Proceedings of the Design Society, Volume 5: ICED25
https://doi.org/10.1017/pds.2025.10217

Identifying the needs for a Requirements Authoring Design Enabler

Shanae Edwards, Oredola Adebayo and Joshua Summers ,✉

University of Texas at Dallas, USA

✉ joshua.summers@utdallas.edu

ABSTRACT: Requirements engineering is in the design process, translating stakeholder needs into actionable and well-defined specifications. While existing design enablers and tools provide partial solutions, they often fall short in addressing essential aspects such as real-time feedback, lifecycle management, and the use of controlled vocabularies. To bridge these gaps, the Requirements Authoring Design Enabler (RADE), a macro-enabled Excel tool, is presented to support requirement authoring, tracking, and management. RADE integrates features like automated feedback, a dual-mode interface, robust change tracking, and controlled vocabularies. The tool was tested with pre-service engineers with user feedback informing iterative refinements. RADE addresses key challenges in requirements engineering, demonstrating its potential to enhance design outcomes across various domains.

KEYWORDS: requirements, design process, computer aided design (CAD), design education

1. Motivation: The need for a new requirements authoring tool

Requirements engineering is a cornerstone of the design process, transforming stakeholder needs into actionable, well-defined specifications (McLellan et al., 2011; Shankar et al., 2012). Effective requirements authoring ensures alignment between project goals and outcomes, while incomplete requirements often result in project delays and cost overruns (Shankar et al., 2012). Despite advancements in computational tools for requirements engineering, significant gaps remain in supporting requirement prioritization, lifecycle management, and authoring using controlled vocabularies (Bao et al., 2022; Dai et al., 2013; Jain et al., 2009; Stoller, 1988; Wali & Nordin, 2024). This paper outlines the need for a comprehensive Requirements Authoring Design Enabler (RADE) to address these gaps by integrating features like real-time feedback, controlled vocabularies, and robust change tracking. Design enablers such as the Requirements Boilerplate Sanity Checker (RUBRIC) and the Tracing and Control of Engineering Requirements (TRACER), offer functions like boilerplate conformance and traceability (Arora et al., 2013; Stoller, 1988). However, these tools lack features for iterative refinement or educational support to improve authoring practices. As engineering projects grow increasingly complex, there is a need for a unified tool that bridges these gaps, ensuring high-quality requirements authoring and lifecycle management. For this design enabler, the initial stakeholder set includes capstone design students with limited formal instruction in requirements authoring. The specific requirements on the design enabler are extracted from the following literature review.

2. Elicited requirements for RADE

Engineering requirements serve as a bridge between stakeholder needs and project execution, guiding teams through the design and development process (Mattson & Sorensen, 2020; Ullman et al., 2024). These requirements must be clear, complete, and actionable to ensure project success. Incomplete or poorly written requirements often result in misaligned goals, increased costs, and delays (Mullis et al., 2024). A well written requirement, at a minimum, includes a subject, verb, and modality (Joshi &

Summers, 2015a). Additional elements, such as an object and adjuncts, enhance clarity and specificity (Joshi & Summers, 2015a). Consider the requirement: "*The tool must have a controlled vocabulary with no synonyms*". "*The tool*" is the subject; "*must*" is the modality; "*[to] have*" is the verb; "*a controlled vocabulary*" is the object; and "*with no synonyms*" is the adjunct describing the object. Without the object, the requirement lacks meaning while the adjunct adds details. Requirements can also be reformulated in multiple ways to convey the same information such as:

1. *The tool must have a controlled vocabulary.*
2. *The controlled vocabulary must not include synonyms.*

Requirements evolve throughout the project lifecycle, with changes such as splitting, refining, adding, or deleting requirements impacting project outcomes (Morkos et al., 2012). Capturing these changes is critical for understanding the relationship between requirement evolution and project health (Asgher-Nadeem et al., 2024). For instance, tracking changes helps identify the propagation of requirement modifications and their influence on cost and timelines.
To address these challenges, 28 specific requirements justified with relevant literature were elicited for the development of RADE (Table 1). These requirements ensure that users write complete requirements while also supporting future project management and reasoning.

