Refactoring in Requirements Engineering: Exploring a methodology for formal verification of safety-critical systems

Oisín Sheridan[1]

[1]Department of Computer Science, Maynooth University, Maynooth, Ireland

Abstract

[Context and motivation] The formal verification of safety-critical software requires formal requirements. Software requirements are often written in natural-language, which then needs to be translated into a formal language for use in verification. **[Question/problem]** As a requirements set becomes larger and more defined over the course of a requirements engineering project, an ever-increasing amount of work is required to ensure consistency, readability and traceability across the set. Often, dependencies and duplication of information between requirements emerges, which then requires additional work from engineers to update all of the affected requirements when changes need to be made. **[Principal ideas/results]** We propose that this can be improved by applying refactoring to requirements. Refactoring is a software engineering technique where code is reorganized to improve its internal structure without changing its behavior; in the case of requirements, we can reduce duplication of information and improve readability of the requirements without changing the behavior that the requirements specify. **[Contribution]** This project aims to provide a working implementation of requirements refactoring in Mu-FRET, a fork of the Formal Requirements Elicitation Tool (FRET) which allows for requirements to be written in a structured natural-language, which is then translated automatically into temporal logic. In addition, we will provide a theory of refactoring that can be generalized to other requirements languages.

Keywords

Requirements Elicitation, Software Verification, FRET, Refactoring, Traceability

1. Introduction & Background

Guaranteeing the trustworthiness of autonomous safety-critical software presents a significant challenge for developers. Industry often relies on manual testing and simulation to verify such systems, but these techniques are time-consuming, expensive, and error-prone. The solution would appear to be formal methods: mathematically-based techniques which can verify correctness through proof of the system's properties and exhaustive checks over its state space. However, applying formal methods requires formal requirements. Software requirements are often specified in natural-language which is too vague for direct formalisation, thus requiring a lengthy elicitation process where additional details about the requirements and the system itself are discussed and encoded in a formal language. This process represents a significant increase in the workload of a development project, especially for engineers with little-or-no experience with formal methods, and this workload only increases as a project develops and the formal requirements need to be updated.

The Formal Requirements Elicitation Tool (FRET) looks to address this challenge. FRET is an open-source tool developed by NASA that allows users to write requirements in FRETISH, a structured natural-language that is readable with minimal training. Each FRETISH requirement is then automatically checked for validity and translated into Past- and Future-Time Linear Temporal Logic (LTL), which can be used in formal methods including model checking and runtime verification. An example of a

In: M. Abbas, F. B. Aydemir, M. Daneva, R. Guizzardi, J. Gulden, A. Herrmann, J. Horkoff, M. Oriol Hilari, S. Kopczyńska, P. Mennig, E. Paja, A. Perini, A. Rachmann, K. Schneider, L. Semini, P. Spoletini, A. Vogelsang. Joint Proceedings of REFSQ-2025 Workshops, Doctoral Symposium, Posters & Tools Track, and Education and Training Track. Co-located with REFSQ 2025. Barcelona, Spain, April 17, 2025.

Oisin.Sheridan@mu.ie (O. Sheridan)

0000-0002-8613-2500 (O. Sheridan)

requirement in FRET, including its FRETISH text and corresponding Future-Time LTL translation, are shown in Figure 1.

The readability of FRETISH requirements makes it much easier to maintain and expand a requirements set as a project progresses. However, requirements engineers still face a number of challenges when managing an industrial-scale specification. As more properties of the system are specified, dependencies between requirements emerge, which can lead to duplication of information across multiple requirements and requires additional work to maintain consistency when updates are made. We propose that this problem can be solved by incorporating refactoring into the requirements engineering process. Refactoring is the process of improving the structure of software without altering its external-facing functionality, and we believe that the techniques used in software engineering can be adapted for requirements engineering. In particular, the aim of this PhD project is to incorporate Refactoring techniques into a fork of FRET, called Mu-FRET[1], for use on FRETISH requirements.

