Extraction and Management of Rationale

Mouna Dhaouadi
DIRO, Université de Montréal
Montreal, Canada

ABSTRACT

Software developers often have to make many design decisions. The underlying logic behind these decisions, also called *design rationale*, represents beneficial and valuable information. In the past, researchers have tried to automatically extract and exploit this information, however, prior techniques are only applicable to specific contexts and there is insufficient progress on an automated end-to-end rationale extraction and management system. In this research project, we propose to use Natural Language Processing (NLP) and Machine Learning (ML) techniques to create a system for the automated extraction, structuring and management of design rationale. This system would support and ensure the consistency and the coherence of the development process.

KEYWORDS

Rationale Extraction, Rationale Management

ACM Reference Format:
Mouna Dhaouadi. 2022. Extraction and Management of Rationale. In *Proceedings of The 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 RESEARCH PROBLEM STATEMENT

Developers often need to make design decisions and to be aware of the reasons behind previously made ones. This knowledge is called *design rationale* [3]. In fact, developers have to spend time and make effort to understand the historical evolution of decisions. This is important as the lack of a shared vision and of a complete understanding of the logic behind the previous decisions would result in duplicated effort, incoherent choices, conflicting decisions and unsuccessful collaborations. It could also result in *design evaporation* [15] and *design erosion* [22], especially when the project spans a long period of time and geographically-distributed teams.

Thus, rationale knowledge would greatly benefit different stakeholders. For instance, developers could exploit it to find relevant past decisions when facing similar situations, or to apprehend the big picture and understand how previous decisions are interconnected, which would help maintain the stability and the permanence of the systems quality. Therefore, several researchers [2, 8] tried to capture it by requiring developers to document it. However, this line of research was unsuccessful because of the extra effort needed for the documentation. The solution is thus to recover rationale automatically. However, this is challenging as it involves not only finding and linking scattered and poorly organized rationale information from different artifacts such as developments documents and written communications [4], but also, structuring it in an exploitable format and proposing a usable system to leverage it. Recently, researchers have tried to automatically extract [1, 10, 20] and exploit this information [7], however, these techniques are applicable to specific contexts and we still lack a complete automated extraction and management system.

We propose to address this gap with an automated end-to-end rationale extraction and management system. For our first milestone, we are building *Kantara*, a pipeline for automated rationale reconstruction. *Kantara* takes as input different textual sources and artifacts (e.g., mails, commit messages) specified by the user, and outputs a knowledge graph that contains the decisions and their rationales as nodes. The graph also captures the different relationships between decisions as edges. Our next steps consist of building a graph-based retrieval system and a tool that supports users by responding to their queries and allowing them to update the graph.

2 DISCUSSION OF THE STATE-OF-THE-ART

In [1], the authors proposed a ML approach to automatically detect and classify design decisions. In [10], the authors compared the performance of several ML configurations to automatically identify decisions. Both these works did not consider rationale information. In [13], the authors proposed a semantic grammar-based approach to automatically capture and structure design rationale. They did not use NLP. In [20], the authors proposed a heuristic-based rationale extraction system and a ranking module. This is different from our generic approach as it is specific to the python development process. In [18] and [9], the authors studied the use of text features for rationale extraction, and in [17], whether some features could be generalizeable to different data sets. They did not consider the structuring and management aspects. In [19], the authors proposed a bot that uses a knowledge graph that creates a domain model from a textual description. The bot can also find and retrieve the rationale behind his modelling decisions. Thus, the rationale they consider is pre-defined by the developers. In [7], the authors proposed tools that use a knowledge graph for the continuous identification, visualization and management of rationale in requirements. They differ from us because they require manual documentation, and they do not consider the historical evolution of decisions.

3 PROPOSED SOLUTION

Our proposed solution is based on the automated *on-demand developer documentation (OD3)* system concept [16]. Our OD3 system has two components: 1) An *information inference* component that extracts and structures the scattered useful information, and 2) A *response generation* component that gives responses to users queries.

