

An ontology-based meta-modelling approach for software test cases

Nehemiah Mung'au[†] and Emanuele Laurenzi^{*,†}

FHNW University of Applied Sciences and Arts Northwestern Switzerland, Riggbachstrasse 16, 4600 Olten, Switzerland

Abstract

Software testing plays a crucial role in the software development lifecycle, ensuring the reliability and quality of software programs. Despite the advancements in the field, software test cases still suffer of poor specifications, leading to communication issues, inefficiencies, and increased costs. This study investigates the suitability of an ontology-based meta-modelling approach, aiming to support the design of adequate software test cases. The approach promotes the human and machine-interpretability of domain-specific models representing the software test cases. This has the advantage of using automated reasoning services to support the creation of adequate test cases. A new domain-specific modelling language, ontoST, has been developed and implemented in the tool AOAME4STC for the proof of concept.

Keywords

software test cases, ontology-based meta-modelling, ontology-based DSML, ontoST, AOAME4STC

1. Introduction

In the domain of software engineering and quality assurance, test cases are vital for ensuring software dependability and efficiency, aligning with reliability standards and customer needs [1, 2]. Despite their importance, software test cases face challenges such as standardization, interoperability, and adaptability across various testing environments [3]. Complex test cases are difficult to manage and modify, risking obsolescence and loss due to decentralized storage [4]. Most importantly, test cases still suffer from poor specifications, which lead to communication issues, inefficiencies, and increased costs [5, 6]. Thus, software test cases need to be adequately specified. According to [7], an adequate test case has the following benefits: it effectively reveals defects with minimal effort, delivers accurate results, improves system performance at a lower cost, and has a strong likelihood of uncovering unknown defects.

In this work, we propose an ontology-based meta-modelling approach to support the design of adequate software test cases. This includes a new ontology-based domain-specific modelling language (DSML), called ontoST. To ensure rigor and extensibility, ontoST has been engineered by supplementing the Design Science Research (DSR) strategy [8] with the Agile Modelling Method Engineering (AMME) methodology [58].

The paper is structured as follows. Section 2 describes the background and related work, ending with the problem statement. Section 3 introduces the artifact requirements. Section 4 discusses the proposed ontology-based DSML ontoST. Section **Error! Reference source not found.** shows a running example of how ontoST has been implemented in the modeling tool AOAME4STC and subsequently used for the proof of concept. The paper concludes with section 6.

BIR-WS 2024: BIR 2024 Workshops and Doctoral Consortium, 23rd International Conference on Perspectives in Business Informatics Research (BIR 2024), September 11-13, 2024, Prague, Czech Rep.

* Corresponding author.

† These authors contributed equally.

✉ nehemiamkubamungau@gmail.com (N. Mung'au); emanuele.laurenzi@fhnw.ch (E. Laurenzi)

ORCID 0000-0001-9142-7488 (E. Laurenzi)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Background and related work

Software test cases are work products of test analysis and design phase of software testing as illustrated in **Error! Reference source not found.** by [9]. “Test Case” (TC) has been recognized as a building block for describing testing items, widely used as a work unit, metric and documentation entity” [3].

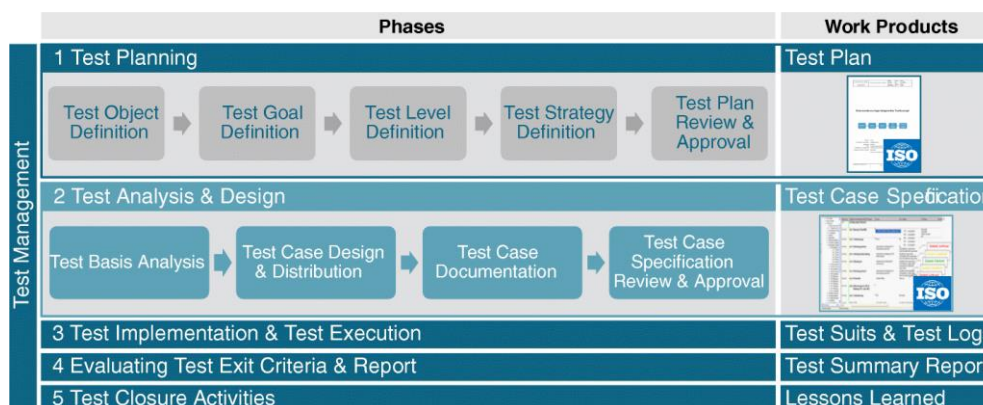


Figure 1: Phases of the fundamental test process and activities of the test planning and test analysis & design phases [9].

Software testing faces some challenges due to the complexities of software, financial and time constraints, and the need for high quality standards [10]. The increasing complexity of modern applications and competitive pressures further raise quality assurance standards [11]. Agile development introduces frequent requirement changes, complicating test case management [11, 12]. Agile methodology's reliance on people over processes presents specific challenges during the software development lifecycle (SDLC) [14]. One major issue is the lack of traceability of test cases to other artifacts and source code [14, 15, 16]. This is exacerbated by inconsistent, incomplete, and inaccurate requirements [18]. Agile methods also lead to inconsistent and inadequate test cases [15], making it difficult for test cases to effectively validate software behaviours [19] and avoid errors [9]. Semantic clarity and consistency in test cases are often lacking, causing misunderstandings due to varying participant knowledge and experiences [9], [20]. Additionally, insufficient tool integration for software test cases hampers seamless testing processes [21]. These challenges highlight the need for improved test case design to ensure high-quality software.

