Custom-MADE – Leveraging Agile Rationale Management by Employing Domain-Specific Languages

Mathias Schubanz

Brandenburg University of Technology,
Platz der Deutschen Einheit 1, 03046 Cottbus, Germany
M.Schubanz@b-tu.de

Abstract. Managing rationale in software development projects can be a cumbersome task with a potentially low return on investment. Especially in the agile context, documentation is therefore very unpopular. Research has not yet properly addressed an agile documentation workflow. In this paper, the author presents an integrated approach to agile rationale management based on a highly-flexible modelling approach using domain-specific languages. It facilitates the complete documentation workflow from capture to reuse, partially automates it and offers various customisation opportunities, making it applicable to agile methods.

Keywords: agile · decision-making · Language Server Protocol · domain-specific languages · documentation · tool support · rationale management

1 Introduction

Managing rationale can considerably improve comprehension in software development. It facilitates change impact analysis as well as requirements traceability and evolution [33]. Moreover, it improves the understanding of architectural decisions and leads to better decisions [34]. In *agile software development* (ASD) this can be particularly important, as empirical work showed that stakeholders in agile teams perceive less architecture involvement [16]. Furthermore, in ASD documentation is questioned with particular scepticism. ASD instead promotes working software while depreciating documentation (cf. *Agile Manifest* [3]). Despite the mentioned and other potential benefits (cf. Tang et al. [32]), the structured and systematic handling of decisions is only applied seldom in practise [2,11]. Even in ASD teaching, it is only dealt with very selectively [20].

As part of an ongoing research project [30] with the focus adapted to ASD, the author developed *Custom-MAnangement of DEcision* (Custom-MADE), an integrated process-centric approach to rationale management. Based on domain-specific languages (DSL) it integrates a highly-flexible modelling approach enabling its users to tailor it to individual or enterprise-wide needs and documentation standards. Another feature is its minimal-invasive and generic approach offering to combine it with existing workflows. All these customization opportunities are vital for use in individually tailored processes in ASD.

J. Manner, S. Haarmann, S. Kolb, N. Herzberg, O. Kopp (Eds.): 13th ZEUS Workshop, ZEUS 2021, Bamberg, held virtually due to Covid-19 pandemic, Germany, 25-26 February 2021, published at http://ceur-ws.org

The remainder of the paper starts with a presentation of related work (Section 2). Section 3 presents a simplified rationale management workflow, while Section 4 elaborates on how *Custom-MADE* enhances it. Subsequently, Section 5 introduces the model architecture and Section 6 the chosen tool architecture. Finally, Section 7 concludes the paper and provides an outlook on future research.

2 Related Work

Approaches to direct process integration of rationale management in ASD are sparse, if at all. Rather, there are many tool-based approaches for rationale management in ASD. In the field of agile requirements gathering, for instance, Lee et al. [22] developed *Echo*, an approach connecting requirements and related design decisions. Around the same time Sauer [28] presented an approach for automating the capture of rationale in ASD. It aims to reduce costs by linking historical data and prototype definitions to an event-based rationale model. More recently, Hadar et al. [15] introduced an approach to document an architecture in the elevator speech concept. Their tool helps architects in organizing relevant information while creating design and architecture blueprints, thus reducing documentation effort. Also recently Voigt et al. [35] presented *sprintDoc*, a tool-based concept based on *DokuWiki* [13]. It integrates the documentation of artefacts into the agile process and thus traces changes in documents along with changes in issues.

There are also a lot of contributions around rationale modelling. These contributions often include tools that support rationale capture or sometimes even the decision-making process, as e. g., Miksovic and Zimmerman [26]. Some approaches even start with requirements analysis, such as RADAR [12]. A considerable share of the model-based contributions stems from research on software architecture documentation, less from the field of agile documentation. Since *Custom-MADE* aims less at architecture-bound documentation and more at integration into lightweight agile processes, please refer to Tang et al. [31] for a comprehensive overview of other approaches to modelling software architecture decisions as well as suitable tool support.

When considering the modeling approach from a more general perspective, one of the approaches that is most similar to *Custom-MADE* is *Frag* [36]. However, Zdun focuses more on DSL-based designs rather than using DSLs to get maximum flexibility for the documentation models.

