# Cause-effect graphing technique: A survey of available approaches and algorithms

Ehlimana Krupalija
*Faculty of Electrical Engineering*
*University of Sarajevo*
Sarajevo, Bosnia and Herzegovina
ekrupalija1@etf.unsa.ba

Emir Cogo
*Faculty of Electrical Engineering*
*University of Sarajevo*
Sarajevo, Bosnia and Herzegovina
ec15261@etf.unsa.ba

Šeila Bećirović
*Faculty of Electrical Engineering*
*University of Sarajevo*
Sarajevo, Bosnia and Herzegovina
sbecirovic1@etf.unsa.ba

Irfan Prazina
*Faculty of Electrical Engineering*
*University of Sarajevo*
Sarajevo, Bosnia and Herzegovina
iprazina1@etf.unsa.ba

Ingmar Bešić
*Faculty of Electrical Engineering*
*University of Sarajevo*
Sarajevo, Bosnia and Herzegovina
ingmar.besic@etf.unsa.ba

***Abstract*— Cause-effect graphs are often used as a method for deriving test case suites for black-box testing different types of systems. This paper represents a survey focusing entirely on the cause-effect graphing technique. A comparison of different available algorithms for converting cause-effect graph specifications to test case suites and problems which may arise when using different approaches are explained. Different types of graphical notation for describing nodes, logical relations and constraints used when creating cause-effect graph specifications are also discussed. An overview of available tools for creating cause-effect graph specifications and deriving test case suites is given. The systematic approach in this paper is meant to offer aid to domain experts and end users in choosing the most appropriate algorithm and, optionally, available software tools, for deriving test case suites in accordance to specific system priorities. A presentation of proposed graphical notation types should help in gaining a better level of understanding of the notation used for specifying cause-effect graphs. In this way, the most common mistakes in the usage of graphical notation while creating cause-effect graph specifications can be avoided.**

***Keywords—cause-effect graphs, test case suites, black-box testing, software testing, software quality***

## I. INTRODUCTION

Black-box testing is an important part of the software development process, where the development of the system is observed as a black box without any knowledge of its interior and tested from the viewpoint of the end user [1]. Many methods have been developed for the purpose of deriving black-box tests for a given system and they are described in [2], [3], [4], and [5], whereas black-box testing automatization techniques are summarized in [6]. Most commonly used black-box testing methods include boundary value analysis, equivalent classes, pairwise testing and cause-effect graphs. Cause-effect graphing was developed in 1970 as a black-box testing technique for testing hardware logical circuits [7]. Since then, this method has gone through many changes regarding the notation used for describing cause-effect graph elements, including graph nodes, logical relations and constraints. The initially proposed algorithm for deriving test cases from the graph specification has also been modified many times and adapted for specific purposes and application on different types of systems.

Depending on the type of the system being tested and available resources such as time and memory, different priorities can be determined when deriving test case suites from system requirements (e.g. prioritization based on coverage or on failing test cases explained in [8]). In all cases, the tests contained in the test suite need to ensure high fault detection in order to prevent software from failure [9]. Some applications may prefer quantity of test cases over test quality, if tests can be executed quickly. However, if test execution is time-costly, the test case subset needs to be as small as possible while ensuring a high level of fault detection [10] [11]. Validating the conformance of system requirements with the software implementation is very important, because mistakes during this process lead to unrepresentative test case suites and, subsequently, low fault detection (as described in [4] and explored in [12]). For this reason, significant efforts have been invested into improving and automatizing the requirement elicitation process so that cause-effect graph specifications can perfectly conform to system requirements, such as [13] and [14].

Cause-effect graphs have been applied to numerous problems in practice, such as black-box testing safety-critical systems for monitoring high-speed trains [15], missile navigation [14], quantum programs [16], automatizing the college registration process [17], ATM machine money withdrawal [18] and knowledge assessment in e-learning [19]. The graphical notation used in these works for representing cause-effect graph elements in some cases only partially conforms to the notation presented in the standard literature ( [2] and [1]), whereas in some cases entirely novel and non-standardized types of notation are used, which are more difficult to understand by non-domain experts. For this reason, a survey explaining different existing types of graphical notation is necessary, so that the usage of non-standardized notation can be avoided in future works focused on applying the cause-effect graphing technique on different real-world problems.