According to [22] Model-Based Testing (MBT) strives to automatically create tests (test cases) based on a model that describes specific behaviours of the system being tested (SUT). MBT offers several key motivations, including facilitating automated test case generation, providing comprehensive test coverage, and simplifying defect discovery. According to [23] MBT is becoming more and more recognized in the market as a cost-cutting strategy that automates test case generation, reducing the need for manual test suite creation and improving test cycle efficiency. Additionally, [23] also emphasize MBT's potential to lower costs and enhance test effectiveness. Compared to manual case generation, [25] assert that a complete abstraction test model allows for more comprehensive testing. According to [26], MBT can test a wider range of scenarios compared to record-based testing. Additionally, [25] demonstrate that MBT automation outperforms manual methods in error detection, citing a case study where MBT found significantly more faults than manual techniques.

Despite these benefits, MBT has some drawbacks too, such as the requirement for specialized skills, initial labour intensity, and model complexity. According to [22] testers need to be familiar with state machines, formal languages, and automata theory, as well as tools and scripts for test automation. Furthermore, [27] highlights the significant initial investment and labour needed for MBT, as careful selection of model types and division of software portions are necessary for effective modelling. The complexity of MBT models is underscored by the state-space explosion,

which complicates maintenance and presents significant challenges, particularly for beginners [24, 26].

Ontologies [27], given their both conceptualization and automation power, can overcome the issues posed by MBT approaches. For example, automated software test case generation is significantly enhanced by combining ontology-based requirement specification with learning-based methods, as proposed by [29]. An Ontology-based framework was proposed by [29] that automates test case generation, execution, and verdict construction using a knowledge-based system and learning-based testing algorithm. Similar approaches are suggested by [30, 31]. In test scenario management, the authors in [32, 33] discuss the use of ontologies to generate relevant test scenarios for complex systems, emphasizing the need for detailed and specific entity descriptions. An ontology-based approach for test case prioritization was suggested by [34] which deemed particularly useful during retesting after software updates. The importance of domain knowledge and knowledge representation in efficient software testing is emphasized in [35, 36], which proposed that ontologies can solve problems such as uneven knowledge representation and focused expertise by creating semantic links between data and knowledge.

These insights collectively advance the field of software testing through improved automation, management, and knowledge sharing. However, pure ontology-based approaches in software test cases face some limitations too. According to [37] the development of ontology-based software test case generation tools is often manual and costly due to the lack of supporting tools. Furthermore, [38] emphasize the need for user-friendly ontology representations that fit the workflow of domain experts, who may not be skilled in ontology development or formal languages. Additionally, [38] highlight that ontologies require input from domain knowledge experts, who may not be familiar with the formal languages or logic needed for ontology development, making the process dependent on both domain and ontology experts.

Using domain-specific modelling language (DSMLs) in software test cases holds the promise to address such non-usability and non-understandability issues raised by ontology-based approaches. According to [38], DSML is a modelling language built for a specific area of discourse, enriching general modelling notions with domain-specific terminology and concepts reconstructed from that domain. Improved communication is one key advantage; DSMLs are expressive and concise, effectively representing concepts and relationships within a specific domain [39, 40]. Empirical studies by [41] compare DSMLs and general-purpose languages (GPMLs) (e.g. UML Class Diagram) based on cognitive dimensions outlined by [42] show that DSMLs perform better in areas such as abstraction gradient, consistency, and error-proneness. This would make test cases created with DSMLs easier for stakeholders to understand and validate, ensuring alignment with domain requirements. Additionally, DSMLs promote increased productivity and consistency from the early stages of development, enhancing the quality of models produced [43]. Additionally, [44] also noted that DSMLs are quickly learnable by domain experts, improving the language's applicability.

To take the full advantage of DSMLs and ontologies, [45, 46], describe an ontology-based meta-modelling approach as being interpretable by both humans and machines. They elaborate by explaining that within the realm of information systems, human interpretability pertains to meta-models, while machine interpretability primarily concerns the formal semantic aspects of models.

To mitigate the requirement for specialized skills in both MBT and ontology-based approaches, the ontology-based meta-modelling approach was extended in this work. Graphics depictions of models are useful for humans, while ontologies make knowledge in models' machine interpretable. The ontology-based metamodeling technique was extended and implemented by [46] with the introduction of AOAME, an Agile and Ontology-Aided Modelling Environment. In this work we extended AOAME to accommodate the new ontology-based DSML for software test cases.

3. Artifacts requirements

To tackle step 1 of the development process for a DSML, as outlined by [47,58], we collected requirements through semi-structured interviews with at least five experts in software testing—such as test engineers, test managers, and developers with testing experience. We also reviewed literature on best practices and existing industry standards such as test cases. An excerpt of the requirements, including their descriptions and sources, is presented in Table 1.

