A Systematic Review of Concolic Testing with Aplication of Test Criteria

Lucilia Y. Araki and Leticia M. Peres
Department of Informatics, Federal University of Paraná UFPR, R. Cel. H. dos Santos, 100, Curitiba, PR, Brazil

Keywords: Software Testing, Test Criteria, Concolic Testing, Dynamic Symbolic Execution.

Abstract: We present in this paper a systematic review, using (Biolchini et al., 2005) approach, of methods, techniques and tools regarding to concolic testing with application of test criteria. The test activity is the process of running a program with the intent of discovering defects. The search for test cases to increase the coverage of structural tests is being addressed by approaches that generate test cases using symbolic and concolic execution. Concolic testing is an effective technique for automated software testing, that aims to generate test inputs to locate failures of implementation in a program. Application of a test criterion is very important to ensure the quality of the test cases used. The number of elements exercised provides a measure of coverage that can be used to evaluate the test data set and consider the test activity to be closed.

1 INTRODUCTION

Software validation is intended to ensure that the software developed complies with the original requirements. The most commonly used validation approach in industry to improve software reliability and quality is software testing. Testing is the process of running a program with the purpose of revealing defects (Myers, 1979) and involves producing a set of tests and then running the system with these test cases.

Software testing is a fundamental software quality assurance activity. Its main purpose is to execute a program with the objective of discovering failures (Myers, 1979). The main challenge of the test is to select input values to create good test cases that are likely to reveal faults.

The problem of generating test data to achieve adequate coverage is an inherently automated activity. This automation ensures a significant impact because the generation of test data is an arduous and time-consuming task.

Several techniques have been proposed to generate input values automatically to improve software testing. Each method covers a set of test cases criteria that are used to determine the test requirements that must be satisfied by test cases. A technique of automatic generation of tests known as concolic testing (Sen and Agha, 2006) has gained importance due to its low number of false positives and high code coverage (Seo and Kim, 2014).

One limitation of current approaches is that they only generate test data for an entire program or function. Current approaches do not generate test data to cover only a single slice of code. Symbolic execution have been used for more than three decades as an effective technique for automatic generation of test data (Cadar and Sen 2013), this can be noticed through works such as (Majumdar and Sen, 2007); (Qu and Robinson, 2011); (Godboley et al., 2013); (Luckow, et al., 2016) that discuss the concolic test, but none of them aims to study the test criteria in conjunction with the concolic test.

The main goal through this work is provides an overview of a research area, identifying the quantity, types of research undertaken, available results, and the frequency of publications over time to identify trends. Another goal is to relate articles that address the concolic testing with test criteria and identify a weakness in the state of the art, related to the concolic test and test coverage. We opted to perform systematic review as a method of scientific investigation that brings along relevant studies.

This paper is organized as follows. Section 2 provides the background. Section 3 presents our methodology of systematic review. Section 3.1 details search strategy. Section 3.2 provides the study selection. Section 4 discusses the results and the analysis. Section 5 related work. Section 6 concludes and discusses future work.

Araki, L. and Peres, L.
A Systematic Review of Concolic Testing with Aplication of Test Criteria.
DOI: 10.5220/0006776301210132
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 121-132
ISBN: 978-989-758-298-1

2 BACKGROUND

2.1 Concolic Testing

Concolic testing (CONCrete + symbOLIC) (Godefroid, Klarlund and Sen, 2005), (Sen, Marinov and Agha, 2005) also known as Dynamic Symbolic Execution (DSE) (Godefroid, Klarlund and Sen, 2005), (Sen, Marinov and Agha, 2005), (Cadar et al., 2008). It is a test method for generating input data in a given program and it is executed both concretely and symbolically at the same time. In other words, test inputs are generated dynamically from a real executable program, rather than statically, from a model (Kähkönen et al., 2010). Concolic testing is considered an important vulnerability detection technique (Wang and Zeng, 2015).

Concolic testing uses concrete values as well as symbolic values for inputs and executes a program both concretely and symbolically. This is called concolic execution. Concrete execution is a part of concolic execution and constitutes the normal execution of the program. Symbolic execution part of concolic execution collects symbolic constraints over the symbolic input values at each branch point encountered along the concrete execution path.

Concolic testing starts by first executing a program under test with any concrete input values. Execution of the program can branch into different execution paths at branching statements that depends on input values. When executing such statements, concolic testing constructs a symbolic constraint that describes the possible input values causing the program to take the true or false branch at the statement in question. A path constraint is a conjunction of these symbolic constraints and new test inputs for the subsequent test runs are generated by solving them. Typically this is done by the using SMT (Satisfiability-Modulo-Theories) solvers with integer arithmetic or bit-vectors as the underlying.

2.2 Test Criteria

In designing test planning, one of the steps is design the test strategy. Test strategy comprises defining following items: the test level, that is, definition of the software development phase in that the test will be applied; test technique to be used; test criterion to be adopted; type of test to be applied in software (Crespo et al., 2004).

The application of a test criterion is very important to ensure quality of test cases used. They assist in choosing the best inputs (test data) from a generally infinite and impractical set. A test criteria is a predicate to be satisfied that establishes certain test requirements, called required elements, that must be exercised during the execution of the program, that is, executed by a test data (Rapps and Weyuker, 1985). The idea is to generate the best data that can reveal most defects with low cost (Maldonado, 1991). The number of elements exercised provides a measure of coverage that can be used to evaluate the test data set and consider the test activity to be closed. Testing criteria help the tester organize the testing process. They should be chosen according to the available testing effort. Test coverage measures are defined as a relationship between test cases required to meet the criteria and those that were performed. Measures are used to obtain information on the completeness of the integration tests.

Test criteria are classified according to the information used to derive the test requirements established from three basic techniques: structural, which uses internal structure of program to derive test data; functional, which derives test data based on functional requirements of software; and based on defects that derive test data based on common defects committed during software development.

