Requirements Formality Levels Analysis and Transformation of Formal Notations into Semi-formal and Informal Notations

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

***Abstract*—It is pivotal to have well-specified requirements to eliminate errors at an early stage of the system development life cycle. Some quality standards recommend the use of formal methods – mandate requirements to be expressed in formal notations – to detect errors. However, formal notations are not suitable for non-experts and may not be understood by all the stakeholder. To fix this, bidirectional transformations among requirement representation levels are required to maintain traceability and facilitate the communication of requirements among all the involved parties. This paper reflects on the different formality levels of requirements specifications including: informal, semi-formal, and formal notations. In addition, an automated multi-layer transformation approach is proposed to enable bi-directional transformation among requirements levels.**

***Index Terms*—Requirements engineering, Informal notation, Semi-formal notation, Formal notations, Transformation**

I. INTRODUCTION

The complexity of modern systems is rapidly increasing as a result of the incorporation of cutting edge technology in various fields (e.g., automotive, robotics, and Internet-of-Things). These systems have special characteristics over classical systems (e.g., integrating multiple subsystems, scalability, reusability, and stringent requirements measures for: reliability, safety and security, etc.) [1], [2]. In addition, depending on the scope of application, development errors in such systems (e.g., inconsistency, incompleteness, and incorrectness) can lead to catastrophic consequences, severe losses, and hazardous operational failures. The earlier the detection and resolution of such errors, the better quality and control through the development life cycle [3].

Requirements Engineering (RE) is the first phase within the development life cycle [4] and thus contributes greatly to the overall quality of the developed system and achieving a successful and efficient development. The core artefact for this process is the requirements specifications document. There are three levels for representing requirements specifications: informal, semi-formal and formal notations. Each representation level has its own strengths and weaknesses. Informal (natural language) offers the easiest form of communicating requirements but is inherently ambiguous, incomplete, and imprecise. Formal notations are precise and unambiguous but require mathematical background and expertise. And semi-formal notations offer an intermediate trade-off between the two, but is not suitable for all scenarios or domains. Thus, the transformation between these levels of formality is required to maintain fast and efficient communication at different stages of the development between all the contributors in the RE team (e.g,, non technical users, formal methods experts, etc) [5]–[7].

In the last three decades, several approaches were proposed for automating the transformation processes. The majority of such approaches focus on transforming requirements from the informal and semi-formal levels to the formal level. This is done to allow formal methods to detect quality issues within the requirements [8], [9]. Yet the problem of transforming requirements from a more formal level of representation to a less formal one is still an open problem. In addition, to the best of our knowledge, very few work has provided a comparison between the different formality levels or discussed their features and the possible transformations between them (e.g., [5], [10]) . To address these gaps, in this paper we provide:

- An analysis of the formality levels for requirements representation and the current state of transformations among them for a better understanding of the problem.
- A multi-layer automated transformation approach from formal notation to semi-formal, then to informal notation.

II. REQUIREMENTS REPRESENTATION FORMALITY LEVELS

Formality levels notations are defined based on the concepts of syntax and semantics visualised in Fig.1. In [11], these concepts are defined from the linguistic perspective and the field of formal language as indicated below:

. **Syntax:** The syntax of a language is the set of rules that define structured sentences or fragments of the language [12] (e.g., grammatical phrases or sentence structure in natural languages).

DOI reference number: 10.18293/SEKE2021-199

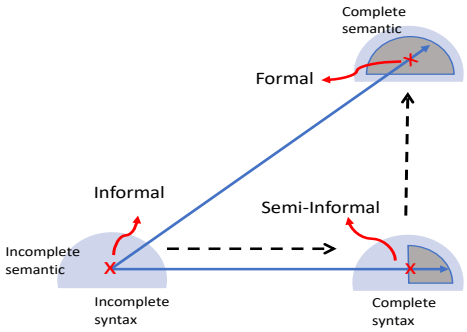

Fig. 1. Formality Levels from syntax and semantic perspectives

. **Completely defined syntax:** The syntax of a language L is completely defined if and only if, for any fragment S, it can be verified whether or not S belongs to L [13].
. **Semantic:** The semantic of a language is the possible interpretations or meanings of a fragment.
. **Completely defined semantics:** The semantic of a language is completely defined if and only if, for any given fragment S that belongs to L, there is only one interpretation. This can be confirmed by transforming the language notation into a mathematical system where proofs can be performed [11].