Table 1

List of Requirements for the software test case DSML.

Number #	Requirement Description	Source of elicitation
6	Test Case Specifications ² - This ensures that we cover the general DSML requirement of the concepts of a modeling language should correspond to concepts prospective users are familiar with as state by [57]	Interview/Questionnaires with software testing professionals Literature [48].
7	Integration with software requirements – This serves as the most important quality factors for a software test case.	Interviews/questionnaires with software testing industry experts Literature review of best practices for software test case [49, 50, 51].
8	The DSML should support the software testing techniques by default or be extensible	Interviews/questionnaires with software testing industry experts Literature review of best practices for software test cases [52].
9	The DSML should support best communication and collaboration techniques for software test cases. For example, Behavior-driven development Gherkin syntax	Interviews/questionnaires with software testing industry experts. Literature review of best practices for software test cases case [49, 51, 53].
10	The DSML should support reusability of test cases	Interviews/questionnaires with software testing industry experts. Literature review of [54, 55].
11	The DSML should support organization and prioritization of software test cases	Interviews/questionnaires with software testing industry experts. Literature review [49, 56].
12	The DSML should support other testing tools by providing an easy way to export test cases	This was derived from the interviews.
13	The DSML should ensure software test cases designed with it are consistent	This was derived from the interviews.

² <https://doi.org/10.1109/IEEESTD.1983.81615>

4. ontoST: The proposed ontology-based DSML

This section aims to show "How can ontology-based DSML of software test cases be conceptualized?" and fulfil the suggestion phase of the Design Science Research methodology. The approach follows [47] DSML development process steps by creating a DSML through three steps: creating concrete syntax, creating abstract syntax, and defining language semantics. The abstract syntax, represented by a metamodel, depicts the language concepts and their relationships, while the concrete syntax explains how these concepts are visually and textually represented as domain-specific modelling elements. Language semantics impose structural and features to govern syntax and semantics. **Error! Reference source not found.** illustrates suggested abstract syntax. The meta model builds on [48] specifications for software test cases in addition to requirements gathered from additional literature and consultations with industry experts as discussed in 3. Test case, Test suite, Test expectation, Test results, and Test inputs are inferred from [48] while Message flow, Sequence flow, Gateways are additional elements added to facilitate the modelling of software test cases. This abstract syntax can be extended to customize the DSML. Table 2 represents the concrete syntax of the DSML and it visually represents the default abstract syntax conceptual elements.

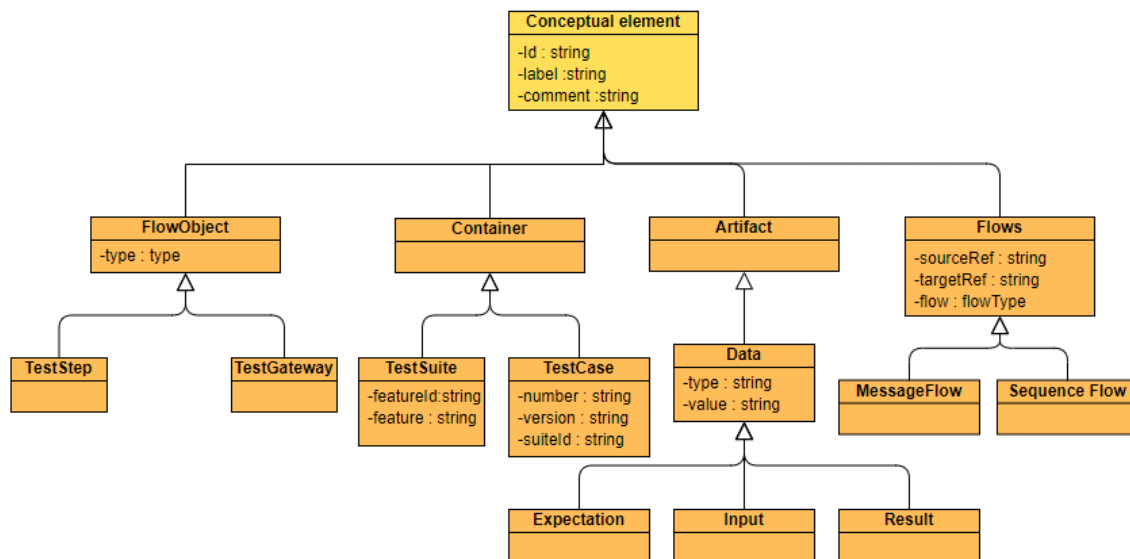





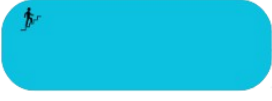





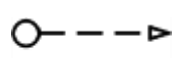



Figure 2: Suggested ontoST abstract syntax.

Table 2
Suggested concrete syntax of ontoST.

Element	Graphical notation	Description
Test Suite		Represents a set of test cases that will be used to test a particular functionality.
Test case		Represents instructions for testers to follow to ensure programs are functioning properly
Exclusive Gateway (XOR)		Represents a decision point where only one outgoing path can be taken based on conditions. It's a mutually exclusive choice.