Branch coverage is a testing requirement that all branches of the program must be exercised. A branch is the result of a decision, so branch coverage simply the measures what the decision results have been tested (Zhu; Patrick and John, 1997). That is, all branches (decision) taken from each path, true and false. This helps validate all branches in the code, ensuring that no branch leads to abnormal application behaviour.

Path coverage criterion requires that all paths from the program input to its output run during the test, though traversing all paths do not guarantee that all errors will be detected. Another problem is that programs with loops can have an infinite number of paths and therefore the criteria of all paths must be replaced by a weaker one that selects only a subset of the paths

Data flow criteria are based on the analysis of the data flow of the program to be tested. It consists in focusing the assignment of values to the variables and the later use of these values, establishing that the occurrence of a variable can be of two types: definition and use (Rapps and Weyuker, 1985; Myers, 1979).

The test criterion is used to select and evaluate the test cases in order to increase the chances of causing failures or, when this does not happen, to establish a high level of confidence in the correctness of the product.

The process of applying a structural criterion consists of analysing the implementation of the application. Next, the paths of implementation to be executed are defined. After that, the test data from program inputs domain is selected to ensure that the selected paths are executed. Then, expected the outputs for each of chosen inputs are determined. And so, the test cases are constructed. Finally, a comparison is made of the outputs obtained with the outputs expected to verify the success or the failure of each test case, and can generate a report with results for analysis.

3 METHODOLOGY OF SYSTEMATIC REVIEW

Considerable progress has been made in defining protocol models and conducting systematic reviews in Software Engineering. Even with all difficulties still existing in the Software Engineering context to perform Systematic Reviews (Mian et al., 2005), some interesting results can already be identified. We conducted a systematic review (Biolchini et al. 2005) with the objective of identifying, analysing and evaluating the reading techniques proposed in the literature. We chose following the steps of Biolchini et al. (2005) because the authors performed a systematic review in the area of software engineering.

The first step of systematic review is define the objectives, main research sources and criteria, selection, validation, inclusion and exclusion of the papers. The research string was being modified in early April 2017 to middle of April 2017 to adapt to the standard of advanced searches and to address the issues of the problem.

The research was restricted to titles, abstracts and introduction of publications. It was initialized in middle of April 2017 and finalized in late of May 2017, using ACM Digital Library (http://dl.acm.org), Periódicos Capes (https://www.periodicos.capes.gov.br), Engineering Village (https://www.engineeringvillage.com), IEEE Xplore Digital Library (http://ieeexplore.ieee.org), Science Direct (http://.sciencedirect.com) and Scopus (https://www.scopus.com) databases. Abstracts of works searched through the search string being read the topics: title, abstract and introduction. As a criterion to search and selection, restriction on publication year was that it must between 2005 and 2017 and were considered only English material. Concolic testing was first appeared in 2005 in Godefroid, Klarlund and Sen (2005) and in Sen, Marinov and Agha (2005), because of it the interval of years of publication to be between 2005 and 2017 in the search string.

3.1 Search Strategy

Search string used to find tools / methods / techniques / theory / model that apply test criteria in concolic testing or concrete test. The following search string were used to research: "(Software OR Program) AND (Concolic OR Concrete) AND (Test OR Testing) AND (Branch OR Structural) AND (Coverage OR Criteria OR Criterion)."

We are looking for papers that present initiatives to evaluate models / techniques / methods / model / theory that apply criteria in concolic or concrete test. We do not select papers that do not consider concrete/concrete test or tools that do not use this test technique.

We used 4 criteria to include papers:

- CI1: Empirical study of concolic /concrete test
- CI2: Tool / methods / techniques / theory / model concolic testing
- CI3: Methods of path generation
- CI4: Concrete test

And 2 criteria to exclude papers:

- CE1: Test tools that are not concolic or concrete
- CE2: Do not indicate how to generate test data and do not use concolic tool

3.2 Study Selection

Although search string defined was very specific to research, we still found a large number of false positives. Because the search string presents concrete, some papers related to construction were found. After we conducted a study selection criterion in those primary studies, selected papers were validated to study selection process, as shown on Table 1. Search string used to find tools / methods / techniques / theory / model.

Since initial selection of papers was made through reading of abstract and introduction of those that were accepted (87), articles were read fully and it was verified if they answered the research questions. Through initial selection it was verified that there were 25 articles that were submitted by more than one database, for example, Majumdar and Sen (2007) is presented by ACM, Engineering Village, IEEE and Scopus databases. It was decided to leave classified paper according to its original base, i.e., if the IEEE base redirected certain article to ACM database, this article is classified as belonging to ACM database and so on. New selection can be seen through column Second Selection - Selected Papers in Table 1.

The extraction of data was done with the aid of spreadsheets that contained forms with general information of the studies as: title, authors, place of publication, year of publication.

Table 1: Number of papers presented by search string.

Database	Total presented by Database	First Selection			Second Selection
		Excluded Papers	Repeated Papers	Included Papers	Selected Papers
ACM	24	10	3	10	9
CAPES	250	247	0	3	1
Engineering Village	31	16	9	6	1
IEEE	103	52	2	49	26
Science Direct	64	64	3	0	0
Scopus	44	20	8	16	8
Manual	-	-	-	3	3
Total	516	409	25	87	48

In addition, we also conducted research in Google (https://www.google.com.br) search engine performing manual technique and 3 more articles were included because they deal with the research topic, increasing the number of included papers to 48.

Papers that presented only coverage criterion without relating methodologies or concolic/concrete test tools were excluded. This phase was supported by a quality questionnaire, in which were adapted 3 questions, according Biolchini et al. (2005):

- Q1: Is used any data generation technique?
- Q2: Is any test criteria applied with concolic testing?
- Q3: Is any concolic tool used or developed?

Figure 1 shows that interest in the subject has been growing over the years. In 2016, 8 articles were published related to the subject and presents this information according to corresponding database. Table 2 shows number of publications over the years in selected database. After checking, it was found 6 articles that before had been included, but were excluded in second selection phase, because they only addressed the topic of coverage, not mentioning concolic/concrete test or concolic/concrete tools. As part of systematic review survey was conducted at end of May, it may not present some papers that are from 2017 year.

