VeriCCM: Improving the Syntax and Semantics of Requirements Models

Danielle Gaither, Kaushik Madala, Hyunsook Do, Barrett R. Bryant
University of North Texas, USA
daniellegaither@my.unt.edu,kaushikmadala@my.unt.edu,hyunsook.do@unt.edu,barrett.bryant@unt.edu

ABSTRACT

Many problems in software systems can ultimately be traced to problematic system requirements. To mitigate this problem, some systems builders have adopted a more systematic approach to requirements elicitation and analysis. However, many of these approaches treat requirements as isolated entities having no direct connection to the actual system implementation. We propose VeriCCM, an approach that addresses potential problems both in the semantics and the syntax of requirements models. We also evaluate our approach with an empirical study.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**; *Software verification and validation*; Domain specific languages; Software development process management; • **Computing methodologies** → *Machine learning*;

KEYWORDS

requirements analysis, machine learning, verification and validation

ACM Reference Format:
Danielle Gaither, Kaushik Madala, Hyunsook Do, Barrett R. Bryant. 2019. VeriCCM: Improving the Syntax and Semantics of, Requirements Models. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3297280.3299745

1 INTRODUCTION

Technological systems can often trace their defects to poorly-specified requirements. One of the most famous examples is the explosion of the Ariane 5 launcher in 1996, where a software bug triggered the spacecraft's self-destruct mechanism. The spacecraft had been tested for a variety of scenarios involving hardware component failure, but none involving design errors in the software [10].

One approach to address the problems of poorly-specified requirements is to build a model of the requirements and analyze the model for potential defects. Some integrate with long-established modeling tools, such as goal modeling [7]. Others use domain-specific ontologies to create their own domain-specific modeling languages [1, 8]. However, like all tools, these approaches have their limitations. For example, goal modeling is concerned with overall system goals, but is less useful for analyzing interactions between system components. Concerning domain-specific modeling languages (DSMLs), while organizations that have adopted DSMLs have seen mostly positive results [3], the formal foundations for such languages are not yet widespread [4]. For example, Aceituna et al. [1] proposed the Causal Component Model (CCM), a domain-specific model-based requirements analysis process with the intention of finding abnormal system behaviors. The results so far have been promising, but their work does not check the quality of the model itself, and much of the process relies on manual analysis.

To address some of the limitations in frameworks such as CCM, we propose VeriCCM, which attempts to improve model quality both by ensuring that model elements are relevant terms and that the models are well-formed. Although there is always some manual effort involved in writing rules for a model, an automatic mechanism to check the structure or quality of the rules can be useful. By adding model element analysis and formalized elements, we can be more confident that our model elements are relevant and that our rules are well-formed. To evaluate the effectiveness of VeriCCM, we performed an empirical study on a set of requirements models. Our approach revealed several instances of erroneously labeled components and incomplete requirements models, including unreachable and unrecoverable states. The results show how even basic text processing and lightweight formalization can reveal numerous defects in a requirements model.

The rest of this paper is organized as follows. Section 2 provides brief background information on topics used in our work and related work. We outline our approach in Section 3. In Section 4, we detail our empirical study and discuss our findings in Section 5. We conclude and suggest potential future work in Section 6.

2 BACKGROUND AND RELATED WORK

2.1 Background

Our approach combines the Causal Component Model (CCM) [1], word embeddings [14] and ANTLR [13]. We verify input model elements and rules corresponding to CCM using word embeddings and ANTLR respectively. The definitions of the model elements in our approach correspond to definitions of model elements in CCM. Due to space constraints, we forgo explaining these topics and refer the readers to their citations.

2.2 Related Work

One of the most difficult problems in requirements engineering is extracting structured information for modeling purposes from unstructured natural language requirements [2, 9, 11]. In an example from construction automation, Lee et al. [9] used a Model View Definition (MVD) validation approach to check the content and format of a given source file against the specifications laid out in

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04.
https://doi.org/10.1145/3297280.3299745

the MVD. The specifications in the MVD are not quite as formal as a grammar, but the MVD provides a useful starting point for their domain. While these approaches extract information, they do not provide ways to validate extracted information. Information extracted using machine learning or natural language processing techniques can be incorrect. Hence, another step for validation is necessary.

