International Journal of Engineering & Technology, 7 (4.19) (2018) 164-167

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Formal Specification Languages: Features, Challenges and Future Directions

Hussin Ahmed[1], Azham Hussain[2*], Fauziah Baharom[3]

Human-Centered Computing Research Lab, School of Computing, Universiti Utara Malaysia, 06000, Malaysia.
**Corresponding Author Email: [2]azham.h@uum.edu.my*

Abstract

Natural language is used as a popular way for Software Requirements Specification (SRS) to ensure successful communication between stakeholders. However, natural language suffers from ambiguity which motivated software community to devise Formal Specification Languages (FSLs) to state requirements precisely. Since the advent of FSLs, a heated debate among researchers and practioners was raised to judge on the practical use of FSLs in industry. In this research, a contemporary review is conducted to shed light upon the features, challenges and future directions of FSLs.

Keywords: *Requirements specification; formal specification, formal methods*.

1. Introduction

It has been acknowledged frequently that the most arduous task in software development is deciding precisely what to build (Ronen, 2017). For this reason, requirements definition is of high importance to produce a high-quality software product. Hussain and Mkpojiogu (2016) stated that software projects that undermine requirements engineering suffer or are likely to suffer from failures, challenges and other attending risks. Requirements are considered as an input to design, implementation and validation phase of software product development(Hussain, Mkpojiogu, & Kamal, 2016).This gives rise to the importance of SRS in software engineering which leads to understanding the proposed functionality of the software for guiding subsequent stages in Software Development Life Cycle (SDLC). SRS is serving as a basis for all communications regarding the software being developed. Moreover, it constructs an agreement between stakeholders including customers, managers and software developers to agree upon the contents of requirements. This agreement will be a roadmap for developing software product. Furthermore,Pressman (2010) referred to SRS as a mean to assess the quality once software is built.

Hence, the basic objective of SDLC is transforming SRS into computer instructions written in programming languages. Programming languages are composed of two parts, syntax and semantics (Bourque & Fairley, 2014). The syntax of a language defines the grammars and rules which must be followed to write a correct program and the semantics provides the meaning of the program (Lee, 1992). The critical problem with SRS is that content suffers from all the inherent limitations that results from using any natural language document (Huertas et al., 2011).These problems include inconsistency, incompleteness and ambiguity. The most critical problem of natural language is ambiguity which results in the possibility of different interpretations of the same requirement statement.

Ambiguity leads to poor definition of requirements which affects the functionality of software and eventually leads to unsatisfied customers. Ambiguous requirements have a passive effect in developing software products in terms of time, cost and quality. Furthermore, writing ambiguous requirements leads to a great deal of iteration to understand what customers really want to achieve regarding the functionality of the software. Sommerville (2010)argued that software developers might interpret ambiguous requirements from their perspectives in a way that facilitates its implementation; however, this is not what the customersreally want. In such scenario,software engineering community was obliged to develop FSLs to overcome the ambiguity of natural language.

2. Literature Review

2.1 Overview of Formal Specification Languages

Over the years, many significant formal languages have been devised to construct formal specifications like object Z (Smith, 2012), Z notation (Spivey, 1992) and VDM (Jones, 1990). Generally, formal methods involve rigorous steps to develop FSLs based on mathematical techniques. These techniques include logic, discrete mathematics and abstract algebra to represent the required information for software construction (Alagar &Periyasamy, 2011). FSLs are the products of formal methods to guide the requirements engineer to write requirements specification. A formal language is defined traditionally as an alphabet of symbols and a set of grammar rules for constructing well formed formulas from the alphabet (Alexander, 1995). The main objective of formal specification languages is to support the delivery of high-quality software based on well defined syntax and semantics.

