluation requirements. The metamodel quality evaluation requirements must be identified, taking into account the evaluation purpose. The aim is to meet all quality requirements of the metamodel and considering restrictions, such as the evaluation purpose and target date. The following should be input for this activity: (i) metamodel quality evaluation needs; (ii) metamodel to be evaluated, including its specifications; and (iii) applicable evaluation tools and methodology. This activity consists of the following tasks:

(1) **Establish the objective of evaluating the metamodel**: the purpose of the metamodel quality evaluation shall be documented as a basis for further evaluation activities and

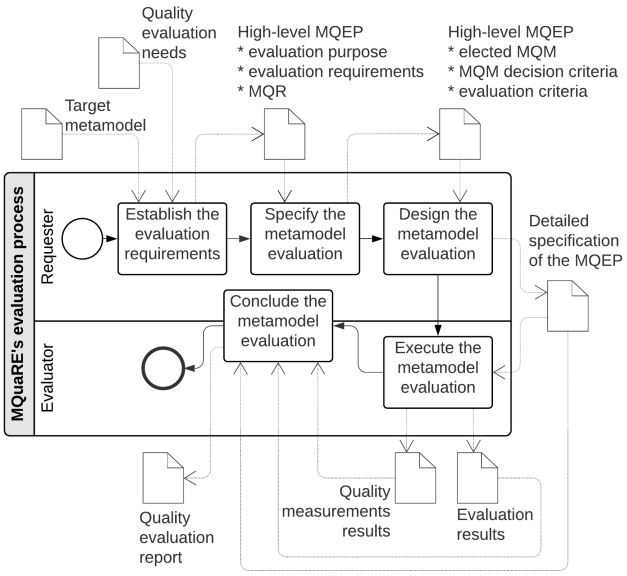

Figure 3: The MQuaRE's Metamodel Evaluation Process.

tasks. The targeted metamodel may be an intermediate version or a final version. Multiple evaluation purposes may take place such as deciding on the acceptance or assuring the quality of an intermediate metamodel version, the comparison between distinct metamodels for a same domain, assessing the positive and negative effects of a final metamodel version, among others.

(2) **Define the metamodel quality requirements**: these shall be specified using the MQuaRE's quality characteristics and sub-characteristics. Should a specification of MQR is available, it may be reused, reviewed, and refined.
(3) **Identify the artifacts to be used in the evaluation**: every metamodel-related artifact required for the evaluation shall be identified and registered. Examples of metamodel artifacts include metamodel specifications (e.g., requirements- and design-oriented), metamodel implementation, metamodel user documentation, just to name a few.

As illustrated in Figure 3, the main output artifact of this activity is a *high-level metamodel quality evaluation plan* (MQEP), which should contain the specification of purposes and requirements for metamodel quality evaluation as well as the specification of metamodel quality requirements.

3.4.2 Specify the metamodel evaluation. This activity consumes a high-level MQEP as input and consists of the following tasks:

(1) **Select the metamodel quality measures**: the requester shall select MQM's to cover all metamodel quality requirements. Measurement procedures should be measured with sufficient accuracy to allow criteria to be set and comparisons to be made.
(2) **Define decision criteria for metamodel quality measures**: decision criteria are numerical thresholds or targets used to determine the need for action or further investigation or to describe the level of confidence in a given result. These shall be defined for the selected individual measures often concerning quality requirements.
(3) **Establish decision criteria for metamodel evaluation**: the requester should prepare a procedure for further summarization, with separate criteria for different quality characteristics, each of which may be in terms of individual quality sub-characteristics and measures. The summarization results should be used as a basis for the metamodel quality assessment. Therefore, the evaluation results of the different characteristics shall be summarized to assess the quality of the metamodel.

The primary output artifact of this activity is a *revised high-level MQEP*, containing the chosen quality measures as well as decision criteria for metamodel quality measures and assessment.

3.4.3 Design the metamodel evaluation. The revised high-level MQEP previously presented is input data for this activity, which includes the following task:

(1) **Plan metamodel evaluation activities**: in this task, those metamodel quality evaluation activities identified shall be scheduled, taking into account the availability of resources such as personnel, evaluation tools, and examples of metamodel application domains. The evaluation plan should detail the following elements: the purpose of the metamodel quality evaluation, quality evaluation requirements including metamodel artifacts and related resources (e.g., personnel, tools, budget, and deadlines), MQR and metamodel quality measures, decision criteria for metamodel evaluation and metamodel quality measures, and an evaluation schedule.

As the evaluation activities evolve, the evaluation plan shall be revised until a thorough level plan. The outcome of this activity is a *detailed specification of the MQEP*.

3.4.4 Execute the metamodel evaluation. A thorough specification of the MQEP represents the input artifact for this activity that consists of the following tasks:

(1) **Compute metamodel quality measurements**: the selected metamodel quality measures shall be applied to the metamodel following the evaluation plan. As a result, values on the measurement scales are computed and assigned to quality measures.
(2) **Apply decision criteria for metamodel quality measures**: the decision criteria for the metamodel quality measures shall be applied to the measured values.
(3) **Apply decision criteria for metamodel quality assessment**: the set of decision criteria shall be summarized into quality characteristics and sub-characteristics. A statement of the extent to which the metamodel meets quality requirements describes the assessment results, which should:
   - establish an appropriate degree of confidence that the metamodel can meet the evaluation requirements;
   - identify any specific deficiencies concerning the evaluation requirements and any additional evaluations needed to determine the scope of those deficiencies;

- identify any particular limitations or conditions placed on the use of the metamodel;
- identify any weaknesses or omissions in the evaluation and any additional evaluation that is needed.

The following should be outcomes of this activity: *metamodel quality measurements results* and *quality evaluation results*.

3.4.5 Conclude the metamodel evaluation. This activity requires as input the detailed specification of MQEP, metamodel quality measurements results, and quality evaluation results. It consists of the following tasks:

(1) **Review metamodel evaluation results**: the evaluator and the evaluation requester shall carry out a joint review of the evaluation results.
(2) **Conclude the metamodel evaluation process**: depending on how the evaluation report is to be used, it should include the following items, among others: the MQEP, computed measurements results, performed analyses, intermediate results or interpretation decisions, the evaluators' profiles, the final result of the metamodel quality evaluation, and any necessary information to be able to repeat or reproduce the assessment.

The final outcome is a *metamodel quality evaluation report*.

4 USE CASE

This section presents a plain use case of the MQuaRE framework. As depicted in Figure 3, the evaluation process begins with the definition of the targeted metamodel and the primary reason for evaluation. In this case, we aimed to estimate the final quality of a metamodel called SoPaMM (Software Pattern MetaModel) [6], which defines how software patterns, in general, can be organized, classified, related, and represented.

A non-detailed MQEP was then created, including the SoPaMM artifacts and related resources both required for the evaluation. For this purpose, we developed a suite of software artifacts documenting the SoPaMM's development: requirements specification, design specification, metamodel implementation, and user documentation. Human resources for the evaluation included two evaluators; in this case, the SoPaMM's and the MQuaRE's original creators.