Approaches [5, 6, 15] to verification and validation of model specification assume more in-depth knowledge of the details of the hardware being used than we do in our work. For example, Busari and Letier [5] apply a probabilistic approach to goal modeling to facilitate architecture decisions and requirements analysis. Their work resulted in some useful models, but their method insists on specific key terminology, which limits the flexibility of what can be modeled with their approach. Our proposed approach aims to address the above mentioned limitations.

3 APPROACH

Figure 1 shows an overview of our approach.

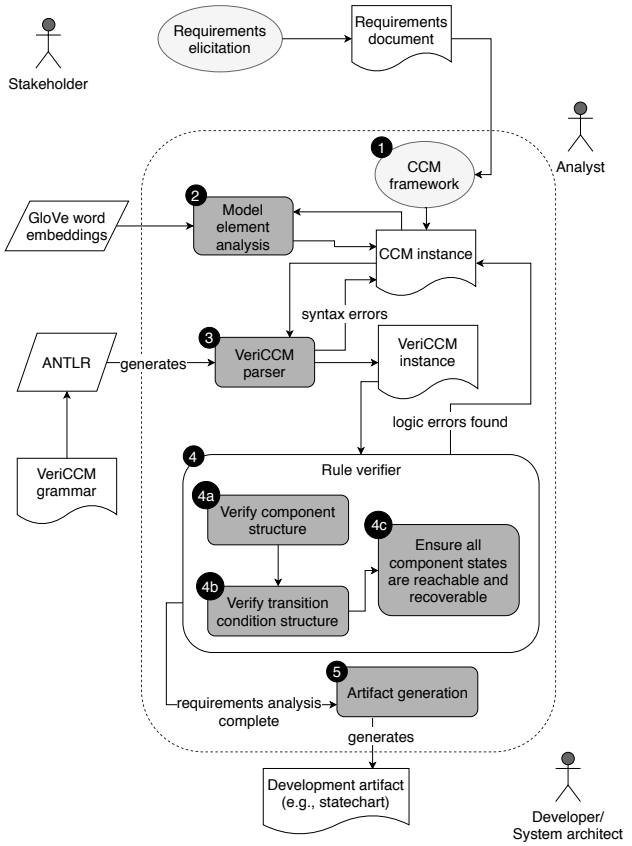

Figure 1: An overview of VeriCCM.

As Figure 1 demonstrates, the requirements analysis process begins with requirements elicitation. For purposes of this paper, we assume the elicitation process has already taken place.

3.1 Apply CCM Framework

In Step 1, the CCM framework [1] is applied to the elicited requirements to form a model in the CCM format. To illustrate our approach, assume that we have obtained the requirements listed below. We will refer to this set of requirements throughout this section as **SR1** [1].

(1) The system shall contain a switch, a motor, and a temperature sensor.
(2) The user will press a switch to turn the motor on and off.
(3) For the motor to turn on, the temperature must be below 80°.
(4) If the temperature is greater than 100°, the system will be considered overheated.
(5) The motor should not be on while the system is overheated.

After applying the CCM framework to **SR1**, we obtain the model shown in Figure 2.

```
#Start_switch|off|on!
Temperature_sensor|safe|overheated!
Motor|off|on!
@USER(on) : Start_switch(off) -> Start_switch(on)
USER(off) : Start_switch(on) -> Start_switch(off)
Start_switch(off) : Motor(on) -> Motor(off)
Start_switch(on) AND Temperature_sensor(safe) : Motor(off) -> Motor(on)
Temperature_sensor(overheated) : Motor(on) -> Motor(off)
Temperature_sensor GE 100 :
  Temperature_sensor(safe) -> Temperature_sensor(overheated)
Temperature_sensor LT 80 :
  Temperature_sensor(overheated) -> Temperature_sensor(safe)
```

Figure 2: Sample VeriCCM input file [1]

3.2 Model Element Analysis

In Step 2, we use word embeddings [14] to find if each model element being used is an actual model element. We create an inference engine based on the word embeddings of known model elements collected from existing requirements documents. We calculate the Euclidean distance between the embedding of each item in the list of model elements and the embedding of each item in a validation list that includes both model elements and non-model elements to determine prediction thresholds. These thresholds are used in the inference engine to determine whether a potential model element is an actual model element. More information about the details and the results of this process will be discussed in Section 4. In this paper, we limit our analysis to components.