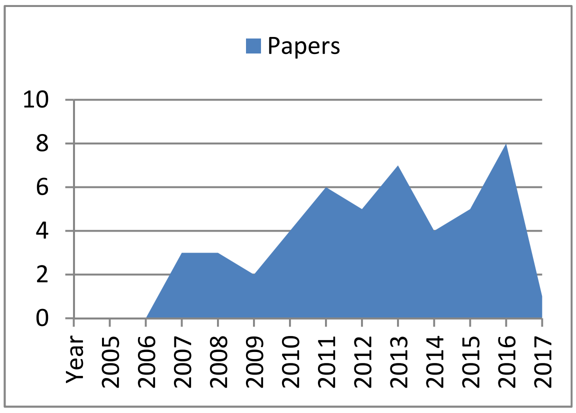

Figure 1: Selected papers per publication year.

Table 2: Selected papers per publication year.

	05	06	07	08	09	10	11	12	13	14	15	16	17
ACM	0	0	2	1	0	2	1	0	0	2	1	0	0
CAPES	0	0	0	0	0	0	1	0	0	0	0	0	0
Engineering Village	0	0	0	0	0	0	0	0	1	0	0	0	0
IEEE	0	0	1	2	1	1	2	4	5	1	3	6	0
Science Direct	0	0	0	0	0	0	0	0	0	0	0	0	0
Scopus	0	0	0	0	0	1	1	1	1	1	1	1	1
Manual	0	0	0	0	1	0	1	0	0	0	0	1	0

Figure 2 shows number of papers found in years 2005 to 2017 in each database. The IEEE portal has the largest number of publications in the search period, except in year 2010, that SCOPUS has surpassed the number of publications on the subject.

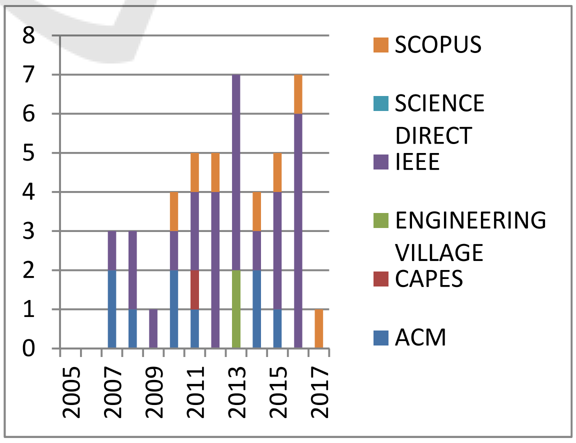

Figure 2: Year of publication of included papers.

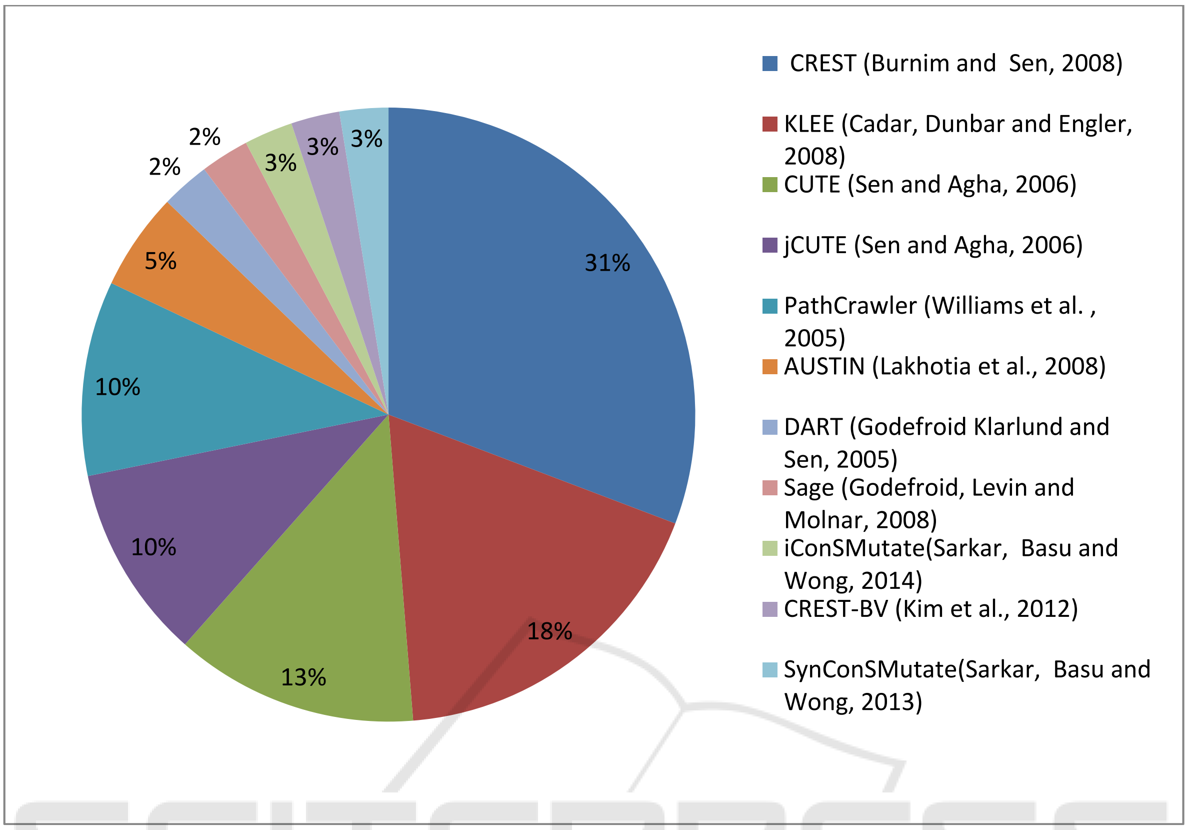

Figure 3: The most used concolic testing tools.

4 RESULTS AND DISCUSSIONS

Among the steps necessary for systematic review practice is to track the best evidence and critically assess the validity, impact, and applicability of the evidence. To achieve the objective of this review, 3 research questions were elaborated as mentioned in section 3.1.