A. *Informal Notations*

Quality standards [14] define informal notations as a technique with incomplete defined syntax, that is used to specify requirements. Having incomplete syntax also means that the semantics aspect is also incomplete.

From the industry perspective, informal notations have a similar definition where it is defined as free-textual or free-styled requirements [5], [15], [16]. However, there are two fundamental constraints on the use of informal notations: (1) auto-completeness – each individual requirement should be fully understandable, and (2) implied references should be avoided as recommended in [16]. The absence of rules controlling the writing style, makes requirements specification easier but affects the uniformity as it depends on the system engineer's writing, background and expressiveness skills. Lacking uniformity makes the requirements vulnerable to ambiguities and inconsistencies [17].

The most commonly used informal notation today is Natural Language (NL) (i.e., usually English) [14]. The principle advantage of NL is that it is well known. Thus, it is a good communication method among non-technical users (e.g., customers, stakeholders, etc.), and technical users (e.g., developers, experts, etc.). In addition, there is no need for training to use NL in specifying requirements. This minimises the time needed to create the first project artifact. More details about what are the sections of the specification document, how each section is structured and what are the main elements that should be contained are available here [15].

Despite the advantages, writing requirements in NL has many drawbacks. First, the expressiveness power of informal notation allows requirements documents to contain all kinds of freedom (e.g., ambiguity, contradiction, inconsistency [18]). As NL is inherently ambiguous, different persons may have different interpretations of the same requirement. This may lead to implementing unintended functions or implementing more functions than the intended ones. Having unnecessary functions increases the complexity of the system and consequently the number of errors [19]. In addition, missing a system function would drive the system into incorrect states. Secondly, NL is hard to maintain as the proper grouping of related requirements can not be ensured. Thirdly, due to the free-style of writing, it is hard to verify requirements and make sure that each process has correct input and output.

Current research witnesses a lot of progress towards improving the quality of informal/textual requirements. The existing approaches can be classified into two categories: (1) detecting quality issues (e.g., ambiguity, inconsistency, incompleteness, etc.) in informal requirements –helping engineers refine written requirements– (e.g., [20], [21]), (2) providing defined formats (templates, patterns, boiler-plates, constrained NL) for engineers to utilise while (re-)writing requirements (e.g., [22]).

B. *Semi-formal Notations*

Semi-formal notations provide an intermediate layer between informal and formal notations. It is a technique for describing requirements with a completely defined syntax and may have incomplete semantics, as defined in most quality standards [14]. From the industry perspective, semi-formal notations are usually either a form of a graphical representation of the system [5], [10], [15] or Constrained Natural Language "CNL" [10], [23] that is developed to minimise ambiguities and improve the readability of the requirements. In fact, there is no contradiction between these definitions because both graphical representations and CNL have completely defined syntax and poorly defined semantics [5], [15].

Graphical representations utilise graphical structures to represent the system [10]. Although not all the requirements issues are eliminated, the effectiveness of graphical representations can be very high. Such representations are popular as they can improve the quality of the requirements with a slight learning curve and give an insight about the system from different points of view (e.g., UML [24], Sysml [25], URN [26], ER-diagrams [27], etc). A graphical representation may be accompanied with some logical languages to raise expressiveness (e.g., OCL is attached to UML in [10]).

CNL is a textual writing with constraints on styling, syntax, vocabulary or mix of them [28]. This limits the engineers to only using the predefined writing style to improve the quality of the written requirements. Commonly, this manifests as boilerplates [29] or Structured English [30] following the defined rule(s). This constrains how the requirements are textually represented by layout and vocabulary (CNL might be impractical if too restrictive [31]).

Semi-formal notations have several advantages: (1) Eliminate, in most cases, issues with the requirements represented in informal notations as the complete syntax limits the expressed variations of a requirement [5]. (2) More comprehensible

than informal notations (detailed descriptions in NL may be confusing when describing complex systems). (3) Graphical representations can give an abstract view of the system providing a better understanding of the system [10]. Visualising the entire system helps find otherwise unnoticed gaps. However, the major drawback is -due to the lack of complete semantics- everyone may have their own interpretation [5], [14].