The idea of refactoring FRETISH requirements originated from experience using FRET in collaboration with an industrial partner as part of the VALU3S project [1], where we used FRET to formalise a set of requirements for an Aircraft Engine Controller [2]. After the requirements were formalised, there were a number of definitions that were repeated numerous times across the set: for example, the phrase "under sensor faults" in the natural-language requirements was formalised as "`if((sensorValue(S) > nominalValue + R) | (sensorValue(S) < nominalValue - R) | (sensorValue(S) = null)`" and appeared in full in 8 different requirements; this made updating this definition difficult, as any changes needed to be manually propagated across the set to maintain consistency [3]. We decided to explore if this "bad smell" and others could be solved by using refactoring, in the same way one might use code refactoring to extract a method in software development.

We examined existing literature and selected four refactoring techniques that we felt were applicable to FRETISH requirements [4]. As a proof-of-concept, we implemented one of these techniques, Extract Requirement, in Mu-FRET and applied it to the Engine Controller requirements. This exploration of requirements refactoring then expanded into this PhD project, which aims to implement the other refactoring techniques for FRETISH requirements and then investigate how refactoring can be applied in requirements engineering in general.

2. Proposed solution

As mentioned above, the main focus of this project is the implementation of four refactoring techniques into Mu-FRET. At the time of writing, three of these techniques have been implemented (Extract Requirement, Inline Requirement, and Rename Requirement), with development on the fourth (Move Definition) currently in progress. We have described these refactorings in detail in previous work [4].

Extract Requirement moves a definition from one or more "source" requirements into a newly-created requirement, which also defines a new variable that captures the meaning of the extracted "fragment". This new variable replaces the fragment in the source requirement(s), preserving the underlying meaning while making the requirements easier to understand and maintain.

Inline Requirement is the opposite of Extract Requirement, taking one requirement and merging it into another. The most basic implementation of Inline is as the direct inverse of Extract, operating on source requirements of the form "`if CONDITION System shall satisfy RESPONSE`", replacing `CONDITION` with `RESPONSE` in any selected destination requirements. However, the applicability of Inline Requirement to requirements of other forms (such as those that include a `timing` constraint that may-or-may-not align with that of the destination) is an open question that I intend to explore.

Rename Requirement changes a requirement's name and then updates any references to it to match its new name. FRET already allows a user to rename a requirement, but this existing functionality does not update the 'Parent Requirement ID' field of any of the target's child requirements. This leaves those child requirements with a broken reference, forcing manual updates across the set. Mu-FRET's implementation of the Rename refactoring solves this problem, maintaining consistency.

