Lecture Notes in Mechanical Engineering

Vijay Kumar Gupta
Prabhakar V. Varde
P. K. Kankar
Narendra Joshi *Editors*

Reliability and Risk Assessment in Engineering

Proceedings of INCRS 2018

Springer

Editors
Vijay Kumar Gupta
Discipline of Mechanical Engineering
PDPM Indian Institute
of Information Technology, Design
and Manufacturing, Jabalpur
Jabalpur, Madhya Pradesh, India

P. K. Kankar
Discipline of Mechanical Engineering
Indian Institute of Technology Indore
Indore, Madhya Pradesh, India

Prabhakar V. Varde
Reactor Group
Bhabha Atomic Research Centre
Mumbai, Maharashtra, India

Narendra Joshi
Research Reactor Services Division
Bhabha Atomic Research Centre
Mumbai, Maharashtra, India

ISSN 2195-4356 ISSN 2195-4364 (electronic)
Lecture Notes in Mechanical Engineering
ISBN 978-981-15-3745-5 ISBN 978-981-15-3746-2 (eBook)
https://doi.org/10.1007/978-981-15-3746-2

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Contents

Optimization and Machine Learning Techniques for Industrial Applications

Editors and Contributors

About the Editors

Dr. Vijay Kumar Gupta is currently working as a Professor in the discipline of Mechanical Engineering at PDPM-Indian Institute of Information Technology, Design and Manufacturing, Jabalpur. He has more than 25 years of teaching and research experience. He obtained his PhD in Mechanical Engineering from Indian Institute of Technology Bombay, India. His research interests include smart structure, vibration, design, reliability, finite element analysis, mechatronics and robotics, etc. He has published more than 30 papers in refereed journals and conferences and one book. He is recipient of ISAME K. Suryanarayan Rao Memorial Senior Student Award for R&D in Smart Technology for year 2003. He is a member of ASME, SPIE, IEEE, SRESA and other professional bodies.

Dr. Prabhakar V. Varde was Associate Director of Reactor Group, Bhabha Atomic Research Center, Mumbai, India and is Senior Professor, Homi Bhabha National Institute, Mumbai. He completed his PhD from IIT Bombay in 1996. He worked as post-doctoral fellow at Korea Atomic Energy Research Institute, South Korea in 2002 and was visiting professor at CALCE, University of Maryland, USA in 2012. His research interests are development of prognostic models for nuclear plants components in general and electronic components in particular. He worked as a consultant with several international organizations such as OECD/NEA (WGRISK) Paris, International Atomic Energy Agency, Vienna, etc. He is a chief editor for International Journal of Life Cycle Reliability and Safety Engineering. He has over 250 publications, which includes 10 conference proceedings/books, in national and international publications.

Dr. P. K. Kankar is an Associate Professor in the discipline of Mechanical Engineering, Indian Institute of Technology Indore. He has over 14 years of teaching and research experience. He obtained his PhD from Indian Institute of Technology Roorkee, India. His research interests include vibration, design,

condition monitoring of mechanical components, nonlinear dynamics, soft computing, etc. He has published more than 100 papers in refereed journals and conferences. His work has been cited more than 1500 times. He also served as a guest editor of special issues of various journals of national and international repute. He is a member of professional bodies like American Society of Mechanical Engineers, Society for Reliability and Safety (SRESA), Tribology Society of India and International Institute of Acoustics & Vibration (IIAV).

Mr. Narendra Joshi is working as Secretary and founder member of the Society for Reliability & Safety, and is the Managing Editor of SRESA-Springer International Journal on Life Cycle Reliability and Safety Engineering. He has over 20 publications to his credit in journals and conferences. Mr. Joshi is currently looking after the activities of Human Resource Development, Simulator Training, Root Cause Analysis and safety documentation in research reactors at the Bhabha Atomic Research Centre, Mumbai. He has also worked in Operation and Maintenance of research reactors for 13 years. He was involved in preparation of Probabilistic Risk Assessment of research reactors at Trombay and Risk Informed In-Service Inspection.

Contributors

Pavan Agrawal Madhav Institute of Technology & Science, Gwalior, MP, India

Suhail Ahmad Indian Institute of Technology Delhi, New Delhi, India

A. Alagesan A. D. Agricultural College & Research Institute, Tamil Nadu Agricultural University, Tiruchirappalli, India

M. Amarnath Department of Mechanical Engineering, PDPM Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India

S. Ananthi Anna University Chennai—BIT Campus, Tiruchirappalli, India

Atul B. Andhare Department of Mechanical Engineering, VNIT, Nagpur, India

Prashant Athanker Department of Mechanical Engineering, MNIT, Jaipur, Rajasthan, India

Mradul Awasthi Indian Institute of Technology Delhi, New Delhi, India

Saurabh Bhalerao Rajarambapu Institute of Technology, Rajaramnagar, India

Sharad Bhardwaj Department of Mechanical Engineering, ABES Engineering College, Ghaziabad, U.P., India