C. Formal Notations

Formal notations are mainly used by formal methods for requirements model verification. They have a completely defined syntax and semantics as defined by the quality standards [14]. Similarly, formal notations are used in the industry as precise notations based on the concept of mathematics to ensure the verification applicability on requirements specifications [15], [32], [33]. Formal notations require strong expertise in mathematics set theory and predicate logic to have the ability to state or understand the formalised requirements of a system [15]. Thus, many (except those with the right expertise) can not easily or clearly understand most of these notations. The most widely used formal notations include (Temporal Logic [34], Z [35], SAL [36], etc.).

A major advantage of formal notations is reducing the development cost and time by discovering errors in the early stages of the software development life cycle. Due to the absence of ambiguity in formal notations, it is reliable to verify whether the system conforms to the specifications or not. Another advantage is improving the quality of the system as the requirements are stated in a precise and consistent manner [32]. Moreover, formal notations may help generate full/partial code [5]. In contrast, non-technical users require extensive time consuming in training to understand formal notations. Large numbers of non-technical users find formal notations very complex [18].

III. FORMALITY LEVELS TRANSFORMATIONS

From the previous section, it is clear that the level of formality of the requirements affects the quality of the represented requirements and the understand-ability among the people involved in the development life cycle (technical and non technical). Thus, maintaining the requirements consistency and traceability is pivotal [5] and can be controlled through transformations. Inspired by the need –in both research and industry– of having viable transformations among the levels of formality [5]–[7], we outline the existing formality levels transformations as follows:-

Informal notations to Semi-formal notations: the popular direction is to transform informal notations into graphical representations. This can be accomplished by applying a set of linguistic parsing rules on free-text [37], [38]. In [39] and [40], NLP-based extraction approaches are proposed for transforming NL-requirements into extracting goal-use-cases and requirements key elements proposed in [41] respectively. Alternatively, in [42], [43], use case models are extracted from NL-requirements through matching the input requirements against a simple set of regular expressions.

Informal notations to Formal notations: type of transformation related to generating a formal notation given free-textual requirements. Linguistic analysis is a viable technique coping with this type of transformation as illustrated in [23], [44]. These approaches are restricted to and reliant on the accepted subset of NL and the corresponding set of hand-crafted rules, that are domain-specific and can only work for limited scenarios [9]. Alternatively, the combined work proposed in [41] and [40] together can be considered as a more flexible and domain-independent approach that transforms clause-based free-textual requirements into formal language.

Semi-formal notations to Formal notations: this type focuses on transforming either graphical representations or textual-based semi-formal notations to formal notations. It is considered a critical transformation due to the value of formal verification on the derived formal models.

- Textual-based to formal notations:- most of the existing approaches adopt NLP techniques to parse the defined meta-model of the used textual format to generate the corresponding formal notation (e.g., [45]–[48]). Much progress has been witnessed within this type of transformation. Many different textual-based notations (along with specific parsing techniques) have been proposed to serve a specific domain or a specific type of requirements.
- Graphical representations to formal notations: similarly to the textual-based type of transformation, existing techniques in this category provide a homomorphic mapping between the meta-models of the graphical representations and the target languages (e.g., [32], [49], [50]). In [41], requirement capturing model (RCM) is proposed to enable automatic transformation into various temporal logic based notations, where it is mapped to metric temporal logic and computational tree logic.

Semi-formal notations to Semi-formal notations: In this class, conceptual mapping between the intended notations is formulated to enable transformation [51], [52]. In [53], system requirements capturing model (SRCM) is constructed by aggregating RCMs presented in [41](i.e., each representing one single requirement), to enable compact and integrated system views and quality issues detection in the semi-formal level.

Formal notations to Semi-formal notations: this type of transformation produces a semi-formal notation given the formal one. The most targeted semi-formal notations are graphical based [54]. In this transformation, homomorphic mapping and construction algorithms support the generation of semi-formal diagrams from formal models to visualise complex models [54].