Additionally, it is necessary to explain and summarize the existing algorithms and approaches for converting cause-effect graph specifications to test suites. This way, the most common drawbacks of the approach used for generating test case suites from cause-effect graph representations identified in [20] can be avoided and a better understanding of the technique can be acquired.

This paper presents a systematic literature review of available cause-effect graphing approaches and algorithms in the form of a secondary study. A survey of this type, focusing entirely on the cause-effect graphing technique, has not been conducted before to the knowledge of the authors of this paper. Research papers (journal articles and conference papers) published from 1990 to 2022 were collected from the following databases: Web of Science (Core Collection), Scopus, EBSCO and IEEE Xplore. Papers were obtained by using the search term "cause-effect graph". The conducted study aims to:

- Present different proposed types of graphical notation for representing cause-effect graph elements (nodes, logical relations and constraints). This may aid the users tasked with creating cause-effect graph specifications so that common mistakes in the usage of graphical notation and improper understanding of different logical relations and constraints can be avoided.
- Give a systematic overview of available algorithms for deriving test case suites from cause-effect graph representations and analyze the strengths and weaknesses of all different approaches. Afterwards, the most appropriate algorithm can be chosen for specific purposes by users of the cause-effect graphing technique.
- Compare available software tools for creating cause-effect graphs and deriving test case suites. In this way, these tools can be used as help for quicker cause-effect graph specification and test case suite derivation.

The rest of this work is structured as follows. Section II describes different forms of cause-effect graphing notation used for graphically describing cause-effect graph elements. An analysis and comparison of available algorithms for deriving test case suites from cause-effect graph specifications is given in Section III. Section IV introduces the available tools for creating cause-effect graph specifications and converting them into test case suites. In Section V, an overall analysis of all presented approaches is given.

## II. Notation used for representing cause-effect graphs

Cause-effect graphs are composed of three types of elements [2] [1]: nodes (causes, effects and intermediates), logical relations between different types of nodes and constraints on same types of nodes. Nodes can be in one of two states – active (1) or inactive (0). Logical relations explain how the change of input variables (graph causes) affects the output variables (graph effects). Constraints explain what combinations of variables are forbidden. The different approaches for representing cause-effect graph elements are summarized in Table I. Cause-effect graphs and all elements they contain are first introduced in [7]. In this work, the purpose of cause and effect nodes is explained. Intermediate nodes are only implicitly defined and are referred to as "unknown dummy nodes". Six types of logical relations – direct, negation, conjunction, disjunction, negated conjunction and negated disjunction are introduced, as well as three types of constraints – mutual exclusion, all-inclusion and mutual exclusion and all-inclusion. This approach is not commonly used in practice, as the usage of intermediate nodes is often necessary for representing complex logical relations.

More detailed explanations of different types of nodes are given in [2]. Intermediates are explicitly defined as a separate type of nodes for the first time. Separate numbering for different types of nodes is introduced as well. This work adapts all six initially proposed types of logical relations and all three different types of constraints. Additional two types of constraints are introduced – requirement and masking. This approach is commonly used in works such as [17], [15], and [16].

A simplification of previous approaches is offered in [1], dropping the negated conjunction and negated disjunction and adopting the remaining four types of logical relations. The usage of intermediates and all constraints is adopted. In this work, cause nodes are numbered incrementally, whereas other types of nodes are numbered according to system requirements. This approach is used in works such as [20] and [21], however representing the negated conjunction and disjunction is more difficult because additional logical relations need to be used.

A different way of representing cause-effect graphs is introduced in [22], making it more suitable for the requirement elicitation process. In this work, only three types of logical relations are used – negation, conjunction and disjunction. No existing types of constraints are adopted. Instead, three new types of constraints are defined – positive, negative and neutral. Other types of connections are also introduced – membership, property and interactions. Node numbering is not used by this approach, due to its intended application for high-level and abstract descriptions of software requirements. Causes are described as "root factors", effects as "top variables/events" and intermediates as "intermediate variables/events".

Table II shows a summary of all approaches for graphically representing cause-effect graph elements (nodes, logical relations and constraints), including the usage of arrow tips. The initial graphical notation for describing cause-effect graphs was introduced in 1970 by [7]. The work contains two graphically illustrated examples which demonstrate the usage of nodes, logical relations and constraints. Logical relations are graphically denoted with straight lines containing different letters depending on the type of relation: D for direct, N for negation, A for conjunction, O for disjunction, $\underline{A}$ for negated conjunction and $\underline{O}$ for negated disjunction. Constraints are graphically denoted with dashed lines containing different letters depending on the type of constraints: E for exclusion, I for inclusion and U for mutual exclusion and all-inclusion.