Parallel Gateway		Splits the flow into multiple parallel paths or synchronizes multiple incoming paths. All outgoing paths are taken simultaneously.
Inclusive Gateway		Splits the flow into one or more paths based on conditions, and all active paths must be completed before merging.
Test Step		Represents test case step. This is a step that should be executed and observed for results.
Input		Represents input data to a test step.
Result		Represents the actual result received after the test.
Expectation		Represents the expectation of a test step after the test.
Assertion		Represents test step asserts.
Sequence flow		Used for connecting Test steps, Gateways and Expectation.
Message flow		Used for showing input or output message flow from input and result elements.
Start		Used to show the beginning of software test cases.

Error! Reference source not found. shows the three main ontologies of AOAME that were extended to design our software test case DSML. Specifically, the Meta-Model Ontology (MMO) that mirrors the abstract syntax, the Domain Ontology (DO) captures the semantic domain; concepts originating from the MMO are aligned with those from the DO. The Palette Ontology (PO) represents the graphical notations of the language and is directly associated with concepts

within the abstract syntax. The PO contains concepts and relations regarding the modelling language's graphical notations, as well as information of how to position the graphical notations on the palette. Thus, the ME palette is supplied by the PO concepts. The MMO contains classes and characteristics that describe a modelling language's abstract syntactic elements, such as modelling elements and modelling relations, as well as the taxonomy and object properties associated with them. MMO consists of one or more modelling languages, either distinct or merged. The DO contains classes and properties that describe the semantic domain.

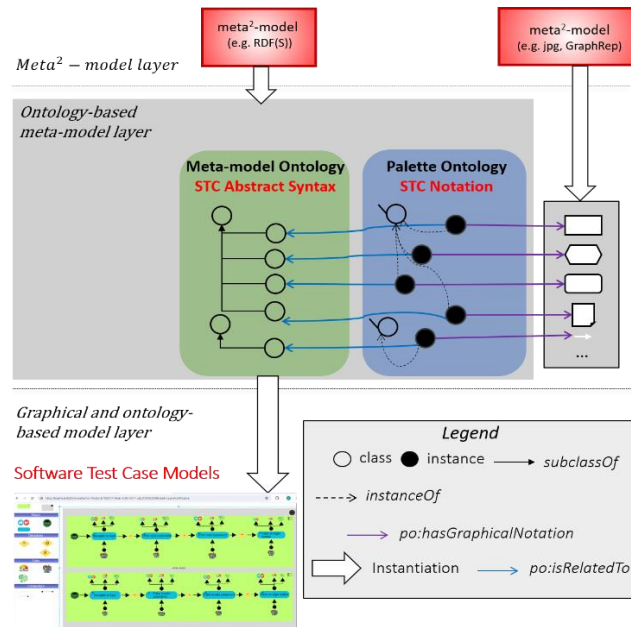


Figure 3: The Ontology-based Meta-modelling Architecture for ontoST. Adapted from [57].

5. Proof of concept

In this section, we evaluate the utility of the approach in two ways: first, we prove that the models can be created with the new ontology-based DSML ontoST, then we show how the ontology can be used to support the design of adequate software test cases. In both cases, a real-world scenario is considered.

5.1. Evaluation of the new ontology-based DSML ontoST

In this section, we will demonstrate the use of ontoST modelling language within the AOAME4STC tool to model test cases created for evaluating the login functionality of a bank website. **Error! Reference source not found.** shows the current record-based software test case representation used by some test case designers. **Error! Reference source not found.** shows the model created using ontoST representing the test cases shown in **Error! Reference source not found.**.

Test Case ID	Test Case Description	Pre Steps	Test Step	Preconditions	Test Data	Expected Result	Actual Result	Status	Comments
Test the Login Functionality in Banking	Verify login functionality with valid username & password		Navigate to login page			Able to see the login page	As expected	Pass	
			Enter valid username	Valid Username	username: choudaryac97@gmail.com	Credential can be entered	As expected	Pass	
			Enter valid password	Valid password	password: XXXXXXX@1	Credential can be entered	As expected	Pass	
			Click on login button			User logged successfully	As expected	Pass	
Test the Login Functionality in Banking	Verify login functionality with valid username & invalid password		Navigate to login page			Able to see the login page	As expected	Pass	
			Enter valid username	Valid Username	username: choudaryac97@gmail.com	Credential can be entered	As expected	Pass	
			Enter valid password	Invalid password	password: XXXXXXX@2	Credential can be entered	As expected	Pass	
			Click on login button			User logged	Unsuccessful login	Fail	

Figure 4: Sample Software test case for testing login functionality [56].