Table 1. Elicited requirements for RADE

Serial No.	Requirement	Justification	Literature
1	The tool should support requirement authoring.	To capture well written requirements	(Carrillo De Gea et al., 2012)
2	The tool must track requirement completeness.	To ensure well written requirements documents	(Dos Santos Soares & Vrancken, 2007)
2.1	The tool should influence users to include a subject in each requirement.	To ensure well written requirement statements	(Hähnle et al., 2002)
2.2	The tool should influence users to include a verb in each requirement.	To ensure well written requirement statements	(Hähnle et al., 2002)
2.3	The tool should influence users to include a predicate phrase in each requirement.	To ensure well written requirement statements	(Hähnle et al., 2002)
2.4	The tool should influence users to include a modality in each requirement	To ensure well written requirement statements	(Hähnle et al., 2002)
3	The tool must link the components of a requirement into a single statement.	To ensure well written requirements documents	(Dos Santos Soares & Vrancken, 2007)
4	The tool must ensure good requirement authoring.	To ensure well written requirement statements	(Carrillo De Gea et al., 2012)
5	The tool must track the date of requirement generation.	To capture historical data	(DelSpina et al., 2018)
6	The tool must ensure that changes in requirements are captured.	To capture historical data	(Morkos et al., 2019)
7	The tool should be used throughout the design process.	To capture historical data	(Ullman et al., 2024)
8	The tool must track whether a requirement is functional or non-functional.	To support reasoning	(Morkos et al., 2019)
9	The tool must implement a criticality for each requirement.	To support reasoning	(Carrillo De Gea et al., 2012)
10	The tool must implement a test method for each requirement.	To support reasoning	(Ullman et al., 2024)

(*Continued*)

Table 1. Continued

Serial No.	Requirement	Justification	Literature
11	The tool must implement a stakeholder for each requirement.	To support reasoning	(Mullis et al., 2024)
12	The tool must implement a justification for each requirement.	To support reasoning	(Carrillo De Gea et al., 2012)
13	The tool must implement risk for each requirement.	To support reasoning	(Ullman et al., 2024)
14	The tool must provide feedback immediately after an error in requirement writing.	To enforce complete requirement writing	(Hähnle et al., 2002)
15	The tool must have a controlled vocabulary with no synonyms.	To enforce complete requirement writing	(Caldwell et al., 2011)
16	The tool should incorporate numeric objects in the predicate phrase field.	To encourage user interaction	(Joshi & Summers, 2015b)
17	The tool must incorporate a user updated subject list.	To encourage user interaction	(Ullman et al., 2024)
18	The tool must incorporate a user updated stakeholder list.	To encourage user interaction	(Carrillo De Gea et al., 2012)
19	The tool must incorporate a user updated testing list.	To encourage user interaction	(Patel et al., 2022)
20	The tool should be available to users 24 hours a day.	To encourage user interaction	(Hoffmann et al., 2004)
21	The tool must implement free response in the predicate phrase field.	To encourage user interaction	(Hoffmann et al., 2004)
22	The tool must incorporate a user updated owner list.	To encourage user interaction	(DelSpina et al., 2018)
23	The tool must support reasoning.	To classify requirement statements	(Morkos et al., 2019)
24	The tool must display requirements in multiple views	To allow for database and document views	(Hoffmann et al., 2004)

3. Comparative analysis of existing tools

Table 2 presents a comparative analysis of 15 existing tools alongside RADE to illustrate identified gaps and feature support. Requirements engineering involves critical activities such as elicitation, modelling, analysis, validation, verification, and management, all of which shape the success of a design project (Barcelos et al., 2024; Umar & Lano, 2024). Requirements play a significant role in the design process by establishing a clear understanding between stakeholders and project teams (Ullman et al., 2024).

Table 2. Comparative analysis of requirements modelling tools

Tools	Feedback (R.14)	Boilerplate	Tracking Changes	Prioritization	Lifecycle Management	Documentation	Controlled Vocabulary
TRACER (Stoller, 1988)			X	X	X	X	
Telelogic DOORS[1]			X		X	X	
Starbase Caliber-RM (Sud, 2003)			X	X	X	X	X
LOLITA (Mich & Garigliano, 2000)	X					X	
OmniVista On YourMark Pro (Sud, 2003)			X		X	X	
QuARS (Fabbrini et al., 2001)	X					X	
RTM Workshop 5.0 (Sud, 2003)		X		X		X	

(*Continued*)

[1] https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/doors/9.7.0?topic=overview-doors (accessed 2024.11.25)