Further work tries to leverage the opportunity of capturing rationale exactly where and when they are made to mitigate a substantial barrier to rationale capture (cf. [11]). For instance, *DesignMinders* [8] complements whiteboard systems so that rationale are directly digitised and browseable. Other approaches directly integrate with the IDE, as implemented by *SEURAT* [10] and *DecDoc* [17]. *ConDec* [19] even goes one step further by additionally integrating with *JIRA* [23] and *Slack* [24].

Other related contributions ignore formal and process aspects of rationale documentation entirely. They focus on retrospectively extracting rationale from existing documents by, e. g., integrating machine learning techniques (cf. Alkadi et al. [1], Bhat et al. [6], or Rogers et al. [27]).

3 Rationale Management Workflow

As presented in the related work (cf. Section 2), many of the approaches to date focus on either modelling, the activity of capturing, or persisting and providing design rationales to developers. However, only a few approaches focus on a holistic workflow and a possible process integration with it. In this chapter, the author presents a simplified rationale management workflow that serves as the basis for presenting the tool-driven approach later on (cf. Figure 1). As of now, *Custom-MADE* deliberately ignores the reasoning and decision-making process and begins with the rationale capture.

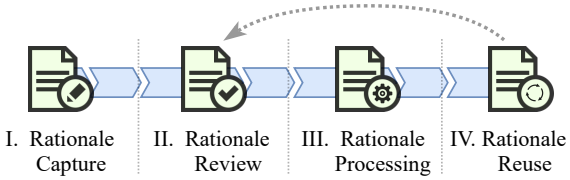

Fig. 1. Rationale management workflow.

Initially, (*I*) rationale that are considered sufficiently important will be documented. As a second step, (*II*) the rationale are reviewed at a given time (e.g. in retrospective, as in [29]). Following the review and potential modifications, (*III*) the documented rationale are processed and archived for later use. The last activity (*IV*) constitutes the reuse of the recorded information. If it is necessary to modify already documented rationale, the workflow can be reiterated from the review.

4 Enhanced Rationale Management Workflow

The aim of *Custom-MADE* is to enable an individual, customisable, flexible and semi-automated rationale management workflow. *Custom-MADE* starts even before the actual workflow (cf. Figure 2).

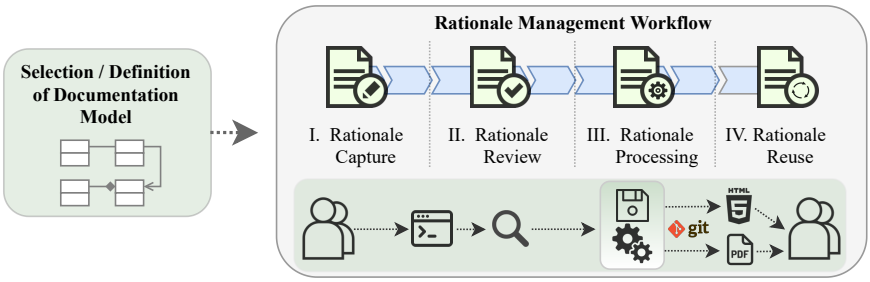

Fig. 2. Custom-MADE – Enhanced rationale management workflow.

With domain experts, developers should either define a particular documentation model or select one from a set of predefined models. Building on the underlying

DSL technology, *Custom-MADE* users can define their particular models and integrate them seamlessly into the toolchain. In the following, the author describes modifications to the workflow and how the tool facilitates the rationale management workflow:

Rationale Capture / Rationale Review

To support software engineers during the *Rationale Capture*, *Custom-MADE* provides an easy and intuitive web-based editor facilitating the previously defined documentation model (cf. Figure 4). Based on the *Xtext* framework [14], this editor offers high levels of flexibility and customisability. It provides a full language infrastructure to the user, i. e., among others, auto-completion, semantic colouring, error checking, quick-fix suggestions. These and many more are individually adapted to the chosen documentation model (defined as DSL). This feature set simultaneously facilitates a semi-automated *Rationale Review* (cf. Figure 2). By offering documentation guidance in the form of customisable rationale capture templates and formal and semantic checks, the developers can fully focus on the content aspects when reviewing the documented design rationale. *Custom-MADE* takes over the remaining part of the quality control in a semi-automated way.

Rationale Processing

Complying with enterprise-wide documentation standards and making documentation available to others can be the most cumbersome documentation task. Here *Custom-MADE* steps in and triggers a processing pipeline when saving the documented rationale. For each documentation model, there are either predefined or specifically customisable generators that transfer the documented rationale into the desired and easily accesible formats, as, e. g., *Markdown Architecture Decision Records* [21], *HTML* [4], or *PDF* [7]. The generators can be easily adapted using *Xtend* [5] to make the documentation comply with existing standards and generate desired formats.