In Table 1 we show the total of articles presented and selected by each database: ACM database obtained 9 articles selected, CAPES 1 article, Engineering Village 1 article, IEEE 26 articles, Scopus 8 and Manual Search 8 articles totalling 48 articles selected.

In order to answer Q1 we had 38 papers that use concolic testing for generating test data. Other 3 papers performing mutation test introduced by Hamlet (1977) and DeMillo et al. (1978), mutation analysis is based on the production of syntactical alterations of code under test aiming at producing semantically different program versions. The different program versions are called mutated versions as each one contains a simple syntactic change of the original code.

For Q2 the analysis shows that the most used test criterion is the branch coverage with 67% as we can see in Figure 4. Then, we have path coverage with 24%. The most significant scalability challenge in path-based testing is how to handle the exponential number of paths in program. Path explosion is mainly due to nested calls, loops and conditions (Krishnamoorthy; Michael and Loganathan, 2010). And with 6% Condition/Decision Coverage (MC/DC) that is a structural coverage criterion requiring that each condition within a decision is shown by execution to independently and correctly affect outcome of the decision (Chilenski and Steven, 1994).

For answering Q3 we have analysed selected papers regarding concolic testing tools that were used. As we can see in Figure 3 the most used concolic testing tool in this systematic review is CREST (Burnim and Sen, 2008), 38%,that is an open-source tool for C programs. Then comes CUTE (Concolic Unit Testing Engine) (Sen and Agha, 2006) and KLEE (Cadar, Dunbar and Engler, 2008) tool, the first one is a tool for C program implement concolic testing and handle input data structures and multi-threading KLEE is implemented as a modified LLVM (Low Level Virtual Machine)virtual machine targeting LLVM bytecode programs (Kim et al. 2012). jCute (Java Concolic Unit Testing Engine) (Sen and Agha, 2006) is created for Java programs. Most notably AUSTIN (AUgmented Search–based TestINg) (Lakhotia et al., 2008) cannot generate meaningful inputs for strings, void and function pointers, as well as union constructs (Lakhotia; Harman and Gross, 2010). PathCrawler (Williams et al., 2005) is an automatic generator of test case inputs to ensure structural coverage of C source code. The exhaustive exploration of the source code can also be used to demonstrate the absence of certain runtime errors or anomalies that may indicate a potential bug (or cause future maintenance problems) in any program (Kosmatov et al., 2013).

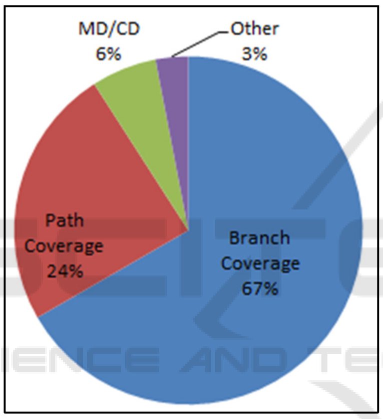

Figure 4: Testing criteria applied.

4.1 Tools and Analysis

Table 3 presents a summary of properties of the concolic testing tools (Godboley et al., 2016). Abbreviation used in Table 3 are given below with its meaning:

- "✓" supports the feature.
- "X" the tool does not support the feature.
- "P" the tool supports partially the feature.
- "-" unknown

For instance, CREST is an open source for C programs which does not support pointer, but KLEE that is also an open source for C programs does. As the goal of concolic testing is to generate test inputs which result in higher path coverage, the ability to generate these inputs depends on efficiency of its underlying constraint solver and most of them do not support float or double types, and they are unable to handle non-linear arithmetic constraints.

We can see in Table 3 that 6 of the concolic testing tools are for C: Austin, CREST, CUTE, DART and KLEE; 5 for Java: CATG, jCUTE, JDART, jFuzz and LCT; 2 for SQL language: iConSMutate and SynConSMutate; 1 for machine code: SAGE. The majority of concolic testing tools do not support variable of float/double type, pointers, native calls, non-linear arithmetic operations and function pointer. Only one of the tools stands out, jDART (Luckow, et al., 2016) a dynamic symbolic tool for Java that supports CORAL, SMTInterpol and Z3solvers and is able to handle software with constraints containing bit operations, floating point arithmetic and complex arithmetic operations. The study showed that 5 of the concolic tools are open source: CREST, CUTE, KLEE, jCUTE and DART (Table 3), only the last one is not for Linux Platform. CREST uses YICES construct solver, CUTE and DART uses LP_Solvers and KLEE uses STP.

Table 4 shows the related concolic testing tools that we separated by data test generation and coverage criteria. For 48 papers listed, 37 generate data using the concolic execution: (Qu and Robinson, 2011), (Xu et al. ,2011), (Kim et al. ,2012), (Dong et al., 2013), (Godboley, Sahani and Mohapatra, 2015), (Williams, 2010), (Godboley et al., 2017), (Su et al., 2014), (Lakhotia; McMinn and Harman, 2009), (Kim; Cha and Bae, 2013), (Baluda, 2011), (Papadakis, and Malevris, 2011), (Baluda; Denaro and Pezze, 2016), (Dutta; Godboley and Mohapatra, 2016), (Lu et al., 2016), (Jin et al., 2015), (Giantsios; Papaspyrou and Sagonas, 2017), (Huang et al., 2012), (Köroglu and Sem, 2016), (Wassermann et al., 2008), (Inkumsah and Xie, 2007), (Wang and Zeng, 2015), (Garg et al., 2013), (Burnim and Sem, 2008), (Majumdar, Saha and Wang, 2010), (Tanno et al. , 2015), (Kosmatov et al. , 2012), (Dhok;Ramanathan and Sinha, 2016), (Luckow, et al., 2016), (Jayaraman, et al., 2009) and (Kähkönen, et al., 2011); 3 use mutation: (Papadakis, and Malevris, 2011), (Sarkar, Tanmoy, Basu and Wong, 2014) and (Sarkar, Basu and Wong, 2013) . Most of the studies (38) (Qu and Robinson, 2011), (Xu et al., 2011), (Kim et al. , 2012), (Dong et al., 2013), (Godboley, Sahani and Mohapatra, 2015), (Godboley et al., 2017), (Su et al., 2014), (Lakhotia; McMinn and Harman, 2009), (Kim; Cha and Bae, 2013), (Baluda, 2011), (Papadakis, and Malevris, 2011), (Baluda; Denaro and Pezze, 2016), (Dutta; Godboley and Mohapatra, 2016), (Lu et al., 2016), (Jin et al., 2015), (Giantsios; Papaspyrou and Sagonas, 2017), (Huang et al., 2012), (Köroglu and Sem, 2016), (Majumdar and Xu, 2007), (Wassermann et al., 2008),

Table 3: Summary of properties and limitations of concolic testing tools adapted from Godboley et al. (2016).

