LEMON: A Tool for Enhancing Software Requirements Communication through Requirements Pattern-based Modelling Assistance

David Mosquera[1,*], Oscar Pastor[2] and Jürgen Spielberger[1]

[1]Zürich University of Applied Sciences, Gertrudstrasse 15, Winterthur 8400, Switzerland
[2]Universitat Politècnica de València, Camí de Vera s/n, Valencia 46022, Spain

Abstract

Context and motivation. Guaranteeing effective software requirement communication among stakeholders and software developers is crucial to ensure stakeholders' satisfaction. Nevertheless, communicating software requirements is error-prone due to misunderstandings between stakeholders and software developers, producing misalignment between the expected solution and the final product. **Problem.** Some authors have proposed software requirements patterns to effectively specify requirements, relying on a reusable experience-based approach. However, industry studies show that using requirements patterns is not yet an established practice due to the lack of tools and scalability of existing solutions. **Solution.** This tool & demonstration paper presents the first version of the architecture of LEMON, a software requirements tool to assist requirements specification based on reusable and documented requirements patterns. In this paper, we exemplify LEMON's components and present a tool prototype through a use case demonstration in the context of digital health software development. **Results and conclusions.** We conclude our paper by reflecting on the challenges, opportunities, and next research efforts towards maximising LEMON's industrial adoption based on practitioners' feedback gathered through an online survey.

Keywords

Requirements patterns, Modelling assistant, Low-code, No-code, Digital Health use case

1. Introduction

Software developers face the challenge of effectively gathering requirements from stakeholders (a.k.a. requirement elicitation phase) to guarantee quality resulting software [1]. However, communicating software requirements is error-prone due to human-related factors, such as the technical knowledge gap between stakeholders and software developers. Some authors have proposed requirements patterns to address such a challenge. Their approaches rely on reusing experience-based and well-documented requirements patterns [2] to reduce the effort of eliciting software requirements from scratch. Based on this premise, some authors have stated that using requirements patterns positively impacts resulting software regarding quality, including requirement communication (a.k.a. requirement identification) [3]. As a result, authors have proposed requirements pattern templates [4], [5], [6], specification languages [7], [8], and catalogues [1], [9], [10], [11] to achieve such benefits.

In industry, software developers have also adopted requirements patterns in their daily work [12], [13]. Nevertheless, the software development industry has taken a different direction. Software developers in the industry have resorted to manual and rudimentary methods to capitalise on software requirements patterns, mainly relying on copying and pasting textual descriptions [12]. These rudimentary solutions prevent the benefits of more sophisticated alternatives to exploit requirements patterns, such as those

In: D. Mendez, A. Moreira, J. Horkoff, T. Weyer, M. Daneva, M. Unterkalmsteiner, S. Bühne, J. Hehn, B. Penzenstadler, N. Condori-Fernández, O. Dieste, R. Guizzardi, K. M. Habibullah, A. Perini, A. Susi, S. Abualhaija, C. Arora, D. Dell'Anna, A. Ferrari, S. Ghanavati, F. Dalpiaz, J. Steghöfer, A. Rachmann, J. Gulden, A. Müller, M. Beck, D. Birkmeier, A. Herrmann, P. Mennig, K. Schneider. Joint Proceedings of REFSQ-2024 Workshops, Doctoral Symposium, Posters & Tools Track, and Education and Training Track. Co-located with REFSQ 2024. Winterthur, Switzerland, April 8, 2024.

*Corresponding author.
mosq@zhaw.ch (D. Mosquera); opastor@disc.upv.es (O. Pastor); spij@zhaw.ch (J. Spielberger)
0000-0002-0552-7878 (D. Mosquera); 0000- 0002-1320-8471 (O. Pastor); 0000-0003-2617-3535 (J. Spielberger)

proposed in the literature. This behaviour can be explained by the fact that adapting proposals from the literature to different software development contexts appears to be more of an art than a science in software requirements patterns and reuse [2].

Considering these limitations, we propose LEMON (acronym for **L**anguage for sp**E**cifiying and **MO**delling patter**N**s). LEMON is a modelling assistant that enhances requirement communication based on requirements patterns. LEMON allows for the specification and modification of requirements patterns using the LEMON language, a Domain Specific Language (DSL) for requirements patterns integrated with the ECORE metamodel. Requirements patterns specified in the LEMON language are then compiled in an execution environment to allow integration into low-code/no-code (LNC) tools for subsequent software prototype generation. In this paper, we refer to a software prototype as the generated set of LNC models representing an instance of a requirement pattern. This early prototyping supports the communication loops between stakeholders and software developers, aiming to reduce miss-alignments between stakeholders' expectations and the final software. We designed LEMON using the concepts already explored by authors in the literature, such as templates, requirements pattern specification languages, and catalogues of patterns, encapsulating them into a modelling assistant integrated with LNC tools.