Rationale Reuse

As mentioned already, it is often cumbersome to use documentation that has already been produced. Accessibility and availability are central obstacles to effective usage here. *Custom-MADE* addresses these by storing the raw documentation and the generated files on the software developers' central workplace, the code repository. The user has the opportunity to connect a project to a *git* repository (cf. [25]) on a remote git server. If connected, the documentation will be stored and versioned on a separate *git* branch. This storage concept not only enables developers to access older versions but also centralises the storage location. Thus, it is always clear where the documentation can be found. The storage concept also enables the direct use of documentation, e.g. in Markdown format, within the IDE. It also mitigates accessibility barriers by enabling additional services, such as full-text search usually offered by IDEs. Corresponding search functions for the web interface are also being developed [1].

[1] The development takes place at https://github.com/schubmat/Custom-MADE.

5 Model Architecture

For the implementation of the desired modelling flexibility, it was necessary to choose a dynamically configurable approach. Accordingly, the author decided to implement a model concept with three levels (cf. Figure 3).

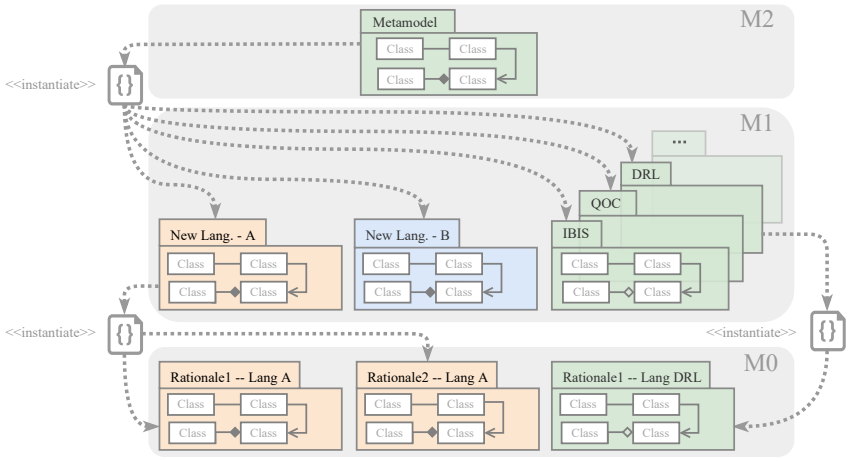

Fig. 3. Language modelling architecture.

On the top level (cf. level M2, Figure 3), a meta-model is used, which can implement the common modelling approaches from rationale management (cf. green packages). This meta-model is now interpreted as a DSL grammar. In this way, a language server can be generated that provides a complete language infrastructure for the definition at the model level (cf. level M1, Figure 3). This way, users can create their documentation model (cf. orange / blue package on the M1 level) with the *Monaco Editor* (cf. Figure 4). Once this is complete, the model itself is interpreted as a DSL grammar to generate the language infrastructure for the rationale documentation (cf. level M0). Starting from this point, developers can now capture their rationale and start the rationale management workflow.

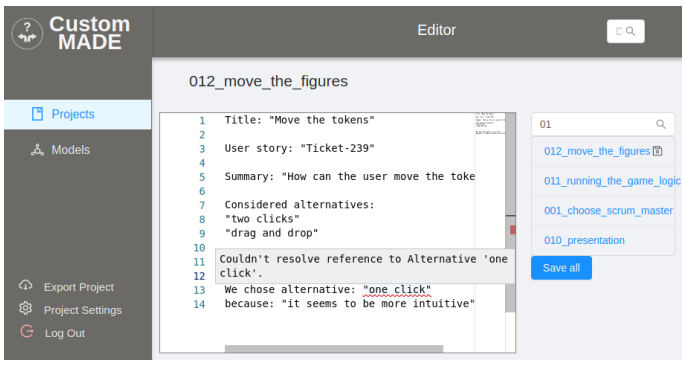

Fig. 4. Screenshot of the integrated editor showing semantic and syntactic hints.