Table 2. Continued

Tools	Feedback (R.14)	Boilerplate	Tracking Changes	Prioritization	Lifecycle Management	Documentation	Controlled Vocabulary
Rational Suite AnalystStudio[2]			X	X	X	X	
RDT 3.0 (Sud, 2003)			X	X		X	
RAT (Jain et al., 2009)	X					X	X
RUBRIC (Arora et al., 2013)	X	X				X	
TRC Requirements Quality Suite (RQS)[3]	X	X					
RM2Doc (Bao et al.,2022)						X	
RCT (Wali & Nordin, 2024)		X				X	
DRed (Dai et al., 2013)	X		X			X	
Requirements Authoring Design Enabler	**X**	**X**	**X**	**X**	**X**	**X**	**X**

A robust requirements management tool must facilitate these activities throughout the project lifecycle, enabling users to track changes, prioritize requirements, and ensure clarity and consistency. Despite the availability of various requirements modelling tools, significant gaps persist. Many existing tools excel in specific areas, such as documentation or tracking changes, but fail to integrate all essential features (Bao et al., 2022; Dai et al., 2013; Jain et al., 2009; Mich & Garigliano, 2000; Stoller, 1988; Sud, 2003; Wali & Nordin, 2024). RADE was developed to fill this void, providing comprehensive support for feedback, boilerplate guidance, change tracking, prioritization, lifecycle management, and controlled vocabulary. Existing requirements modelling tools provide valuable but fragmented support for requirements engineering. RADE integrates critical functionalities; real-time feedback, boilerplate guidance, change tracking, prioritization, lifecycle management, and controlled vocabularies—into a single, user-friendly platform. By doing so, RADE positions itself as a holistic solution to modern requirements engineering challenges.

3.1. Feedback

Providing real-time feedback helps users refine their requirements while writing (Wang et al., 2024). Tools like LOLITA and QuARS provide limited feedback, primarily focused on detecting ambiguities (Fabbrini et al., 2001; Mich & Garigliano, 2000). For instance, LOLITA identifies lexical and sentence ambiguities, offering guidance to improve clarity (Mich & Garigliano, 2000). Similarly, QuARS detects defects such as vagueness and under specification (Fabbrini et al., 2001). However, these tools lack the ability to ensure compliance with boilerplate structures or provide actionable error messages. RADE enhances this capability by offering immediate feedback for incomplete requirements.

3.2. Boilerplates guidance

Boilerplates provide standardized templates for writing requirements, which ensures consistency across requirement documents (Ibrahim et al., 2014). By guiding authors through predefined structures, boilerplates simplify the process of creating requirements that are clear, actionable, and aligned with industry standards (Antoniou et al., 2024; Lim et al., 2024). Tools like RCT and RUBRIC adopt established templates, such as those from the International Requirements Engineering Board (IREB), to support this standardization (Arora et al., 2013; Wang et al., 2024). However, these tools often rely on static templates that may not be easily customizable or user-friendly. RADE advances this functionality by incorporating dynamic boilerplates directly into its interface, using structured columns that guide users through the requirement-writing process. This approach ensures that every requirement includes critical components—such as subject, verb, and modality—while allowing flexibility for user-specific needs, such as custom object or adjunct fields.

3.3. Change tracking

Requirements often evolve during a project due to shifting stakeholder needs, unforeseen constraints, or technological advancements (Morkos et al., 2019). Effective change tracking allows teams to document these modifications and analyse their impact on project goals (Stoller, 1988). Tools like TRACER and

[2] https://public.dhe.ibm.com/software/rational/web/datasheets/2003/d801f-analyst-studio.pdf (accessed 2024.11.25)

[3] https://www.reusecompany.com/ (accessed 2024.11.25)

Telelogic DOORS provide hierarchical tracing of requirements, enabling users to follow the propagation of changes from parent to child requirements (McLellan et al., 2011; Stoller, 1988; Sud, 2003). This feature is particularly useful for impact assessments, as it highlights dependencies and potential conflicts (Sud, 2003). These tools can be complex to set up and require significant manual effort to maintain traceability. RADE simplifies change tracking by integrating dedicated columns for adding, modifying, or deleting requirements. Each change is date-stamped, ensuring a complete history of requirement evolution. This functionality not only supports traceability but also enables teams to monitor trends in requirement modifications, such as frequent additions or deletions.

