29th IEEE International Requirements Engineering Conference (RE2021), 20-24 Sept 2021, South Bend, USA

DBRG: Description-based Non-quality Requirements Generator

1st Mohamed Osama *, 2nd Aya Zaki-Ismail *, 3rd Mohamed Abdelrazek *, 4th John Grundy †, and 5th Amani Ibrahim *
Information Technology
* Deakin University , † Monash University
Melbourne, Australia
* {amohamedzakiism, mdarweish, mohamed.abdelrazek, amani.ibrahim}@deakin.edu.au , †<john.grundy@monash.edu>

***Abstract*—Requirements quality checking is a key process in requirements engineering. For complex and large scale systems, it is recommended to use automated requirements quality checking tools because of the size and complexity of requirements. However, such tools are typically evaluated on a small set of manually curated requirements. This limitation affects the comprehensiveness and reliability of the evaluation and leaves several possible quality issues undetected. In this paper, we describe a novel quality-checking-oriented synthesised requirements generator. We provide an input description language so that several quality checking issues and scenarios can be defined. The generator utilises an input dictionary of nouns and verb frames, and generates requirements sentences complying to a user-defined description of a quality affected requirement.**

***Index Terms*—Requirements Generation, Requirements Engineering**

I. Introduction

Evaluating the performance of quality checking tools is limited and constrained by the scope of quality issues available within a given suite of requirements in a provided dataset. In [1] specific requirements are selected through a case study to illustrate their defined quality indicators in the evaluation. Similarly, in [2], the defined indicators are evaluated on a synthesised case study. In [3], the authors evaluated six indicators on three datasets with a total of 244 requirements. These datasets had only three instances in total, for the two indicators being evaluated.

Such manually curated small requirements datasets are not enough to robustly evaluate approaches performing requirements manipulation and translation [4]. In many cases, the evaluation dataset is also not accessible due to confidentiality issues and this makes it very difficult to replicate or benchmark new tools against existing work. It would be very helpful to have a requirements generator that is configurable to create a large set of example requirements with seeded quality issues that conform to specific scenarios described by the researchers.

Motivated by these limitations, we aim to facilitate the evaluation and benchmarking of requirements quality checking tools by providing researchers with a new description-based requirements generator (DBRG) tool. DBRG can be used for the automated generation of synthesised non-quality textual requirements. To generate requirements that satisfy certain properties or reflect specific scenarios, DBRG supports describing these scenarios at different levels of granularity using a novel description language. It then parses these scenarios and generates requirements instances (sentences) that satisfy the specified description. In addition, DBRG can also provide detailed breakdowns of the generated requirements sentences. Further details about DBRG is available here [1].

II. DBRG Tool

Figure 1 summarises the process flow and interactions within DBRG. It consists of two main parts: the input description language (governing both the single, and multi level description formats), and the generation approach.

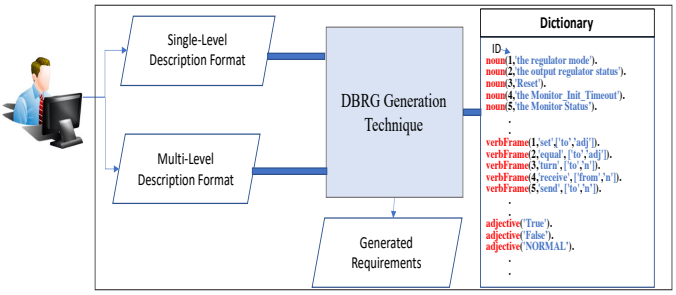

Fig. 1: DBRG Framework

DBRG accepts requirements descriptions as input. The structure of the utilised description language is based on an analysis that we conducted on several quality checking tools to identify the key quality indicators required to be supported. We identified their key quality indicators and added more hand crafted indicators to further exploit and demonstrate the potential of DBRG. This allows DBRG to generate requirements with both single and multi level quality issues (i.e., quality issues defined on a single requirement, or multiple requirements). We achieved this by supporting the description of inter-requirements relations through the multi-level format of the developed language.

To parse the descriptive language, DBRG utilises the Prolog variable binding approach [5] to assign concrete identifiers for representing the variables in the language. Parsing constructs