IV. APPROACH

In our previous work [41], we analysed the existing requirements representation models and proposed RCM as a comprehensive semi-formal model supporting behavioural requirements. RCM defines the key requirements elements required for the automatic generation of the corresponding formal notations. We also formulated the mapping rules between the

RCM's key elements and the metric temporal logic "MTL" formal notation. In addition, we developed a fully automated approach for generating MTL formulas for requirements expressed in RCM. In [40], we extended the work by proposing an automatic NLP-based approach for extracting RCMs from NL-requirements. In this paper we provide requirements transformation in the reverse direction (i.e., MTL to RCM, then RCM to NL-requirements) to maintain consistency among the different levels of formality. Our approach takes a text file containing MTL formulas and a definition file (for interpreting formal symbols) as input, and provides XML and text files for the corresponding generated RCMs and NL-requirements respectively. Figure 2 shows the two transformation layers.

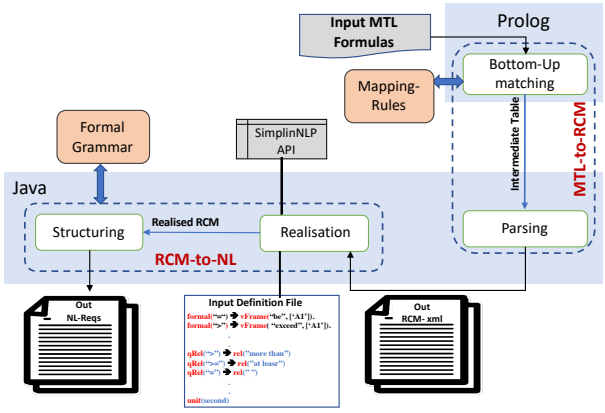

Fig. 2. Multi-Layer Transformations Flow

A. MTL to RCM Transformation

We utilise the same mapping rules proposed in [41]. We represent such rules as regular expressions to facilitate matching the underlying formal grammar of the rules. Table I shows the crafted regular expressions mapped to the RCM-MTL mapping formulas. However, regex rules 1, 2, 3, and 7 are crafted according to the formalisation approach proposed in [41]. We apply a bottom-up abstraction approach on the input formula to construct an intermediate table (IT) containing the required details for RCM. The approach consists of five steps shown in Algorithm IV-A, each designed to match a specific set of rules against the input formulas to extract specific information.

TABLE I
MAPPING REGEX RULES

RCM-Role	MTL-Id	Re-Id	Regex
Predicate Structure1	-	1	$(\backslash(\backslash w+(=\mid>\mid<\mid>=\mid<=)\backslash w+\backslash))$
Predicate Structure2	-	2	$(\backslash(\backslash w+\backslash((\backslash w+,)*\backslash w+\backslash)\backslash))$
Predicate Structure3	-	3	$(\backslash(\backslash w+(=\mid>\mid<\mid>=\mid<=)\backslash w+\backslash((\backslash w+,)*\backslash w+\backslash)\backslash))$
Pre-Time Element	10:14	4	$F\{t(=\mid>\mid<\mid>=\mid<=)\backslash d\}\backslash([A-Z]\backslash)$
Valid-Time Element	15:19	5	$G\{t(=\mid>\mid<\mid>=\mid<=)\backslash d\}\backslash([A-Z]\backslash)$
In-Time Element	20:24	6	$G\backslash(F\{t(=\mid>\mid<\mid>=\mid<=)\backslash d\}\backslash([A-Z]\backslash)\backslash)$
Coordinated Elements	-	7	$(\backslash(\backslash([0-9]\|\|AND\|OR\|(?R))*\backslash))$
Scope StartUP	5	8	$G\backslash([A-Z]==>F\backslash([A-Z]\backslash)\backslash)$
Scope EndUP	6-7	9	$F[A-Z]==>\backslash(F\backslash([A-Z]\backslash]\backslash\|[A-Z])U[A-Z]\backslash)\backslash)$
Scope both	8-9	10	$G\backslash(\backslash([A-Z]\backslash[A-Z]\backslash F[A-Z]\backslash)==>\backslash(F\backslash([A-Z]\backslash]\backslash\|[A-Z])U[A-Z]\backslash)\backslash)$
Single PreCond	2-3	11	$G\backslash([A-Z]==>[A-Z]\backslash)$
Composite PreCond	4	12	$G\backslash(\backslash([A-Z]\backslash[A-Z]\backslash)==>[A-Z]\backslash)$
Action	1	13	$G\backslash([A-Z]\backslash)$