3.4. Prioritization

Prioritization is a critical feature for managing competing requirements, particularly in resource-constrained projects (Berntsson Svensson & Torkar, 2024). Existing tools like Starbase Caliber-RM and Rational Suite AnalystStudio use attributes such as cost, value, or risk to rank requirements (Sud, 2003). These rankings help teams focus on high-impact requirements that deliver the greatest value to stakeholders. However, prioritization in these tools is often subjective, relying heavily on user inputs without standardized criteria. RADE addresses this challenge by introducing a criticality scale, where requirements are classified into three levels of importance: low, medium, and high. This scale is intuitive and aligns with common project management frameworks, allowing teams to quickly identify critical requirements and allocate resources accordingly (Ullman et al., 2024).

3.5. Project lifecycle management

The lifecycle of a requirement extends from its initial elicitation to its validation, implementation, and eventual retirement (Sud, 2003). Effective lifecycle management ensures that requirements remain relevant, aligned with project goals, and adequately documented at each stage (Halbleib, 2003; Hoffmann et al., 2004). Tools like Telelogic DOORS offer multi-level traceability, assigning states (e.g., "new," "in review," or "approved") to requirements, while Starbase Caliber-RM maintains version histories for easy comparison of changes over time (Sud, 2003). While these features are valuable, they often operate in silos, requiring manual effort to integrate with other project management tools. RADE enhances lifecycle management by combining tracking with date-stamped documentation, ensuring that every change is logged and visible. RADE's integration with prioritization and controlled vocabulary features allows users to manage requirements holistically, ensuring consistency and relevance throughout the design process.

3.6. Documentation

Documentation is essential for managing requirements across diverse stakeholders and project phases (Behutiye et al., 2022). Tools such as RDT 3.0 and Starbase Caliber-RM, provide basic documentation features like traceable links and version control (Sud, 2003). However, these tools often lack flexibility in organizing and presenting requirements, which can hinder accessibility and usability. RADE addresses this limitation by offering multiple views: a database view for granular analysis and a document view for higher-level summaries. Requirements in RADE are organized using serialized numbering and structured fields, making it easy to track relationships and dependencies. Additionally, RADE's integration of documentation with change tracking and lifecycle management ensures that every requirement is not only well-documented but also kept up-to-date as projects evolve.

3.7. Controlled vocabulary

Controlled vocabularies standardize the language used in requirement statements, reducing ambiguity and ensuring consistency across documents. Tools like Starbase Caliber-RM and RAT employ predefined glossaries to define acceptable terms for specific industries or projects (Jain et al., 2009; Sud, 2003). Starbase Caliber-RM includes glossaries for industry-specific terms, while RAT offers entity and action glossaries for system components and functionalities. However, these vocabularies are often static and may not accommodate project-specific terminology. RADE improves on these features by incorporating two controlled vocabularies. The verb vocabulary classifies requirements as functional or non-functional, while the modality vocabulary captures criticality (e.g., "must," "should"). Users can also update these vocabularies to reflect unique project needs, providing both standardization and flexibility.

4. RADE: development approach and system architecture

RADE was developed with a robust system architecture (Figure 1) to address the complexities of requirement authoring, management, and lifecycle tracking. Its design prioritizes user accessibility, real-time feedback, and structured documentation, leveraging a macro-enabled Excel workbook as its graphical user interface (GUI). The architecture integrates multiple column types, controlled vocabularies, and feedback mechanisms to ensure high-quality requirements and seamless user interaction.