Next, we describe an excerpt of the list of MQR chosen for the SoPaMM metamodel:

MQR2. The metamodel must cover the concepts[1] found in its specifications.
MQR7. The evaluator must be able to recognize whether a metamodel is appropriate for their needs through evident concepts found in the specifications.
MQR8. The evaluator must be able to recognize whether the metamodel concepts' purpose is correctly interpreted through the specifications.
MQR13. The evaluator must be able to recognize modifications in the metamodel based on documented changes in the specifications.

[1]Concepts are foundation elements of a metamodel, e.g., a concept may be a class, relationship, or attribute in a UML metamodel. As we propose MQuARE as a generic-purpose framework, concepts may vary according to the specification language of the metamodel under evaluation.

Next, we refined MQEP with MQuaRE's quality measures required for addressing the SoPaMM's quality requirements (i.e., 2, 7, 8, and 13). Given that every measure is associated with a sub-characteristic in the MQuaRE's quality model, we show in Table 1 the correspondence among MQR, measure, and quality sub-characteristic and characteristic. In sequence, Table 2 details each measure required for the evaluation of the SoPaMM. Also, we assigned a target value and an acceptable tolerance value to each quality measure in the evaluation of the SoPaMM.

As we chose multiple quality characteristics for evaluation, we defined a quality formula as the weighted mean of measures grades per characteristic. Equation 1 presents a final quality (FQ) formula with the measures grades described in Table 2:

$$FQ = 10 * ((CCp1 \quad 3 * ((UAp3 \quad UAp4)/2) \quad MMd1)/5) \quad (1)$$

, i.e., in this hypothetical case, usability (UAp-3 and UAp-4) is three times more relevant than conceptual suitability (CCp1) and maintainability (MMd1).

From this point, we elaborated a detailed specification of the MQEP, including information about MQR, quality measures, measurement functions, the correct interpretation of measurement results, and the SoPaMM metamodel's specifications. All this body of information was organized into a simple schedule with the resources available (i.e., evaluators and tool support).

Table 3 shows the calculus of each measure based on the SoPaMM's documentation and a non-subjective interpretation of each measurement value. Based on this information, SoPaMM's final quality score is 9.2, which was carefully revised by the evaluators later.

5 CONCLUSIONS AND FUTURE WORK

We target a general solution for metamodel quality evaluation, most influenced by general standards. The idea is to establish a general quality approach which later, based on our experimentation, can also be instantiated for a more specific context. We think having a general quality evaluation model can make even different metamodels comparable through some general quality metrics.

It is noteworthy that the MQuaRE requirements, model, and measures are flexible and, thus, can be adapted to the specific contexts of a metamodel. That is, new characteristics and sub-characteristics can be included in the MQuaRE's quality model as well as corresponding measures and requirements. The MQuaRE framework should serve as a guide for metamodel quality evaluation, but it should not hamper execution.

We have already finished the development of the whole framework and several supporting documents to facilitate MQuaRE usage in practice. For this, we make available the full documentation of MQuaRE [7], including a detailed description of the 19 quality requirements and 23 quality measures, and the specific contributions from related work and international quality standards.

The validation of the MQuaRE framework is a work in progress so that we comprehend how much consistent MQuaRE is regarding its requirements. Soon, MQuaRE will also be evaluated through a survey with experts in metamodeling or metamodel quality. A tool to support the MQuaRE framework is also under development.

Table 1: Evaluation of the SoPaMM metamodel: the correspondence among MQR, measures, and quality model.

Requirement	Measure	Sub-characteristic	Characteristic
MQR2	Conceptual coverage	Conceptual completeness	Conceptual Suitability
MQR7	Evident concepts	Appropriateness recognizability	Usability
MQR8	Concept understandability	Appropriateness recognizability	Usability
MQR13	Conceptual stability	Modifiability	Maintainability

Table 2: Evaluation of the SoPaMM metamodel: quality measures' id, name, description, and measurement function.

ID	Measure	Measure description	Measurement function
CCp-1	Conceptual coverage	What proportion of the specified concepts has been modeled?	X = 1 - A / B A = No. of missing concepts B = No. of concepts described in the specification 0 <= X <= 1 (the closer to 1, the more complete)
UAp-3	Evident concepts	What proportion of metamodel concepts are evident to the user?	X = A / B A = No. of concepts evident to the user B = No. of concepts described in the specification 0 <= X <= 1 (the closer to 1, the better)
UAp-4	Concept understandability	What proportion of metamodel concepts are correctly understood by a first-time user without prior study or training or seeking external assistance?	X = A / B A = No. of concepts correctly understood by the evaluator B = No. of concepts described in the specification 0 <= X <= 1 (the closer to 1, the better)
MMd-1	Conceptual stability	How stable is the metamodel specification during the development life cycle?	X = 1 - A / B A = No. of concepts changed during development life cycle B = No. of concepts described in the specification 0 <= X <= 1 (the closer to 1, the more stable)

Table 3: Evaluation of the SoPaMM metamodel: the respective results for each quality measure.

Measure	Measurement	Results interpretation
Conceptual coverage	A = 0; B = 20; **X = 1**	The SoPaMM's implementation covers all specified concepts
Evident concepts	A = 18; B = 20; **X = 0.9**	90% of the SoPaMM's concepts are evident to the user
Concept understandability	A = B = 20; **X = 1**	A user with no prior help correctly understands all of SoPaMM's concepts
Conceptual stability	A = 5; B = 20; **X = 0.75**	75% of the SoPaMM's concepts remain stable after changes

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The authors thank reviewers for their valuable comments.

REFERENCES

[1] Markus Herrmannsdorfer and Guido Wachsmuth. 2014. *Evolving Software Systems.* Springer, Berlin, Heidelberg, Chapter Coupled Evolution of Software Metamodels and Models, 33–63.

[2] ISO/IEC. 2001. ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model. *ISO/IEC 9126-1:2001* 1 (2001), 1–25.

[3] ISO/IEC. 2003. ISO/IEC TR 9126-3:2003 Software engineering — Product quality — Part 3: Internal metrics. *ISO/IEC 9126-3:2003* 2 (2003), 1–62.

[4] ISO/IEC. 2014. ISO/IEC 25000:2014 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE. *ISO/IEC 25000:2014* 2 (2014), 1–27.

[5] ISO/IEC. 2016. 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 (2016), 1–45.

[6] Taciana Novo Kudo, Renato F. Bulcão Neto, and 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*. ACM, New York, NY, USA, 155–160. https://doi.org/10.1145/3350768.3351300

[7] Taciana Novo Kudo, Renato F. Bulcão-Neto, and Auri M. R. Vincenzi. 2020. Metamodel Quality Requirements and Evaluation (MQuaRE). arXiv:2008.09459

[8] Haohai Ma, Weizhong Shao, Lu Zhang, Zhiyi Ma, and Yanbing Jiang. 2004. Applying OO Metrics to Assess UML Meta-models. In *UML 2004 — The Unified Modeling Language. Modeling Languages and Applications*, Thomas Baar, Alfred Strohmeier, Ana Moreira, and Stephen J. Mellor (Eds.). Springer, Berlin, Heidelberg, 12–26.

[9] Zhiyi Ma, Xiao He, and Chao Liu. 2013. Assessing the quality of metamodels. *Frontiers of Computer Science* 7, 4, Article 558 (2013), 12 pages.