	Language	Open Source	Platform	Construct Solvers	Support for float/double	Support for pointer	Support for native call	Support for non-linear arithmetic op.	Support for function pointer
Austin	C	-	-	-	✓	✓	-	-	✓
CATG	Java	✓	-	-	X	–	-	-	-
CREST	C	✓	Linux	YICES	X	X	X	-	X
CUTE	C	✓	Linux	LP_Solver	X	✓	X	P	X
DART	C	✓	-	LP_Solver	X	X	X	-	X
iConSMutate	SQL	–	-	-	-	-	-	-	-
jCUTE	Java	✓	Linux	-	X	-	-	X	X
jDART	Java	✓	-	CORAL, SMTInterpol, and Z3 solvers,	✓	✓	✓	✓	✓
jFuzz	Java	-	-	-	-	-	-	-	-
KLEE	C	✓	Linux	STP	X	✓	P	✓	-
LCT	Java	✓	-	SMT Boolector or YICES	✓	-	-	-	-
Path Crawler	C and C++	-	-	-	-	-	X	-	-
SAGE	Machine Code	-	Windows	Disolver	X	-	-	-	-
SynConSMutate	SQL	-	-	-	-	-	-	-	-

Table 4: Related papers of Data Test Generation, Test Criteria and Concolic testing Tools.

Papers	Data Test Generation		Coverage Criteria			Concolic testing Tools	
	Concolic	**Mutation**	**Branch Coverage**	**Path Coverage**	**MC/DC**	**Used**	**Implemented**
(Qu and Robinson, 2011)	✓		✓			DART, CUTE, jCUTE, PathCrawler, SAGE	
(Xu et al. ,2011)	✓		✓			CREST	
(Kim et al. ,2012)	✓		✓				SCORE
(Dong et al., 2013)			✓			CREST	
(Godboley, Sahani and Mohapatra, 2015)	✓		✓			jCUTE	Architectural model for branch coverage Enhancement (ABCE)
(Williams, 2010)	✓			✓		PathCrawler	
(Godboley et al., 2017)	✓		✓			jCUTE	Green Analysis of Branch Coverage Enhancement
(Su et al., 2014)	✓		✓			KLEE and CREST	
(Lakhotia; McMinn and Harman, 2009)	✓		✓			CUTE and AUSTIN	
(Kim; Cha and Bae, 2013)	✓		✓			SAGE	
(Baluda, 2011)	✓		✓				
(Papadakis, and Malevris, 2011)		✓	✓				Concolic execution tool
(Baluda; Denaro and Pezze, 2016)	✓		✓			CREST and KLEE	
(Dutta; Godboley and Mohapatra, 2016)	✓		✓			CREST	COLT
(Lu et al., 2016)	✓		✓				

Table 4: Related papers of Data Test Generation, Test Criteria and Concolic testing Tools (cont.).

Papers	Data Test Generation		Coverage Criteria			Concolic testing Tools	
	Concolic	**Mutation**	**Branch Coverage**	**Path Coverage**	**MC/DC**	**Used**	**Implemented**
(Jin et al., 2015)	✓		✓				COMEDY
(Giantsios; Papaspyrou and Sagonas, 2017)	✓		✓				
(Huang et al., 2012)	✓		✓			KLEE	CRAX
(Köroglu and Sem, 2016)	✓		✓			CREST	
(Majumdar and Xu, 2007)			✓			CUTE	CESE
(Wassermann et al., 2008)	✓		✓	✓			
(Godboley et al., 2013a)					✓	CREST	
(Baluda, 2011)			✓			CREST and KLEE	
(Inkumsah and Xie, 2007)	✓		✓				Evacon
(Wang and Zeng, 2015)	✓		✓			SAGE	CrashFinderHB
(Garg et al., 2013)	✓		✓				
(Burnim and Sem, 2008)	✓		✓			CREST	
(Seo and Kim, 2014)			✓			CREST	
(Majumdar and Sen, 2007)	✓		✓			CUTE	
(Sarkar, Tanmoy, Basu and Wong, 2014)		✓	✓			iConSMutate	
(Inkumsah and Xie, 2008)	✓		✓				Evacon
(Godboley et al., 2013b)	✓				✓	CREST	
(Kim et al., 2012)	✓		✓			CREST-BV and KLEE	
(Gao et al. , 2016)	✓		✓			KLEE	LLSPLAT
Mouzarani, Sadeghiyan and Zolfaghari, 2015)	✓		✓			KLEE	
(Baluda et al., 2010)	✓		✓				Star (Software Testing by Abstraction Refinement)
(Kosmatov et al. , 2013)	✓			✓		PathCrawler	
(Mao, Yu and Chen, 2012)			✓				
(Sarkar, Basu and Wong, 2013)		✓	✓			SynConSMutate	
(Majumdar, Saha and Wang, 2010)	✓		✓				SPLAT
(Dinges and Agha, 2014)				✓		jCUTE.	
(Tanno et al. , 2015)	✓			✓			CATG
(Kosmatov et al. , 2012)	✓			✓		PathCrawler	
(Dhok;Ramanathan and Sinha, 2016)	✓		✓				
(Luckow, et al., 2016)	✓			✓			jDart
(Jayaraman, et al., 2009)	✓			✓			jFuzz
(Kähkönen, et al., 2011)	✓			✓			LCT