AlgorithmIV-A: MTL-to-RCM Transformation

States:
R: MTL-to-RCM indexed Regex Rules
Formula: input string representing one formula
Tbl: Table contians extracted information
procedure
Step 1: Identify predicate elements
Formula ← abstractMatchedRegx(R{1:3},Formula)
updateTable(Tbl)
Step 2: Identify attached time elements and their types
Formula ← abstractMatchedRegx(R{4:6},Formula)
updateTable(Tbl)
Step 3: Identify predicate elements with same type
Formula ← abstractMatchedRegx(R{7},Formula)
updateTable(Tbl)
Step 4: Identify Scope elements attached to main elements
Formula ← abstractMatchedRegx(R{8:10},Formula)
updateTable(Tbl)
Step 5: Identify main elements types
Formula ← abstractMatchedRegx(R{11:13},Formula)
updateTable(Tbl)
end procedure

In Step 1, each predicate element is identified and represented with one alphabet. Each matched group is replaced with a Unique alphabet, where the letter and the corresponding matched group are stored in the IT. Step 2 matches regular expression rules R{4:6} against the MTL formula in an iterative manner to identify time elements. Each identified time element and its type are attached to the corresponding alphabet representing the predicate element in the IT (to keep their composition relation). In addition, the matched elements are replaced with their related predicate alphabets. Step 3 groups the coordinated elements (same type). Step 4 identifies the scope of the main elements. In Step 5, the main elements are identified. The IT is then parsed into RCM. Figure 3 shows a step by step annotation of the transformation algorithm on an example MTL formula along with the constructed IT. The generated RCM is shown in the first column in Figure 4.

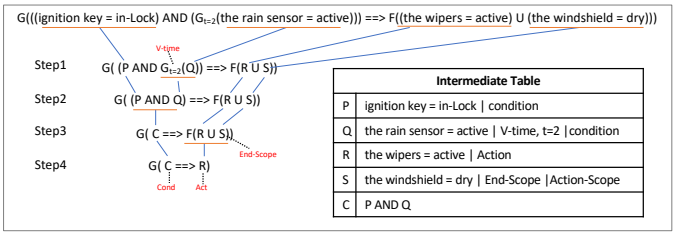

Fig. 3. Tracing example for MTL transformation

B. RCM to NL-Requirements Transformation

This layer consists of two tasks: realisation and sentence structuring. The generated sentences are governed by the following grammar:

```
<Sentence> ::= <SubordinatingClause>*.<MainClauses>
                .<SubordinatingClause>*
<SubordinatingClause> ::= Subordinator.<MainClauses>
<MainClauses> ::= <Clause>.<CoordinatingClause>*
<CoordinatingClause> ::= CoordinatingRel.<Clause>
CoordinatingRel ::= "and"|"or"
<Clause> ::= [<Subj>].[<RelativeClause>].<VerbPhrase>.<Time>*
<RelativeClause> ::= RelHead.[Property].<VerbPh>
<Subj> ::= NounPh
<VerbPh> ::= [Modality].<MainVerb>.<Complement>+
<MainVerb> ::= Verb |(be).Verb.(ed)
<Complement> ::= Preposition.(NounPh|Adj)
Modality ::= "shall"
Subordinator ::= TriggerHead|ConstraintHead|ScopeHead
ConstraintHead ::= "if"
TriggerHead ::= "when"
ScopeHead ::= "after"|"before"|"until"|"while"
RelHead ::= "whose"|"that"
<Time> ::= TimeHead. [QuantifyingRel]. Value. Unit
TimeHead ::= ValidTimeHead|PreElapsedTimeHead|InTimeHead
ValidTimeHead ::= "for"
PreElapsedTimeHead ::= "within"
InTimeHead ::= "every"
QuantifyingRel ::= "at least"|"less than"|"at most"|"more than"
```

Where, "*" means zero or more items, "+" indicates the presence of one or more, "." specifies composition of different items, "<>" refers to non-terminal, and "[ ]" refers to an optional item. NounPh, Adj, Value, Verb, QuantifyingRel and Unit are terminals. The first three exist in the extracted elements, while the rest are selected from the input definition file. A requirement sentence in the proposed grammar consists of at least one clause. A clause is built up from at least a verb phrase expressing the core meaning of the clause. Optionally, a subordinator can be attached (setting the grammatical role to a subordinating clause).