This tool & demonstration paper exemplifies how LEMON can be used using a use case in the context of digital health software. Finally, we reflect on the challenges of adopting LEMON in an industrial environment based on an online survey collecting practitioners' opinions about LEMON. Based on the initial results from experimentation and practitioners' feedback, we observe that LEMON has the potential to enhance requirement communication among stakeholders and software developers employing requirements patterns. We discuss future work, including further developments and validation efforts to demonstrate the value of LEMON.

This tool and demonstration paper is structured as follows: In Section 2, we introduce our use case to exemplify LEMON; In Section 3, we present the LEMON architecture and tool; and, finally, in Section 5, we draw some conclusions and reflect on preliminary evaluation data.

2. The Need to Reuse Requirements for Enhancing Requirement Communication: a Use Case at Whatscount GmbH

Whatscount GmbH (WGmbH) is a young and innovative Swiss start-up in the field of digitalisation that has dedicated itself to creating digital health software. WGmbH has broad experience in developing such software and has software products, including EPDs (Electronic Patient Dossiers), patient appointment systems, and patient visualisation test data. In addition, this start-up has found great added value in developing using an LNC tool rather than traditional programming tools. Therefore, the developers of WGmbH do not program but model.

Requirements patterns at WGmbH. A few years back, software developers at WGmbH noticed their stakeholders often communicated software requirements they had addressed earlier. For instance, TestCorp, BioTest, and PatientTest have requested different visualisations for patient test data. TestCorp required software for visualising antibiotic-resistant tests, BioTest for blood cell counts, and PatientTest for viral infection tests. Despite varied sources and challenges, all have a common requirement: visualising test data. WGmbH observed recurring LNC models describing such requirements across projects. To prevent misunderstandings in future communications, they adopted "requirements patterns," leveraging similarities and relying on previous experience.

Limitations on adopting requirements patterns at WGmbH. After deciding to implement requirements patterns in their start-up, WGmbH encountered limitations with available literature solutions, initially exploring templates and pattern catalogues. However, these focused on text-level requirements, not aligning with WGmbH's LNC approach. Efforts to transform text to LCN models outweighed the benefits, prompting consideration of existing pattern specification languages. Unfortunately, these were often model-specific (fixed to notations such as UML or BPMN), hindering compatibility with WGmbH's LNC models. Then, how could WGmbH fully benefit from requirements

patterns in their software development context (i.e., developing digital health software using LNC tools)? Figure 1 shows the gap in a desired solution based on requirement patterns for WGmbH.

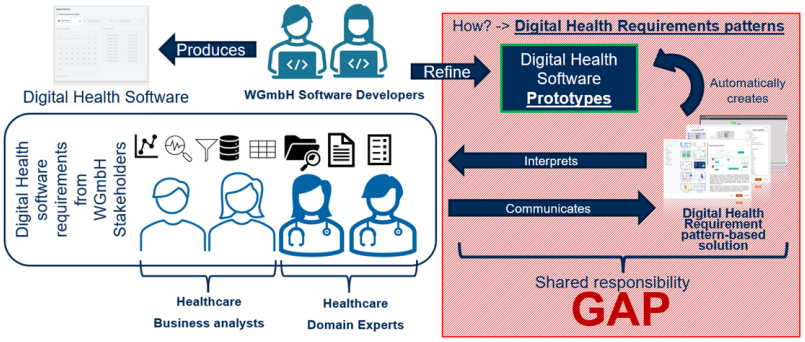

Figure 1: Gap in a desired digital health requirements patterns solution for WGmbH.

3. LEMON: a Requirements Pattern-Based Modelling Assistant

The limitations outlined in Section 2 call for a novel solution. This motivates us to propose LEMON. LEMON is a modelling assistant. Modelling assistants are tools, methods, or techniques that aim to assist in one or more tasks during software modelling (e.g., requirements communication). In this case, we conceive LEMON as a tool aiming to reduce the complexity of software requirement communication, transforming stakeholder input into LNC models using requirements patterns. LEMON has two users: stakeholders and software developers. Stakeholders are the ones that interact with LEMON to communicate their requirements. We refer to them in Section 2 and Figure 1 as business analysts and domain experts in digital health. On the other hand, software developers are in charge of configuring and populating LEMON with requirement patterns. In Section 2 and Figure 1, we refer to them as the software developers from WGmbH.