(Inkumsah and Xie, 2007), (Wang and Zeng, 2015), (Garg et al., 2013), (Burnim and Sem, 2008), (Seo and Kim, 2014), (Majumdar and Sen, 2007), (Sarkar, Tanmoy, Basu and Wong, 2014), Inkumsah and Xie, 2008), (Kim et al., 2012), (Gao et al. , 2016), (Mouzarani, Sadeghiyan and Zolfaghari, 2015), (Baluda et al., 2010) (Mao, Yu and Chen, 2012), (Sarkar, Basu and Wong, 2013), (Majumdar, Saha and Wang, 2010) and (Dhok;Ramanathan and Sinha, 2016) use the branch coverage and 9 use path coverage (Williams, 2010), (Wassermann et al., 2008), (Kosmatov et al. , 2013), (Dinges and Agha, 2014), (Tanno et al. , 2015), (Kosmatov et al. , 2012), (Luckow, et al., 2016), (Jayaraman, et al., 2009) and (Kähkönen, et al., 2011); and 2 MC/ DC (Godboley et al., 2013a) and (Godboley et al., 2013b).

None of the 48 analysed articles works with the data flow test criteria, showing the necessity to study concolic testing along with this criterion.

5 RELATED WORK

For more than three decades, symbolic execution has been used in the context of software testing to generate test data with test criteria (King, 1976), (Cadar and Sen, 2013). However, symbolic execution has challenges. Several approaches have been proposed to improve problems such as path explosion, competition, complex data, constraint solvers, and integration with external libraries (Pasareanu and Visser 2009), (Godefroid 2012), Cadar and Sen 2013).

In (Majumdar and Sen, 2005) they implement a hybrid concolic testing using the CUTE tool, to achieve branch coverage for C programs. They present an algorithm that merges the application of random testing with concolic testing for exploitation in depth and width of the program.

In (Baluda et al., 2010) they proposed a technique that combines static and dynamic analysis approaches to identify infeasible program elements that can be eliminated from the structural coverage calculation to obtain accurate coverage data. The approach identifies a relevant number of impractical elements, the elements that belong statically to the code, but cannot be executed under any input condition. They implemented a prototype tool Star (Software Testing by Abstraction Refinement), built based on the Crest. The technique can also generate new test cases that execute the discovered elements, thus increasing the structural coverage of the program.

In (Majumdar and Xu, 2007) they address the problem of automatic generation of test inputs for large programs. The authors have developed the CESE tool, which implements the generation of test using symbolic grammars for C programs. The work presents a test input generation algorithm that combines the advantages of the selective and enumerative test generation and the generation of directed symbolic test.

In (Qu and Robison, 2009) identify existing concolic testing techniques and tools, identifying the languages and the platforms that it run. Then they identify the limitations of identified concolic testing techniques and tools, also study the limitations, as well as how they may affect the effectiveness (measured in branch coverage) of test suites generated in large programs.

In (Seo and Kim, 2014) they introduce the context-guided search (CGS) strategy, in that the search is guided by the set of branches. The CGS selects a branch from a new set to the next input. In addition CGS excludes irrelevant branches in the context information calculating domain. They implement the CGS strategy using two concolic test tools: CREST and CarFast.

In (Dhok's, Ramanathan and Sinha, 2016) they found an extension of the concolic testing for Java Script (JS) programs that causes the generation of a large number of inputs. The authors have proposing an approach that incorporates a type of intelligent awareness to the conventional test, thus reducing the number of inputs generated for JS programs.

The main difference between our work and the ones above is that through this work is provides an overview of a research area, identifying the quantity, types of research undertaken, and the frequency of publications over time to identify trends. We could verify the necessity of studies that combine the concolic testing and the structural criteria of test considering data flow to evaluate the quality of the applied test.

6 CONCLUSION

The systematic review was conducted by means of a review protocol that specified the methods used during the conduction of the work. The methodology defined in the protocol were necessary and sufficient to obtain the primary studies necessary to achieve the research. The systematic review proved to be an effective, though time-consuming, methodology which involved hard work in reading and analyzing primary studies in order to obtain answers to the questions raised for the research.

Through the systematic review, 48 related papers that addresses concolic testing and test criteria. Through systematic review process, we could identify a deficiency related the application of test criteria that are not branch coverage or path (Figure 4).

The problem of generating test data to achieve adequate coverage is an inherently automated activity. This automation ensures a significant impact because the generation of test data is an arduous and time-consuming task.

One limitation for elaborating this article was that this work was originated from a software engineering lectures, being executed in just a few months. Having the main challenge the definition of the search string covering the largest number of works related to the research theme within the range of 2005 to 2017.

This research allowed to verify the necessity of studies that combine concolic testing and structural criteria of test to evaluate the quality of the applied test.

Our objective with this work was to perform a study about papers that deal with concolic testing and test criteria. Thus, we can see a lack with respect to data flow criteria of software test. We intend to make an approach to generate test data to cover only test requirements selected by users considering data flow and control criteria along concolic test which will be our future work.

ACKNOWLEDGEMENTS

We thank the support from Department of Informatics from Federal University of Paraná (UFPR) and Federal Technological University of Paraná (UTFPR – Câmpus Pato Branco).