Mahesh Bhiwapurkar O. P. Jindal University, Raigarh, India

Neeraj Kumar Bhoi Department of Mechanical Engineering, PDPM IIITDM Jabalpur, Jabalpur, India

Abhishek Chavan Rajarambapu Institute of Technology, Rajaramnagar, India

N. J. Chotai Department of Mechanical Engineering, Marwadi Education Foundation Group of Institutions, Rajkot, India

A. A. Darji Department of Mechanical Engineering, C. U. Shah University, Surendranagar, Gujarat, India;
Department of Mechanical Engineering, LDRP Institute of Technology, Gandhinagar, India

P. H. Darji Department of Mechanical Engineering, C. U. Shah University, Surendranagar, Gujarat, India

N. Dave Department of Mechanical Engineering, PDPU, Gandhinagar, India

Vipul Dave Pandit Deendayal Petroleum University, Gandhinagar, India

V. Deenadayalan Anna University, Tiruchirapalli, India

J. V. Desai Paher University, Udaipur, Rajasthan, India;
Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

Nishant Doshi Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Pratiksha Gautam Department of Computer Science & Engineering, Amity School of Engineering and Technology, Amity University Madhya Pradesh, Gwalior, Madhya Pradesh, India

A. B. Gholap Marathwada Mitra Mandal's College of Engineering Pune, Pune, India

Ashish Gupta Discipline of Electrical Engineering, NIT Agartala, Agartala, Tripura, India

Karunesh Kumar Gupta Birla Institute of Technology and Science, Pilani, Rajasthan, India

S. P. Harsha Indian Institute of Technology Roorkee, Roorkee, India

Ankush C. Jahagirdar Birla Institute of Technology and Science, Pilani, Rajasthan, India

Arihant Jain Research Reactor Services Division, Reactor Group, Bhabha Atomic Research Centre, Trombay, India

Arvind Kumar Jain Discipline of Electrical Engineering, NIT Agartala, Agartala, Tripura, India

Pratesh Jayaswal Madhav Institute of Technology & Science, Gwalior, MP, India

M. D. Jaybhaye Department of Production Engineering and Industrial Management, College of Engineering Pune, Pune, India

N. S. Joshi Research Reactor Services Division, Reactor Group, Bhabha Atomic Research Centre, Trombay, India

U. Kagathara Department of Mechanical Engineering, PDPU, Gandhinagar, India

P. K. Kankar Discipline of Mechanical Engineering, Indian Institute of Technology Indore, Indore, India;
PDPM Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India

M. Karthick AERB-Safety Research Institute, Kalpakkam, India;
College of Engineering, Anna University, Chennai, India

R. Karthikeyan Anna University Chennai—BIT Campus, Tiruchirappalli, India

Abhay Khalatkar Department of Mechanical Engineering, G. H. Raisoni College of Engineering, Nagpur, India

M. B. Kiran Department of Industrial Engineering, PDPU, Gandhinagar, India

K. L. A. Khan Department of Mechanical Engineering, KIET Group of Institutions, Muradnagar, Ghaziabad, U.P., India

Basavraj S. Kothavale Department of Mechanical Engineering, MAEER's MIT College of Engineering, Kothrud, Pune, Maharashtra, India

Harshwardhan Kulkarni Indian Institute of Technology, Mumbai, India

Prabhat Kumar Bharatiya Nabhikiya Vidyut Nigam (BHAVINI), Kalpakkam, Tamilnadu, India

Sachin Kumar Department of Mechanical Engineering, Indian Institute of Technology Ropar, Rupnagar, India

Devesh Kumar Lal School of Computer Science & IT, Devi Ahilya University, Indore, India

Rashmi Lalwani Manipal University Jaipur, Jaipur, Rajasthan, India

V. B. Lalwani Department of Mechanical Engineering, LDRP-ITR, Gandhinagar, Gujarat, India

A. R. Majmudar Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

Ashish Malik Department of Mechanical Engineering, ABES Engineering College, Ghaziabad, U.P., India

D. P. Mavani Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

S. P. Mehta Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

C. J. Mevada Department of Mechanical Engineering, LDRP Institute of Technology, Gandhinagar, India

R. K. Mishra CEMILAC, Defence R&D Organization, Bangalore, India

M. Monica Bhavani Department of CSE, Anna University, BIT Campus, Trichy, Tamilnadu, India

D. H. Pandya Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

Adesh Paramane Rajarambapu Institute of Technology, Rajaramnagar, India

Anand Parey Department of Mechanical Engineering, Indian Institute of Technology Indore, Indore, India

S. Parsana Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Chintan Patel Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Jaydeep Patel Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Rajesh Patel Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, India