4. LEMON Architecture

We have designed LEMON using a module-based architecture. We reused concepts available in literature to design LEMON, such as pattern specification languages, templates, and catalogues combined with a modelling assistant interaction. Up to this point, LEMON contains four modules (see Figure 2):

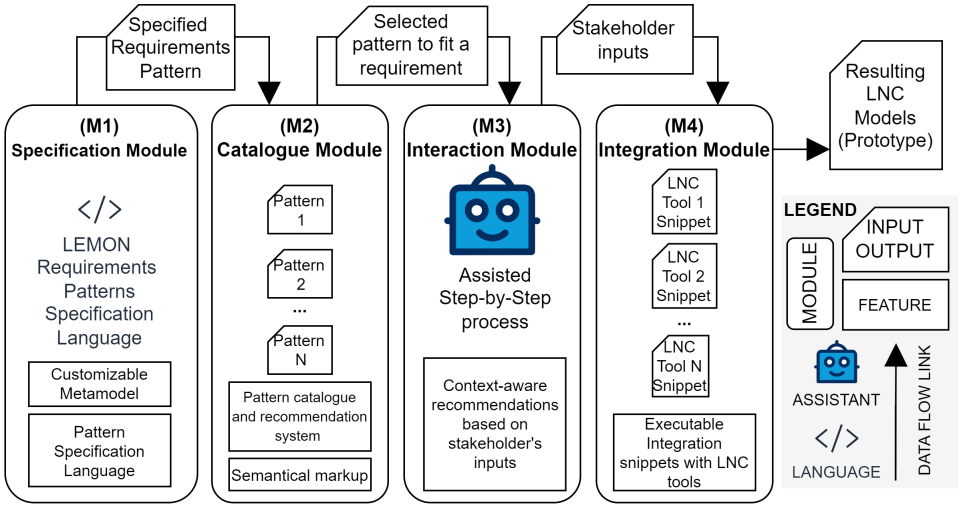

Figure 2: LEMON Architecture overview.

(M1) Requirements Pattern Specification module. We provide LEMON with a requirements pattern specification language (DSL), allowing software developers to specify their requirements patterns. Unlike previous literature in requirements pattern languages [7], [8], software developers can use M1 to specify requirements patterns without depending on the output notation, such as UML or BMPN.

(M2) Requirements pattern Catalogue module. We provide LEMON with a requirements pattern catalogue, allowing software developers to store previously specified requirements patterns. We reuse the concept of a catalogue from the literature [1], [9], [10], [11], where requirements patterns are classified and tagged for further search. This catalogue is not only an abstract description of the requirements pattern but contains a semantical markup, allowing LEMON to suggest requirement patterns depending on stakeholders' input.

(M3) Requirements pattern Interaction module. We provide LEMON with a user interface that allows stakeholders and developers to interact with the requirements patterns. We base the LEMON interaction on templates in the literature [4], [5], [6], with a step-by-step guided process. As a novelty, LEMON interact with stakeholders as an assistant. LEMON offers possible options to fulfil the templates and allow stakeholders to provide information without technological jargon.

(M4) Integration module. M4 allows software developers to integrate LNC tools to reuse the requirements pattern results. In practice, several LNC tools exist. Software developers provide code snippets using M4 to connect their requirement patterns to their desired LNC tool. This implies that for each LNC tool integrated with LEMON, there is a specific integration code snippet that LEMON executes to transform stakeholders' input into LNC models automatically. So far, we have integrated LEMON with one LNC tool. However, we provide LEMON's DSL with the ECORE metamodel [14]. Ecore allows software developers to represent other LNC tool metamodels through meta-modelling. Thus, software developers can reference an LNC tool metamodel within a LEMON pattern and later use M4 to integrate the result into the LNC tool. Not all LNC tools' metamodels can be represented using ECORE. This is a current limitation of our work. In future work, we aim to provide compatibility with other meta-modelling tools, such as the ADOxx metamodel [15], to increase the integration with LNC tools.

4.1. LEMON: Workflow and Interaction

We use a step-by-step explanation using the context of our use case at WGmbH to demonstrate how LEMON works (see Figure 3).