REFERENCES

Ammann, Paul, and Jeff Offutt. Introduction to software testing. *Cambridge University Press, 2016.*

Baluda, Mauro, et al. "Structural coverage of feasible code." *Proceedings of the 5th Workshop on Automation of Software Test. ACM, 2010.*

Baluda, Mauro. "Automatic structural testing with abstraction refinement and coarsening." *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011.*

Baluda, Mauro, Giovanni Denaro, and Mauro Pezze. "Bidirectional symbolic analysis for effective branch testing." *IEEE Transactions on Software Engineering 42.5 (2016): 403-426.*

Biolchini, J., Mian, P. G., Natali, A. C., Travassos, G. H. (2005) "Systematic Review in Software Engineering: Relevance and Utility", *Technical Report, PESC - COPPE/UFRJ.*

Burnim, Jacob, and Koushik Sen. "Heuristics for scalable dynamic test generation." Automated Software Engineering, *2008. ASE 2008. 23rd IEEE/ACM International Conference on. IEEE, 2008.*

Cadar, Cristian, Daniel Dunbar, and Dawson R. Engler. "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs." *OSDI. Vol. 8. 2008.*

Cadar, C. and Sen, K. Symbolic execution for software testing: three decades later. *Communications of the ACM, 56(2):82–90. 2013.*

Chilenski, John Joseph, and Steven P. Miller. "Applicability of modified condition/decision coverage to software testing." *Software Engineering Journal 9.5 (1994): 193-200.*

Crespo, Adalberto Nobiato, et al. "Uma metodologia para teste de Software no Contexto da Melhoria de Processo." *Simpósio Brasileiro de Qualidade de Software (2004): 271-285.*

DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer, 11(4), 34–41. doi:10.1109/C-M.1978.218136.*

Dhok, Monika, Murali Krishna Ramanathan, and Nishant Sinha. "Type-aware concolic testing of JavaScript programs." Software Engineering (ICSE*), 2016 IEEE/ACM 38th International Conference on. IEEE, 2016.*

Dinges, Peter, and Gul Agha. "Targeted test input generation using symbolic-concrete backward execution." *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. ACM, 2014.*

Dong, Qixing, et al. "A Search Strategy Guided by Uncovered Branches for Concolic testing." *Quality Software (QSIC), 2013 13th International Conference on. IEEE, 2013.*

Dutta, Arpita, Sangharatna Godboley, and Durga Prasad Mohapatra. "COLT: Extending Concolic testing to measure LCSAJ Coverage." *Region 10 Conference (TENCON), 2016 IEEE. IEEE, 2016.*

Gao, Min, et al. "LLSPLAT: Improving Concolic testing by Bounded Model Checking." Source Code Analysis and Manipulation (SCAM), *2016 IEEE 16th International Working Conference on. IEEE, 2016.*

Garg, Pranav, et al. "Feedback-directed unit test generation for C/C++ using concolic execution." *Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.*

Giantsios, Aggelos, Nikolaos Papaspyrou, and Konstantinos Sagonas" Concolic testing for functional languages." *Science of Computer Programming*

(2017).

Godboley, Sangharatna, et al. "Enhanced modified condition/decision coverage using exclusive-nor code transformer." Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), *2013 International Multi-Conference on. IEEE, 2013 (a).*

Godboley, Sangharatna, et al. "Increase in modified condition/decision coverage using program code transformer." *Advance Computing Conference (IACC), 2013 IEEE 3rd International. IEEE, 2013 (b).*

Godboley, Sangharatna, et al. "Making a concolic testinger achieve increased MC/DC." *Innovations in Systems and Software Engineering 12.4 (2016): 319-332.*

Godboley, Sangharatna, Arun Sahani, and Durga Prasad Mohapatra. "ABCE: a novel framework for improved branch coverage analysis." *Procedia Computer Science 62 (2015): 266-273.*

Godboley, Sangharatna, et al. "An Automated Analysis of the Branch Coverage and Energy Consumption Using Concolic testing." *Arabian Journal for Science and Engineering 42.2 (2017): 619-637.*

Godefroid, Patrice, Nils Klarlund, and Koushik Sen. "DART: directed automated random testing*." ACM Sigplan Notices. Vol. 40. No. 6. ACM, 2005.*

Godefroid, Patrice, Michael Y. Levin, and David A. Molnar. "Automated whitebox fuzz testing*." NDSS. Vol. 8. 2008.*

Hamlet, R. G. (1977). Testing programs with the aid of a compiler. *IEEE Transaction on Software Engineering, 3(4), 279–290. doi:10.1109/TSE.1977.231145.*

Huang, Shih-Kun, et al. "Crax: Software crash analysis for automatic exploit generation by modeling attacks as symbolic continuations." Software Security and Reliability (SERE), *2012 IEEE Sixth International Conference on. IEEE, 2012.*

Inkumsah, Kobi, and Tao Xie. "Evacon: A framework for integrating evolutionary and concolic testing for object-oriented programs." *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. ACM, 2007.*

Inkumsah, Kobi, and Tao Xie. "Improving structural testing of object-oriented programs via integrating evolutionary testing and symbolic execution." *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2008.*

Jayaraman, Karthick, et al. "jFuzz: A concolic whitebox fuzzer for Java." (2009).

Jin, Hao, et al. "Concolic metamorphic debugging." Computer Software and Applications Conference (COMPSAC), *2015 IEEE 39th Annual. Vol. 2. IEEE, 2015.*

Kähkönen, K., Kindermann, R., Heljanko, K., & Niemelä, I. Experimental comparison of concolic and random testing for Java Card applets. In: *Model Checking Software. Springer Berlin Heidelberg, 2010. p. 22-39.*

Kähkönen, Kari, et al. "LCT: An open source concolic testing tool for Java programs." *Proceedings of the 6th Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE). 2011.*

Kim, Su Yong, Sungdeok Cha, and Doo-Hwan Bae. "Automatic and lightweight grammar generation for fuzz testing." *Computers & Security 36 (2013): 1-11.*

Kim, Yunho, et al. "Industrial application of concolic testing approach: A case study on libexif by using CREST-BV and KLEE." *Software Engineering (ICSE), 2012 34th International Conference on. IEEE, 2012.*

Kosmatov, Nikolai, and Nicky Williams. "Tutorial on automated structural testing with pathcrawler." *Proceedings of the 6th international conference on Tests and Proofs. Springer-Verlag, 2012.*

Kosmatov, Nikolai, et al. "Structural Unit Testing as a Service with PathCrawler-online.com." Service Oriented System Engineering (SOSE), *2013 IEEE 7th International Symposium on. IEEE, 2013.*

Köroglu, Yavuz, and Alper Sen. "Design of a Modified Concolic testing Algorithm with Smaller Constraints." CSTVA@ ISSTA. 2016.