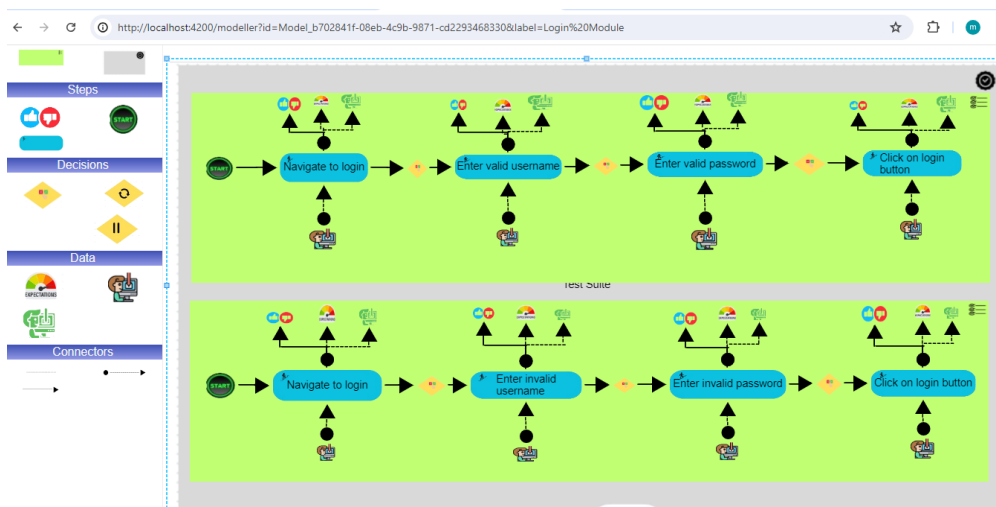


Figure 5: Representation of the sample Test case in Figure 4 using ontoST in AOAME4STC.

5.2. Validation of the modeled test cases for conformance with test case specifications

In this section we evaluated the test case model created against one of [56] test case specifications constraint that states that every test case must have at least one test step. In **Error! Reference source not found.** we have 2 test cases in our model where one does not meet the requirement of having a test step. We have manually labelled it as wrong. As seen in **Error! Reference source not found.**, when we run the SPARQL query against the ontology created for the model, we receive an incorrect testcase triple.

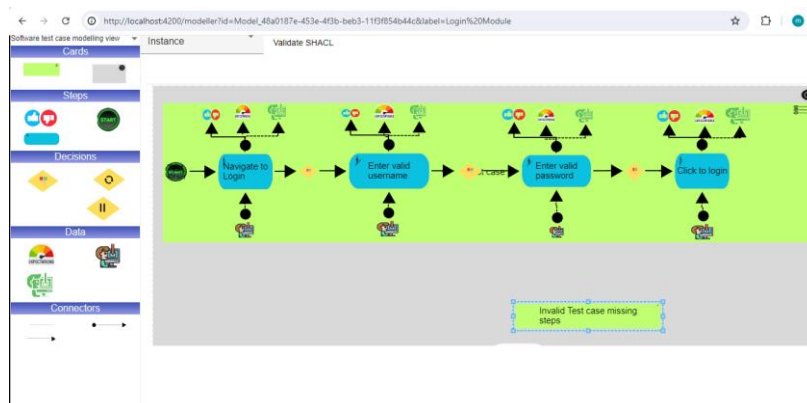


Figure 6: Sample test cases modelled with the ontology-based DSML

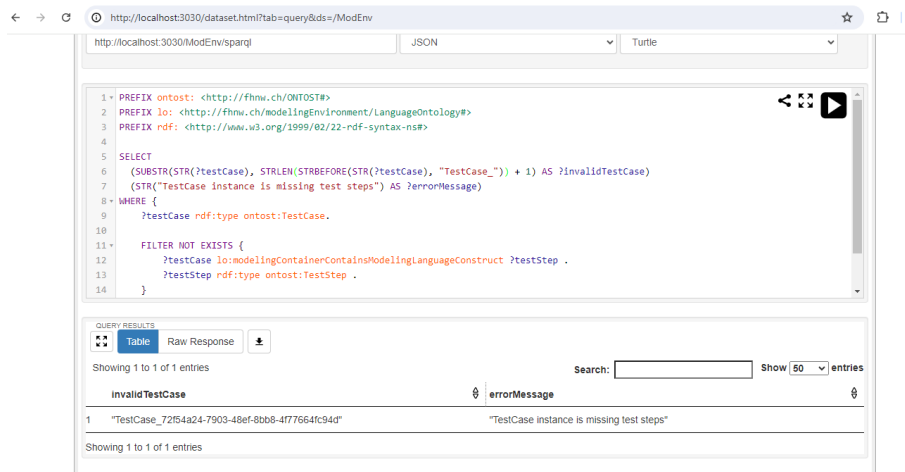


Figure 7: Result of evaluating test cases missing test steps

6. Conclusion

This paper demonstrated the suitability of an ontology-based meta-modelling approach for the support of the design of adequate software test cases. For this, a new ontology-based domain-specific modelling language, ontoST, was created. The latter was implemented in the modelling environment AOAME4STC, which was used for the proof of concept.

As a future work, we regard important to evaluate the perceived usefulness of the new DSML ontoST with software testers and to continue validating ontoST by modeling additional software test cases.