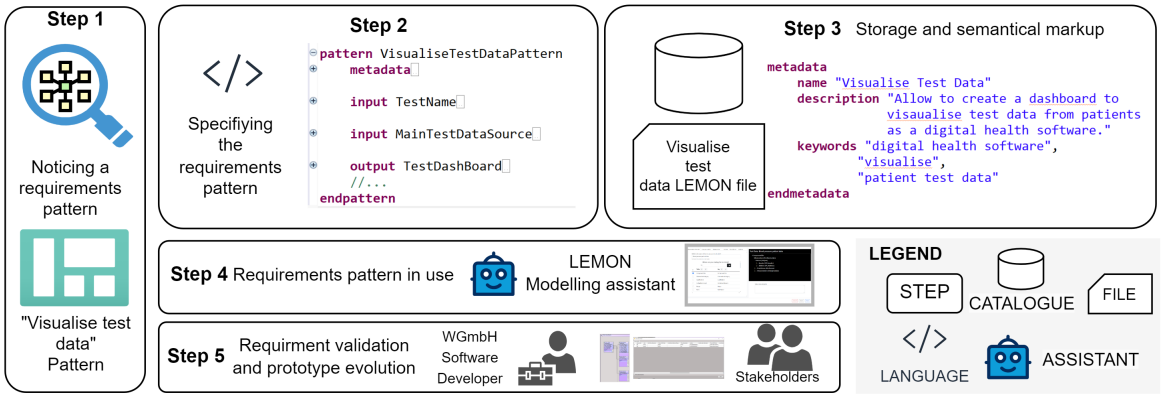

Figure 3: LEMON workflow and interaction.

(Step 1) Noticing a requirements pattern. The first step to using LEMON requires at least one requirements pattern identified in the software development context. In the case of WGmbH, software developers noticed by experience that their stakeholders often need to "visualise test data" software. This triggered the identification of a pattern in digital health software development. In this case, the requirements pattern is identified a posteriori (i.e., software developers have placed the pattern after

creating several solutions that have something in common to be considered a pattern). However, a pattern could be identified a priori with existing catalogues in the literature or other sources in digital health software (such as FHIR: Fast Healthcare Interoperability Resources).

(Step 2) Specifying requirements patterns. Having identified the "visualise test data" requirement pattern, WGmbH software developers use the LEMON requirement specification language (M1) to specify the pattern's inputs, outputs, and methods. For instance, inputs for the 'visualise test data' requirement pattern include the specific test name for visualisation (e.g., blood pressure test data), the corresponding data model table where the test data is stored (e.g., a component observation based on the FHIR standard), and additional data model tables containing related information (e.g., patient data model). On the other hand, examples of outputs comprise the LNC models that allow the visualisation of selected data from the LNC tool. Concerning methods, they delineate the process of transforming inputs into outputs.

(Step 3) Storage and semantical markup. WGmbH software developers store the specified "visualise test data" requirement pattern in the LEMON requirements pattern catalogue (M2). To do so, they provide the pattern with semantic markup for inputs and outputs that will help search for the pattern later when needed. For example, they provide the requirements pattern with metadata and keywords (semantical markup) related to their specific development context, such as "digital health software", "visualise", and "patient test data."

(Step 4) Requirements pattern in use. Later, when a new requirement arises during a meeting or by request of a WGmbH stakeholder, WGmbH software developers deploy the LEMON modelling assistant, search for the "visualise test data" supported by a recommendation system and start using the pattern. Then, WGmbH stakeholders provide the required input, using suggestions according to their needs (M3). After collecting all the required inputs, the LEMON modelling assistant (see Figure 4) is ready to store the answers from the stakeholders and generate a prototype (i.e., generate the models in the LNC tool from WGmbH). LEMON automatically generates such prototypes into their LNC tool, executing the code snippets provided by WGmbH software developers in the integration module (M4).