ASE '22, October 10–14, 2022, Rochester, MI

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: *Kantara* pipeline overview

Stage	Step	Definition	Proposed Method
Extracting DR triples	1. Decisions extraction	Identify decision-containing sentences	Classification
	2. Rationale extraction	Extract sentence-level rationale	Semantic Role Labeling [14]
Extracting DD relationships	3. Relatedness relationship	Capture related decisions (about the same topic)	Classification
	4. Similar relationship	Capture semantically similar decisions	Semantic Similarity
	5. Contradicts relationship	Capture contradictory decisions	Natural Language Inference [11]
	6. History relationship	Check if a decision is a previous version of another	Heuristics

3.1 *Information inference* component

We exposed our plans for this component in [6]. Our contributions in this part are: 1) *Rationale and Decision Graph*: a knowledge graph-based representation of decisions and their rationale expressions that takes their historical evolution into consideration and that provides traceability to artifacts. 2) *Kantara Approach*: an end-to-end generic NLP-based information extraction pipeline to extract and structure rationale in a knowledge graph. 3) *Validation Mechanisms*: NLP-based heuristics to detect inconsistencies in the graph.

Rationale and Decision Graph. The main components of our knowledge graph are the triples *(decision, "rationale", rationale)*, which associate each decision node with its corresponding rationale node through the "rationale" edge. Each decision has an associated *source* attribute to ensure traceability (the URI of the file from which the decision has been extracted), and is related to a specific *topic*. Topics organize the extracted decisions into clusters of related decisions. The graph also allows representing different relationship types between decisions. For now, we introduce three types: a *similar* relationship, a *contradicts* relationship and a *history* relationship, and we plan to introduce other types in the future.

Kantara Approach. This information extraction pipeline has two stages: 1) extracting decision-rationale (DR) triples, and 2) extracting decision-decision (DD) relationships. Table 1 explains the steps of each stage, and mentions the proposed methods to use. We use context-independent techniques for each step: in step 1, we plan to use the software embeddings proposed in [12] that were shown to help with transferring contexts; in step 2, we only consider a sentence grammatical structure; in step 3, we choose a training dataset with 300K entries spanning the entire StackOverflow community [21]; in steps 4 and 5, we use pre-trained context-independent models; in step 6, we use heuristics.

Validation Mechanisms. We propose two mechanisms: 1) a mechanism for detecting inconsistencies between the decisions rationales and the decisions (e.g., if two decisions have a *similar* relationship, detecting a contradiction between their rationales could reflect a problem in the graph construction or a problem in the actual development), and 2) a conflict detection mechanism to check whether new decisions may cause conflict with former ones.

Evaluation plan. We conducted a preliminary evaluation of*Kantara* on a small example sourced from the Linux Kernel, which showed promising results [6]. Our future evaluation plan is two-fold: 1) a detailed step-by-step evaluation of the 6 steps of the *Kantara* pipeline, and 2) an overall evaluation of the pipeline as a whole. For the first detailed evaluation, we focus on the performance of our proposed methods and we plan to leverage commonly-used metrics (e.g., accuracy, F1-score). We plan to use publicly available datasets (e.g., we will use [10] for step 1), and create our own datasets through manual labelling (e.g., for step 2). For the second overall evaluation, we plan to use a case study to evaluate the end-to-end effectiveness and performance of *Kantara*, as well as its usefulness and that of the validation mechanisms. We also plan to evaluate the genericity of our pipeline, by applying it on case studies from different open-source systems (e.g., the Linux Kernel and Rust).

3.2 *Response generation* component

Our proposed contributions in this part are: 1) *Retrieval system:* a two-step graph-based retrieval system that can produce responses to a natural language query about a certain decision. 2) *Tool:* an interactive tool that allows the user to query the graph, see the results and refine the search. The tool should also allow the user to visualize and traverse the graph, and to correct or improve it.

Retrieval system. In order to select relevant decisions, our proposed retrieval system uses an index of the decision nodes of the *Rationale and Decision graph*. This is done as follows: 1) We compute a relevance score (e.g., similarity score) to find and rank the relevant decisions using the decisions index. We plan to experimentally propose a threshold and filter out results with relevance scores below it. 2) For each relevant decision selected, we leverage the knowledge to retrieve its corresponding rationale, the decisions associated with its topic and their rationales, and the decisions connected to it within a parameterized scope, and their rationales.

Tool. Our envisioned user interface is a web-based bot with a form field to type the query, and a list field to show the results (a list of reports). For each selected decision, the bot creates a report composed of two parts: 1) an image of the corresponding sub-graph retrieved as explained in step 2 of the *retrieval system*, and 2) a textual description of the image that coherently integrates the sub-graph information. Our bot can also open an interactive web page for the *Rationale and Decision Graph* visualization and traversal. This page should allow the user to update the graph (e.g. correct some nodes, or add another data source and recreate the graph). The tool could be used at any point in the development process.