2.2 Formal Specification Languages Review

Many publications addressed the applicability of FSLs in industry. Clarke and Wing (1996) reviewed the practices of formal methods and identified future directions for supporting the wide use of formal methods. The emphasis on future directions are based on developing reusable models and theories to prevent the repetitive work in developing formal methods, creating different kinds of abstractions, using efficient methods for decomposing global properties into local properties, ease of learning and technology transfer to the targeted audience. Moreover, the authors expected that using formal methods will increase dependent on the continuous support of developing new specification languages and new verification techniques.

Lamsweerde (2000) reviewed the main formal specification paradigms and discussed their evaluation criteria. Also, he provided a brief assessment of the strengths and weaknesses of formal specification technology. Lamsweerde (2000)concluded that there is a long way to go before formal specifications can be used by the average software engineer to gain visible reward.Woodcock et al., (2009) conducted a survey to assess the applicability of formal methods in industry. The focus was on the increasing use of formal methods at the earlier stages of specification and design. Using a structured questionnaire, data was collected between November 2007 and December 2008 on 62 industrial projects. The survey illustrated that the respondents in general were satisfied with the formal techniques used in their projects.Kassab et al. (2014) used the Web-based QuestionProtool to conduct a survey for assessing the use of formal methods. The findings revealed that formal specifications techniques are still not commonly utilized. Szmuc and Szpyrka (2015) have conducted a short overview to determineadvantages and disadvantages of formal methods. This overview led to a conclusion that formal methods are applicable in specific areaswhere high reliability is needed like Safety Critical Systems.Based on the aforementioned publications, there is a need for contemporary and complementary view to document the features, challenges and future directions of FSLs.

On the basis of this research, the following research question and research objective are formulated.

RQ: What are the features, challenges and future directions of FSLs?

RO: The objective is to identify the features, challenges and future directions of FSLs from the existing literature.

3. Methodology

In order to conduct this research, an analysis of literature review is performed to answer the research question. There are three main keywords used to collect relevant information regarding this topic such as “requirements specification”, “formal method” and “formal specification”. The review conducted based onmultiple resources such as software engineering textbooks, journals and conference publications.

4. Formal Specification Languages Features

FSLs have many features in contrast with informal ones. These features will be discussed in the following sub-sections.

4.1 Revealing Design Flaws

If formal methods used early in software development, it can reveal design flaws and detect potential errors that otherwise would only possible be discovered in the costly testing and debugging phases (Wing, 1990).Meyer (1993)has referred to some potential flawsthat can be eliminated from SRS document. These flaws include contradiction, inconsistency within the requirements document; noise, extraneous information; silence, the lack of information concerning a required component of the software; forward reference, the use of items not yet defined in requirements document; over specification, the burden of dealing withirrelevant information; ambiguity, a lack of precise information concerning a particular component and wishful thinking, the inclusion of requirements not required by the software.However, using formal methods earlier has a side effect to verify building the right product from perspective of users. Hence, this feature can be obtained when the stakeholders are familiar about formal methods which can be applicable in certain sophisticated domains.

4.2 Overcoming the Conceptual Gap

The most marvelous feature of FSLs is the success of overcoming the conceptual gap between requirements specification and programming languages. This feature helps to overcoming the heterogeneous concepts between requirements specification and programming languages. As a result, this characteristic will secure producing a clear understanding about the software product functionality. This is important to resolve ambiguity in requirements specification which creates a clear view for software developers. Moreover, this feature will sustainthe capability of producing code directly from requirements specification.

4.3 Automatic Analysis and Validation

FSLs enable an automatic analysis and promotevalidation. This advantage will smooth the usage of tools for reasoning about most of the representedknowledge in SRS which can help to reduce time and effort. Automatic analysis will contribute to avoid violation of syntax and semantics in SRS document. This is important to facilitate writing SRS based on the predefined grammar and help to avoid semantics errors like adding textual variable to numerical.Moreover, it will boost the validation process of the software product to ensure building the product right.

4.4 Reusability

FSLs can support reusability in software engineering projects. This feature results from the capability of formal representations to store the concepts and relations in a software repository which could be employed to reuse existingspecifications in similar software projects (Diamantopoulos et al., 2017). In other words, software analysts have the capability to use the same building blocks in FSLs to represent similar functionalities in ongoing projects. Employing reusability will save time via avoiding repetitive work and enhance the productivity of SDLC. Furthermore, this feature will accelerate software development to produce many software products.