[10] OMG. 2002. Meta Object Facility (MOF) Specification, Version 1.4. *Object Management Group, Inc.* (2002).

[11] Juri Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2014. Mining metrics for understanding metamodel characteristics. In *Proceedings of the 6th International Workshop on Modeling in Software Engineering (MiSE 2014)*. ACM, New York, NY, USA, 55–60.

[12] Vjeran Strahonja. 2007. The Evaluation Criteria of Workflow Metamodels. In *29th International Conference on Information Technology Interfaces*. IEEE, New York, NY, USA, 553–558.

[13] Éric Vépa, Jean Bézivin, Hugo Bruneliere, and Frédéric Jouault. 2006. Measuring model repositories. In *Model Size Metrics Workshop - a MODELS 2006 Satellite Event*. Springer, Genoa, Italy, 1–5.

[14] James Williams, Athanasios Zolotas, Nicholas Matragkas, Louis Rose, Dimitios Kolovos, Richard Paige, and Fiona Polack. 2013. What do metamodels really look like?. In *Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling*. CEUR-WS, Miami, USA, 55–60.

Capítulo 8

ALINHANDO REQUISITOS E TESTES USANDO METAMODELAGEM E PADRÕES: PROJETO E AVALIAÇÃO

Artigo submetido ao Requirements Engineering Journal.

KUDO, T. N.; BULCÃO-NETO, R. F.; GRACIANO NETO, V. V.; VINCENZI, A. M. R. Aligning requirements and testing through metamodeling and patterns: Design and evaluation. Requirements Engineering v. XX. n. XX, p. 1-20. 2020. (submetido para).

Noname manuscript No.
(will be inserted by the editor)

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

Taciana Novo Kudo · Renato Bulcão-Neto · Valdemar Vicente Graciano Neto · Auri Marcelo Rizzo Vincenzi

Received: date / Accepted: date

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. Consider that 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 SoPaMM (Software Pattern MetaModel), 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

Departamento de Computação, Universidade Federal de São Carlos
São Carlos, São Paulo, Brazil
Instituto de Informática, Universidade Federal de Goiás
Goiânia, Goiás, Brazil
E-mail: taciana@ufg.br (*corresponding author*)

Instituto de Informática, Universidade Federal de Goiás
Goiânia, Goiás, Brazil

Departamento de Computação, Universidade Federal de São Carlos
São Carlos, São Paulo, Brazil

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 [9,42].

In the last decade, requirements reuse [6,7,12,15,10] 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 [44]. Usually, RP guides requirements elicitation and specification through well-defined templates that can be reused in later specifications [8,27,1].

A promising approach for representing RP is through metamodeling [2] because it raises the level of abstraction at which software is conceived, implemented, and evolved. As related work, Franch et al. [11] 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. [47]'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 towards 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 [22].

For instance, consider the alignment between RE and testing. The popular V-model [37] 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 companies still struggle with the cost, rework, and delay effects resulting from a weak alignment between requirements and testing [4].

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 SoPaMM (Software Pattern MetaModel) [19]. Currently, SoPaMM aligns functional requirement patterns (FRP) and acceptance test patterns (ATP), which structure generic testing solutions to recurrent behaviors arising from different scenarios [29] and help a tester understand the context of a testing practice [28]. 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 [5] by describing FRP via user stories and associating it with behaviors through the Gerkhin language.

As metamodel quality impacts the terminal models' quality[1], we evaluated SoPaMM using the Metamodel Quality Requirements and Evaluation (MQuaRE) framework [23] that comprises an evaluation process, metamodel quality require-

[1] Terminal models are metamodel instances as defined in the Meta-Object Facility (MOF) architecture [31].

ments 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 three-fold:

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.

The organization of this paper is as follows: Section 2 discusses related work, Section 3 presents background, Section 4 describes the SoPaMM metamodel and its behavior-driven approach, Section 5 presents the evaluation of the metamodel using the metamodel quality assessment framework and threats to validity, and Section 6 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.

Two proposals [11,43] cover both functional and non-functional requirement patterns (FRP and NFRP, respectively), whereas one work represents one type of requirement pattern (NFRP [47]). To the best of the authors' knowledge, the SoPaMM metamodel we propose is the only one proposing the alignment of functional requirement patterns (FRP) and acceptance test patterns (ATP).

Multiple representation formats have been used: traditional natural language (NL) [11,47], and controlled natural language (CNL) with a subset of meaningful terms for pattern representation [43]. 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 [38] for requirements and behaviors specification, respectively.

Tools are an effective strategy to assist practitioners' practices in the use of requirement patterns. In previous work [21], we report the development of TMEd (Terminal Model Editor), 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 [11,33] 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 [11,34,8]. 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 [19,21]. In this paper, we go further by evaluating the quality of the SoPaMM metamodel using a quality evaluation framework [23,18].

Table 1 Metamodeling approaches for requirement patterns

Ref.	Pattern	Formalism	Tool	Evaluation
[43]	(N)FRP	CNL	-	-
[47]	NFRP	NL	-	-
[11]	(N)FRP	NL	PABRE-Man	Catalogues
SoPaMM	(N)FRP ATP	CNL	TMEd	Catalogues and quality

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 [30]. 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 [5]. 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 software 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 what is happening

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 "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. Features' description as user stories integrated with behaviors as executable scenarios is BDD's contribution to our metamodeling proposal.

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 [23].

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 [18].

The MQuaRE's quality model categorizes the MQRs into five characteristics (C) subdivided into eleven sub-characteristics (SC) described in Figure 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 Figure 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 [23, 18].

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 accomplishment 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

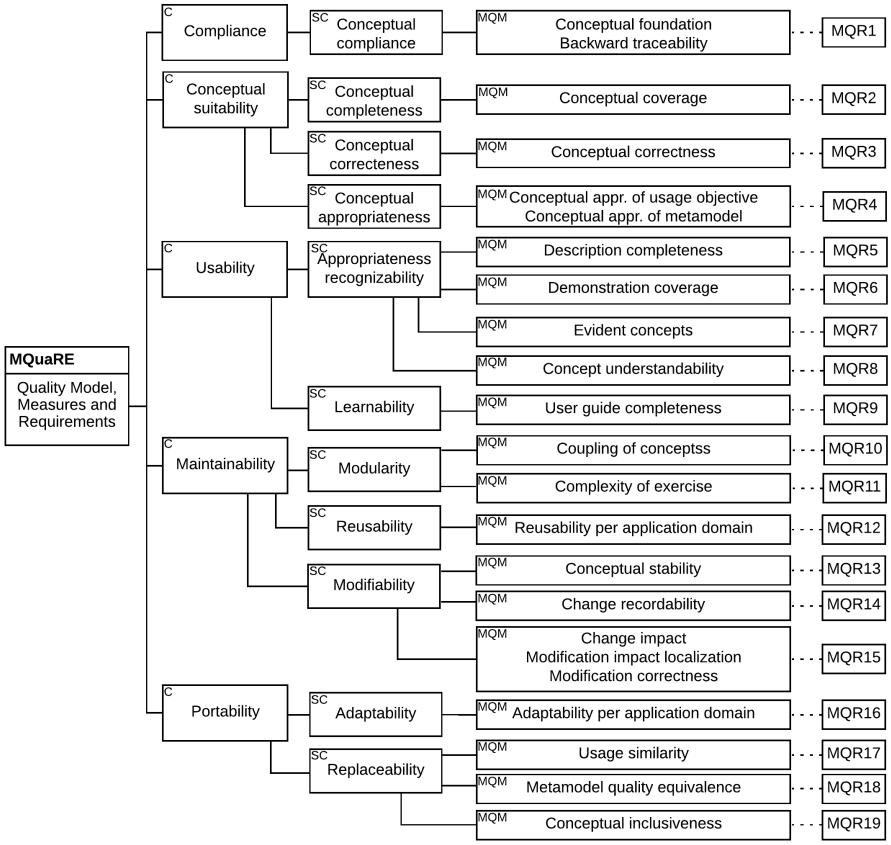

Fig. 1 The MQuaRE quality model, measures and requirements.