[1]Mu-FRET repository: https://github.com/valu3s-mu/mu-fret

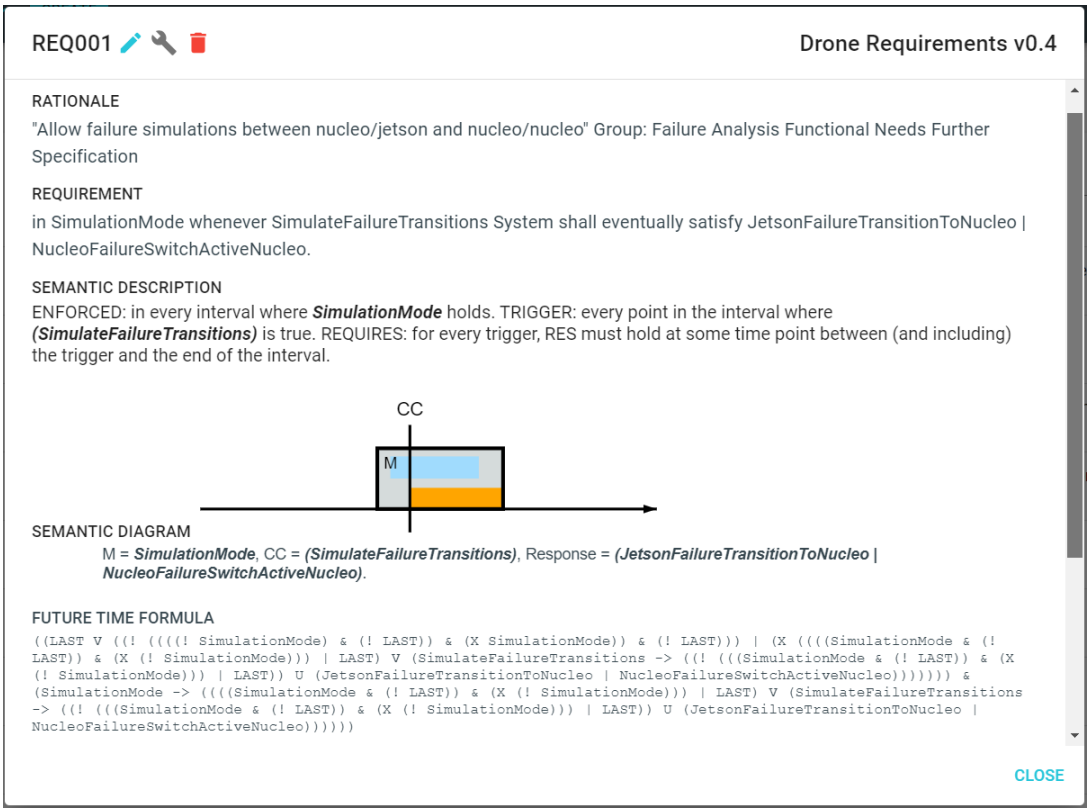

Figure 1: A requirement in FRET. The **Requirement** field shows the FRETISH definition of the requirement. From this is generated the **Semantic Description**, **Semantic Diagram**, **Future Time LTL Formula**, and **Past Time LTL Formula** (not shown). The **Rationale** field includes notes on the requirement from the user.

In addition, robust renaming is very important when considered alongside Extract Requirement, which creates a variable which directly corresponds to the extracted definition. It is important that the names of this requirement and variable stay aligned so that the connection is maintained, and thus Rename Requirement is able to rename both an extracted requirement and it's corresponding variable in a single user-facing action, with the variable name being updated in every requirement where that variable is used. This ability to rename a variable and have it propagate across multiple requirements is not present in the original build of FRET.

Move Definition takes part of one requirement and moves it to another, focusing a requirement on a single responsibility. It differs from Extract Requirement and Inline Requirement in that it does not alter the number of requirements in the set, it only moves definitions between requirements.

Each of these refactoring techniques will be integrated with formal checks that the behaviour of the requirements before the refactoring is the same as after the refactoring, by checking equivalence of the temporal logic formulae using the NuSMV model checker. The exact nature of this check varies depending on the refactoring in question; for example, Extract Requirement checks that the updated source requirement combined with the newly-extracted requirement matches the original version of the source requirement before refactoring, and the extraction is only performed if this check succeeds.

3. Related Work

This work is primarily based on the framework for refactoring natural-language requirements outlined by Ramos et al. [5]. The authors propose five refactorings that are applicable to textual requirements; we determined that four of these could be applied to FRETISH ("Extract Alternative Flows" is not considered applicable as FRETISH does not have a notion of branching requirements). These refactoring techniques are adapted from software refactoring, as laid out by Fowler et al. [6].

As far as we are aware, there is little existing work on applying refactoring to formal or semi-formal requirements, and there is essentially no tool-support for it. Similar work has been conducted by Xuan et al. on using refactoring to improve test suites for analysis of source code [7]. Yu et al. have proposed 10 refactoring rules for episode models [8]. There is also significant existing literature on the application of refactoring to system modeling languages, such as Feature Models [9], UML [10][11] and ADORA [12]. However, these languages are based on diagrams rather than textual descriptions or logics, and as such the results are not directly transferrable.

4. Methodology

This project seeks to answer the following research questions:

- **RQ1:** To what extent are software refactorings applicable to functional requirements?
- **RQ2:** Under precisely what conditions are requirement refactoring techniques applicable? What elements of the structure of a requirement might allow or prevent the use of one or more refactoring techniques?
- **RQ3:** How can refactoring techniques developed for FRETISH requirements be generalised for other requirements languages and formalisms?