6 Custom-MADE Tool Architecture

The author briefly presents the tool architecture based on the overview illustration in Figure 5. The left-hand side shows the part of *Custom-MADE* that is visible to the user. Based on a *ReactJS* web application, a *spring*-backend and a special implementation of the *Monaco Editor*, all functionalities described in Section 4 are accessible to the user. Particular attention is to be paid to the *Monaco Editor*, as its support of the *Language Server Protocol* (LSP) (cf. [9]) enables the flexibility in modelling that is one of the core features of the *Custom-MADE* approach. With the help of this web interface, developers can initially select or individually define their documentation model. Using *Xtext*, an individual language server is generated, with which the *Monaco Editor* supports the complete language infrastructure. The language server and the editor communicate using JSON-RPC [18]. The user can now start to record rationales based on the defined documentation model. These are validated instantly in the editor and in the processing step (cf. centre of Figure 5). If required, test cases can also be implemented for the model. Subsequently, with the generators' help the defined document formats are created and stored in the project's code repository (cf. right side of Figure 5).

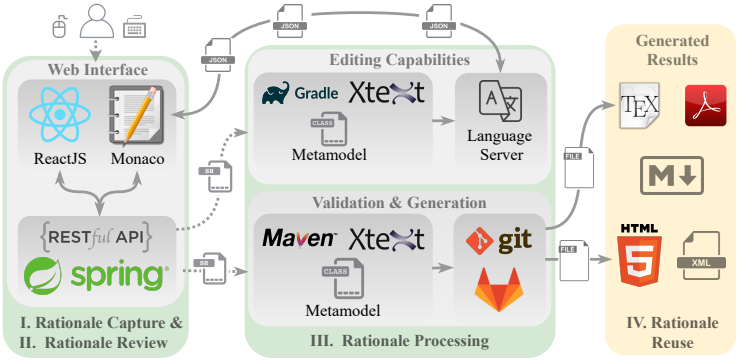

Fig. 5. Abstract tool architecture.

7 Conclusion

In this paper, the author presented a process-centric approach for the partial automation of rationale documentation, called *Custom-MADE*. Special features include its model-flexibility and customisability at various points in the toolchain. It can be easily applied to existing workflows and configured with documentation models already in use.

Future work includes, but is not limited to, the implementation of profound traceability down to the ticket level and automatically generated review requests becoming necessary due to changes that affect other rationale documentation.

Acknowledgments

Special gratitude goes to Jost-V. Schultz and Sebastian Brüggemann for their contributions to *Custom-MADE* and to Claus Lewerentz for his valuable feedback.

References

1. Alkadhi, R., Laţa, T., Guzman, E., Bruegge, B.: Rationale in Development Chat Messages: an Exploratory Study. In: Proceedings of the 14^{th} International Conference on Mining Software Repositories. pp. 436–446. IEEE Press (2017)
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edn. (2003)
3. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunnigham, W., Fowler, M., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development. http://agilemanifesto.org/ (February 2001), http://agilemanifesto.org/
4. Berners-Lee, T., Connolly, D.: Hypertext markup language-2.0 (1995)
5. Bettini, L.: Implementing Domain-Specific Languages With Xtext and Xtend. Packt Publishing Ltd (2016)
6. Bhat, M., Shumaiev, K., Biesdorf, A., Hohenstein, U., Matthes, F.: Automatic Extraction of Design Decisions From Issue Management Systems: A Machine Learning Based Approach. In: European Conference on Software Architecture. pp. 138–154. Springer (2017)
7. Bienz, T., Cohn, R., Adobe Systems (Mountain View, C.: Portable document format reference manual. Citeseer (1993)
8. Bortis, G.: Informal Software Design Knowledge Reuse. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering. vol. 2, pp. 385–388. IEEE (2010)
9. Bünder, H.: Decoupling Language and Editor – The Impact of the Language Server Protocol on Textual Domain-Specific Languages. In: Proceedings of the 7^{th} International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019). pp. 131–142 (2019)
10. Burge, J., Brown, D.: SEURAT: Integrated Rationale Management. In: 2008 ACM/IEEE 30th International Conference on Software Engineering. pp. 835–838. IEEE (2008)
11. Burge, J.E., Brown, D.C.: Software Engineering Using RATionale. Journal of Systems and Software **81**(3), 395–413 (2008)
12. Busari, S.A., Letier, E.: RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp. 552–562. IEEE (2017)
13. DokuWiki: DokuWiki – It's Better When its Simple (Jul 2004), https://www.dokuwiki.org/
14. Eysholdt, M., Behrens, H.: Xtext: Implement Your Language Faster Than the Quick and Dirty Way. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. pp. 307–309 (2010)
15. Hadar, I., Sherman, S., Hadar, E., Harrison, J.J.: Less is More: Architecture Documentation for Agile Development. In: 6^{th} International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). pp. 121–124. IEEE (2013)
16. Heijenk, F., van den Berg, M., Leopold, H., van Vliet, H., Slot, R.: Empirical Insights Into the Evolving Role of Architects in Decision-Making in an Agile Context. In: European Conference on Software Architecture. pp. 247–264. Springer (2018)
17. Hesse, T.M., Kuehlwein, A., Roehm, T.: DecDoc: A Tool for Documenting Design Decisions Collaboratively and Incrementally. In: 1^{st} International Workshop on Decision Making in Software ARCHitecture (MARCH). pp. 30–37. IEEE (2016)
18. JSON-RPC Working Group, .: JSON-RPC 2.0 Specification (2013)