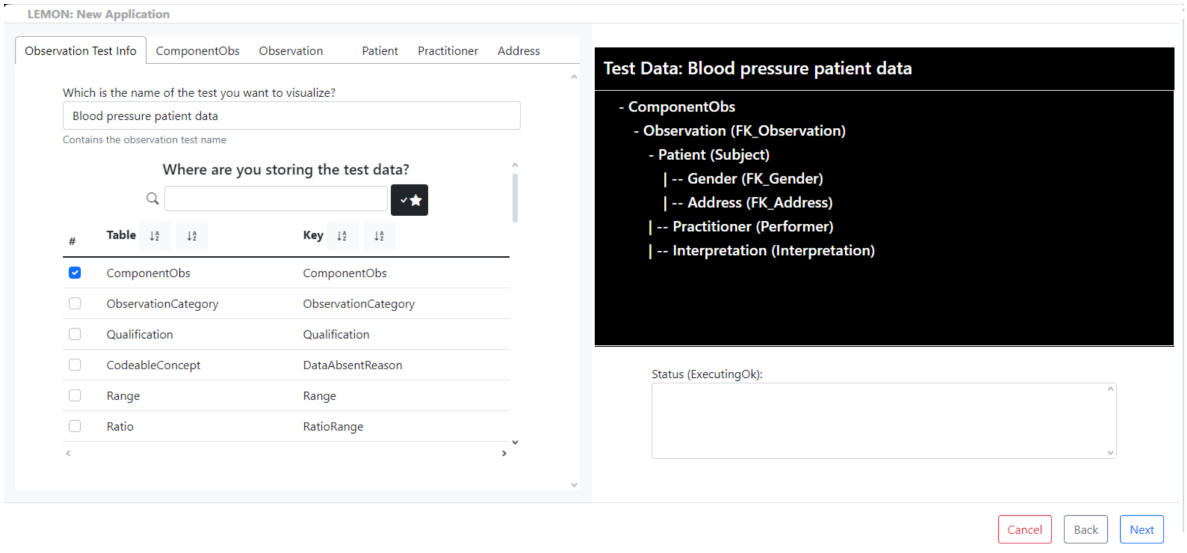

Figure 4: LEMON in action: Screenshot from modelling assistant at Step 4.

(Step 5) Requirement validation and prototype evolution. Now, software developers are ready to validate if the generated prototype fulfils the stakeholders' expectations. Step 5 is a significant benefit of using LEMON since WGmbH software developers can gather feedback when the stakeholders provide the inputs to LEMON. Finally, WGmbH software developers collect feedback from stakeholders about the "visualise test data" prototype and evolve it to produce the final software product. Notably, at the end of step 5, software developers can be brought back to step 4 to use a different requirements pattern that fits better with the stakeholder's requirements and continue with the prototype evolution.

5. Conclusions and Preliminary Evaluation

In this tool & demonstration paper, we presented LEMON: an LNC-powered tool for enhancing requirement communication based on requirements patterns. We have described the architecture and workflow of LEMON, exemplified in the digital health software development context, using a use case at a Swiss start-up, Whatscount GmbH.

To gather preliminary practitioners' feedback about LEMON architecture and tool, we surveyed 11 subjects with experience (81.8%) and without experience (18.2%) in requirements engineering in digital health software as potential software developers or stakeholders that could adopt LEMON in practice[1]. After introducing LEMON with a similar use case as the one explained in this paper, practitioners answered questions such as: What could be the barriers to the successful adoption of LEMON in your team? What do you think is missing in LEMON to be useful in practice? Which would be the perfect interaction for reusing requirements patterns in your projects? After processing the data, we identified some challenges in LEMON industrial adoption, primarily related to LEMON's i) user interaction, ii) quantity and quality of requirements patterns, and iii) integrability with tools other than LNC tools. Regarding LEMON interaction, we gathered comments such as: "...how and where LEMON is deployed?..." and "...who is the target group of LEMON? Decision makers, product managers, scrum masters, developers, etc? all may have different needs." Thus, we observed that subjects consider identifying which users would be targeted to use LEMON and how LEMON will be used as a critical factor for LEMON success. Moreover, subjects pointed out that LEMON's success would depend on the catalogue's quantity and quality of requirements patterns based on quotes such as "...if LEMON has a small database and after 2 enquiries it doesn't find anything, I would give up using it." Regarding this concern, we expect that the catalogue of patterns continue growing with the availability of the LEMON language. Finally, we gathered feedback on LEMON's interoperability with no-LNC tools such as JIRA, GitHub, etc., having comments such as "...I would like to reuse issues (e.g., on GitHub) for LEMON..." and "how LEMON build the software (to connect to other existing software)? Will it produce a microservice-based system?" LEMON integration with LNC and no-LNC tools is still a work in progress and will require future research efforts.

In future work, we plan to continue developing to finalise and improve LEMON's architecture[2] and conduct more empirical efforts involving interviews and quasi-experiments. We are confident that LEMON holds the potential to enhance communication of requirements between stakeholders and software developers by leveraging requirements patterns.