[1] DBRG-UI: https://github.com/ABC-7/DBRG/blob/main/DBRG_Annex.pdf

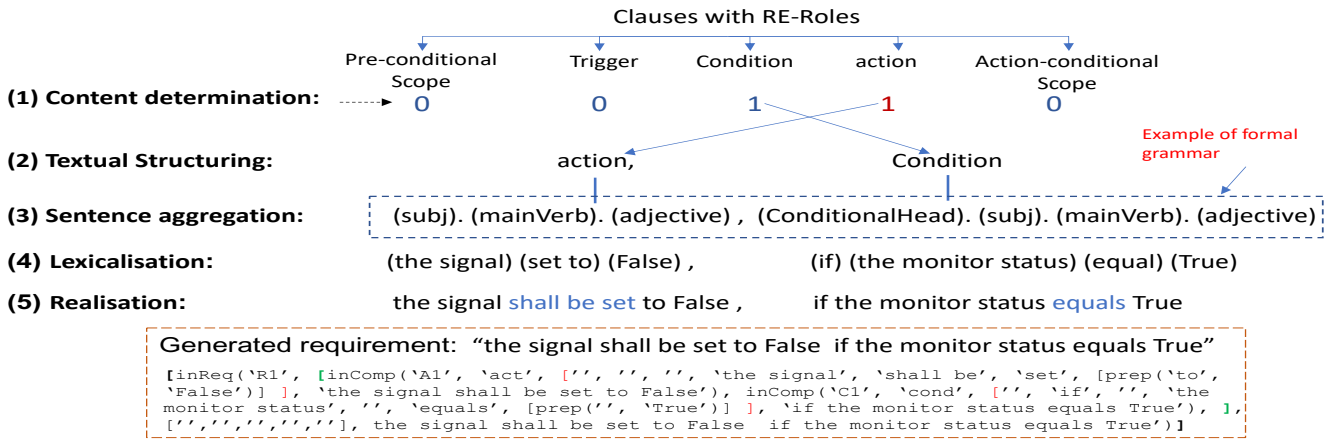

Fig. 2: Requirement Sentence Generation Example

the requirement sentence in a top-down paradigm. The existing variables in the input description determine the required parts in the sentence to be generated. Then, a suitable grammatical rule is selected by our generation approach [6]. Finally, the input variables are bound with concrete values (lexical words) to produce a requirement sentence. The utilised tokens are sourced from the supplied dictionary containing the lexical words, grammatical words, and verb frames to be used in the generation process.

A. Generation Approach

Our generation approach [6] follows the common generation tasks: content determination, textual structuring, sentence aggregation, lexicalisation and realisation. The role of the five main tasks within our approach are:

1) Content determination: is responsible for deciding which parts or blocks are included in the text. In this task we decide the number and type of clauses that will be present in the requirement sentence.
2) Textual structuring: determines the order of the selected parts in the text. It refers to the order of the clauses within the sentence.
3) Sentence aggregation: is responsible for deciding which parts are included in an individual clause. In this task, the clause breakdowns (predicate and subject structures) are decided (e.g., is the subject simple noun or noun-phrase, which structure is selected for the complement)
4) Lexicalisation: replaces the included parts with suitable words and phrases. Each clause is filled with the suitable terminal words from the input dictionary while adhering to the syntactic constraints within the selected structures).
5) Realisation: combines all the words and phrases in a well-formed sentence .

Figure 2 shows the step by step generation process for a single requirement sentence. We benefit from the backtracking support of Prolog (a logic descriptive language [7]), and its underlying inference engine to bind the free variables and match the input values.

III. Evaluation

We evaluated DBRG from two different aspects, applicability and effectiveness. For the applicability evaluation, we crafted 19 single-level and 10 multi-level input descriptions each corresponding to a different quality metric. The input descriptions and the corresponding outputs are available online [2]. For evaluating the effectiveness, we used DBRG to generate 110 non-quality requirements scenarios (i.e., 10 different scenarios for 11 different quality multi-level quality indicators) with a total of 240 synthesised requirements sentences.

IV. Conclusion

We presented DBRG, a tool supporting the generation of synthesised non-quality requirements for evaluating quality checking tools. It enables the users to describe the requirement(s) to be generated according to their needs.

References

[1] F. Konig, L. C. Ballejos, and M. A. Ale, "A semi-automatic verification tool for software requirements specification documents," in *Simposio Argentino de Ingeniería de Software (ASSE)-JAIIO 46 (Córdoba, 2017)*., 2017.
[2] A. Ciemniewska, J. Jurkiewicz, Ł. Olek, and J. Nawrocki, "Supporting use-case reviews," in *International Conference on Business Information Systems*. Springer, 2007, pp. 424–437.
[3] G. Lucassen, F. Dalpiaz, J. M. E. Van Der Werf, and S. Brinkkemper, "Forging high-quality user stories: towards a discipline for agile requirements," in *2015 IEEE 23rd international requirements engineering conference (RE)*. IEEE, 2015, pp. 126–135.
[4] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner, "Arsenal: automatic requirements specification extraction from natural language," in *NASA Formal Methods Symposium*. Springer, June 2016, pp. 41–46.
[5] J. Cheney and C. Urban, "αprolog: A logic programming language with names, binding and α-equivalence," in *International Conference on Logic Programming*. Springer, 2004, pp. 269–283.
[6] A. Zaki-Ismail, M. Osama, M. Abdelrazek, J. Grundy, and A. Ibrahim, "Corg: A component-oriented synthetic textual requirements generator," in *Requirements Engineering: Foundation for Software Quality: 27th International Working Conference, REFSQ 2021, Essen, Germany, April 12–15, 2021, Proceedings*. Springer Nature.
[7] F. C. Pereira and S. M. Shieber, *Prolog and natural-language analysis*. Microtome Publishing, 2002.

[2]Descriptive Input and Generated requirements: https://github.com/ABC-7/DBRG/tree/main/Evaluation