The significance features of FSLs make them an ideal choice for requirements specification in critical systems such as nuclear missile system, auto pilot system and air traffic system. However, using FSLs come with disadvantages and huge cost as well. In the following section, an overview of these disadvantages.

5. Formal Specification Languages Challenges

5.1 Learning

It is observable that the vast majority of software projects still rely upon natural language for specifying requirements. Normally, people use natural language on a daily basisin comparison with FSLs. In reality, FSLs are usually complex and require a sophisticated mathematical knowledge which is difficult to be achieved by naive users and in most of the designers and practicing analysts (Fraser et al., 1991).This phenomena has been documented in software engineering textbooks (Wiegers &

Beatty, 2013;Sommerville, 2010). Also, Pressman (2010) stated that formal methods are so unfamiliar to most people. In practice, formal languages require a high learning curve (Olajubu, 2015;Szmuc & Szpyrka, 2015). Taking into consideration that newcomer in software projects who take the responsibility of writing requirements specification might not be familiar about the discipline of formalism. This results in enormous effort and time to learn FSLsfor achieving a certain level of professionalism.

5.2 Verification

The second challenge of using FSLs is the difficulty of verification. It is important that stakeholder understand the presented requirements thus they would be able to verify it (Robie, Baharom, & Mohd, 2014). Thus, the verification process requires the involvement of customers to agree upon the requirements to be developed. Reviewing requirements will be onerous task from perspective of customers who cannot grasp the formal specification terminologies. Unfortunately, this means that a FSL usually does not serve as a basis for discussion and communication. From an industrial point of view, Agerholm and Larsen (1998) stated that only parts of the systems would benefit from formal methods and the general skill level in industry is not adequate to keep up with the techniques for fully formal development.Additionally, Ryan (1993) clarified that some requirements cannot easily be formalised. For instance, demands that a user interface be "user-friendly" or a piece of code can be "easily maintained". As a consequence, software projects cannot rely upon formal methods completely to verify requirements. From this perspective, using natural language will be inevitable to smooth the verification process and accelerate the agreement upon SRS.

5.3 High Cost and Strenuous Effort

A third challenge of using FSLs is the high cost and strenuous effort for stating comprehensive and precise requirements. Bollin and Rauner-reithmayer (2014)emphasized that a lot of revisions are needed to develop a specification that is close to the concepts of the developers and customers have in mind. Interestingly, Pang et al. (2016) argued that even an experienced users may commit mistakes particularly in the case of developing complex formal specifications. The main reason behind this difficulty is the limited expressive capability of formal specification in comparison with natural language which can be easily used to express any sort of requirements.

5.4 Lack of Community Support

The fourth challenge is lack of support from government, academic institutions and industry. For instance, Mandrioli (2015) declared that among 157 project proposals by young researchers to be funded by the Italian government, none of 157 proposals had a minimum reference to formal system analysis and verification. If there is a certain direction from the government to support the applications of FSLs, many proposals will contribute to make advantage of FSLs. In addition, formal methods and the mathematics necessary to support them have not found their way into standard software engineering curriculum (Alexander, 1995). Tse and Pong (1991) noticed that users are not willing to try a new method with which they are not familiar. These observations revealed the magnificent role of academic communities to encourage authors for writing interesting curriculum about the discipline of FSLs. Enriching the knowledge domain of FSLs with sufficient and interesting curriculum will help students and practioners as well to be knowledgeable about using FSLs effectively. Thirdly,the need of industrial support is highly required to facilitate using FSLs. This requires developing tools and Integrated Development Environment (IDE) to make the life of using FSLs easier. Although model checkers, proof tools and the like exist for formal models, they provide little support for the software life cycle (Alexander, 1995). The lack of proper tools has often been claimed to be a main obstacle to the industrial take up of formal methods (Agerholm & Larsen, 1998). As a result, many software companies would be reluctant to use FSLs due to lack of productive tools to ease the work of software development.