Realisation task is responsible for: (1) replacing formal symbols in elements with English words and (2) assigning correct grammatical syntax to these elements. To replace the formal symbols, we feed our approach with an extensible definition file mapping the formal-symbols to English frames. To adjust the grammar, we utilise SimplingNLG. All the elements are assigned a present tense except for the action – assigned a future tense. The second column in Figure 4 shows the changes in RCM after applying the realisation task.

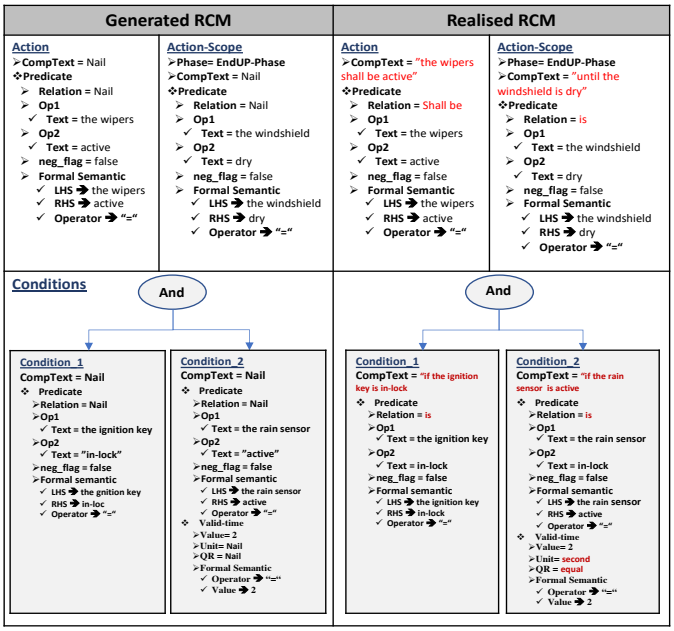

Fig. 4. Generated and realised RCM of the MTL formula in Fig.3

Sentence Structuring task orders the existing elements in a given RCM to construct a sentence. First, we assign priority indices to all RCM elements as shown in the third row in Table II. The numbers indicate the elements occurrence order in the sentence. The last four rows show the order of the sub-components within each component (e.g., the action component exists in the sixth place with its sub-components ordered as: pre-time, core-segment, valid-time, and in-time). Empty cells mean a specific sub-component is not eligible to the corresponding component. These indices are used to structure the sentence of a given RCM based on the existing elements.

TABLE II
COMPONENTS AND SUB-COMPONENTS PRIORITY INDICES

		PreCond-Scope		Action-Scope		Condition	Trigger	Action
		StartUp	EndUP	StartUP	EndUP			
	Components	1	4	5	7	3	2	6
SubComp	Core-segment	1.1	4.1	5.1	7.1	3.2	2.1	6.2
SubComp	Valid-time	1.2	4.2	5.2	7.2	3.3	2.2	6.3
SubComp	Pre-time					3.1		6.1
SubComp	In-time						2.3	6.4

Figure 5 shows the sentence structure of the realised RCM in Figure 4 in compliance with the elements priority listed in Table II.

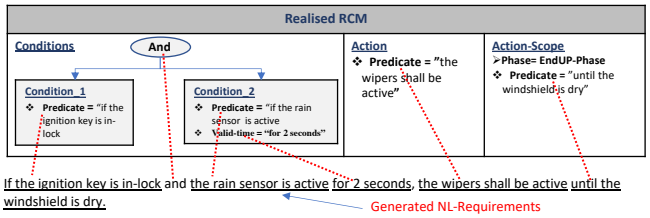

Fig. 5. Structured sentence of the realised RCM in compliance with Table II

V. CONCLUSION

In this paper, we provided an insight about the different levels of formality for representing requirements while highlighting the (dis)advantages of each level. We also presented the current state of research for the transformation among these levels. Finally, we proposed a multi-layer transformation approach to bridge the gap of maintaining traceability between formality level and enabling (non-)technical people to understand system requirements.

REFERENCES

[1] A. Pretschner, M. Broy, I. H. Krüger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *Proceedings of Future of Software Engineering FOSE*. IEEE, 2007, pp. 55–71.

[2] M. Weber and J. Weisbrod, "Requirements engineering in automotive development-experiences and challenges," in *Requirements Engineering,Proceedings.* IEEE, 2002, pp. 331–340.