Rhythm Patel Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

S. A. Patel Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

Shivdayal Patel Discipline of Mechanical Engineering, Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India

Vivek Patel Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Rajkumar Bhimgonda Patil Center for Advanced Life Cycle Engineering (CALCE), University of Maryland, College Park, USA;
Department of Mechanical Engineering, Annasaheb Dange College of Engineering and Technology, Ashta, Sangli, Maharashtra, India

T. Paul Robert College of Engineering, Anna University, Chennai, India

N. B. Prajapati Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

Saurabh Pratap Department of Mechanical Engineering, PDPM IIITDM Jabalpur, Jabalpur, India

N. Radadia Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

B. D. Raja INDUS University, Ahmedabad, Gujarat, India

T. Ramprabhu Materials and Manufacturing Processes, Palakkad, Kerala, India

K. S. Ramprasad Sathyabama Institute of Science and Technology, Chennai, India

Arun Kumar Rouniyar Department of Mechanical Engineering, Motilal Nehru National Institute of Technology, Allahabad, Uttar Pradesh, India

Neelesh Kumar Sahu Department of Mechanical Engineering, Medi-Caps University, Indore, India

D. Saikiran Department of Mechanical Engineering, Motilal Nehru National Institute of Technology, Allahabad, Uttar Pradesh, India

Hemraj Saini Jaypee University Information Technology, Waknaghat, Solan, Himachal Pradesh, India

Poonam Savsani Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Vimal Savsani Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Prakash Kumar Sen O.P. Jindal University, Raigarh, India

C. Senthil Kumar AERB-Safety Research Institute, Kalpakkam, India

Pragya Shandilya Department of Mechanical Engineering, Motilal Nehru National Institute of Technology, Allahabad, Uttar Pradesh, India

Aditya Sharma Department of Mechanical Engineering, Faculty of Engineering, Dayalbagh Educational Institute, Dayalbagh, Agra, India

Vikas Sharma Department of Mechanical Mechatronics Engineering, The LNM Institute of Information Technology, Jaipur, India

Mohak Sheth Canadore college, North Bay, Canada

N. Sheth Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

Parnika Shrivastava Mechanical Engineering Department, National Institute of Technology Hamirpur, Hamirpur, India

Amit Singh Department of Mechanical Engineering, MNIT, Jaipur, Rajasthan, India

Harpreet Singh Department of Mechanical Engineering, PDPM IIITDM Jabalpur, Jabalpur, India

Suneet Singh Indian Institute of Technology, Mumbai, India

R. B. Solanki Atomic Energy Regulatory Board, Mumbai, India

Ugrasen Suman School of Computer Science & IT, Devi Ahilya University, Indore, India

Puneet Tandon Mechanical Engineering Department, PDPM Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India

V. K. Tapas Research Reactor Services Division, Reactor Group, Bhabha Atomic Research Centre, Trombay, India

H. M. Trivedi Department of Mechanical Engineering, LDRP Institute of Technology, Gandhinagar, India

N. Upadhyay Department of Mechanical Engineering, Indian Institute of Technology Delhi, New Delhi, India

O. G. Vaghela Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

P. Vaishnavi Department of Computer Applications, Anna University, Chennai—BIT Campus, Tiruchirappalli, India

S. Vaishnavi Department of Computer Applications, Anna University, Chennai—BIT Campus, Tiruchirappalli, India

V. Vakharia Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, India

A. Valarmathi Department of MCA, Anna University, BIT Campus, Trichy, Tamilnadu, India

P. V. Varde Bhabha Atomic Research Center, Mumbai, India;
Research Reactor Services Division, Reactor Group, Bhabha Atomic Research Centre, Trombay, India

Akash Vasani Department of Mechanical Engineering, Pandit Deendayal Petroleum University, Gandhinagar, Gujarat, India

A. K. Verma Western Norway University of Aplied Sciences, Haugesund, Norway

Jay Govind Verma PDPM Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India

G. Vidhyalakshmi Department of Computer Applications, Anna University, Chennai—BIT Campus, Tiruchirappalli, India

Anurag Vijaywargiya O.P. Jindal University, Raigarh, India

Venkata Ravi Vusa Discipline of Mechanical Engineering, Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India

H. K. Yadav Department of Mechanical Engineering, LDRP Institute of Technology and Research, Gandhinagar, Gujarat, India

Mutation Testing-Based Evaluation Framework for Evaluating Software Clone Detection Tools

Pratiksha Gautam and Hemraj Saini

Abstract Mutation testing has become a prominent research area in the past few decades. The mutation testing has been basically used in the testing society. It is a type of software testing where we mutate (small change, modification in the program) source code using mutant operators by introducing potential new bugs in the program code without changing its behavior. Analogously, mutant operators generate new clones by copy/paste editing activities. However, several software clone detection tools and techniques have been introduced by numerous scientists and a large number of tools comprises for a perceivable evaluation. Moreover, there have been a lot of efforts to empirically assess and analyze variant state-of-the-art tools. The current abstraction exhibits that various aspects that could leverage the legitimacy of the outcome of such assessment have been roughly anticipated due to lack of legitimized software clone benchmark. In this paper, we present a mutation testing-based automatic evaluation structure for valuating software clone detection tools and techniques. The proposed framework uses the edit-based taxonomy of mutation operator for assessing code clone detection tools. The proposed structure injects software clones in the source code automatically, and after that, we evaluate clone detection tools. The clone detection tools are evaluated on the basis of precision (number of corrected clones) and recall (total number of clones). We visualize that such a framework will present a valuable augmentation to the research community.