6. Conclusion and Future Directions

Generally, the difficulty of using FSLs leads to the gap between its advantages and the real practice in software development(Cataño, 2017;Martin et al., 2016;Wang & Miao, 2016). In order to increase the adoption of FSLs, many efforts have to be done in three interlocking directions (see Figure 1). Firstly, the direction of community support which requires support from government, academic institutions and industry to initiate projects based on formal methods. Secondly, developmentdirection whichmotivates cooperation between government, academic community and industrial institutions for developinginteractive tools in order to support the users for learning and using FSLs in developing software projects. This entails developinguser friendly syntaxand interactive interfaces. Szmuc and Szpyrka (2015) noticed that the engineers are discouraged not only by more complicated notation but also lack of advanced tools for specification and analysis of properties. Creating interactive interfaces and productive tools of FSLswillenhance and accelerate writing requirements in a timely manner. Thirdly, teachingdirectionwhich requires creating engagingcurriculum and comprehensive user guides to facilitate the comprehension of formal methods. Moreover, the lecturers have to change their methodology of teaching via providing in depth explanation and sufficient examples of using FSLs to raise the level of comprehension. The assumption is that, by raising comprehensibility, one is also very likely raising acceptability (Bollin & Rauner-reithmayer, 2014). This will encourage students, researchers and practioners to be familiar about the conceptsof formal methods.

Fig. 1: Future directions of formal specification languages

References

[1] Agerholm, S., & Larsen, P. G. (1998). A lightweight approach to formal methods. In In International Workshop on Current Trends in Applied Formal Methods (pp. 168–183).

[2] Alagar, V. S., & Periyasamy, K. (2011). Specification of software systems. (D. Gries & F. B. Schneider, Eds.) (2nd ed.). Springer Science & Business Media.

[3] Alexander, P. (1995). Best of both worlds [formal and semi-formal software engineering]. IEEE Potentials, 14(5), 29–32.

[4] Bollin, A., & Rauner-reithmayer, D. (2014). Formal specification comprehension: the art of reading and writing z. In In Proceedings of the 2nd FME Workshop on Formal Methods in Software Engineering (pp. 3–9). ACM.