The first step in answering these questions will be to complete an intitial implementation of the four chosen refactoring methods. By developing support for refactoring in a requirements engineering tool, we will be able to test how it can be applied to real sets of requirements. Also, the process of developing the software forces us to codify exactly how the refactorings behave; this behaviour can be updated as the theory behind the refactoring process matures, but it is valuable to have a concrete implementation to build from.

Once development on the four refactorings is complete, we aim to evaluate the applicability of each on a number of case studies. This will address Research Question 1. These case studies include the aforementioned Aircraft Engine Controller requirements from VALU3S, a set of requirements for a Mechanical Lung Ventilator developed for the ABZ 2024 case study [13], and the requirements for a tilt-rotor drone in development for the ProVANT Emergentia project. Other case studies from the literature are being considered, but we feel that these case studies in particular are valuable because we have access to the complete elicitation process and artifacts. We believe that refactoring is most useful in the early and middle stages of a project, when the requirements are still a work-in-progress, and thus many completed case studies are too "clean" to gain a marked benefit from applying refactoring. This is worth investigating further, however.

In addition, we will examine the effect that refactoring has on the underlying meaning of the requirements, represented by the LTL formulae. At present, the Mu-FRET refactorings are focused primarily on improving the structure of the FRETISH text of the requirements, with the LTL translations used for checking consistency. However, we want to investigate if a requirement's logical representation can be used to determine whether a given refactoring can be applied. In doing so, we hope to arrive at a set of well-defined criteria for each refactoring that describes exactly when that technique can be used, and when some part of the requirement prohibits its use, addressing Research Question 2. Also, any results gathered from focusing on the LTL specification should be more easily generalisable to requirements languages other than FRETISH, as temporal logics are commonly used in requirements engineering and formal methods. This aligns with Research Question 3.

5. Next Steps & Future Challenges

The implementation of refactoring in Mu-FRET is nearing completion, with two tasks remaining: implement Move Definition, and incorporate NuSMV checks into Inline, Rename, and Move. These tasks are the current highest priority for this project.

Although we have described the Move Definition refactoring in previous work, the exact details of the implementation are not entirely clear. Exactly what parts of a requirement can be moved, and which other requirements are valid options for the movement, will need to be decided during the development process. We have also considered a specialised form of Move Definition called **Pull-Up Definition**, which would move a definition from a child requirement into its parent, representing the idea that the definition is common to all of that requirement's children. Implementing this in Mu-FRET is being considered.

Implementing model checking with NuSMV into the remaining refactorings should not pose a significant challenge, as the capability has already been demonstrated with Extract Requirement. The difficulty will be in deciding exactly what properties should be checked, and ensuring that the implementation is robust when multiple different refactorings are performed on the requirements set over the course of a project.

After all four refactorings have been fully implemented, we will investigate the limitations of the implementation and the conditions that are required to perform refactoring. In essence, our goal is to find the "weakest precondition" for each refactoring, and thus define the widest possible set of requirements that can be refactored. We suspect that the most challenging refactoring to define will be Inline Requirement, because it is often performed with the intention of deleting the source requirement after the inlining has been performed. This means that we need to consider all of the information that a requirement contains across all of its fields, to ensure that none of that information is lost in the refactoring process. Expanding the application of Inline Requirement to arbitrary requirements of a suitable structure, rather than being limited to those generated by Extract Requirement, represents a notable challenge for this project but also a significant result once complete.

After this, as mentioned above, we will investigate how the refactorings change not only the FRETISH text, but also the equivalent Temporal Logic semantics as well. We will need to examine the temporal logic semantics generated by FRET to find any consistent effects of refactoring the FRETISH text. Also, there are questions on how exactly our extracted requirements are templated, which arose from the temporal logic translation. We will need to examine what the optimal template is for specifying the extracted definitions, and this requires a deeper understanding of the relationships between the requirements and how refactoring may affect them.

6. Conclusion

We are underway with an investigation into the application of refactoring on functional requirements. Four refactoring techniques are being implemented in the Mu-FRET tool; three are already implemented and available for use, while the fourth is set to be developed in the near future. The usefulness of these refactorings will be evaluated on at least three case studies, with others from the literature being considered. We will use the temporal logic semantics for the requirements generated by Mu-FRET to draw conclusions about requirements refactoring that can be applied more broadly to other requirements languages.