Acknowledgments

We would like to express our sincere gratitude to the 11 subjects who allowed us to gather their data during our survey. Moreover, we would like to extend our sincere gratitude to the anonymous reviewers for their valuable insights and constructive feedback. Their contributions have greatly enhanced the quality and clarity of this paper. This research is fully funded by the ZHAW Institute for Applied Information Technology (InIT), School of Engineering, and the Innosuisse Flagship SHIFT project.

References

[1] S. Renault, O. Mendez-Bonilla, X. Franch, C. Quer, Pabre: Pattern-based requirements elicitation, in: Third International Conference on Research Challenges in Information Science, RCIS, IEEE, 2009, pp. 81–92. doi:10.1109/RCIS.2009.5089271.

[1] For the sake of space, we report all raw data, traceability between subjects' answers and not yet solved challenges, and threats to validity in the following repository: https://doi.org/10.5281/zenodo.10579372

[2] LEMON is a prototype not yet openly available as an open-source project. In the future, we plan to make LEMON open-source. We provide a version of the LEMON DSL syntax at: doi.org/10.5281/zenodo.10579372

[2] P. Mahendra, A. Ghazarian, Patterns in the requirements engineering: A survey and analysis study, in: Transactions on Information Science and Applications, WSEAS, 2014, pp. 214–230.

[3] T. N. Kudo, R. F. Bulcão-Neto, A. M. Vincenzi, Requirement patterns: a tertiary study and a research agenda, IET Software 14 (2020) 18–26. doi:10.1049/iet-sen.2019.0016.

[4] A. D. Toro, A. Ruiz-Cortés, M. Toro, A. D. Toro, B. B. Jiménez, A. R. Cortés, M. T. Bonilla, A requirements elicitation approach based in templates and patterns a requirements elicitation approach based in templates and patterns?, in: Workshop em Engenharia de Requisitos, 1999, pp. 1–13.

[5] B. I. Ya'u, A. Nordin, N. Salleh, I. Aliyu, Requirements patterns structure for specifying and reusing software product line requirements, in: International Conference on Information and Communication Technology for the Muslim World, ICT4M, IEEE, 2018, pp. 185–190. doi:10.1109/ICT4M.2018.00042.

[6] K. Kumar, R. K. Saravanaguru, Context aware requirement patterns (carepa) methodology and its evaluation, Far East Journal of Electronics and Communications 16 (2016) 101–117. doi:10.17654/EC016010101.

[7] S. Robertson, Requirements patterns via events/use cases, in: PLoP, 1996, pp. 1–16.

[8] A. R. da Silva, D. Savić, S. Vlajić, I. Antović, S. Lazarević, V. Stanojević, M. Milić, A pattern language for use cases specification, in: Proceedings of the 20th European Conference on Pattern Languages of Programs, ACM, 2015, pp. 1–18. doi:10.1145/2855321.2855330.

[9] L. Sardi, A. Idri, L. Redman, H. Alami, J. Fernández-Alemán, A reusable catalog of requirements for gamified mobile health applications, in: Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering, SCITEPRESS, 2022, pp. 435–442. doi:10.5220/0011071700003176.

[10] R. Wahono, J. Cheng, Extensible requirements patterns of web application for efficient web application development, in: First International Symposium on Cyber Worlds, IEEE, 2002, pp. 412–418. doi:10.1109/CW.2002.1180908.

[11] S. Srivastava, A repository of software requirement patterns for online examination system, International Journal of Computer Science Issues 10 (2013) 247–255.

[12] C. Palomares, C. Quer, X. Franch, Requirements reuse and requirement patterns: a state of the practice survey, Empirical Software Engineering 22 (2017) 2719–2762. doi:10.1007/s10664-016-9485-x.

[13] B. I. Ya'u, A. Nordin, N. Salleh, Analysis of expert's opinion on requirements patterns for software product families framework using gqm method, in: International Conference on Computational Science and Technology, ICCST, Springer, 2020, pp. 135—-144. doi:https://doi.org/10.1007/978-981-15-0058-9_14.

[14] E. Foundation, Ecore: A metamodel for models, 2010. URL: https://wiki.eclipse.org/Ecore.

[15] D. Karagiannis, M. Lee, R. A. Buchmann, Using metamodeling for requirements engineering: A best-practice with adoxx, in: 27th International Requirements Engineering Conference, RE, IEEE, 2019, pp. 498–499. doi:10.1109/RE.2019.00073.