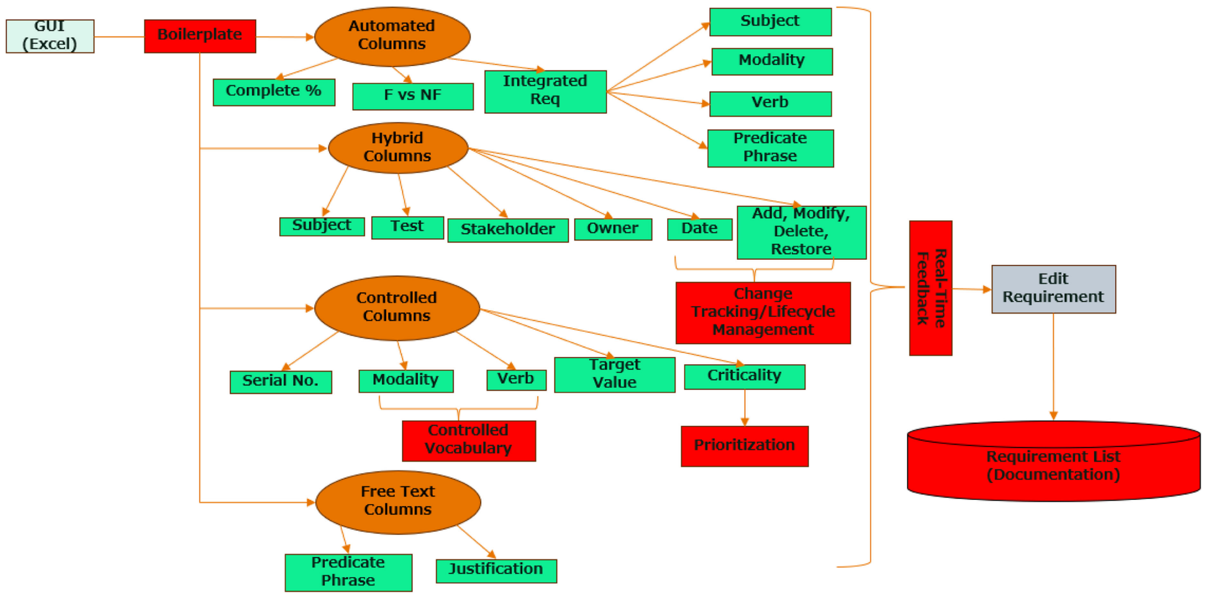

Figure 1. RADE system architecture

4.1. System architecture

The RADE system architecture consists of **automated, hybrid, controlled,** and **free-text columns**, each playing a distinct role in requirement authoring and management. These components are integrated into a boilerplate structure, ensuring consistency and adaptability across projects.

4.1.1. Automated colums

Automated columns perform predefined functions without requiring user input, simplifying the requirement authoring process. The *Complete %* column calculates the percentage completeness of a requirement by assessing the presence of its components, ensuring users are aware of missing elements. The *Functional vs. Non-Functional (F vs NF)* column categorizes requirements automatically as either functional, such as actions the system must perform, or non-functional, such as performance criteria. Additionally, the *Integrated Requirement* column combines key components, including the subject, modality, verb, and predicate phrase into a single, comprehensive requirement statement.

4.1.2. Hybrid columns

Hybrid columns allow for limited user input while maintaining structured options for consistency. For instance, the *Subject, Test, Stakeholder*, and *Owner Lists* can be customized by users to reflect project-specific contexts, ensuring adaptability. Change tracking is facilitated by the *Add, Modify, Delete*, and *Restore* columns, which capture updates to requirements throughout their lifecycle. These are complemented by the Date column, which tracks changes to aid traceability and historical analysis.

4.1.3. Controlled columns

Controlled columns enforce uniformity by restricting inputs to predefined options derived from controlled vocabularies or standard conventions. The *Modality* and *Verb* columns, as core elements of RADE's controlled vocabulary, ensure consistent phrasing across requirements. The *Modality* column defines criticality levels, distinguishing between constraints such as "must" and criteria such as "should," while the *Verb* column classifies actions as either functional or non-functional. Other controlled columns include the *Serial Number*, which assigns unique identifiers to each requirement, and the *Target Value* and *Criticality* columns, which capture prioritization details, allowing requirements to be ranked based on their importance.

4.1.4. Free-text columns

Free-Text columns provide users with the flexibility to elaborate on requirement details. The *Predicate Phrase* column allows users to add essential context to a requirement statement, ensuring precision and completeness. Similarly, the *Justification* column provides space to explain the rationale behind each requirement, fostering transparency and aiding stakeholder understanding.

4.2. Controlled vocabulary

Controlled vocabularies are a fundamental element of RADE's architecture, ensuring consistency, clarity, and precision in requirement authoring. RADE incorporates two controlled vocabularies: verbs and modalities, both developed through a systematic and rigorous process.
The *Verb Vocabulary* was created by analysing multiple sources, including a wind turbine requirements repository, NASA's NPR 7120, and academic studies on design vocabularies (Hirtz et al., 2002). Overlapping terms from these sources were identified, redundancies removed, and synonyms consolidated, resulting in a refined list of 64 standardized verbs. This vocabulary categorizes requirements as functional or non-functional, aiding classification and analysis.
The *Modality Vocabulary* focuses on criticality, distinguishing between constraints and criteria. This classification allows users to prioritize requirements based on their necessity and importance within the project scope. By guiding users in selecting appropriate terms, the modality vocabulary ensures uniformity and facilitates decision-making in requirement prioritization. Together, these controlled vocabularies serve as essential tools for minimizing ambiguity, ensuring consistency, and maintaining high standards in requirement documentation.