Evaluation plan. To evaluate the effectiveness of the retrieval component and the usefulness of the OD3 system, we plan to do a user study. Specifically, we will ask participants to query the tool and to assess the relevance of the results. Then, we will use standard retrieval metrics: Reciprocal Rank and Precision@K [5] to evaluate the effectiveness of the retrieval process, and Mean Average Precision, Recall and F1-score to evaluate the usefulness of the whole OD3 system. To evaluate the usability aspect, we will ask participants to take a post-study questionnaire regarding the intuitiveness and the ease of learning of our tool.

REFERENCES

[1] Manoj Bhat, Klym Shumaiev, Andreas Biesdorf, Uwe Hohenstein, and Florian Matthes. 2017. Automatic extraction of design decisions from issue management systems: a machine learning based approach. In *European Conference on Software Architecture*. Springer, 138–154.

[2] Janet E Burge. 2005. *Software engineering using design RATionale*. Ph. D. Dissertation. Worcester Polytechnic Institute.

[3] Janet E Burge and David C Brown. 2008. Software engineering using rationale. *Journal of Systems and Software* 81, 3 (2008), 395–413.

[4] Rafael Capilla, Anton Jansen, Antony Tang, Paris Avgeriou, and Muhammad Ali Babar. 2016. 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software* 116 (2016), 191–205.

[5] Ben Carterette and Ellen M Voorhees. 2011. Overview of information retrieval evaluation. In *Current challenges in patent information retrieval*. Springer, 69–85.

[6] Mouna Dhaouadi, Bentley James Oakes, and Michalis Famelis. 2022. End-to-End Rationale Reconstruction. In *2022 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, to appear.

[7] Anja Kleebaum, Barbara Paech, Jan Ole Johanssen, and Bernd Bruegge. 2021. Continuous Rationale Identification in Issue Tracking and Version Control Systems. (2021).

[8] Jintae Lee. 1991. Extending the Potts and Bruns model for recording design rationale. In *Proceedings-13th International Conference on Software Engineering*. IEEE Computer Society, 114–115.

[9] Miriam Lester, Miguel Guerrero, and Janet Burge. 2020. Using evolutionary algorithms to select text features for mining design rationale. *AI EDAM* 34, 2 (2020), 132–146.

[10] Xueying Li, Peng Liang, and Zengyang Li. 2020. Automatic identification of decisions from the hibernate developer mailing list. In *Proceedings of the Evaluation and Assessment in Software Engineering*. 51–60.

[11] Bill MacCartney. 2009. *Natural language inference*. Stanford University.

[12] Alvi Mahadi, Neil A Ernst, and Karan Tongay. 2022. Conclusion stability for natural language based mining of design discussions. *Empirical Software Engineering* 27, 1 (2022), 1–42.

[13] Raymond McCall. 2018. Using argumentative, semantic grammar for capture of design rationale. In *International Conference on-Design Computing and Cognition*. Springer, 519–535.

[14] Martha Palmer, Daniel Gildea, and Nianwen Xue. 2010. Semantic role labeling. *Synthesis Lectures on Human Language Technologies* 3, 1 (2010), 1–103.

[15] Martin P Robillard. 2016. Sustainable software design. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 920–923.

[16] Martin P Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, et al. 2017. On-demand developer documentation. In *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 479–483.

[17] Benjamin Rogers, Connor Justice, Tanmay Mathur, and Janet E Burge. 2017. Generalizability of document features for identifying rationale. In *Design Computing and Cognition'16*. Springer, 633–651.

[18] Benjamin Rogers, Yechen Qiao, James Gung, Tanmay Mathur, and Janet E Burge. 2015. Using text mining techniques to extract rationale from existing documentation. In *Design Computing and Cognition'14*. Springer, 457–474.

[19] Rijul Saini, Gunter Mussbacher, Jin LC Guo, and Jörg Kienzle. 2022. Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *Software and Systems Modeling* 21, 3 (2022), 1015–1045.

[20] Pankajeshwara Nand Sharma, Bastin Tony Roy Savarimuthu, and Nigel Stanger. 2021. Extracting Rationale for Open Source Software Development Decisions—A Study of Python Email Archives. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1008–1019.

[21] Amirreza Shirani, Bowen Xu, David Lo, Thamar Solorio, and Amin Alipour. 2019. Question relatedness on stack overflow: the task, dataset, and corpus-inspired models. *arXiv preprint arXiv:1905.01966* (2019).

[22] Jilles Van Gurp and Jan Bosch. 2002. Design erosion: problems and causes. *Journal of systems and software* 61, 2 (2002), 105–119.