The most commonly used graphical notation for describing logical relations is introduced in [2] and [1] – "~" for negation, "∧" for conjunction, "∨" for disjunction and a combination of these symbols for the negated conjunction and negated disjunction. The notation for representing exclusion and inclusion is adopted from [7], mutual exclusion and all-inclusion is represented by using the letter O and masking is represented

TABLE I. SUMMARY OF DIFFERENT APPROACHES FOR REPRESENTING CAUSE-EFFECT GRAPH ELEMENTS

| Source | Year | Node types | Node numbering | Logical relations | Constraints |
|---|---|---|---|---|---|
| [7] | 1970 | 1. Causes<br>2. Effects<br>3. Unknown dummy nodes | Incremental | 1. Direct<br>2. Negation<br>3. Conjunction<br>4. Disjunction<br>5. Negated conjunction<br>6. Negated disjunction | 1. Mutual exclusion<br>2. All-inclusion<br>3. Mutual exclusion and all-inclusion |
| [2] | 2010 | 1. Causes<br>2. Effects<br>3. Intermediates | Separate for each type | 1. Direct<br>2. Negation<br>3. Conjunction<br>4. Disjunction<br>5. Negated conjunction<br>6. Negated disjunction | 1. Mutual exclusion<br>2. All-inclusion<br>3. Mutual exclusion and all-inclusion<br>4. Requirement<br>5. Masking |
| [1] | 2012 | 1. Causes<br>2. Effects<br>3. Intermediates | - Incremental for causes<br>- Named for effects and intermediates | 1. Direct<br>2. Negation<br>3. Conjunction<br>4. Disjunction | 1. Mutual exclusion<br>2. All-inclusion<br>3. Mutual exclusion and all-inclusion<br>4. Requirement<br>5. Masking |
| [22] | 2017 | 1. Root factors<br>2. Top variables/events<br>3. Intermediate variables/events | Named for all types | 1. Negation<br>2. Conjunction<br>3. Disjunction | 1. Positive<br>2. Negative<br>3. Neutral |

by using the letter M. Small differences are present between the notations used in these two works. In [2], the required constraint does not use any letters for representation, whereas in [1] the letter R is used for this purpose. In [1], an arrow tip is used for describing the required and masking constraints. This notation is often used in works such as [15], [14] and [16].

Many differences to the previously adopted approaches were introduced in 2017 by [22]. This work proposes the representation of logical relations as logical AND, OR and NOT circuits, which makes cause-effect graphs more similar to hardware logical circuits, as initially intended in [7]. Different types of nodes are represented with different types of bounding boxes (causes as ellipses, effects as circles and intermediates as boxes with rounded corners). Positive relations are described by using the "+" sign and negative relations by using the "-" sign. This approach only uses arrow tips for types of connections for which they are explicitly defined (e.g. causality and correlation).

Some works focused on applying the cause-effect graphing technique to real-world problems do not conform to any of the previously described notation. In [18], arrow tips are used for describing directions of logical relations. "{ ∧" is used for describing conjunction and "¬" is used for describing negation. A type of notation where logical relations are represented as separate nodes (through the usage of rounded bounding boxes) is also presented, where logical relations are named by using words (e.g. conjunction is represented as "and"). A similar approach is also used in [19], where differentiation of relations from nodes is done by representing logical relations through the usage of square bounding boxes.

## III. ALGORITHMS FOR DERIVING TEST CASE SUITES FROM CAUSE-EFFECT GRAPH SPECIFICATIONS

Different algorithms which have been proposed for converting cause-effect graphs to test case suites are summarized in Table III. Two algorithms for generating test case suites from cause-effect graph specifications are introduced in [7] – a brute-force approach which works by propagating values forward through the graph in order to determine resulting effect values, and a backward-propagation approach which forcefully activates effects and determines cause values by propagating the effect values backward through the graph. All subsequent algorithms represent some form of the initially proposed backward-propagation approach. The forward-propagation approach was immediately abandoned due to its high computational complexity, which makes it too slow to be able to generate resulting test case suites in real time. A more detailed explanation of the backward-propagation algorithm is offered in [2] and [1], as well as examples of its application on cause-effect graphs of different sizes.