Keywords Mutation analysis · Software clone · Mutation techniques · Mutation operators

P. Gautam (✉)
Department of Computer Science & Engineering, Amity School of Engineering and Technology, Amity University Madhya Pradesh, Gwalior, Madhya Pradesh, India
e-mail: pratikshamtech20@gmail.com

H. Saini
Jaypee University Information Technology, Waknaghat, Solan, Himachal Pradesh, India
e-mail: hemraj1977@yahoo.co.in

V. K. Gupta et al. (eds.), *Reliability and Risk Assessment in Engineering*,
Lecture Notes in Mechanical Engineering,
https://doi.org/10.1007/978-981-15-3746-2_3

1 Introduction

The experiential estimation of testing techniques plays a vital role in software testing as well as in software clone detection research. One frequent usage is inserting flaws, either manually or by using mutation operators [1]. Generally, empirical evaluations are used to actuate two or more approaches, which are exclusive for complying with some detection-related activity. Testing experiments usually entail a set of subject programs with known faults or errors. Furthermore, in software clone detection, it also requires some faulty version or duplicated code, to evaluate software clone detection approaches. Many researchers consequently have taken the methods of introducing bugs into the correct program to generate faulty variants. These bugs can be pioneered by hand or generated automatically through a faulty version of program text. Typically, we view an automatically generated version as an outcome of employing some editing activities in the source code. However, these editing activities are performed according to well-defined rules which are called mutant operators, and the resulting flawed variants are called mutation generation [2]. The mutation operators are identically used to generate potential bugs in software code clones so as to change the original source code [3, 4]. Reusing source code segment by copying and pasting is a common practice in software development. As a consequence, similar copied code fragments are called software clones, and the process is called software cloning [3]. Earlier study exhibits that a major portion (20–59) of program code in the software code was duplicated [5–10]. An error detected in one segment of code, and then, all segments of source code should be checked for the same error [11]. Copied code can also considerably augment the effort to be thorough when intensifying or complying source code [8]. However, several software clone detection approaches have been anticipated, and there have been a number of comparisons and valuation abstractions relating them in different contexts [12–16]. Nonetheless, it is ambitious to analyze different software clone detection tools, due to software clone detection techniques having specific features, and thus, these researches present considerable contributions to the software clone detection research [17]. Typically, insufficient assessment is aggravated as there are no universal evaluation benchmarks. It is thorny to find such a general norm as each technique has its own tunable features and is designed for distinctive reasons.

In this paper, we present a layout of mutation testing-based evaluation framework. The main objective of our abstraction is to draft a mutation-based evaluation framework which has been used to assess clone detection tools and techniques. The proposed framework is still under implementation and is manifested for quantifying and expanding various software clone detection tools. In this paper, we start off by abstracting the basic introduction of software code cloning and mutation testing from which an editing taxonomy of mutant operators can be derived. Moreover, on the basis of these mutation operators, we assert a framework for valuating code clone detection tools.

2 Background

Copying source code fragments from one section of source text and reusing them into another section with or without minor adaptation are frequent activities in software development process [5]. Software clones can be classified on the basis of their attributes as textual similarity and functional similarity. Further, the textual similarity-based clones are also categorized into three types such as (1) type 1(identical code segment except some variations in whitespace and comments), (2) type 2 (structurally similar code segments except some modifications in variable names, white spaces and comments) (3) type 3 (similar code segments with further alterations as statements deleted, inserted). Functional similarity-based clones are type 4 which have similar functionality but different structures.