4.3. Real-time feedback mechanism

Real-time feedback is a critical feature of RADE, designed to enhance the accuracy and completeness of requirement authoring. The system dynamically generates feedback based on the completeness and correctness of user entries, helping to identify and rectify errors during the writing process. For example, if key fields such as *Subject, Verb*, or *Predicate Phrase* are left blank, RADE triggers context-specific prompts such as "*The Requirement Needs a Subject*" or *"Please Complete the Requirement."* Additionally, incomplete requirements are flagged in the *Complete %* column, providing users with a visual cue to address gaps promptly. This feedback loop improves the quality of requirements and functions as an educational tool. By exposing users to best practices in real-time, RADE helps users develop a deeper understanding of structured requirement writing, making it especially valuable for novice authors.

4.4. Change tracking and lifecycle management

RADE integrates change tracking and project lifecycle management features to support iterative and dynamic design processes. These features ensure that requirement documents remain aligned with project goals while adapting to changes over time. The *Add, Modify, Delete*, and *Restore* columns, combined with date-stamped *Date* entries, create a detailed record of requirement evolution. This structured tracking allows users to monitor the history of modifications, facilitating traceability and ensuring accountability throughout the project lifecycle. Requirements are kept relevant and actionable from their initial elicitation to their final documentation.

4.5. Dual-mode interface

RADE enhances usability through its dual-mode interface, catering to technical and non-technical users. The *Database View* provides a comprehensive display of all requirement fields in a structured format, ideal for detailed analysis, tracking, and technical review. The *Document View* summarizes requirements, focusing on key elements and justifications, making it well-suited for stakeholder communication. This dual-mode interface ensures accessibility for diverse audiences, allowing teams to seamlessly switch between detailed technical insights and high-level summaries.

5. Implementation of the design enabler

RADE underwent extensive testing across diverse user groups, including pre-service engineers (undergraduate students), the UTDesign-R lab, and the NASA Micro-g NExT team. These testing environments offered valuable insights into the tool's usability, effectiveness, and areas for improvement. The iterative feedback gathered from these groups informed RADE's development, driving refinements that enhanced its functionality and overall user experience.

The feedback process was integral to RADE's iterative development. Undergraduate students using version 1.2 of RADE identified several issues that were addressed in subsequent updates. For instance, inconsistencies in drop-down columns and malfunctions in the real-time feedback mechanism were observed in some workbooks. Users also found certain column definitions unclear and deemed some fields unnecessary. Additionally, the *Integrated Requirement* column, which concatenated key components such as subject, modality, and predicate phrase, did not display correctly in all cases.

Another area of concern was the data validation setup required in version 1.2, which made updating stakeholder, owner, and test columns time-consuming. To resolve this, version 1.3.1 automated data validation on the back end, allowing users to directly update their options in the vocabulary sheet. These adjustments, along with fixes to other inconsistencies, significantly improved RADE's usability and functionality. Feedback from the NASA team highlighted the tool's practical utility but also underscored the need for further enhancements to support large-scale, iterative projects.

6. Conclusions and future work

Requirement modelling tools play a critical role in optimizing the engineering design process, yet many existing tools fail to address key needs such as prioritization, comprehensive lifecycle management, and real-time feedback for requirement authors. RADE was developed to fill these gaps, incorporating features like controlled vocabularies, automated feedback mechanisms, and robust tracking capabilities. Its development involved creating a controlled vocabulary, implementing the authoring design enabler in Excel, writing a user manual, and conducting usability studies.

While RADE has demonstrated significant promise, limitations remain. One challenge is the need to unblock macros when accessing the tool on certain devices, such as tablets. Future work will involve analysing the evolution of requirements authored with RADE in capstone industry sponsored projects. The design enabler will be used to capture historical data, focusing on typology and the balance between constraints and criteria based on (Joshi & Summers, 2014; Summers et al., 2014).