Many attempts at improving the initial algorithm have been made. In [23], the usage of binary decision trees was proposed in order to further optimize the final test case table so that it contains test cases with multiple simultaneously active effects. In this approach, a binary tree is created for every effect of the graph and the backward-propagation algorithm is executed for every separate tree. At the end of the process, multiple test case tables are merged in order to obtain the optimized test case table. A similar approach was proposed in [24], without the usage of binary trees but instead by performing optimization on the test case table itself. Transforming UML models to cause-effect graphs for achieving a higher level of standardization was proposed in [25]. In this approach, a process for converting UML state machines to cause-effect graphs is proposed in order to enable the application of this technique for systems which are easily represented by UML models. This approach was further explored and applied for deriving test case suites in [26], through the usage of XMI files and the ATLAS programming language.

The first approaches that propose the representation of cause-effect graph elements as Boolean expressions are [27] and [28]. These works introduce the Boolean Operator (BOR) strategy for generating test case suites from cause-effect graph specifications. Combining pairwise testing with cause-effect graphing was proposed in [29], by using the approach based on

TABLE II. SUMMARY OF DIFFERENT APPROACHES FOR THE GRAPHICAL REPRESENTATION OF CAUSE-EFFECT GRAPHS

| Source | Year | Node representation | Usage of arrow tips | Notation for logical relations | Notation for constraints |
|---|---|---|---|---|---|
| [7] | 1970 | Rounded bounding boxes | No | Straight lines with letters (D, N, A, O, A and O) | Dashed lines with letters (E, I, U) |
| [2] and [1] | 2010 and 2012 | Rounded bounding boxes | Yes (only for specific constraint types) | Straight lines with symbols (∧, ∨, ~ and combinations of these symbols) | Dashed lines with letters (E, I, O, R, M) |
| [18] and [19] | 2012 and 2016 | Rounded bounding boxes | Yes (for logical relations) | Rounded/square bounding boxes | Not discussed |
| [22] | 2017 | Causes – ellipses<br>Effects – circles<br>Intermediates – boxes with rounded corners | Yes (only for specific connection types) | Logical circuits | Straight lines with letters (+, -) |

TABLE III. SUMMARY OF DIFFERENT APPROACHES FOR THE USAGE OF THE BACKWARD-PROPAGATION ALGORITHM IN DERIVING TEST CASE SUITES FROM CAUSE-EFFECT GRAPH SPECIFICATIONS

| Approach | Source | Year | Type of algorithm | Algorithm specifics | Consideration of simultaneously active effects |
|---|---|---|---|---|---|
| Brute force | [7] | 1970 | Forward-propagation | Not implemented due to hardware limitations | No |
| Initially proposed | [7] | 1970 | Backward-propagation | APL/360 program for testing hardware circuits | No |
| Boolean expressions | [27] and [28] | 1993 and 1997 | Backward-propagation | Boolean operator (BOR) strategy | No |
| Optimization | [23] | 2009 | Backward-propagation | Usage of binary tress for considering simultaneously active effects | Yes |
| Combination with pairwise testing | [29] | 2014 | Combined | Combined approach with the usage of Boolean expressions and pairwise testing | Yes |
| UML model transformation | [25] and [26] | 2008 and 2014 | Combined | Usage of UML models for transformation | No |
| Boolean expressions | [30] | 2015 | Backward-propagation | Masking modified condition/decision coverage (MCDC) approach | Yes |
| Optimization | [24] | 2017 | Backward-propagation | Decision table reduction based on equivalent test cases | Yes |
| Boolean expressions | [31] | 2021 | Backward-propagation | Mutant-based spectral testing for Boolean specification models | No |

Boolean expressions for representing cause-effect graph elements. The usage of Boolean expressions is further explored in [30] in order to increase the formalization of the cause-effect graph specification process so that it can be less error-prone. Cause-effect graphs are represented mathematically by using Boolean expressions, making it possible to use mathematical formulas when deriving test case suites by using the backward-propagation algorithm. In this way, a smaller test case table is obtained while maintaining a high fault detection rate. This concept was perfected and successfully applied by using the GraphML format in [31].