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.

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 define the MQRs according to the general evaluation purpose (e.g., estimate the final quality of 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 the 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, to name a few.
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 Section 5.

4 The SoPaMM Metamodel

The first important question to answer is what the structure of the behavior-driven requirement pattern is. Figure 2 depicts the most significant components of a functional requirement pattern (FRP) related to an acceptance test pattern (ATP). For the sake of conciseness, some FRP and ATP metadata is not shown.

Influenced by the BDD agile methodology, a *Functional Requirement Pattern* (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 Figure 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.*

```
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
```

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

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

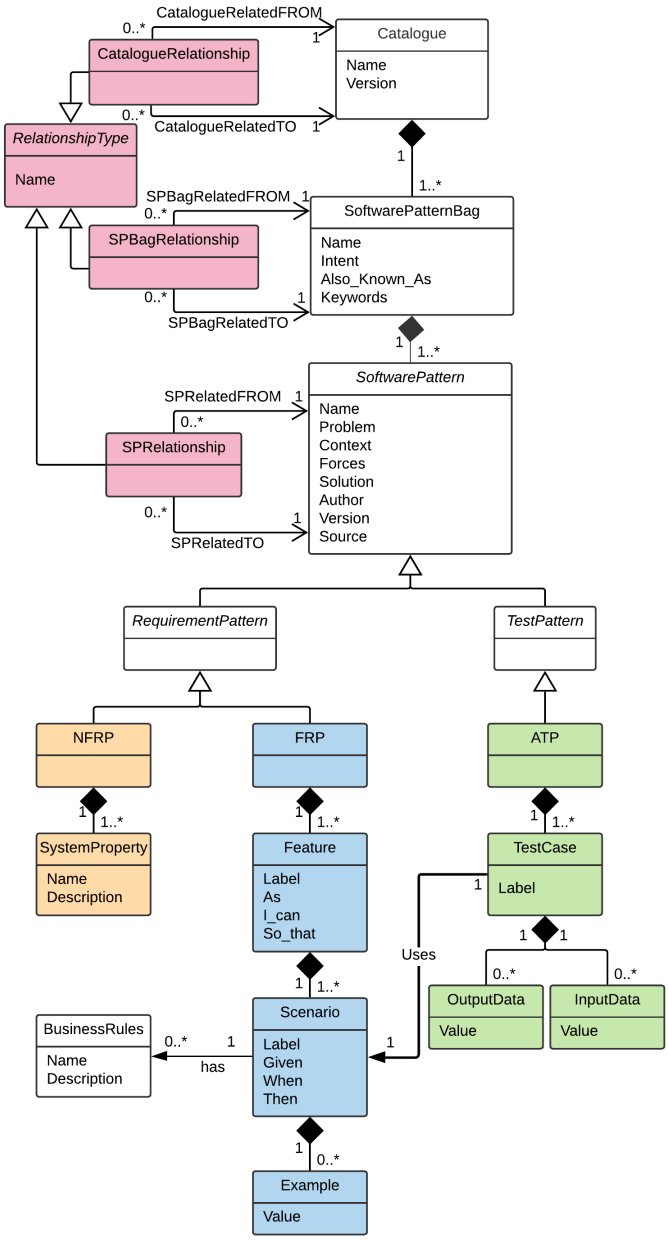

Fig. 3 The SoPaMM metamodel.

Once presented the FRP and ATP concepts, we outline the entire SoPaMM metamodel illustrated in Figure 3. *Catalogue* is a means of systematically gathering patterns, usually addressing the most common problems for a particular application domain. This 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 literature, the SPB concept in SoPaMM allows organizing, in a same catalogue, software patterns for problems at different stages of the SDLC.

Note that industry standards [32] and classic literature on software patterns [13, 35, 14, 44] 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*) [44], 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 (*CatalogueRelationship*, *SPBagRelationship*, and *SPRelationship*, respectively).

Non-functional Requirement Pattern (NFRP) is a reuse unit that states general properties (behavioral constraints or quality attributes). 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 (i.e., a behavioral constraint for user authentication).

Noteworthy that this is an enhanced version of SoPaMM. The main differences from its previous versions [19, 21] 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 user interface (UI) elements represented by the *Page Object* design pattern [24, 40]. UI elements were of two types: for web and mobile applications. However, the removal of UI elements in the current version makes the SoPaMM metamodel more flexible and technology-independent.

5 Quality Evaluation

This section details the quality evaluation of the SoPaMM metamodel using the MQuaRE framework described in Section 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.

5.1 Evaluation Planning and Design

As evaluation requesters, the SoPaMM's developers performed the first three activities of the MQuaRE's evaluation process described in Section 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 Figure 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 Figure 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, 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's 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 [19, 21]; 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.

[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 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 correcteness	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

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.

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.

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 correcteness	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-4	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

5.2 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 de-

scribed 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 SoPaMM's concepts' changes 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.

5.3 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 not only about SoPaMM but also 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 catalogs 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 show 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.

5.4 Threats to Validity

The validity of experiment results depends on experiment settings, and it can be of four types [45]: 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 small number of participants might have negatively affected

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 the ISO/IEC standards for metamodel quality purposes, this missing information in the evaluation plan may have hampered the SoPaMM's portability results. Furthermore, mostly impacted by the covid-19 outbreak, the participants did not receive extensive training, but only the evaluation plan, supporting artifacts, tutorial videos, and the evaluation report to be filled in.

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 [25, 41, 26, 36] and international standards for software quality, such as ISO/IEC 25010 [16] and ISO/IEC 25023 [17].

External validity concerns the generalization of research findings outside the experiment setting. Once again, we selected a heterogeneous group as a representative population, despite the group size. 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.

6 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 MQuaRE.

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

1. Most of SoPaMM's quality properties were well-judged, namely: conceptual compliance, completeness, correctness, appropriateness, and learnability, appropriateness recognizability, reusability, and adaptability.
2. Although the quality in use evaluation was not performed, the participants experienced and approved the alignment of requirements and testing through pattern catalogs built upon SoPaMM. High scores of SoPaMM's reusability and adaptability suggest subjects' approval.
3. 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.
4. Maybe not all metamodels should exhibit all quality properties present in MQuaRE, as noted by the expert subject.
5. MQuaRE-aware tool support would undoubtedly be helpful.
6. The most significant threats to this study's validity include the low number of participants and the lack of more in-depth training on evaluating a metamodel using MQuaRE.