References

Antoniou, C., Kravari, K. & Bassiliades, N. (2024). Semantic requirements construction using ontologies and boilerplates. *Data & Knowledge Engineering*, 152, 102323. https://doi.org/10.1016/j.datak.2024.102323

Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F. & Gnaga, R. (2013). RUBRIC: a flexible tool for automated checking of conformance to requirement boilerplates. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 599–602. https://doi.org/10.1145/2491411.2494591

Asgher-Nadeem, M., Hasnain, M., Saleemi, M. M., Awais-Mohsin, M., Adeel-Ansari, M. & Essalah, W. (2024). *Challenges in Requirements Engineering for IoT Solutions | Journal of Computing & Biomedical Informatics*. 6(2). https://doi.org/0.56979/602/2024

Bao, T., Yang, Y., Yang, J. & Yin, Y. (2022). RM2Doc: A Tool for Automatic Generation of Requirements Documents from Requirements Models. 2022 *IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 188–192.

Barcelos, L. V., Antonino, P. O. & Nakagawa, E. Y. (2024). Requirements engineering in industry 4.0: State of the art and directions to continuous requirements engineering. *Systems Engineering*, 27(5), 955–971. https://doi.org/10.1002/SYS.21753

Behutiye, W., Rodríguez, P., Oivo, M., Aaramaa, S., Partanen, J. & Abhervé, A. (2022). Towards optimal quality requirement documentation in agile software development: A multiple case study. *Journal of Systems and Software*, 183, 111112. https://doi.org/10.1016/J.JSS.2021.111112

Berntsson Svensson, R. & Torkar, R. (2024). Not all requirements prioritization criteria are equal at all times: A quantitative analysis. *Journal of Systems and Software*, 209, 111909. https://doi.org/10.1016/J.JSS.2023.111909

Caldwell, B. W., Sen, C., Mocko, G. M. & Summers, J. D. (2011). An empirical study of the expressiveness of the functional basis. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AI EDAM*, 25(3), 273.

Carrillo De Gea, J. M., Nicolás, J., Fernández Alemán, J. L., Toval, A., Ebert, C. & Vizcaíno, A. (2012). Requirements engineering tools: Capabilities, survey and assessment. *Information and Software Technology*, 54(10), 1142–1157. https://doi.org/10.1016/J.INFSOF.2012.04.005

Dai, W., Aurisicchio, M. & Armstrong, G. (2013). An IBIS Based Approach for the Analysis of Non-Functional Requirements. *Proceedings of the ASME Design Engineering Technical Conference*, 7, 591–602. https://doi.org/10.1115/DETC2012-71023

DelSpina, B., Gilliam, S., Summers, J. & Morkos, B. (2018). Corporate requirement culture in development of a large scale medical system: A case study, 2621–2632.

Dos Santos Soares, M. & Vrancken, J. (2007). Requirements specification and modeling through SysML. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 1735–1740. https://doi.org/10.1109/ICSMC.2007.4413936

Fabbrini, F., Fusani, M., Gnesi, S. & Lami, G. (2001). The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. *26th Annual NASA Goddard Software Engineering Workshop, IEEE/NASA SEW* 2001, 97–105. https://doi.org/10.1109/SEW.2001.992662

Hähnle, R., Johannisson, K. & Ranta, A. (2002). An Authoring Tool for Informal and Formal Requirements Specifications. In G. Goos, J. Hartmanis & J. van Leeuwen (Eds.), *International Conference on Fundamental Approaches to Software Engineering* (pp. 233–248). Springer.

Halbleib, H. (2003). Requirements management. *Information Systems Management*, 21(1), 8–14. https://doi.org/10.1201/1078/43877.21.1.20041201/78982.2

Hirtz, J., Stone, R. B., McAdams, D. a., Szykman, S. & Wood, K. L. (2002). A functional basis for engineering design: reconciling and evolving previous efforts. *Research in Engineering Design*, 13(2), 65–82. https://doi.org/10.1007/s00163-001-0008-3

Hoffmann, M., Kühn, N., Weber, M. & Bittner, M. (2004). Requirements for requirements management tools. *Proceedings of the IEEE International Conference on Requirements Engineering*, 301–308. https://doi.org/10.1109/ICRE.2004.1335687

Ibrahim, N., Wan Kadir, W. M. N. & Deris, S. (2014). Documenting requirements specifications using natural language requirements boilerplates. 2014 *8th Malaysian Software Engineering Conference, MySEC 2014*, 19–24. https://doi.org/10.1109/MYSEC.2014.6985983

Jain, P., Verma, K., Kass, A. & Vasquez, R. G. (2009). Automated review of natural language requirements documents: Generating useful warnings with user-extensible glossaries driving a simple state machine. *Proceedings of the 2nd India Software Engineering Conference, ISEC* 2009, 37–45. https://doi.org/10.1145/1506216.1506224