A systematic review of drawbacks of the initial backward-propagation algorithm is offered in [20]. The main identified weaknesses are:

- Many inconsistencies in the description of the algorithm in [7] and [1], which lead to ambiguous interpretations of the steps of the algorithm and, subsequently, to its incorrect application for the derivation of test case suites.
- The imprecise definition of the algorithm, which leads to the possibility of creating multiple different test case suites for the same set of system requirements which can be modelled in multiple equivalent ways, resulting in multiple cause-effect graphs.
- The usage of the “don’t care” combination for output values, which needs to be resolved to either active or inactive output, but without any clear explanation as to how the resolving process should be done. The “don’t care” combination is still present in more recent works such as [2] and [18].
- Insufficient amount of explanation for different constraint types, which discourages their usage and leads to incomplete test cases suites.
- Lack of consideration for different combinations of output values. In the initially proposed backward-propagation algorithm, only effect activations are

considered, whereas the absence of effect activations may be of interest for different types of systems.

The importance of simultaneous effect activation is emphasized in [23], which is also not considered by the initially proposed algorithm. An attempt to enable the automatization of the test case derivation process and decrease the error-proneness of the initial algorithm is made in all approaches based on Boolean expressions (e.g. [30] and [31]), by introducing a mathematically formulated algorithm with explicitly described steps, removing all ambiguities and inconsistencies. A method for resolving the drawback concerning the misuse of cause-effect graph elements by users of the technique is proposed in [14]. This method automatizes the process of cause-effect graph specification from system requirements by using natural language processing methods to derive cause-effect graph elements from the formal definition of the system. Novel notation more suited for domain experts is introduced in [22] in an attempt to resolve this drawback.

## IV. CAUSE-EFFECT GRAPHING TOOLS

Several software tools for aiding the process of cause-effect graph specification and test case suite derivation have been introduced, as summarized in Table IV. The first such tool, BenderRBT, was developed by [32] in 2002. This tool is a desktop application that allows the user to graphically define the desired cause-effect graph specification, although the graphical elements do not use any of the graphical notation types explained in Section II. The option to derive test case suites is also supported. The tool is commercial and not available for free usage.

Another desktop tool for cause-effect graphing was introduced in [33] (Cause-Effect Graph Software Testing Tool – CEGSTT). This tool allows the user to define cause-effect graph elements through the user interface, however the tool does not support the usage of graphical elements. Test case suites can be derived by using the created cause-effect graph specifications. The tool also enables automatic calculation of the effect coverage metric, which describes the ratio of graph effects covered by the generated test case suite. CEGSTT has not been disclosed by its authors and is therefore unavailable for free usage. Its usability is unclear, because its initial interface was executed on the Microsoft Windows 7 operating system.

Recently, a new tool named Test Generator for Cause-Effect Graphs (TOUCH) has been developed in [31] and this tool is available for free usage at [34]. It is cross-platform and can be executed on the latest operating systems. This tool does not enable the user to graphically define cause-effect graph elements, instead specifying the graph through the user interface. It has the functionality of deriving test case suites from cause-effect graph specifications by using different algorithms based on Boolean expressions and the original backward-propagation algorithm from [1]. In 2022, a new software tool designed for automatic conversion of system requirement descriptions to cause-effect graph specifications, Korean Requirement Analyzer for the Cause-Effect Graph (KRA-CE) [35] was introduced. This tool is web-based, but it has not yet been disclosed by its authors.

TABLE IV. SUMMARY OF AVAILABLE CAUSE-EFFECT GRAPHING TOOLS

| Name | Source | Year | Open-source | Graphical elements | Platform |
|---|---|---|---|---|---|
| BenderRBT | [32] | 2002 | No | Yes | Microsoft Windows |
| CEGSTT | [33] | 2017 | No | No | Microsoft Windows |
| TOUCH | [31] and [34] | 2021 | Yes | No | Cross-platform |
| KRA-CE | [35] | 2022 | No | No | Web-based |

## V. CONCLUSION

This paper gives a systematic summary of different aspects of the cause-effect graphing black-box testing technique. The process of defining graphs is complex and many ways of representation and graphical notations for different graph elements (nodes, logical relations and constraints) have been proposed. Many works which apply cause-effect graphs on different types of systems use different types of notation, which can lead to many errors in the process of defining cause-effect graph specifications and all further steps in black-box testing through the usage of this technique.