[3] R. Thayer and W. Royce, "Software systems engineering, ieee system and software requirements engineering," *IEEE Software Computer Society Press Tutorial. Los Alamos, California*, 1990.

[4] B. W. Boehm, "Verifying and validating software requirements and design specifications," *IEEE software*, vol. 1, no. 1, p. 75, 1984.

[5] K. Pohl, "The three dimensions of requirements engineering," in *International Conference on Advanced Information Systems Engineering*. Springer, 1993, pp. 275–292.

[6] C. Rolland and C. Proix, "A natural language approach for requirements engineering," in *Seminal Contributions to Information Systems Engineering*. Springer, 2013, pp. 35–55.

[7] L. Antonelli, G. Rossi, J. C. S. do Prado Leite, and A. Oliveros, "Deriving requirements specifications from the application domain language captured by language extended lexicon." in *WER*, 2012.

[8] I. Buzhinsky, "Formalization of natural language requirements into temporal logics: a survey," pp. 400–406, Jul 2019.

[9] A. Brunello, A. Montanari, and M. Reynolds, "Synthesis of ltl formulas from natural language texts: State of the art and research directions," in *26th International Symposium on Temporal Representation and Reasoning*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[10] V. Sládeková, "Methods used for requirements engineering," Master's thesis, Mar, 2007.

[11] P. Sternudd, "Unambiguous requirements in functional safety and iso 26262: dream or reality?" 2011.

[12] J. Harrison, *Handbook of practical logic and automated reasoning*. Cambridge University Press, 2009.

[13] R. W. Sebesta, *Concepts of programming languages*. 4th ed. Addison-Wesley, 1999. ISBN: 0201385961, 1993.

[14] I. ISO, "26262: Road vehicles-functional safety," *International Standard ISO/FDIS*, vol. 26262, 2011.

[15] W. C. In and M. H. LinLee, "Informal, semi-formal, and formal approaches to the specification of software requirements," Ph.D. dissertation, University of British Columbia, 1994.

[16] V. Ambriola and V. Gervasi, "Processing natural language requirements," in *Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference*. IEEE, 1997, pp. 36–45.

[17] V. Johannessen, "Cesar-text vs. boilerplates," 2012.

[18] I. C. S. S. E. S. Committee and I.-S. S. Board, "Ieee recommended practice for software requirements specifications." Institute of Electrical and Electronics Engineers, 1998.

[19] P. Mary and P. Tom, "Lean software development: an agile toolkit," 2003.

[20] J. Kocerka, M. Krześlak, and A. Gałuszka, "Analysing quality of textual requirements using natural language processing: A literature review," in *2018 23rd International Conference on Methods & Models in Automation & Robotics (MMAR)*. IEEE, 2018, pp. 876–880.

[21] M. Osama, A. Zaki-Ismail, M. Abdelrazek, J. Grundy, and A. Ibrahim, "Score-based automatic detection and resolution of syntactic ambiguity in natural language requirements," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 651–661.

[22] T. Kuhn, "A survey and classification of controlled natural languages," *Computational Linguistics*, vol. 40, no. 1, pp. 121–170, 2014.

[23] 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.

[24] OMG, "Object management group. omg unified modeling language infrastructure. omg specification." https://www.omg.org/spec/UML/2.5.1, Dec, 2017.

[25] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[26] D. Amyot, "Introduction to the user requirements notation: learning by example," *Computer Networks*, vol. 42, no. 3, pp. 285–301, 2003.

[27] P. Shoval, R. Danoch, and M. Balabam, "Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation," *Requirements Engineering*, vol. 9, no. 4, pp. 217–228, 2004.

[28] W. Scott and S. Cook, "A context-free requirements grammar to facilitate automatic assessment," Ph.D. dissertation, UniSA, 2004.

[29] H. Elizabeth, J. Ken, and D. Jeremy, "Requirements engineering," 2011.

[30] T. DeMarco, *Structured analysis and system specification*. Yourdon Press, 1979.

[31] M. Osborne and C. MacNish, "Processing natural language software requirement specifications," in *Proceedings of the Second International Conference on Requirements Engineering*. IEEE, 1996, pp. 229–236.