3 Mutation Testing Overview

Mutation is a type of white-box testing, which is mainly used for unit testing where we transform (mutate) certain statements of a program text and verify if the test cases are able to find the fault. The adaptation in a mutant program is kept enormously small, so that it does not change the overall intention of the program. The perception of mutation testing is to introduce a syntactic transformation into the original source code, to create a flawed variant (termed as a mutant) as stated by well-defined rules (mutant operators) [18]. The mutation testing first time emerged in the 1970s by [19], in a class term paper. The first research paper was published by the authors 1 [20–22]. The author [23] conducted a survey work on the subarea of mutation testing, which was a weak survey, while the firm mutation provided by [24–27] and they gave an introductory chapter on mutation testing in their books. In 2000, [28] carried out a survey work on mutation testing as well as they recapitulated the history of mutation testing and presented an outline of the existing optimization techniques for mutation testing. Authors [29] defined mutation testing as a fault-based testing, which presents a testing standard known as "mutationcompetence score." The potency of test set can be quantified by using this score in terms of its facility to detect flaws. A recent review was accomplished [30]. They provide a methodical literature survey on application perception of mutation testing. The mutation testing approaches are classified into three types, and mutant operators can be assorted into two types which are shown in Fig. 1. Figure 1a depicts the classification of mutation testing techniques such as (1) the value mutation testing, which alters the key parameter's value as well types of mutation operators. (2) The decision mutation testing, modification in the control statement of the source code using and, or, not. (3) The statement mutation testing method, reorder statements of the program. Figure 1b demonstrates types of mutant operators. The mutant operators can be categorized as traditional mutation operators and class mutant operators. (1) The traditional mutant operators are further classified as arithmetic, logical, relational, conditional, etc. (2) The class mutant operator as

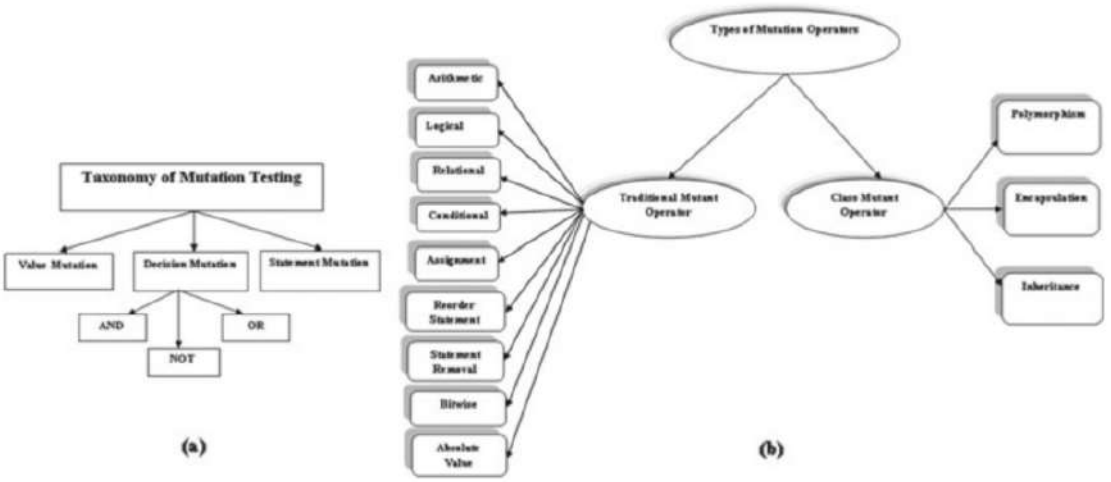

Fig. 1 Taxonomy of **a** mutation testing, **b** mutation operators

inheritance (delete a hidden variable), polymorphism (to check methods have the same name) and encapsulation (transforms, delete and insert instance variables for generating mutants).

The mutation testing is used to create flawed variants from an original program. Further, we used mutation operators for creating various types of software clones which are shown in Fig. 2. Figure 2 shows the mutation operators for software clones. Software clones can be classified into four types. Type-1 clones can be generated by using several mutation operators as mCW removes white space, mCW—changes in blank spaces, mCC—changes in comments, mCF—changes in formatting and mCNWs—changes in new line spaces. Type-2 clones can be generated by using mutation operators as mARV—arbitrary renaming variables, mRPE—replacement of parameters with expressions, mARDT—renaming data types and mARL—arbitrary renaming of literals. Type-3 clones can be generated by mSDL—small deletions within a line, mSIL—small insertions within a line, mMLs—modification in the whole line, mAOR—changes in arithmetic operators and mDSV—variation data statements. Type-4 clones can be created by using mROS—reorder the statements, mCR—replaces one type of control statement with another type of control statement, mCSS—constant substitution.

4 Related Work

The basic idea behind mutation testing is creating exactly similar variants from original program. Each mutant contains at least one artificial modification [31]. There is no mutation testing-based assessment structure presented in the literature; however, there are a number of experiments that contrast and valuate software clone detection