19. Kleebaum, A., Johanssen, J.O., Paech, B., Bruegge, B.: Sharing and Exploiting Requirement Decisions. In: In Proceedings: Fachgruppentreffen Requirements Engineering (FGRE'19) (2019)
20. Kleebaum, A., Johanssen, J.O., Paech, B., Bruegge, B.: Teaching Rationale Management in Agile Project Courses. In: Tagungsband des 16. Workshops "Software Engineering im Unterricht der Hochschulen" (2019)
21. Kopp, O., Armbruster, A., Zimmermann, O.: Markdown Architectural Decision Records: Format and Tool Support. In: ZEUS. pp. 55–62 (2018)
22. Lee, C., Guadagno, L., Jia, X.: An Agile Approach to Capturing Requirements and Traceability. In: Proceedings of the 2^{nd} International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE). vol. 20 (2003)
23. Li, P.: Jira 7 Essentials. Packt Publishing Ltd (2016)
24. Lin, B., Zagalsky, A., Storey, M.A., Serebrenik, A.: Why developers are slacking off: Understanding how software teams use slack. In: Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion. pp. 333–336 (2016)
25. Loeliger, J.: Version control with Git. O'Reilly Series, O'Reilly (2009), http://books.google.de/books?id=e9FsGUHjR5sC
26. Miksovic, C., Zimmermann, O.: Architecturally Significant Requirements, Reference Architecture, and Metamodel for Knowledge Management in Information Technology Services. In: 9^{th} Working IEEE/IFIP Conference on Software Architecture. pp. 270–279. IEEE (2011)
27. Rogers, B., Qiao, Y., Gung, J., Mathur, T., Burge, J.E.: Using Text Mining Techniques to Extract Rationale From Existing Documentation. In: Design Computing and Cognition'14, pp. 457–474. Springer (2014)
28. Sauer, T.: Using Design Rationales for Agile Documentation. In: Proceedings of the 12^{th} IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE). pp. 326–331. IEEE (2003)
29. Schubanz, M., Lewerentz, C.: What Matters to Students – A Rationale Management Case Studyin Agile Software Development. In: Tagungsband des 17. Workshops "Software Engineering im Unterricht der Hochschulen", Innsbruck, Österreich (2020)
30. Schubanz, M.: Design Rationale Capture in Software Architecture: What has to be Captured? In: Proceedings of the 19^{th} International Doctoral Symposium on Components and Architecture. pp. 31–36. ACM (2014)
31. Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M.A.: A Comparative Study of Architecture Knowledge Management Tools. Journal of Systems and Software **83**(3), 352–370 (2010)
32. Tang, A., Babar, M.A., Gorton, I., Han, J.: A Survey of Architecture Design Rationale. Journal of Systems and Software **79**(12), 1792–1804 (2006)
33. Thurimella, A., Schubanz, M., Pleuss, A., Botterweck, G.: Guidelines for Managing Requirements Rationales. Software, IEEE **34**(1), 82 – 90 (2017). https://doi.org/10.1109/MS.2015.157
34. Tofan, D., Galster, M., Avgeriou, P.: Difficulty of Architectural Decisions – A] Survey With Professional Architects. In: European Conference on Software Architecture. pp. 192–199. Springer (2013)
35. Voigt, S., Hüttemann, D., Gohr, A.: SprintDoc: Concept for an Agile Documentation Tool. In: 11^{th} Iberian Conference on Information Systems and Technologies (CISTI). pp. 1–6. IEEE (2016)
36. Zdun, U.: A DSL Toolkit for Deferring Architectural Decisions in DSL-Based Software Design. Information and Software Technology **52**(7), 733–748 (2010)