3.3 Syntax Checking With the VeriCCM Parser

We use ANTLR [13] to generate the VeriCCM parser automatically, based on a created VeriCCM grammar. The parser then performs Step 3 of our process, checking the syntax for CCM input files. The sample input file in Figure 2 is checked against the VeriCCM grammar. If the input conforms to the grammar specification, the parser generates a parse tree for the provided input.

3.4 Rule Verification

In Step 4, we identify any inconsistent or incomplete requirements by checking that components, component states, and transitions are properly formed using a rule verifier. Our approach takes a VeriCCM instance as input and returns a display of any errors found during the course of our analysis. The first stage is to read in the input file containing the model. If any portion of the model is found not to meet the grammar specification, the problem is noted and an appropriate error message is displayed.

After all the components and rules have been scanned, the set of all current states and next states is searched to make sure that each component state is represented in both sets. This is to ensure that all states are both reachable and recoverable. In our VeriCCM tool, if there has been no indication of any error states after completing all the steps of our analysis, the rule verifier outputs a message indicating that the model has passed the verifier without any errors.

In the context of our chosen modeling framework, each model should satisfy the following criteria: (1) every component must have at least 2 states; (2) every component state must have at least one incoming transition and one outgoing transition; (3) the same transition condition cannot cause transitions to different states for the same component; and (4) a transition condition must be a well-formed component state, a guard condition, or a user interaction. If the model violates any of the specified conditions, the rule verifier prints out an appropriate error message along with the line in the model that triggered the error. In the event that inconsistent or incomplete requirements are found, clarification is sought from system stakeholders. The process is repeated until all parties are satisfied with the requirements document.

4 EMPIRICAL STUDY

To evaluate VeriCCM, we performed an empirical study on various test models developed by students on a robotics system. The test models used in our analysis came from a human study performed in the fall of 2017 with 28 students in an upper-level undergraduate software engineering course. We discuss the details of the human study and our empirical study in subsequent subsections.

4.1 Research Questions

We investigated the following research questions:

RQ1 Can VeriCCM identify erroneously labeled components in an automated fashion more effectively than manual analysis?

RQ2 Can formal syntax analysis be used to identify defects in requirements models more effectively than manual analysis?

4.2 Variables and Measures

Independent Variable: Our independent variable is our requirements analysis technique, with manual analysis of CCM models as the control and VeriCCM as the heuristic approach.

Dependent Variables: Our dependent variables for RQ1 are the F_1-measure of classified components and the time taken for analysis. In our study, we calculate two types of F_1-measures: strict and lenient. The strict F_1-measure is calculated by considering all entities that are classified as 'may be components' to be 'non-components', whereas the lenient F_1-measure is calculated by considering all component classified as 'may be components' to be 'components'. The dependent variables for RQ2 are the time taken for verifying rules and the number of defects found.

4.3 CCM Model Preparation

We used requirements models created by students as part of a human study conducted in fall 2017. The students were divided randomly into two groups (14 per group) and asked to create CCM instances for a robot vacuum cleaner requirements document [16] using two different approaches: (1) directly creating CCM instances from natural language requirements (Group 1) and (2) converting natural language requirements into modified Easy Approach to Requirements Syntax (EARS) patterns [12], which are then transformed into rules (Group 2). The result of the study showed that the majority of participants claimed it is easy and comfortable to write transition rules directly than using structured sentence patterns as an intermediate step for creating rules. However, the study's results also showed that both the approaches can result in incorrect models (between the two techniques, generating rules directly results in more incorrect models than using sentence patterns). We believe that if we have a mechanism to check the model for its validity, we can aid users in creating a better model, even when we create rules directly from requirements. Hence, we chose the models from both the groups to evaluate if our approach can aid in creating better models, especially for novice users. We used 26 out of 28 models developed by students, as two of the models were unusable due to incorrectness.