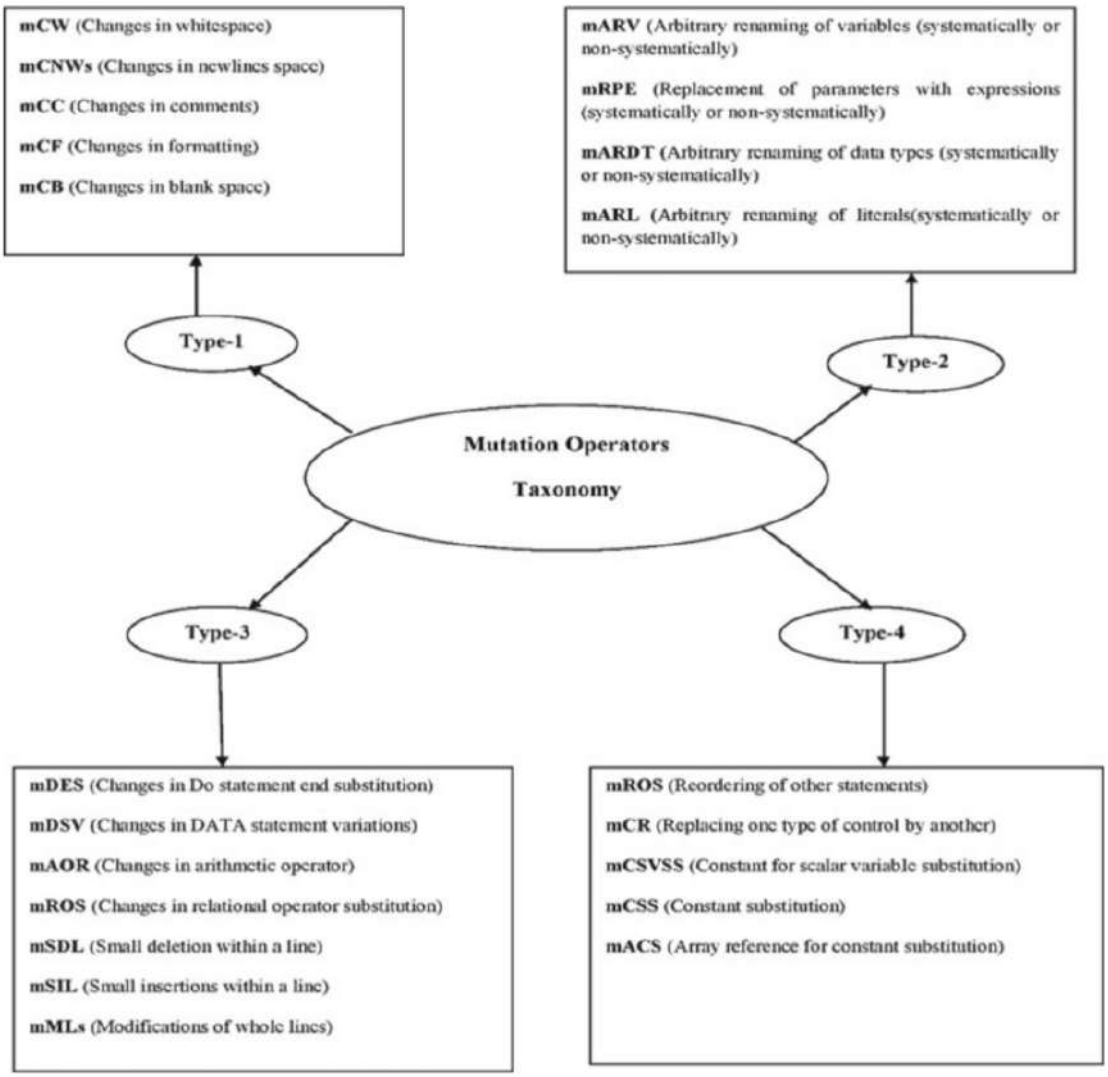

Fig. 2 Taxonomy of mutation operators for software clones

techniques/tools. We present an outline of the existing tool assessment experimentation from the literature. The first experiment was accomplished [14] on two plagiarism detection tools and three state-of-the-arts clone detection tools to evaluate their performance. The first thing they did was verifying the entire duplicated code clones using all the techniques of their experimentation. Various techniques were then compared against various parameters by a human oracle, which in turn uses several metrics that are responsible for measuring different facets of the detected code clones. Though they efficiently identified all the candidate clones, the main constraints of this case study are in terms of size and system quality. The primary objective of their study was to help in a preventive maintenance task which had leverage in validating the candidate clone. Authors have [12] accomplished tool comparison which was invented after considering the limitations of [14] experiment. Further, the authors [12] conducted this experiment with similar three software clone detection tools which are used [14] in their experiment, and they also used three additional clone detection tools in their experiments. The system software [14] had used a diverse set,

accounting to four Java and four C. Moreover, when the authors' [14] studies were taken under consideration while validating the candidate clones, a human oracle was then used for the same. Although being the most comprehensive study till now, the clone candidate being oracled was only a small proportion considering that other factors might have affected the results [17]. However, authors [15] extended this study with a prototype application of several tools, but without addressing anything to conquer the shortcomings of Bellon et al.'s study. The three envoy software clone detection techniques valuated [16], and they presented relative fallouts in perspective of portability, types of replication addressed, scalability, a number of false test matches and the number of ineffectual matches. Nevertheless, the little size cases and paradigm implementation were used by them instead of the actual tools. They intended to conclude appropriateness of the detection technique for a meticulous task as a substitute to the perceptible evaluation of the detection approaches. The researchers [13] carried out an interesting study; they evaluated numerous clone detection techniques in respect of detecting crosscutting apprehensions.