References

- [1] A. S. Verma, A. Choudhary, and S. Tiwari, 'Software Test Case Generation Tools and Techniques: A Review', *Int. J. Math. Eng. Manag. Sci.*, vol. 8, no. 2, pp. 293–315, Apr. 2023, doi: 10.33889/IJMEMS.2023.8.2.018.
- [2] P. Kamde, V. Nandavadekar, and R. Pawar, 'Value of Test Cases in Software Testing', in *2006 IEEE International Conference on Management of Innovation and Technology*, Singapore, China: IEEE, Jun. 2006, pp. 668–672. doi: 10.1109/ICMIT.2006.262303.
- [3] D. Almog and T. Heart, 'What Is a Test Case? Revisiting the Software Test Case Concept', in *Software Process Improvement*, vol. 42, R. V. O'Connor, N. Baddoo, J. Cuadrado Gallego, R. Rejas Muslera, K. Smolander, and R. Messnarz, Eds., in *Communications in Computer and Information Science*, vol. 42., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 13–31. doi: 10.1007/978-3-642-04133-4_2.
- [4] T. Parveen, S. Tilley, and G. Gonzalez, 'A case study in test management', in *Proceedings of the 45th annual southeast regional conference*, Winston-Salem North Carolina: ACM, Mar. 2007, pp. 82–87. doi: 10.1145/1233341.1233357.
- [5] N. Shete and A. Jadhav, 'An empirical study of test cases in software testing', in *International Conference on Information Communication and Embedded Systems (ICICES2014)*, Chennai, India: IEEE, Feb. 2014, pp. 1–5. doi: 10.1109/ICICES.2014.7033883.
- [6] K. Juhnke, M. Tichy, and F. Houdek, 'Challenges concerning test case specifications in automotive software testing: assessment of frequency and criticality', *Softw. Qual. J.*, vol. 29, no. 1, pp. 39–100, Mar. 2021, doi: 10.1007/s11219-020-09523-0.
- [7] S. O. Barraood, H. Mohd, and F. Baharom, 'An initial investigation of the effect of quality factors on Agile test case quality through experts' review', *Cogent Eng.*, vol. 9, no. 1, p. 2082121, Dec. 2022, doi: 10.1080/23311916.2022.2082121.

- [8] V. Vaishnavi and B. Kuechler, 'Design Science Research in Information Systems', *Assoc. Inf. Syst.*, Jan. 2004.
- [9] K. Juhnke, M. Tichy, and F. Houdek, 'Challenges Concerning Test Case Specifications in Automotive Software Testing', in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Prague: IEEE, Aug. 2018, pp. 33–40. doi: 10.1109/SEAA.2018.00015.
- [10] P. Mudholkar, M. Mudholkar, and S. Kulkarni, 'Software testing', in *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, Mumbai Maharashtra India: ACM, Feb. 2010, pp. 1024–1024. doi: 10.1145/1741906.1742242.
- [11] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, 'Software Testing Techniques: A Literature Review', in *2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, Jakarta, Indonesia: IEEE, Nov. 2016, pp. 177–182. doi: 10.1109/ICT4M.2016.045.
- [12] P. Nidagundi and L. Novickis, 'Introducing Lean Canvas Model Adaptation in the Scrum Software Testing', *Procedia Comput. Sci.*, vol. 104, pp. 97–103, 2017, doi: 10.1016/j.procs.2017.01.078.
- [13] K. Beck, *Test Driven Development By Example*. Hoboken: Pearson Education, Limited, 2002.
- [14] T. Dyba, 'Improvisation in Small Software Organizations', *IEEE Softw.*, vol. 17, no. 5, pp. 82–87, Sep. 2000, doi: 10.1109/52.877872.
- [15] J. Fischbach, H. Femmer, D. Mendez, D. Fucci, and A. Vogelsang, 'What Makes Agile Test Artifacts Useful? An Activity-Based Quality Model from a Practitioners' Perspective', 2020, doi: 10.48550/ARXIV.2009.01722.
- [16] B. V. Rompaey and S. Demeyer, 'Establishing Traceability Links between Unit Test Cases and Units under Test', in *2009 13th European Conference on Software Maintenance and Reengineering*, Kaiserslautern, Germany: IEEE, 2009, pp. 209–218. doi: 10.1109/CSMR.2009.39.
- [17] K. Kärkliņa and R. Pirta, 'Quality metrics in Agile Software Development Projects', *Inf. Technol. Manag. Sci.*, vol. 21, pp. 54–59, Dec. 2018, doi: 10.7250/itms-2018-0008.
- [18] S.-T. Lai, 'A Maintainability Enhancement Procedure for Reducing Agile Software Development Risk', *Int. J. Softw. Eng. Appl.*, vol. 6, no. 4, pp. 29–40, Jul. 2015, doi: 10.5121/ijsea.2015.6403.
- [19] R. Romli, S. Sarker, M. Omar, and M. Mahmood, 'Automated Test Cases and Test Data Generation for Dynamic Structural Testing in Automatic Programming Assessment Using MC/DC', *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 10, no. 1, pp. 120–127, Feb. 2020, doi: 10.18517/ijaseit.10.1.10166.
- [20] R. Lachmann and I. Schaefer, *Towards Efficient and Effective Testing in Automotive Software Development*. in GI-Jahrestagung. 2014. [Online]. Available: <https://api.semanticscholar.org/CorpusID:4870726>
- [21] M. Broy, 'Challenges in automotive software engineering', in *Proceedings of the 28th international conference on Software engineering*, Shanghai China: ACM, May 2006, pp. 33–42. doi: 10.1145/1134285.1134292.
- [22] A. Meriem and M. Abdelaziz, 'A Meta-Model for Model-Based Testing Technique: A Review', *J. Softw. Eng.*, vol. 12, no. 1, pp. 1–11, Dec. 2017, doi: 10.3923/jse.2018.1.11.
- [23] S. Schulz, J. Honkola, and A. Huima, 'Towards Model-Based Testing with Architecture Models', in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, Tucson, AZ, USA: IEEE, Mar. 2007, pp. 495–502. doi: 10.1109/ECBS.2007.73.
- [24] S. R. Dalal, A. Jain, Karunanithi, N., Leaton, J. M., Lott, C. M., Patton, G. C., & Horowitz, B. M, 'Model-based testing in practice', in *Proceedings of the 21st international conference on Software engineering*, Los Angeles California USA: ACM, May 1999, pp. 285–294. doi: 10.1145/302405.302640.
- [25] S. Dhawan, N. Kumar, and S. Saini, 'MODEL BASED TESTING CONSIDERING STEPS, LEVELS, TOOLS & STANDARDS OF SOFTWARE QUALITY', vol. 2, no. 4, 2011.