Acknowledgments

This work is supported by the Maynooth University John & Pat Hume Fellowship.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] R. Barbosa, S. Basagiannis, G. Giantamidis, H. Becker, E. Ferrari, J. Jahic, A. Kanak, M. L. Esnaola, V. Orani, D. Pereira, et al., The VALU3S ECSEL Project: Verification and Validation of Automated Systems Safety and Security, in: Euromicro Conference on Digital System Design, IEEE, 2020, pp. 352–359.

[2] M. Farrell, M. Luckcuck, O. Sheridan, R. Monahan, FRETting About Requirements: Formalised Requirements for an Aircraft Engine Controller, in: Requirements Engineering: Foundation for Software Quality, Springer, 2022, pp. 96–111. doi:10.1007/978-3-030-98464-9_9.

[3] M. Farrell, M. Luckcuck, O. Sheridan, R. Monahan, Towards refactoring fretish requirements, in: NASA Formal Methods Symposium, Springer, 2022, pp. 272–279.

[4] M. Luckcuck, M. Farrell, O. Sheridan, Why just fret when you can refactor? retuning fretish requirements, 2022. URL: https://arxiv.org/abs/2202.05816. doi:10.48550/ARXIV.2202.05816.

[5] R. Ramos, E. K. Piveta, J. Castro, J. a. Araújo, A. Moreira, P. Guerreiro, M. S. Pimenta, R. T. Price, Improving the Quality of Requirements with Refactoring, in: Anais do VI Simpósio Brasileiro de Qualidade de Software (SBQS 2007), Sociedade Brasileira de Computação - SBC, Brasil, 2007, pp. 141–155. doi:10.5753/sbqs.2007.15573.

[6] M. Fowler, K. Beck, Refactoring: improving the design of existing code, The Addison-Wesley object technology series, Addison-Wesley, 1999.

[7] J. Xuan, B. Cornu, M. Martinez, B. Baudry, L. Seinturier, M. Monperrus, B-refactoring: Automatic test code refactoring to improve dynamic analysis, Information and Software Technology 76 (2016) 65–80. URL: https://www.sciencedirect.com/science/article/pii/S0950584916300714. doi:https://doi.org/10.1016/j.infsof.2016.04.016.

[8] W. Yu, J. Li, G. Butler, Refactoring use case models on episodes, in: Proceedings. 19th International Conference on Automated Software Engineering, 2004., 2004, pp. 328–335. doi:10.1109/ASE.2004.1342757.

[9] M. Tanhaei, J. Habibi, S.-H. Mirian-Hosseinabadi, Automating feature model refactoring: A model transformation approach, Information and Software Technology 80 (2016) 138–157. URL: https://www.sciencedirect.com/science/article/pii/S0950584916301422. doi:https://doi.org/10.1016/j.infsof.2016.08.011.

[10] M. Misbhauddin, M. Alshayeb, Uml model refactoring: a systematic literature review, Empirical Software Engineering 20 (2013) 206–251. URL: http://dx.doi.org/10.1007/s10664-013-9283-7. doi:10.1007/s10664-013-9283-7.

[11] R. Van Der Straeten, V. Jonckers, T. Mens, A formal approach to model refactoring and model refinement, Software & Systems Modeling 6 (2006) 139–162. URL: http://dx.doi.org/10.1007/s10270-006-0025-9. doi:10.1007/s10270-006-0025-9.

[12] R. Stoiber, S. Fricker, M. Jehle, M. Glinz, Feature unweaving: Refactoring software requirements specifications into software product lines, in: 2010 18th IEEE International Requirements Engineering Conference, 2010, pp. 403–404. doi:10.1109/RE.2010.59.

[13] M. Farrell, M. Luckcuck, R. Monahan, C. Reynolds, O. Sheridan, Fretting and formal modelling: A mechanical lung ventilator, in: International Conference on Rigorous State Based Methods, 2024, pp. 360–383.