Joshi, S. & Summers, J. D. (2014, August 17). Tracking Project Health Using Completeness and Specificity of Requirements: A Case Study. Volume 3: *16th International Conference on Advanced Vehicle Technologies; 11th International Conference on Design Education; 7th Frontiers in Biomedical Devices*. https://doi.org/10.1115/DETC2014-35020

Joshi, S. & Summers, J. D. (2015a). Requirements change: Understanding the type of changes in the requirements document of novice designers. *International Journal of Mechanical Engineering Education*, 43(4), 286–304. https://doi.org/10.1177/0306419015612348

Joshi, S. & Summers, J. D. (2015b). Requirements Evolution: Understanding the Type of Changes in Requirement Documents of Novice Designers. *ICoRD'15–Research into Design Across Boundaries Volume* 2, 471–481.

Lim, J. W., Chiew, T. K., Su, M. T., Ong, S., Subramaniam, H., Mustafa, M. B. & Chiam, Y. K. (2024). Test case information extraction from requirements specifications using NLP-based unified boilerplate approach. *Journal of Systems and Software*, 211, 112005. https://doi.org/10.1016/j.jss.2024.112005

Mattson, C. A. & Sorensen, C. D. (2020). Product Development: Principles and Tools for Creating Desirable and Transferable Designs (1st ed.). *Springer*.

McLellan, J. M., Morkos, B., Mocko, G. G. & Summers, J. D. (2011). Requirement Modeling Systems for Mechanical Design: A Systematic Method for Evaluating Requirement Management Tools and Languages. *Proceedings of the ASME Design Engineering Technical Conference*, 3(PARTS A AND B), 1247–1257. https://doi.org/10.1115/DETC2010-28989

Morkos, B., Joshi, S. & Summers, J. D. (2019). Investigating the impact of requirements elicitation and evolution on course performance in a pre-capstone design course. *Journal of Engineering Design*, 30(4–5), 155–179.

Morkos, B., Shankar, P. & Summers, J. D. (2012). Predicting requirement change propagation, using higher order design structure matrices: an industry case study. *Journal of Engineering Design*, 23(12), 905–926. https://doi.org/10.1080/09544828.2012.662273

Mullis, J., Chen, C., Morkos, B. & Ferguson, S. (2024). Deep Neural Networks in Natural Language Processing for Classifying Requirements by Origin and Functionality: An Application of BERT in System Requirements. *Journal of Mechanical Design*, 146(4). https://doi.org/10.1115/1.4063764/1169299

Patel, A. R., Murphy, A., Summers, J. D. & Tahera, K. (2022). Testing in Engineering Design: What Are We Teaching. *Proceedings of the Design Society*, 2, 2363–2372. https://doi.org/10.1017/PDS.2022.239

Shankar, P., Morkos, B. & Summers, J. D. (2012). Predicting Requirement Change Propagation using higher order design structure matrices: an industry case study. *Research in Engineering Design*, 23(12), 905–926.

Stoller, R. L. (1988). TRACER: A tool for tracing and control of engineering requirements. *IEEE International Conference on Engineering Management*, 27–36. https://doi.org/10.1109/IEMC.1988.34907

Summers, J. D., Joshi, S. & Morkos, B. (2014, August 17). Requirements Evolution: Relating Functional and Non-Functional Requirement Change on Student Project Success. Volume 3: *16th International Conference on Advanced Vehicle Technologies; 11th International Conference on Design Education; 7th Frontiers in Biomedical Devices*. https://doi.org/10.1115/DETC2014-35023

Ullman, D. G., Summers, J. D. & Fielding, J. (2024). *Product Design Best Practices (1st ed.)*. BVT Publishing.

Umar, M. A. & Lano, K. (2024). Advances in automated support for requirements engineering: a systematic literature review. *Requirements Engineering*, 29(2), 177–207. https://doi.org/10.1007/s00766-023-00411-0

Wali, S. S. A. S. & Nordin, A. (2024). The Design and Development of a Requirements Conformance Tool (RCT). *International Journal on Perceptive and Cognitive Computing*, 10(2), 74–79. https://doi.org/10.31436/IJPCC.V10I2.487

Wang, S., Mitchell, J. & Piech, C. (2024). A Large Scale RCT on Effective Error Messages in CS1. *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, 1, 1395–1401. https://doi.org/10.1145/3626252.3630764