- [26] 'Model based testing vs. Record based testing', Tricentis. Accessed: Dec. 09, 2023. [Online]. Available: <https://www.tricentis.com/blog/model-based-vs-record-based-testing/>
- [27] L. Anton, 'How to Improve Your Workflow Using Model-Based Testing', freeCodeCamp.org. Accessed: Dec. 10, 2023. [Online]. Available: <https://www.freecodecamp.org/news/improve-your-workflow-using-model-based-testing/>
- [28] T. R. Gruber, 'Toward principles for the design of ontologies used for knowledge sharing?', *Int. J. Hum.-Comput. Stud.*, vol. 43, no. 5–6, pp. 907–928, Nov. 1995, doi: 10.1006/ijhc.1995.1081.
- [29] S. Ul Haq and U. Qamar, 'Ontology Based Test Case Generation for Black Box Testing', in *Proceedings of the 2019 8th International Conference on Educational and Information Technology*, Cambridge United Kingdom: ACM, Mar. 2019, pp. 236–241. doi: 10.1145/3318396.3318442.
- [30] H. N. Anjalika, M. T. Yasantha, and P. I. Siriwardhana, 'An Ontology Based Test Case Generation Framework', 2017.
- [31] H.-J. Happel and S. Seedorf, 'Applications of Ontologies in Software Engineering', 2006. Accessed: Dec. 11, 2023. [Online]. Available: <https://www.semanticscholar.org/paper/Applications-of-Ontologies-in-Software-Engineering-Happel-Seedorf/11e2c3bfe1dd68446180f17e476addc947dad095>
- [32] T. Hamilton, 'Test Case vs Test Scenario – Difference Between Them'. Accessed: Dec. 11, 2023. [Online]. Available: <https://www.guru99.com/test-case-vs-test-scenario.html>
- [33] M. Zipfl, N. Koch, and J. M. Zöllner, 'A Comprehensive Review on Ontologies for Scenario-based Testing in the Context of Autonomous Driving', Apr. 21, 2023, *arXiv: arXiv:2304.10837*. Accessed: Dec. 11, 2023. [Online]. Available: <http://arxiv.org/abs/2304.10837>
- [34] M. Hasnain, I. Ghani, M. F. Pasha, and S.-R. Jeong, 'Ontology-Based Regression Testing: A Systematic Literature Review', *Appl. Sci.*, vol. 11, no. 20, p. 9709, Oct. 2021, doi: 10.3390/app11209709.
- [35] S. Vasanthapriyan, J. Tian, and J. Xiang, 'An Ontology-Based Knowledge Framework for Software Testing', in *Knowledge and Systems Sciences*, vol. 780, J. Chen, T. Theeramunkong, T. Supnithi, and X. Tang, Eds., in *Communications in Computer and Information Science*, vol. 780., Singapore: Springer Singapore, 2017, pp. 212–226. doi: 10.1007/978-981-10-6989-5_18.
- [36] Z. Sun, C. Hu, C. Li, and L. Wu, 'Domain Ontology Construction and Evaluation for the Entire Process of Software Testing', *IEEE Access*, vol. 8, pp. 205374–205385, 2020, doi: 10.1109/ACCESS.2020.3037188.
- [37] V. Tarasov, H. Tan, A. Adlemo, A. Andersson, M. Ismail, M. Johansson, & D. Olsson, 'Ontology-based Software Test Case Generation (OSTAG)', in *European Projects in Knowledge Applications and Intelligent Systems*, Lisbon, Portugal: SCITEPRESS - Science and Technology Publications, 2015, pp. 135–159. doi: 10.5220/0007901301350159.
- [38] A. Westerinen and R. Tauber, 'Ontology development by domain experts (without using the "O" word)', *Appl. Ontol.*, vol. 12, pp. 1–13, Aug. 2017, doi: 10.3233/AO-170183.
- [39] A. Van Deursen, P. Klint, and J. Visser, 'Domain-specific languages: an annotated bibliography', *ACM SIGPLAN Not.*, vol. 35, no. 6, pp. 26–36, Jun. 2000, doi: 10.1145/352029.352035.
- [40] U. Frank, 'The MEMO meta modelling language (MML) and language architecture', *ICB Res. Rep.*, Art. no. 24, 2008, Accessed: Apr. 14, 2024. [Online]. Available: <https://ideas.repec.org/p/zbw/udeicb/24.html>
- [41] T. Kosar N. Oliveira, M. Mernik, M. João, M. V. Pereira, M. Repinšek, D. C. da Cruz, & P. R. Henrique, 'Comparing general-purpose and domain-specific languages: An empirical study', *Comput. Sci. Inf. Syst.*, vol. 7, no. 2, pp. 247–264, 2010, doi: 10.2298/CSIS1002247K.
- [42] T. R. G. Green and M. Petre, 'Usability Analysis of Visual Programming Environments: A "Cognitive Dimensions" Framework', *J. Vis. Lang. Comput.*, vol. 7, no. 2, pp. 131–174, Jun. 1996, doi: 10.1006/jvlc.1996.0009.