[5] Bourque, P., & Fairley, R. E. (2014). Guide to the Software Engineering Body of Knowledge (3.0). IEEE Computer Society Press.
[6] Cataño, N. (2017). An empirical study on teaching formal methods to millennials. In In Proceedings of the 1st International Workshop on Software Engineering Curricula for Millennials (pp. 3–8). IEEE Press. http://doi.org/10.1109/SECM.2017.1
[7] Clarke, E. M., & Wing, J. M. (1996). Formal methods: State of the art and future directions. ACM Computing Surveys (CSUR), 28(4), 626–643.
[8] Diamantopoulos, T., Roth, M., Symeonidis, A., & Klein, E. (2017). Software requirements as an application domain for natural language processing. Language Resources and Evaluation, 51(2), 495–524. http://doi.org/10.1007/s10579-017-9381-z
[9] Fraser, M. D., Kumar, K., & Vaishnavi, V. K. (1991). Informal and Formal Requirements Specification Languages: Bridging the Gap. IEEE Transactions on Software Engineering, 17(5), 454–466. http://doi.org/10.1109/32.90448
[10] Huertas, C., Gómez-ruelas, M., Juárez-Ramírez, R., & Plata, H. (2011). A formal approach for measuring the lexical ambiguity degree in natural language requirement specification: Polysemes and Homonyms focused. In In Uncertainty reasoning and knowledge engineering (URKE), 2011 International Conference (pp. 115–118).
[11] Hussain, A., & Mkpojiogu, E. O. (2016). Requirements: Towards an understanding on why software projects fail. In In AIP Conference Proceedings. AIP Publishing.
[12] Hussain, A., Mkpojiogu, E. O., & Kamal, F. M. (2016). The role of requirements in the success or failure of software projects. International Review of Management and Marketing, 6, 306–311.
[13] Jones, C. B. (1990). Systematic software development using VDM (2nd ed.). Englewood Cliffs: Prentice Hall.
[14] Kassab, M., Neill, C., & Laplante, P. (2014). State of practice in requirements engineering: contemporary data. Innovations in Systems and Software Engineering, 10(4), 235–241. http://doi.org/10.1007/s11334-014-0232-4
[15] Lamsweerde, A. Van. (2000). Formal Specification : a Roadmap. In In Proceedings of the Conference on the Future of Software Engineering (pp. 147–159). ACM.
[16] Lee, S. (1992). A formal methodology for the specification of distributed systems from an object perspective. Louisiana State.
[17] Mandrioli, D. (2015). On the heroism of really pursuing formal methods: title inspired by Dijkstra's On the Cruelty of Really Teaching Computing Science. In In Proceedings of the Third FME Workshop on Formal Methods in Software Engineerin (pp. 1–5). IEEE Press. http://doi.org/10.1109/FormaliSE.2015.8
[18] Martin, B., Bogusch, R., Fraga, A., & Rudat, C. (2016). Bridging the Gap between Natural Language Requirements and Formal Specifications. In In REFSQ Workshops.
[19] Meyer, B. (1993). On formalism in specifications. In Program Verification (pp. 155–189).
[20] Olajubu, O. (2015). A textual domain specific language for user interface modelling. In In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 1060–1062). ACM. http://doi.org/10.1007/978-1-4614-3558-7_84
[21] Pang, C., Pakonen, A., Buzhinsky, I., & Vyatkin, V. (2016). A study on user-friendly formal specification languages for requirements formalization. In In Industrial Informatics (INDIN), 2016 IEEE 14th International Conference (pp. 676–682). IEEE.
[22] Pressman, R. S. (2010). Software engineering: a practitioner's approach (7th ed.). Palgrave Macmillan.
[23] Robie, M. A. M., Baharom, F., & Mohd, H. (2014). Functional requirements specification for e-tendering system using requirement template. In In Knowledge Management International Conference (KMICe), Langkawi, Malaysia.
[24] Ronen, B. (2017). Excessive software development : Practices and penalties, 35, 13–27.
http://doi.org/10.1016/j.ijproman.2016.10.002
[25] Ryan, K. (1993). The role of natural language in requirements engineering. In In Requirements Engineering, 1993., Proceedings of IEEE International Symposium (pp. 240–242).
[26] Smith, G. (2012). The Object-Z specification language. Springer Science & Business Media.
[27] Sommerville, I. (2010). Software Engineering: Pearson New International Edition. Pearson Education Limited.
http://doi.org/10.1111/j.1365-2362.2005.01463.x
[28] Spivey, J. (1992). The Z notation : A Reference Manual. Prentice Hall International.
[29] Szmuc, T., & Szpyrka, M. (2015). Formal methods—Support or scientific decoration in software development? In In Mixed Design of Integrated Circuits & Systems (MIXDES), 2015 22nd International Conference (pp. 24–31). IEEE.
[30] Tse, T. H., & Pong, L. (1991). An examination of requirements specification languages. The Computer Journal, 34(2), 143–152.
[31] Wang, X., & Miao, W. (2016). Automatic support for formal specification construction using pattern knowledge. In In Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) (pp. 363–372).
[32] Wiegers, K., & Beatty, J. (2013). Software requirements (3rd Editio). Pearson Education.
[33] Wing, J. M. (1990). A specifier's introduction to formal methods. Computer, 32(9), 8–22. http://doi.org/10.1109/2.58215
[34] Woodcock, J. I. M., PETER GORM, L., Bicarregui, J., & JOHN, F. (2009). Formal Methods : Practice and Experience Engineering College of Aarhus. ACM Computing Surveys (CSUR), 41(4), 1–40.