4.4 Artifact Preparation

To perform our components analysis, we created an inference engine by performing the following steps: (1) We chose seven requirements documents for robotics systems and generated a master list of 75 unique components. (2) We then used GloVe embeddings [14] to create embeddings of each component in the master list. (3) To tune parameters that aid in classifying potential model elements, we created validation data with 5 components (e.g., braking system) and 15 non-components (e.g., zebra, mortar), for which we looked up the associated GloVe embeddings [14]. (4) We measured pairwise Euclidean distances between each entity in the master list and each entity in the validation data. (5) Using these distances, we identified two thresholds 'th1' and 'th2'. If the value is less than 'th1', we classify it as a 'component'. If it is between 'th1' and 'th2', it is considered as a 'may be component'. If it is greater than 'th2' we classify it as a 'non-component'. (6) We adjusted the thresholds to maximize F_1-measure for validation data. (7) We applied our inference engine on the components of unseen requirements models.

4.5 Experimental Procedure

We followed the steps as detailed in Section 3 to perform our study. Two doctoral students performed the manual analysis and collected data in our study. For analysis of components, we generated embeddings of each term identified as a component in each of the students' models and classified if the term is a 'component', 'may be component' or a 'non-component' using our inference engine. We evaluated both strict and lenient F_1-measures to examine the effectiveness of our technique. We also compared the time taken for manual analysis of components to automatic analysis. For our next step, we performed syntax checking using the grammar specified in ANTLR. To perform rule verification, the models are analyzed by the VeriCCM tool which gives us the list of errors in the requirements models. We also performed manual analysis of models and recorded the number of errors found. We measured time taken for manual analysis and automatic analysis and compared them.

4.6 Results

In this section, we will summarize our findings for our research questions. RQ1 investigates if VeriCCM can identify erroneously labeled components in an automated fashion. The results of RQ1

Table 1: RQ1 results.

Measurement	Strict scores	Lenient scores
Precision	0.95	0.83
Recall	0.75	0.96
F1	0.84	0.90

Table 2: Aggregated RQ2 results.

Doc. #	Manual analysis		VeriCCM	
	Time taken (s)	# of defects	Time taken (s)	# of defects
Avg.	71.4	13.8	0.04	21.5
Min.	32	4	0.04	9
Max.	180	30	0.04	39

can be shown in Table 1. Our automated analysis was able to predict components with a strict F_1 measure of 0.84 and a lenient F_1-measure of 0.90. This result indicates that while the component analysis tool considers all 'may be components' as 'components', it can miss fewer components when compared to considering all 'may be components' as 'non-components'. We found that the term 'Power', which is not a component is always classified as a component. We believe this is because of its close association to sensors and controllers which are often components.

RQ2 investigates if VeriCCM can identify defects in models effectively than manual analysis. The aggregated results of RQ2 can be found in Table 2. The number of defects found by VeriCCM is 1.5 times higher than the defects found via manual analysis, whereas the time taken by VeriCCM is 99.94% less than time consumed for manual analysis. The number of defects found via VeriCCM ranges from 9 to 39, which is 24 to 44% more than the range of defects found via manual analysis (4 to 30). The manual analysis time ranged from 32 seconds to 180 seconds, while VeriCCM took only 0.04 seconds for each of the models.

There are rare cases (5 out of 26) in which manual analysis found more errors than VeriCCM. We believe this is mostly because of significantly underspecified models, leaving fewer opportunities for an automated analysis tool to detect defects.

5 DISCUSSION

Our results show that VeriCCM aids in reducing the number of iterations to build a correct model when compared to manual analysis. We found that even if some entities that are not components are identified as components, the rule verifier helps identify the issue. As a result, this technique can reduce human effort. Our technique is also scalable and takes less time even for a large system. This is because the technique does not expand states as a combination of components, but rather uses simple input and output nodes to analyze recoverability and reachability.