As future work, we plan to enhance SoPaMM's capabilities by bridging non-functional requirement patterns (NFRP) to test patterns. NFRP is a subject widely investigated in the literature [1,46,39,3] but often restricted to requirements engineering and not other software life cycle phases.

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 catalogs generated by TMEd are input for another tool we have been working on, called DREAM (behavior-DRivEn Application Model generator). 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 [20].

Finally, we learned that a software tool could better assist MQuaRE users' tasks. A wizard would guide evaluation requesters towards 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.

References

1. Amorndettawin, M., Senivongse, T.: Non-functional requirement patterns for agile software development. In: Proceedings of the 2019 3rd International Conference on Software and E-Business, ICSEB 2019, p. 66–74. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3374549.3374561. URL https://doi.org/10.1145/3374549.3374561

2. Baudry, B., Nebut, C., Traon, Y.L.: Model-driven engineering for requirements analysis. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 459–459 (2007)
3. Beckers, K., Côté, I., Goeke, L.: 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 (2014)
4. Bjarnason, E., Borg, M.: Aligning requirements and testing: Working together toward the same goal. IEEE Softw. **34**(1), 20–23 (2017). DOI 10.1109/MS.2017.14. URL `https://doi.org/10.1109/MS.2017.14`
5. Chelimsky, D., Astels, D., Helmkamp, B., North, D., Dennis, Z., Hellesoy, A.: The RSpec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, 1st edn. Pragmatic Bookshelf, Raleigh, NC (2010)
6. Cheng, B.H.C., Atlee, J.M.: Current and future research directions in requirements engineering. In: K. Lyytinen, P. Loucopoulos, J. Mylopoulos, B. Robinson (eds.) Design Requirements Engineering: A Ten-Year Perspective, pp. 11–43. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
7. Chernak, Y.: 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, pp. 46–53. IEEE Computer Society, Los Alamitos, CA, USA (2012)
8. Costal, D., Franch, X., López, L., Palomares, C., Quer, C.: On the use of requirement patterns to analyse request for proposal documents. In: A.H.F. Laender, B. Pernici, E. Lim, J.P.M. de Oliveira (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 (2019). DOI 10.1007/978-3-030-33223-5_45. URL `https://doi.org/10.1007/978-3-030-33223-5\_45`
9. Franch, X.: Software requirements patterns: A state of the art and the practice. In: Proceedings of the 37th International Conference on Software Engineering - Volume 2, ICSE '15, pp. 943–944. IEEE Press, Piscataway, NJ, USA (2015)
10. Franch, X., Palomares, C., Quer, C.: Industrial practices on requirements reuse: An interview-based study. In: N.H. Madhavji, L. Pasquale, A. Ferrari, S. Gnesi (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 (2020). DOI 10.1007/978-3-030-44429-7_6. URL `https://doi.org/10.1007/978-3-030-44429-7\_6`
11. Franch, X., Palomares, C., Quer, C., Renault, S., De Lazzer, F.: A metamodel for software requirement patterns. In: R. Wieringa, A. Persson (eds.) Requirements Engineering: Foundation for Software Quality, pp. 85–90. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
12. Fricker, S., Grau, R., Zwingli, A.: Requirements Engineering: Best Practice, pp. 25–46. Springer International Publishing (2015)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
14. Haskins, C.: Using patterns to share best results - a proposal to codify the sebok. INCOSE International Symposium **13**(1), 15–23 (2003)
15. Irshad, M., Petersen, K., Poulding, S.: A systematic literature review of software requirements reuse approaches. Inf. Softw. Technol. **93**(C), 223–245 (2018)
16. 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)
17. 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)
18. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R.: Metamodel Quality Requirements and Evaluation (MQuaRE). arXiv e-prints arXiv:2008.09459 (2020)
19. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R.: A conceptual metamodel to bridging requirement patterns to test patterns. In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering, SBES 2019, p. 155–160. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3350768.3351300. URL `https://doi.org/10.1145/3350768.3351300`

20. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R.: Requirement patterns: a tertiary study and a research agenda. IET Softw. **14**(1), 18–26 (2020). DOI 10.1049/iet-sen.2019.0016. URL https://doi.org/10.1049/iet-sen.2019.0016
21. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R.: Uma ferramenta para construção de catálogos de padrões de requisitos com comportamento. In: G.D.S. Hadad, J.H. Pimentel, I.S.S. Brito (eds.) Anais do WER20 - Workshop em Engenharia de Requisitos, São José dos Campos, SP, Brasil, August 24-28, 2020. Editora PUC-Rio (2020). URL http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER20/12_WER_2020_paper_16.pdf
22. Kudo, T.N., Bulcão-Neto, R.F., Macedo, A.A., Vincenzi, A.M.R.: A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle. Journal of Software Engineering Research and Development **7**, 9:1–9:11 (2019). DOI 10.5753/jserd.2019.458. URL https://sol.sbc.org.br/journals/index.php/jserd/article/view/458
23. Kudo, T.N., Bulcão-Neto, R.F., Vincenzi, A.M.R.: Toward a metamodel quality evaluation framework: Requirements, model, measures, and process. In: Proceedings of the XXXIV Brazilian Symposium on Software Engineering, SBES 2020, Rio Grande do Norte, Brazil, October 19-23, 2020, SBES 2020. Association for Computing Machinery, New York, NY, USA (2020)
24. Leotta, M., Clerissi, D., Ricca, F., Spadaro, C.: Improving test suites maintainability with the page object pattern: An industrial case study. In: ICST Workshops, pp. 108–113. IEEE Computer Society, Washington, DC, USA (2013)
25. Ma, H., Shao, W., Zhang, L., Ma, Z., Jiang, Y.: Applying oo metrics to assess uml metamodels. In: T. Baar, A. Strohmeier, A. Moreira, S.J. Mellor (eds.) UML 2004 — The Unified Modeling Language. Modeling Languages and Applications, pp. 12–26. Springer, Berlin, Heidelberg (2004)
26. Ma, Z., He, X., Liu, C.: Assessing the quality of metamodels. Frontiers of Computer Science **7**(4), 558 (2013)
27. Macasaet, R.J., Noguera, M., Rodríguez, M.L., Garrido, J.L., Supakkul, S., Chung, L.: Micro-business requirements patterns in practice: Remote communities in developing nations. J. Univers. Comput. Sci. **25**(7), 764–787 (2019). URL http://www.jucs.org/jucs_25_7/micro_business_requirements_patterns
28. Meszaros, G.: XUnit Test Patterns: Refactoring Test Code. Prentice Hall PTR, Upper Saddle River, NJ, USA (2006)
29. Moreira, R.M.L.M., Paiva, A.C.R.: 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 (2014)
30. Oliveira, G., Marczak, S., Moralles, C.: How to evaluate BDD scenarios' quality? In: I. do Carmo Machado, R. Souza, R.S.P. Maciel, C. Sant'Anna (eds.) Proceedings of the XXXIII Brazilian Symposium on Software Engineering, SBES 2019, Salvador, Brazil, September 23-27, 2019, pp. 481–490. ACM (2019). DOI 10.1145/3350768.3351301. URL https://doi.org/10.1145/3350768.3351301
31. OMG: Meta object facility (mof) specification, version 1.4. Object Management Group, Inc. (2002)
32. OMG: Structured patterns metamodel standard. OMG - Object Management Group (2017)
33. Palomares, C., Quer, C., Franch, X.: 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 (2011). DOI 10.1109/RE.2011.6051666. URL https://doi.org/10.1109/RE.2011.6051666
34. Palomares, C., Quer, C., Franch, X., Renault, S., Guerlain, C.: 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, NY, USA (2013)
35. Rising, L.: Patterns: A way to reuse expertise. IEEE Communications Magazine **37**(4), 34–36 (1999)
36. Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: 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, NY, USA (2014)
37. Rook, P.: Controlling software projects. Software Engineering Journal **1**, 7 (1986)