The initially proposed backward-propagation algorithm for deriving test cases from cause-effect graph representations was analyzed and many of its improvements and combinations with other methods were mentioned. Known drawbacks of the initially proposed algorithm and attempts at its enhancement were explained, as well as different existing tools for aiding the usage of the cause-effect graphing method. Rather small number of available software tools for aiding the cause-effect graphing process indicates that new tools should be created, preferably with the usage of graphical elements which are not supported in currently available tools.

The existing approaches were introduced, explained and compared to help users determine which notation they should use when describing cause-effect graphs and what elements they can use so that the cause-effect specifications can be complete and contain all necessary logical relations and constraints. Depending on their desired goal, the users can choose one of the many available algorithms for deriving test case suites from cause-effect graph specifications and optionally use the available tools for aiding this process. This can help remove some of the inconsistencies of the most widely used backward-propagation algorithm, resulting in cause-effect graph specifications which are easy to understand by future users due to the usage of standard accepted graphical notation and test case suites generated by using algorithms most suited for the desired user priorities.

Cause-effect graphs have many applications in practice, mainly black-box testing of different types of systems without specific requirements for system boundaries, class similarities and correlation. In the future, the presented concepts can be used for applying the cause-effect graphing technique for black-box testing of safety-critical, industrial, embedded or real-time systems, or any combination thereof. A case study of this type

can be conducted while applying different types of proposed graphical notation and different types of existing algorithms, using the forward-propagation, backward-propagation, or the combined approach. The usability, ease of understanding for domain experts and the fault detection rate of the generated black-box testing suites for these approaches can be compared. Existing approaches can be merged in order to combine the positive aspects of each different approach, such as enhancing the existing graphical notation while using all different proposed types of logical relations and constraints.

## REFERENCES


[1] G. J. Myers, T. Badgett and C. Sandler, The Art of Software Testing, 3rd ed., New Jersey: John Wiley & Sons, Inc., 2012.

[2] S. L. Pfleeger and J. M. Atlee, Software Engineering, Theory and Practice, 4th ed., New Jersey: Pearson Higher Education, 2010.

[3] S. Nidhra and J. Dondeti, "Black box and white box testing techniques - A literature review," *International Journal of Embedded Systems and Applications (IJSEA),* vol. 2, no. 2, pp. 29-50, 2012.

[4] M. J. Pramod and M. Prasanna, "A comparative analysis on black box testing strategies," in *2016 International Conference on Information Science (ICIS)*, Kochi, 2016.

[5] N. Anwar and S. Kar, "Review paper on various software testing techniques & strategies," *Global Journal of Computer Science and Technology,* vol. 19, no. 2, pp. 43-49, 2019.

[6] L. Mariani, M. Pezzè and D. Zuddas, "Recent advances in automatic black-box testing," *Advances in Computers,* vol. 99, pp. 157-193, 2015.

[7] W. R. Elmendorf, "Automated design of program test libraries," IBM Technial Report TR 00.2089, 1970.

[8] S. Kukolj, V. Marinković, M. Popović and S. Bognár, "Selection and prioritization of test cases by combining white-box and black-box testing methods," in *2013 Third Eastern European Regional Conference on the Engineering of Computer Based Systems*, Budapest, 2013.

[9] M. J. Pramod, S. Priyadarsini, R. V. Renju, S. Sumisha and M. Prasanna, "A comparative analysis on software testing tools and strategies," *International Journal of Scientific & Technology Research,* vol. 9, no. 4, pp. 3510-3515, 2020.

[10] T. Y. Chen, P. L. Poon, S. F. Tang and Y. T. Yu, "White on black: A white-box-oriented approach for selecting black-box-generated test cases," in *Asia-Pacific Conference on Quality Software*, Hong Kong, 2000.

[11] S. Singhal, N. Jatana, B. Suri, S. Misra and L. Fernandez-Sanz, "Systematic literature review on test case selection and prioritization: A tertiary study," *Applied Sciences,* vol. 11, no. 24, 2021.

[12] C. Burnay, S. Bouraga, J. Gillain and I. J. Jureta, "What lies behind requirements? A quality assessment of statement grounds in requirements elicitation," *Software Quality Journal,* vol. 28, pp. 1615-1643, 2020.

[13] B. Vogel-Heuser, V. Karaseva, J. Folmer and I. Kirchen, "Operator knowledge inclusion in data-mining approaches for product quality assurance using cause-effect graphs," *IFAC PapersOnLine,* vol. 50, no. 1, pp. 1358-1365, 2017.