Limitations: Our model elements analysis is restricted to only components but not state and transition conditions. We plan to add such analysis in future work. Our current technique cannot deal with undesired states. We plan to address this limitation by expanding the capabilities of the rule verifier. Our current tool can handle only environmental transition conditions written in a restricted format. We can address this limitation by adding new rules to the VeriCCM grammar.

Threats to validity: The results might not be generalizable in their current form, as industrial applications often tend to use vocabulary that does not exist in GloVe word embeddings. We can address this threat by considering embeddings trained on different domains and by adding automated embedding generation for new words.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed VeriCCM, an approach built on CCM, which verifies correctness of transition rules and components. We used word embeddings to analyze if a listed component was likely to be an actual system component. To automate the rule verification, we built a syntax checker and rule verifier. To evaluate our approach, we conducted an empirical study. The results show that our approach can uncover inconsistent and incomplete requirements.

One possibility for future work is to keep pursuing further formalization, eventually giving a tool like VeriCCM the same solid logical ground as any programming language. Another possibility is applying the same fundamental approach to other types of requirements analysis methods, such as goal modeling.

ACKNOWLEDGMENTS

This work was supported, in part, by NSF CAREER Award CCF-1564238 to the University of North Texas.

REFERENCES

[1] D. Aceituna and H. Do. 2015. Exposing the Susceptibility of Off-Nominal Behaviors in Reactive System Requirements. In *Proceedings of the IEEE International Requirements Engineering Conference*, Vol. 23. IEEE, 136–145.
[2] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. 2016. Extracting domain models from natural-language requirements. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems - MODELS '16*. ACM Press, New York, New York, USA, 250–260.
[3] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice: Second Edition* (2nd ed.). Morgan & Claypool Publishers.
[4] B. R. Bryant, J. Gray, M. Mernik, P. J. Clarke, R. B. France, and G. Karsai. 2011. Challenges and directions in formalizing the semantics of modeling languages. *Computer Science and Information Systems* 8, 2 (2011), 225–253.
[5] S. A. Busari and E. Letier. 2017. RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 552–562.
[6] A. Ferraiuolo, R. Xu, D. Zhang, A. C. Myers, and G. E. Suh. 2017. Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis. *ACM SIGOPS Operating Systems Review* 51, 2 (Apr 2017), 555–568.
[7] J. Hassine and D. Amyot. 2016. A questionnaire-based survey methodology for systematically validating goal-oriented models. *Requirements Engineering* 21, 2 (2016), 285–308.
[8] J. Holt, S. Perry, R. Payne, J. Bryans, S. Hallerstede, and F. O. Hansen. 2015. A Model-Based Approach for Requirements Engineering for Systems of Systems. *IEEE Systems Journal* 9, 1 (Mar 2015), 252–262.
[9] Y. Lee, C. M. Eastman, W. Solihin, and R. See. 2016. Modularized rule-based validation of a BIM model pertaining to model views. *Automation in Construction* 63 (Mar 2016), 1–11. https://doi.org/10.1016/J.AUTCON.2015.11.006
[10] N. G. Leveson. 2004. The role of software in spacecraft accidents. *Journal of Spacecraft and Rockets* 41, 4 (2004), 564–575. https://doi.org/10.2514/1.11950
[11] Kaushik Madala, Danielle Gaither, Rodney Nielsen, and Hyunsook Do. 2017. Automated identification of component state transition model elements from requirements. In *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*. https://doi.org/10.1109/REW.2017.73
[12] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. 2009. Easy Approach to Requirements Syntax (EARS). In *17th IEEE International Requirements Engineering Conference*. IEEE, 317–322. https://doi.org/10.1109/RE.2009.9
[13] T. Parr. 2014. ANTLR. http://www.antlr.org/
[14] J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162
[15] O. Semeráth, A. Barta, A. Horváth, Z. Szatmári, and D. Varró. 2017. Formal validation of domain-specific languages with derived features and well-formedness constraints. *Software & Systems Modeling* 16, 2 (May 2017), 357–392.
[16] E. So, J. Ajtum, Y. Moy, and Y. L. Quach. 2005. Requirements Specification. http://www.ecs.umass.edu/ece/sdp/sdp05/preston/sdp_data/RequirementSpecification.doc