Recently, various researchers proposed number of techniques for mutation testing. Author [32] proposed an efficient mutation testing framework for multi-threaded code which can reduce time required for mutation testing of multi-threaded code. They proposed a tool named as MuTMuT which is based on four optimizations and one heuristic. The mutation testing method used mutation testing in safety-critical industry using high-integrity subsets of C and ADA [33]. They recognized most adequate mutant types and analyzed main reasons of failure in test cases. Moreover, they also provided a practical evaluation of the application of mutation testing to airborne software system. One of the main issues regarding mutation testing was high cost, due to the creation of mutants, execution of mutants and calculation of their scores. Mateo and Usaola [34] proposed a mutant schema with extra code (MUSIC) which reduces the mutation cost through uncovered mutants. Though this technique defines the statements enclosed by the tests in the original system, in order to out the mutant executions, because tests are only executed against the mutants whose mutated statement is covered by tests. Authors [35] measured the complexity of mutants and prioritized them on the basis of how easy or hard to manifest them. The mutation testing is presented in perspective of Python program [36]. They showed how mutation testing can be effectively handled in Python environment. A mutation reduction method is proposed in terms of program structure [37]. Although they used two path-aware heuristic rules named as loop-depth and module-depth rules and combined these two rules with operator-based selection and statements for the development of four mutant reduction approaches. In addition to this, the researchers [38] have also evaluated mutation at the class level while the existing method analyzes mutation at the traditional level. Further, they proposed a MuCPP system which is based on the class mutation operators of C++ programming language. An improved genetic algorithm is presented [39] which is a search-based approach to reduce the computational cost of mutation testing. However, they used state-based and control-oriented fitness function in their tool eMuJava and compared it with other standard fitness functions. However, aforementioned mutation-based testing approaches have been presented in the literature. All these approaches were focused on mutation

testing, while mutation word first time was used [3, 4] in clone detection research field in a brief way. Although they provided an editing taxonomy for various types of clone generation and also proposed a mutation/injection-based framework for evaluation clone detection tools.

As per the authors' best knowledge, there is no thorough work on mutation testing operators in perspectives of software clone detection tools. Yet, there is no standard benchmark available for evaluating clone detection, and there is no empirical evaluation, which explicitly determines the utilization of mutants in clone detection research field. The author contribution in this manuscript is summarized as follows:

- We used mutation testing concept in clone detection research area.
- We used mutation operators for generating various types of software clones.
- Proposed an evaluation framework using mutated code clones.

We evaluated clone detection tools on the basis of mutated code.

5 Proposed Evaluation Framework

This section presents details of our proposed framework which is based on mutation testing as shown in Fig. 5. We start with the basic components of the proposed framework and then thrash out clones terminology, precision and recall of the tools. Although, this framework is based on mutation testing and it acts in accordance with the principle of mutation testing.

Figure 5 illustrates conceptual layout of the proposed framework. The proposed framework is categorized into two main phases. Firstly, Clone Generation Phase, in this phase, software clones are generated from original code with the help of mutation operator-based editing taxonomy. The second phase is Tools Evaluation Phase, in which mutated codes are used to evaluate the software clone detection tools performance. The detailed discussions of the proposed framework are shown below.

5.1 Validation Study

Input Original Code Base: At the primary step of the framework, we input the target code base which is shown in Fig. 3. To find out which tool would be significant for such a valuation in terms of recall and precision.

Figure 3 shows an example of code base which is taken as input in the proposed framework. The original code is retrieved from an open-source project named as wet lab [40] which is an open-source project of C++ language.

Figure 4 depicts an illustration of duplicated code which is generated using mutation operators-based editing taxonomy as well as original source code.

```
//Date: Oct 4 Start :5:23PM End :5:30PM
//Find The Pair With Given Sum
//Complexity : O(n)
#include<iostream>
#include<vector>
using namespace std;
int main()
{
vector<int> v = {1,2,4,5,6,7};
int sum ,left =0 ,right=v.size()-1;
cin>>sum;
while(left<right)
{
 if(v[left]+v[right]==sum)
 break;
 else if(v[left]+v[right]>sum)
 right--;
 else
 left++;
}
if(left<right)
cout<<"Pair Having Sum  "<<sum <<" found at location "<<left+1<<" and "<<right+1<<" value
s are "<<v[left]<<" and "<<v[right];
else
```