[14] W. S. Jang and R. Y. C. Kim, "Automatic generation mechanism of cause-effect graph with informal requirement specification based on the Korean language," *Applied Sciences,* vol. 11, no. 24, 2021.

[15] L. Dou and W.-D. Yang, "Design of test case for ATP speed monitoring function based on cause-effect graph," in *2019 CAA Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS)*, Xiamen, 2019.

[16] N. Oldfield, T. Yue and S. Ali, "Investigating quantum cause-effect graphs," in *2022 IEEE/ACM 3rd International Workshop on Quantum Software Engineering (Q-SE)*, Pittsburgh, 2022.

[17] D. Jagli, T. Mamatha, S. Mahalingam and N. Ojha, "The application of cause effect graph for the college placement process," *International Journal of Software Engineering & Applications (IJSEA),* vol. 3, no. 6, pp. 77-85, 2012.

[18] J. Lal and S. Singh, "From cause to effect: An empirical study of cause-effect graphing testing techniques and its test-measurement: A review," *International Journal of Computer Science and Technology,* vol. 3, no. 3, pp. 89-92, 2012.

[19] N. Gavrilović and L. Lazić, "Knowledge assessment using cause-effect graphing methods," in *The Seventh International Conference on eLearning (eLearning-2016)*, Belgrade, 2016.

[20] K. Nursimulu and R. L. Probert, "Cause-effect graphing analysis and validation of requirements," in *Proceedings of CASCON'95: Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, 1995.

[21] M. E. Khan, "Different approaches to black box testing technique for finding errors," *International Journal of Software Engineering & Applications (IJSEA),* vol. 2, no. 4, pp. 31-40, 2011.

[22] F. Huang and C. Smidts, "Causal mechanism graph - A new notation for capturing cause-effect knowledge in software dependability," *Reliability Engineering and System Safety,* vol. 158, pp. 196-212, 2017.

[23] P. R. Srivastava, P. Patel and S. Chatrola, "Cause effect graph to decision table generation," *SIGSOFT Software Engineering Notes,* vol. 34, no. 2, 2009.

[24] M. Agrawal and U. Badhera, "Approach for minimization of test cases from decision table generated from cause effect graph," *International Journal of Computer Applications,* vol. 172, no. 7, pp. 7-10, 2017.

[25] S. Weißleder and D. Sokenou, "Cause-effect graphs for test models based on UML and OCL," *Softwaretechnik-Trends,* vol. 28, no. 3, 2008.

[26] H. S. Son, R. Y. C. Kim and Y. B. Park, "Test case generation from cause-effect graph based on model transformation," in *2014 International Conference on Information Science & Applications (ICISA)*, Seoul, 2014.

[27] K.-C. Tai, A. Paradkar, H.-K. Su and M. A. Vouk, "Fault-based test generation for cause-effect graphs," in *CASCON'93: Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, 1993.

[28] A. Paradkar, K.-C. Tai and M. A. Vouk, "Specification-based testing using cause-effect graphs," *Annals of Software Engineering,* vol. 4, no. 1, pp. 133-157, 1997.

[29] I. Chung, "Modeling pairwise test generation from cause-effect graphs as a Boolean satisfiability problem," *International Journal of Contents,* vol. 10, no. 3, pp. 41-46, 2014.

[30] T. Ayav and F. Belli, "Boolean differentiation for formalizing Myers' cause-effect graph testing technique," in *2015 IEEE International Conference on Software Quality, Reliability and Security - Companion*, Vancouver, 2015.

[31] D. K. Ufuktepe, T. Ayav and F. Belli, "Test input generation from cause-effect graphs," *Software Quality Journal,* vol. 29, pp. 733-782, 2021.

[32] I. BenderRBT, "BenderRBT: The Software Quality and Testing Experts," [Online]. Available: https://www.benderrbt.com/bendersoftware.htm. [Accessed 22 September 2022].

[33] B. Bekiroglu, "A cause-effect graph software testing tool," *European Journal of Computer Science and Information Technology,* vol. 5, no. 4, pp. 11-24, 2017.

[34] D. K. Ufuktepe, "TOUCH: Test Generator for Cause Effect Graphs," [Online]. Available: https://github.com/denizkavzak/TOUCH. [Accessed 22 September 2022].

[35] W. S. Jang and R. Y. C. Kim, "Automatic cause–effect graph tool with informal Korean requirement specifications," *Applied Sciences,* vol. 12, no. 18, 2022.