38. Smart, J.F.: BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle, first edn. Manning Publications (2014)
39. de Souza Cunha, H., do Prado Leite, J.C.S., Duboc, L., Werneck, V.: 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 (2013). DOI 10.1109/RePa.2013.6602668. URL https://doi.org/10.1109/RePa.2013.6602668
40. Stocco, A., Leotta, M., Ricca, F., Tonella, P.: 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, NJ, USA (2015)
41. Strahonja, V.: The evaluation criteria of workflow metamodels. In: 29th International Conference on Information Technology Interfaces, pp. 553–558. IEEE, New York, NY, USA (2007)
42. Tockey, S.: Insanity, hiring, and the software industry. Computer **48**, 96–101 (2015)
43. Videira, C., da Silva, A.R.: Patterns and metamodel for a natural-language-based requirements specification language. In: O. Belo, J. Eder, J.F. e Cunha, O. Pastor (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 (2005). URL http://ceur-ws.org/Vol-161/FORUM_31.pdf
44. Withall, S.: Software Requirement Patterns. Best practices. Microsoft Press, Redmond, Washington (2007)
45. Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A.: Experimentation in Software Engineering. Springer Publishing Company, Incorporated (2012)
46. Xuan, X., Wang, Y., Li, S.: Privacy requirements patterns for mobile operating systems. In: L. Zhao, J.C.S. do Prado Leite, S. Supakkul, L. Chung, Y. Wang (eds.) 4th IEEE International Workshop on Requirements Patterns, RePa 2014, Karlskrona, Sweden, August 26, 2014, pp. 39–42. IEEE Computer Society (2014). DOI 10.1109/RePa.2014.6894842. URL https://doi.org/10.1109/RePa.2014.6894842
47. Ya'u, B., Nordin, A., Salleh, N.: 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 (2016)

Capítulo 9

CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo são apresentadas as nossas conclusões, contribuições, lições aprendidas e propostas de trabalhos futuros.

9.1 Conclusão

O problema de pesquisa tratado nesta tese envolve a carência de trabalhos que apoiam a integração de PRS ao longo do CVS, em especial na fase de testes. Conclui-se que é possível demonstrar o alinhamento entre requisitos e testes por meio de metamodelagem, que permite o reúso integrado de padrões de requisitos e de testes de software, PRS e PTS, respectivamente. Este metamodelo é usado como referência para criação de modelos de catálogos de PRS e PTS, que por sua vez são usados para geração de especificações de requisitos com comportamento associado. Neste trabalho, o reuso ocorre quando PRS são selecionados e os casos de testes já estão integrados e disponíveis para uso na forma de PTS.

A Figura 9.1 ilustra as principais contribuições desta pesquisa:

- Revisão da literatura sobre os estados da arte e da prática de PRS, na forma de um estudo terciário, resultando em uma agenda de pesquisa sobre o tema (Kudo et al., 2020a);
- Mapeamento sistemático sobre uso de PRS no CVS (Kudo et al., 2019b,c), motivado pelo item 2 da agenda de pesquisa mencionada – resultados apontaram a carência de estudos de PRS em outras etapas do CVS;
- Especificação, projeto e construção do metamodelo SoPaMM (Kudo et al., 2019a), motivado pelo item 5 da agenda de pesquisa e indiretamente pelos resultados do mapeamento sistemático;

- Especificação, projeto e construção da ferramenta TMEd para criação de catálogos de padrões baseados no metamodelo SoPaMM (Kudo et al., 2020b), motivada pelo item 6 da agenda de pesquisa e indiretamente pelos resultados do mapeamento sistemático;
- Especificação do framework MQuaRE (Kudo et al., 2020d) para avaliação de qualidade de metamodelos, motivado pelo item 4 da agenda de pesquisa; e
- Avaliação de qualidade do metamodelo SoPaMM utilizando o framework de avaliação de qualidade MQuaRE (Kudo et al., 2020c).

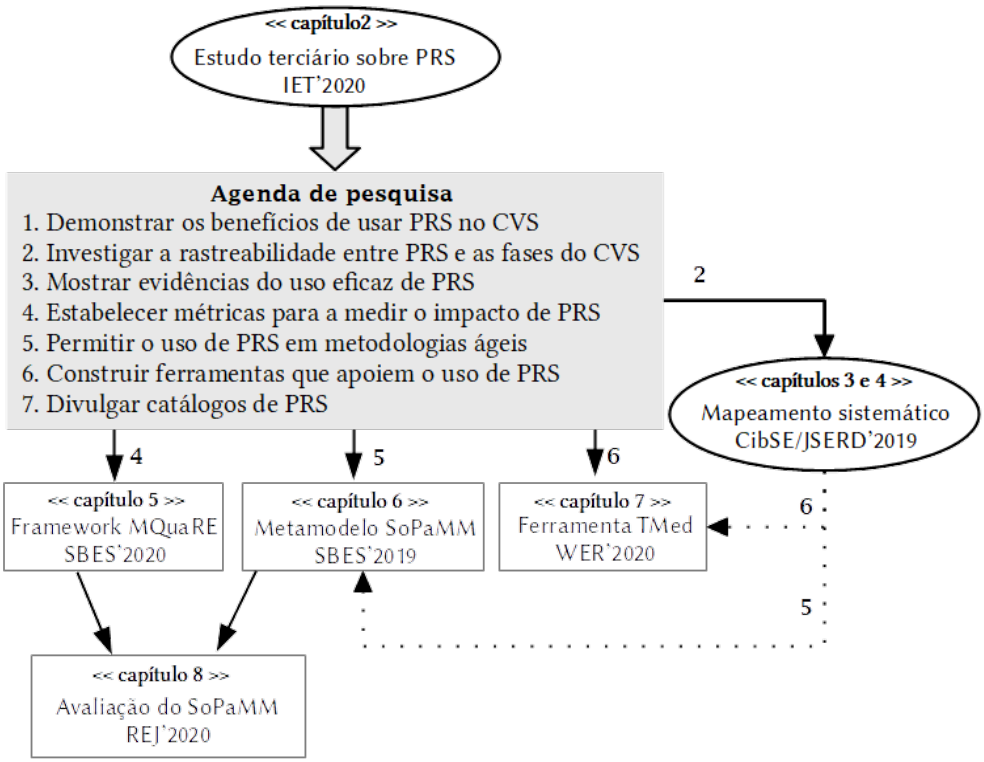

Figura 9.1: Principais contribuições deste trabalho de doutorado.