- [43] M. Mernik, J. Heering, and A. M. Sloane, 'When and how to develop domain-specific languages', *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, Dec. 2005, doi: 10.1145/1118890.1118892.
- [44] P. Hudak, 'Building domain-specific embedded languages', *ACM Comput. Surv.*, vol. 28, no. 4es, p. 196, Dec. 1996, doi: 10.1145/242224.242477.
- [45] K. Hinkelmann, E. Laurenzi, A. Martin, and B. Thönssen, 'Ontology-Based Metamodeling', in *Business Information Systems and Technology 4.0*, vol. 141, R. Dornberger, Ed., in *Studies in Systems, Decision and Control*, vol. 141., Cham: Springer International Publishing, 2018, pp. 177–194. doi: 10.1007/978-3-319-74322-6_12.
- [46] E. Laurenzi, K. Hinkelmann, and A. Van Der Merwe, 'An Agile and Ontology-Aided Modeling Environment', in *The Practice of Enterprise Modeling*, vol. 335, R. A. Buchmann, D. Karagiannis, and M. Kirikova, Eds., in *Lecture Notes in Business Information Processing*, vol. 335., Cham: Springer International Publishing, 2018, pp. 221–237. doi: 10.1007/978-3-030-02302-7_14.
- [47] H. Cho, J. Gray, and E. Syriani, 'Creating visual Domain-Specific Modeling Languages from end-user demonstration', in *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, Zurich, Switzerland: IEEE, Jun. 2012, pp. 22–28. doi: 10.1109/MISE.2012.6226010.
- [48] 'How to Write Effective Test Cases (With Templates) - TestRail'. Accessed: Apr. 07, 2024. [Online]. Available: <https://www.testrail.com/blog/effective-test-cases-templates/>
- [49] A. Beer, M. Junker, H. Femmer, and M. Felderer, 'Initial Investigations on the Influence of Requirement Smells on Test-Case Design', in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, Lisbon, Portugal: IEEE, Sep. 2017, pp. 323–326. doi: 10.1109/REW.2017.43.
- [50] R. D. Craig and S. P. Jaskiel, *Systematic software testing*. in Artech House computing library. Boston: Artech House, 2002.
- [51] S. Ganji, 'Test Case Design - A Guide for QA Engineers With Examples', ACCELQ Inc. Accessed: Apr. 21, 2024. [Online]. Available: <https://www.accelq.com/blog/test-case-design/>
- [52] H. K. V. Tran, N. B. Ali, J. Börstler, and M. Unterkalmsteiner, 'Test-Case Quality – Understanding Practitioners' Perspectives', in *Product-Focused Software Process Improvement*, X. Franch, T. Männistö, and S. Martínez-Fernández, Eds., Cham: Springer International Publishing, 2019, pp. 37–52. doi: 10.1007/978-3-030-35333-9_3.
- [53] R. Li and S. L. Ma, *The Use of Ontology in Case Based Reasoning for Reusable Test Case Generation*. 2015. doi: 10.2991/aiie-15.2015.102.
- [54] Zhang Juan, L. Cai, W. Tong, Yuan Song, and Li Ying, 'Test Case Reusability Metrics Model', in *2010 2nd International Conference on Computer Technology and Development*, Cairo, Egypt: IEEE, Nov. 2010, pp. 294–298. doi: 10.1109/ICCTD.2010.5645869.
- [55] W. Afzal, *Metrics in Software Test Planning and Test Design Processes*. 2007. Accessed: Apr. 28, 2024. [Online]. Available: <https://urn.kb.se/resolve?urn=urn:nbn:se:bth-6193>
- [56] 'IEEE Standard for Software Test Documentation', *IEEE Std 829-1983*, pp. 1–48, Feb. 1983, doi: 10.1109/IEEESTD.1983.81615.
- [57] U. Frank, 'Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines', in *Domain Engineering*, I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 133–157. doi: 10.1007/978-3-642-36654-3_6.
- [58] D. Karagiannis, 'Agile modeling method engineering', in *Proceedings of the 19th Panhellenic Conference on Informatics*, Athens Greece: ACM, Oct. 2015, pp. 5–10. doi: 10.1145/2801948.2802040.