Krishnamoorthy, Saparya, Michael S. Hsiao, and Loganathan Lingappan. "Tackling the path explosion problem in symbolic execution-driven test generation for programs." Test Symposium (ATS), *2010 19th IEEE Asian. IEEE, 2010.*

Lakhotia, Kiran, Mark Harman, and Phil McMinn. "Handling dynamic data structures in search based testing." *Proceedings of the 10th annual conference on Genetic and evolutionary computation. ACM, 2008.*

Lakhotia, Kiran, Phil McMinn, and Mark Harman. "Automated test data generation for coverage: Haven't we solved this problem yet?." Testing: Academic and Industrial Conference-Practice and Research Techniques, *2009. TAIC PART'09.. IEEE, 2009.*

Lakhotia, Kiran, Mark Harman, and Hamilton Gross. "AUSTIN: A tool for search based software testing for the C language and its evaluation on deployed automotive systems." Search Based Software Engineering (SSBSE), *2010 Second International Symposium on. IEEE, 2010.*

Lu, Jiawen, et al. "Complexity analysis and comparison of test paths based on DSE." Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), *2016 17th IEEE/ACIS International Conference on. IEEE, 2016.*

Luckow, Kasper, et al. "JDart: A dynamic symbolic analysis framework." *International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2016*

Maldonado, J. C. Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software. Orientador: Prof. Dr. Mário Jino (Tese) ICMC/USP. São Carlos/SP, 1991.

Majumdar, Rupak, and Ru-Gang Xu. "Directed test generation using symbolic grammars." *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. ACM, 2007.*

Majumdar, Rupak, and Koushik Sen. "Hybrid concolic testing." Software Engineering, *2007. ICSE 2007. 29th International Conference on. IEEE, 2007.*

Majumdar, Rupak, Indranil Saha, and Zilong Wang. "Systematic testing for control applications." Formal

Methods and Models for Codesign (MEMOCODE), *2010 8th IEEE/ACM International Conference on. IEEE, 2010.*

Mao, Chengying, Xinxin Yu, and Jifu Chen. "Swarm intelligence-based test data generation for structural testing." Computer and Information Science (ICIS), *2012 IEEE/ACIS 11th International Conference on. IEEE, 2012.*

Mian, Paula; Conte, Tayana Uchoa; Natali, Ana Candida Cruz; Biolchini, Jorge; Travassos, G. H.. Lessons Learned on Applying Systematic Reviews to Software Engineering. In: *WSESE2005- Workshop Series in Empirical Software Engineering, 2005, Oulu. Proceedings of the 3rd International Workshop "Guidelines For Empirical Work" in the Workshop Series on Empirical Software Engineering 2005. Kaiserslautern: Fraunhofer Center, 2005. v. 1. p. 1-6.*

Mouzarani, Maryam, Babak Sadeghiyan, and Mohammad Zolfaghari. "Smart fuzzing method for detecting stack-based buffer overflow in binary codes." *IET Software 10.4 (2016): 96-107.*

Myers, G. J. Art of Software Testing. John Wiley & Sons, *Inc., New York, NY, USA, 1979.*

Papadakis, Mike, and Nicos Malevris. "Automatically performing weak mutation with the aid of symbolic execution, concolic testing and search-based testing." *Software Quality Journal 19.4 (2011): 691.*

Qu, Xiao, and Brian Robinson. "A case study of concolic testing tools and their limitations." Empirical Software Engineering and Measurement (ESEM*), 2011 International Symposium on. IEEE, 2011.*

Rapps, S., e Weyuker, E. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering 11, 04 (April), 367–375, 1985.*

Sarkar, Tanmoy, Samik Basu, and Johnny Wong. "Synconsmutate: Concolic testing of database applications via synthetic data guided by sql mutants." Information Technology: New Generations (ITNG), *2013 Tenth International Conference on. IEEE, 2013.*

Sarkar, Tanmoy, Samik Basu, and Johnny Wong. "iConSMutate: Concolic testing of Database Applications Using Existing Database States Guided by SQL Mutants." Information Technology: New Generations (ITNG), *2014 11th International Conference on. IEEE, 2014.*

Sen, Koushik, Darko Marinov, and Gul Agha. "CUTE: a concolic unit testing engine for C." *ACM SIGSOFT Software Engineering Notes. Vol. 30. No. 5. ACM, 2005.*

Sen, Koushik, and Gul Agha. "CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools." *CAV. Vol. 6. 2006.*

Seo, Hyunmin, and Sunghun Kim. "How we get there: A context-guided search strategy in concolic testing." *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014.*

Su, Ting, et al. "Automated coverage-driven test data generation using dynamic symbolic execution." Software Security and Reliability*, 2014 Eighth International Conference on. IEEE, 2014.*

Tanno, Haruto, et al. "TesMa and CATG: automated test generation tools for models of enterprise applications." *Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 2015.*

Xu, Zhihong, et al. "A hybrid directed test suite augmentation technique." Software Reliability Engineering (ISSRE), *2011 IEEE 22nd International Symposium on. IEEE, 2011.*

Wang, W. and Zeng, Q. Evaluating Initial Inputs for Concolic testing. In: Theoretical Aspects of Software Engineering (TASE*), 2015 International Symposium on. IEEE, 2015. p. 47-54.*

Wassermann, Gary, et al. "Dynamic test input generation for web applications." *Proceedings of the 2008 international symposium on Software testing and analysis. ACM, 2008.*

Williams, Nicky, et al. "PathCrawler: Automatic Generation of Path Tests by Combining Static and Dynamic Analysis." *EDCC. Vol. 3463. 2005.*

Williams, Nicky. "Abstract path testing with PathCrawler." *Proceedings of the 5th Workshop on Automation of Software Test. ACM, 2010.*

Zhu, Hong, Patrick AV Hall, and John HR May. "Software unit test coverage and adequacy." *ACM computing surveys (csur) 29.4 (1997): 366-427.*