9.2 Lições aprendidas

Algumas lições aprendidas ao longo do desenvolvimento deste trabalho, incluem:

- Ao propor o metamodelo SoPaMM, foi importante seguir padrões, fundamentos e diretrizes (por exemplo, MOF e SPMS) para que tenha embasamento reconhecido.
- Durante a proposta de um metamodelo SoPaMM, ciclos incrementais de instanciação foram realizados construindo catálogos válidos. Novos conceitos emergiram

à medida que o metamodelo SoPaMM foi usado na prática como referência para a criação de catálogos.

- Problemas não identificados na construção do metamodelo SoPaMM foram identificados com a avaliação de sua qualidade, o que poderia impactar na qualidade dos catálogos gerados a partir do SoPaMM.
- É imprescindível uma documentação detalhada do metamodelo SoPaMM para que ele seja amplamente divulgado, facilmente compreendido e avaliado.
- O uso de padrões de requisitos não é aconselhável para todo tipo de sistema. Sistemas que exigem certificação, que precisam ser auditados, ou que possuem funcionalidades comuns a vários tipos de sistemas são bons candidatos para serem modelados como PRS.
- A qualidade de metamodelos em geral pode ser medida pelas propriedades de conformidade, completude, correção e adequação conceituais, facilidade de aprendizado, modularidade, reusabilidade, adaptabilidade e capacidade de substituição.
- Nem todos os metamodelos precisam contemplar todas as propriedades de qualidade presentes no MQuaRE; isto dependerá do objetivo da avaliação do metamodelo.
- A avaliação de qualidade com apoio do MQuaRe é uma tarefa que pode ser facilitada por meio de uma ferramenta de software.

9.3 Trabalhos futuros

É possível vislumbrar algumas propostas de trabalhos futuros a partir do que foi construído ao longo desta pesquisa. Na Figura 9.2 são ilustradas as ideias de trabalhos futuros enumeradas e integradas com o que já foi desenvolvido até o momento (em destaque, SoPaMM, TMEd, Catálogos e MQuaRE).

1. Estender o metamodelo SoPaMM para tratar outros padrões de requisitos não funcionais integrados com PTS.
2. Estender o metamodelo SoPaMM para inclusão de outros tipos de padrões de software.
3. Aprimorar a ferramenta TMEd para permitir a busca por catálogos de PRS e PTS.

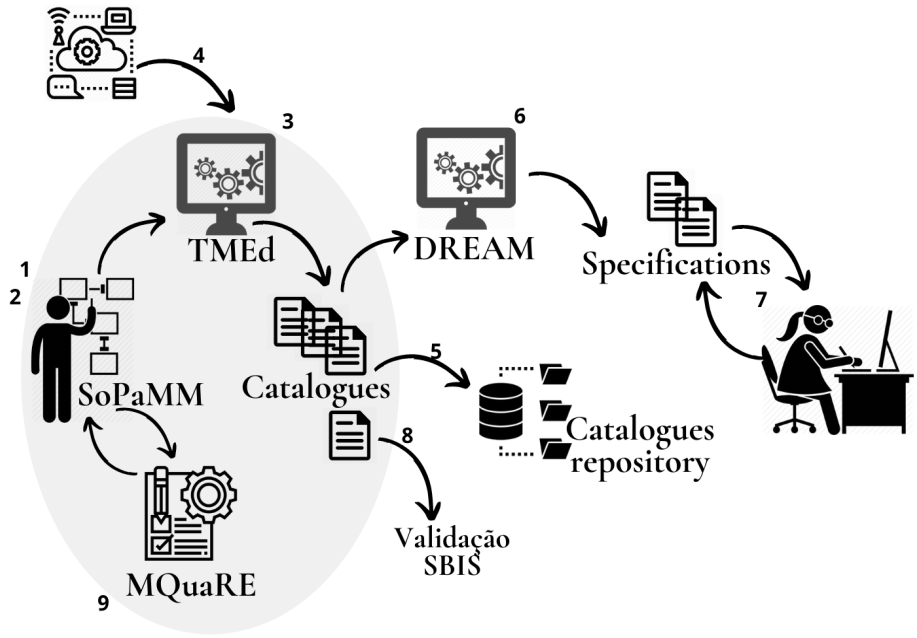

Figura 9.2: Trabalhos futuros.

4. Acrescentar à ferramenta TMEd: um mecanismo de coleta de especificações em repositórios on-line, e algoritmos de mineração de texto sobre essas especificações; dessa forma, TMEd poderia servir de apoio a serviços de descoberta e de recomendação de padrões, como PRS e PTS.

5. Criar um repositório de catálogos de padrões de software que utilize o metamodelo SoPaMM, como modelo de referência, e a ferramenta TMEd, para produção dos catálogos - motivada pelo item 7 da agenda de pesquisa citada na Figura 9.1.

6. Utilizar os catálogos produzidos pela TMEd para apoiar a geração automática de especificações de requisitos e de testes. Essa pesquisa tem sido desenvolvida no contexto de trabalhos de mestrado e de iniciação científica, nos quais desenvolve-se a ferramenta DREAM (*behavior-DRivEn Application Model generator*) (Ribeiro et al., 2020).

7. Fomentar, junto a empresas de desenvolvimento de software, o uso prático das especificações de requisitos e de testes geradas pela ferramenta DREAM, especificações estas baseadas em catálogos de padrões produzidos pela ferramenta TMEd e que seguem a gramática do metamodelo SoPaMM - motivada pelos itens 1 e 3 da agenda de pesquisa citada na Figura 9.1.

8. Validar o catálogo de padrões de requisitos de Sistemas de Registro Eletrônico de Saúde (S-RES) com membros da Sociedade Brasileira de Informática em Saúde (SBIS) e, posteriormente, disponibilizá-lo para as comunidades de interesse. Esta

iniciativa encontra-se em andamento no âmbito de um trabalho de pesquisa de mestrado.

9. Melhorar o processo de avaliação de qualidade de metamodelos utilizando o MQuaRE, por exemplo, por meio do desenvolvimento de uma ferramenta de software.

De acordo com as idealizações de trabalhos futuros, espera-se contribuir na orientação de trabalhos que promovam avanços no estado da arte e da prática sobre padrões de requisitos de software em geral, e de sua integração com outros tipos de padrões de software, como os padrões de teste.

REFERÊNCIAS

ASSAR, S. Model driven requirements engineering: Mapping the field and beyond. 2014.

BARROS-JUSTO, J. L.; BENITTI, F. B. V.; LEAL, A. C. Software patterns and requirements engineering activities in real-world settings: A systematic mapping study. *Computer Standards & Interfaces*, v. 58, p. 23–42, 2018.

BAUDRY, B.; NEBUT, C.; TRAON, Y. L. Model-driven engineering for requirements analysis. In: *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, p. 459–459, 2007.

BECKERS, K.; CÔTÉ, I.; GOEKE, L. A catalog of security requirements patterns for the domain of cloud computing systems. In: *Proceedings of the ACM Symposium on Applied Computing*, p. 337–342, 2014.

BJARNASON, E.; BORG, M. Aligning requirements and testing: Working together toward the same goal. *IEEE Softw.*, v. 34, n. 1, p. 20–23, 2017.
Disponível em `https://doi.org/10.1109/MS.2017.14`

BOURQUE, P.; FAIRLEY, R. E., eds. *SWEBOK: Guide to the software engineering body of knowledge*. Version 3.0 ed. Los Alamitos, CA: IEEE Computer Society, 2014.