[32] S. Dupuy, Y. Ledru, and M. Chabre-Peccoud, "An overview of roz: A tool for integrating uml and z specifications," in *Advanced Information Systems Engineering*. Springer, 2000, pp. 417–430.

[33] T. Teige, T. Bienmüller, and H. J. Holberg, *Universal pattern: Formalization, testing, coverage, verification, and test case generation for safety-critical requirements*. Universität, 2016.

[34] S. Konur, "A survey on temporal logics for specifying and verifying real-time systems," *Frontiers of Computer Science*, vol. 7, no. 3, pp. 370–403, 2013.

[35] J. M. Spivey, "An introduction to z and formal specifications," *Software Engineering Journal*, vol. 4, no. 1, pp. 40–50, 1989.

[36] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Munoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saıdi, N. Shankar *et al.*, "An overview of sal," in *Proceedings of the 5th NASA Langley Formal Methods Workshop*. Williamsburg, VA, 2000.

[37] A. M. Moreno, "Object-oriented analysis from textual specifications," in *Ninth International Conference on Software Engineering and Knowledge Engineering, Madrid, Spain (June 1997)*, 1997.

[38] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with circe," *Automated Software Engineering*, vol. 13, no. 1, pp. 107–167, 2006.

[39] T. H. Nguyen, J. Grundy, and M. Almorsy, "Rule-based extraction of goal-use case models from text," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 591–601.

[40] A. Zaki-Ismail., M. Osama., M. Abdelrazek., J. Grundy., and A. Ibrahim., "Rcm-extractor: Automated extraction of a semi formal representation model from natural language requirements," in *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, INSTICC. SciTePress, 2021, pp. 270–277.

[41] ——, "Rcm: Requirement capturing model for automated requirements formalisation," in *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, INSTICC. SciTePress, 2021, pp. 110–121.

[42] M. Kamalrudin, J. Hosking, and J. Grundy, "Maramaaic: tool support for consistency management and validation of requirements," *Automated software engineering*, vol. 24, no. 1, pp. 1–45, 2017.

[43] ——, "Improving requirements quality using essential use case interaction patterns," in *Software engineering (ICSE), 2011 33rd international conference on*. IEEE, 2011, pp. 531–540.

[44] R. Nelken and N. Francez, "Automatic translation of natural language system specifications into temporal logic," in *International Conference on Computer Aided Verification*. Springer, 1996, pp. 360–371.

[45] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Property specification patterns for finite-state verification," in *Proceedings of the second workshop on Formal methods in software practice*, 1998, pp. 7–15.

[46] S. Flake, W. Müller, and J. Ruf, "Structured english for model checking specification." in *MBMV*, 2000, pp. 99–108.

[47] S. Konrad and B. H. Cheng, "Real-time specification patterns," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 372–381.

[48] R. Yan, C.-H. Cheng, and Y. Chai, "Formal consistency checking over specifications in natural languages," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 1677–1682.

[49] W. E. McUmber and B. H. Cheng, "Uml-based analysis of embedded systems using a mapping to vhdl," in *High-Assurance Systems Engineering, 1999. Proceedings. 4th IEEE International Symposium on*. IEEE, 1999, pp. 56–63.

[50] ——, "A general framework for formalizing uml with formal languages," in *Proceedings of the 23rd international conference on Software engineering*. IEEE Computer Society, 2001, pp. 433–442.

[51] H. Afreen, I. S. Bajwa, and B. Bordbar, "Sbvr2uml: A challenging transformation," in *2011 Frontiers of Information Technology*. IEEE, 2011, pp. 33–38.

[52] A. Rodríguez, I. G.-R. de Guzmán, E. Fernández-Medina, and M. Piattini, "Semi-formal transformation of secure business processes into analysis class and use case models: An mda approach," *Information and Software Technology*, vol. 52, no. 9, pp. 945–971, 2010.

[53] M. Osama., A. Zaki-Ismail., M. Abdelrazek., J. Grundy., and A. Ibrahim., "Srcm: A semi formal requirements representation model enabling system visualisation and quality checking," in *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,*, INSTICC. SciTePress, 2021, pp. 278–285.

[54] K. Moremedi and J. A. van der Poll, "Transforming formal specification constructs into diagrammatic notations," in *International Conference on Model and Data Engineering*. Springer, 2013, pp. 212–224.