CHELIMSKY, D.; ASTELS, D.; HELMKAMP, B.; NORTH, D.; DENNIS, Z.; HELLESOY, A. *The rspec book: Behaviour driven development with rspec, cucumber, and friends*. 1st ed. Raleigh, NC: Pragmatic Bookshelf, 2010.

CHERNAK, Y. 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*, p. 46–53, 2012.

COSTAL, D.; FRANCH, X.; LÓPEZ, L.; PALOMARES, C.; QUER, C. On the use of requirement patterns to analyse request for proposal documents. In: LAENDER, A. H. F.; PERNICI, B.; LIM, E.; DE OLIVEIRA, J. P. M., eds. *Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, November 4-7, 2019, Proceedings*, Springer, p. 549–557, 2019 (*Lecture Notes in Computer Science*, v.11788).
Disponível em `https://doi.org/10.1007/978-3-030-33223-5_45`

FRANCH, X. Software requirements patterns: A state of the art and the practice. In: *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, Piscataway, NJ, USA: IEEE Press, p. 943–944, 2015 (*ICSE '15*,).

FRANCH, X.; PALOMARES, C.; QUER, C. Industrial practices on requirements reuse: An interview-based study. In: MADHAVJI, N. H.; 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]*, Springer, p. 78–94, 2020 (*Lecture Notes in Computer Science*, v.12045).
Disponível em https://doi.org/10.1007/978-3-030-44429-7_6

FRANCH, X.; PALOMARES, C.; QUER, C.; RENAULT, S.; DE LAZZER, F. A metamodel for software requirement patterns. In: WIERINGA, R.; PERSSON, A., eds. *Requirements Engineering: Foundation for Software Quality*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 85–90, 2010.

IRSHAD, M.; PETERSEN, K.; POULDING, S. A systematic literature review of software requirements reuse approaches. *Inf. Softw. Technol.*, v. 93, n. C, p. 223–245, 2018.

KONRAD, S.; CHENG, B. H. Requirements patterns for embedded systems. In: *Proceedings IEEE Joint International Conference on Requirements Engineering*, Essen, Germany: IEEE, p. 127–136, 2002.

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. A conceptual metamodel to bridging requirement patterns to test patterns. In: *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES 2019, New York, NY, USA: Association for Computing Machinery, p. 155–160, 2019a (*SBES 2019*,).
Disponível em https://doi.org/10.1145/3350768.3351300

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Requirement patterns: a tertiary study and a research agenda. *IET Softw.*, v. 14, n. 1, p. 18–26, 2020a.
Disponível em https://doi.org/10.1049/iet-sen.2019.0016

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Uma ferramenta para construção de catálogos de padrões de requisitos com comportamento. In: HADAD, G. D. S.; PIMENTEL, J. H.; BRITO, I. S. S., eds. *Anais do WER20 - Workshop em Engenharia de Requisitos, São José dos Campos, SP, Brasil, August 24-28, 2020*, Editora PUC-Rio, 2020b.
Disponível em http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER20/12_WER_2020_paper_16.pdf

KUDO, T. N.; BULCÃO-NETO, R.; GRACIANO NETO, V. V.; VINCENZI, A. M. R. Aligning requirements and testing through metamodeling and patterns: Design and evaluation. *Requirements Engineering*, p. 1–20, submetido para, 2020c.

KUDO, T. N.; BULCÃO-NETO, R. F.; MACEDO, A. A.; VINCENZI, A. M. Padrão de requisitos no ciclo de vida de software: Um mapeamento sistemático. In: *XXII Ibero-American Conference on Software Engineering, CIbSE 2019*, New York, USA: Curran Associates, Inc., p. 420–433, 2019b.

KUDO, T. N.; BULCÃO-NETO, R. F.; MACEDO, A. A.; VINCENZI, A. M. R. A revisited systematic literature mapping on the support of requirement patterns for the software development life cycle. *Journal of Software Engineering Research and Development*, v. 7, p. 9:1–9:11, 2019c.
Disponível em `https://sol.sbc.org.br/journals/index.php/jserd/article/view/458`

KUDO, T. N.; BULCÃO-NETO, R. F.; VINCENZI, A. M. R. Toward a metamodel quality evaluation framework: Requirements, model, measures, and process. In: *Proceedings of the XXXIV Brazilian Symposium on Software Engineering, SBES 2020, Rio Grande do Norte, Brazil, October 19-23, 2020*, New York, NY, USA: Association for Computing Machinery, 2020d.

LONIEWSKI, G.; INSFRAN, E.; ABRAHÃO, S. A systematic review of the use of requirements engineering techniques in model-driven development. In: PETRIU, D. C.; ROUQUETTE, N.; HAUGEN, Ø., eds. *Model Driven Engineering Languages and Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, p. 213–227, 2010.

MACASAET, R. J.; NOGUERA, M.; RODRÍGUEZ, M. L.; GARRIDO, J. L.; SUPAKKUL, S.; CHUNG, L. Micro-business requirements patterns in practice: Remote communities in developing nations. *J. Univers. Comput. Sci.*, v. 25, n. 7, p. 764–787, 2019.
Disponível em `http://www.jucs.org/jucs_25_7/micro_business_requirements_patterns`

MOREIRA, R. M. L. M.; PAIVA, A. C. R. 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*, Lisbon, Portugal: IEEE, p. 126–135, 2014.

MYERS, G. J.; SANDLER, C. *The art of software testing*. USA: John Wiley & Sons, Inc., 2004.

PALOMARES, C. Definition and use of software requirement patterns in requirements engineering activities. In: *International Working Conference on Requirements*

Engineering: Foundation for Software Quality, p. 60–66, 2014.
Disponível em `http://ceur-ws.org/Vol-1138/ds3.pdf`

PALOMARES, C.; QUER, C.; FRANCH, X. Requirements reuse and requirement patterns: a state of the practice survey. *Empirical Software Engineering*, v. 22, n. 6, p. 2719–2762, 2017.

PALOMARES, C.; QUER, C.; FRANCH, X.; RENAULT, S.; GUERLAIN, C. 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, Coimbra, Portugal, March 18-22, 2013*, p. 1260–1265, 2013.

RIBEIRO, P. Q.; KUDO, T. N.; VINCENZI, A. M. R.; BULCÃO-NETO, R. F. Automatic generation of requirements and test cases specifications based on behavior-driven requirement patterns. *Journal of Software Engineering Research and Development*, p. 1–17, a submeter para, 2020.

ROOK, P. Controlling software projects. *Software Engineering Journal*, v. 1, n. 1, p. 7–, 1986.

TOCKEY, S. Insanity, hiring, and the software industry. *Computer*, v. 48, n. 11, p. 96–101, 2015.

WIEGERS, K.; BEATTY, J. *Software requirements*. third ed. Microsoft Press, 2013.

WITHALL, S. *Software requirement patterns*. Best practices. Redmond, Washington: Microsoft Press, 2007.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014*, p. 38:1–38:10, 2014.

YA'U, B.; NORDIN, A.; SALLEH, N. 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*, Jakarta, Indonesia: ICT4M, p. 188–193, 2016.