Fig. 3 An example of original code base

```
//Date: Oct 4 Start :5:23PM End :5:30PM
//Find The Pair With Given Sum
\\Testing Comment Inserted//Complexity : O(n)
#include<iostream>
#include<vector>
using namespace std;
int main()
{
vector<int> v = {1,2,4,5,6,7};
int sum ,left =0 ,right=v.size()-1;
cin>>sum;
while(left<right)

{
 if(v[left]+v[right]==sum)
 break;
 else if(v[left]+v[right]>sum)
 right--;
 else
 left++;
}
if(left<right)
cout<<"Pair Having Sum  "<<sum <<" found at location "<<left+1<<" and "<<right+1<<" value
s are "<<v[left]<<" and "<<v[right];
```

Fig. 4 An illustration of code clone (type-1)

Random Selection of Code Segment

Once the source code is elected, then any preferred fractional number of accessible source code segment is either selected automatically or randomly from the code base for clone mutation.

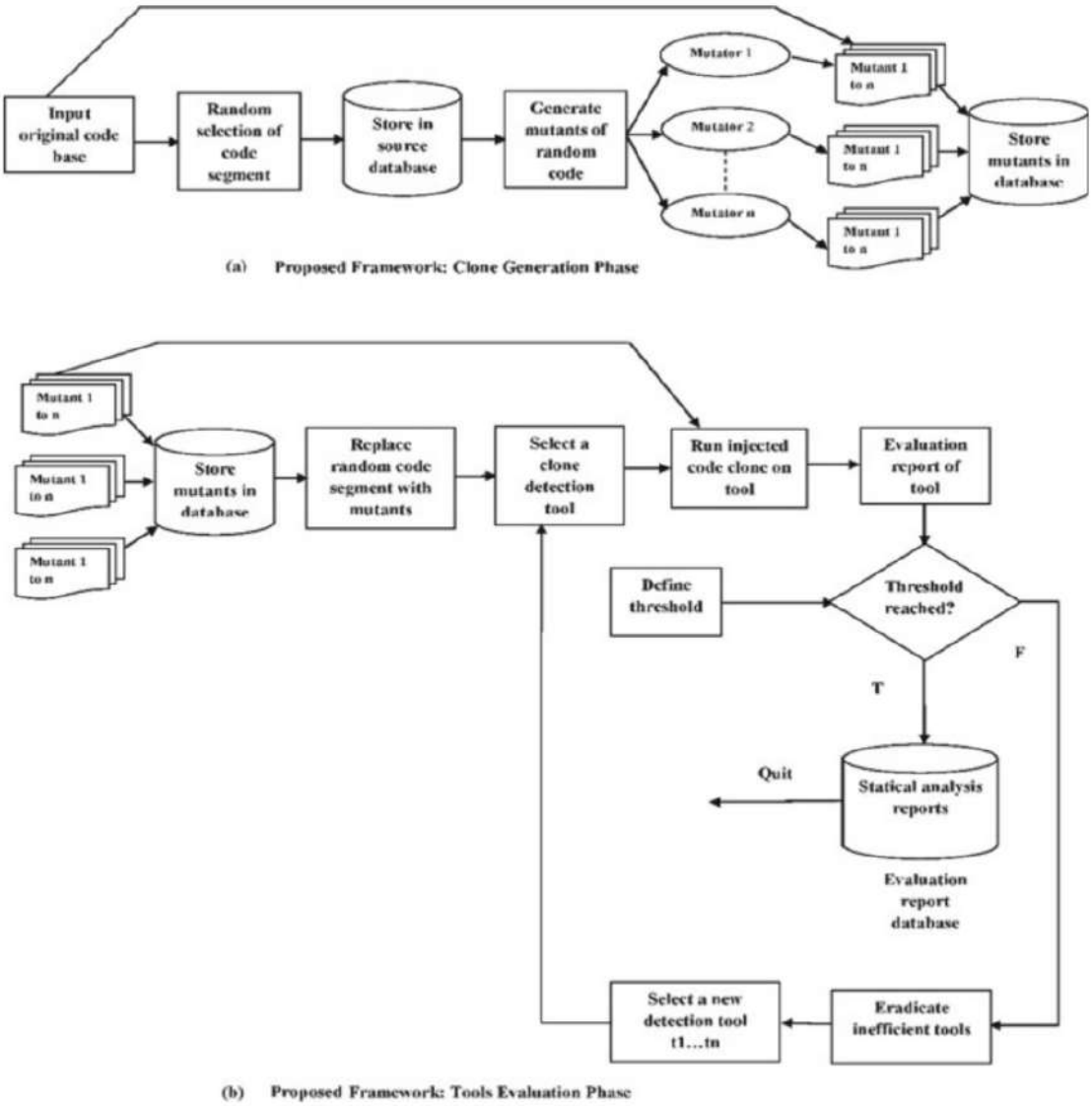

Fig. 5 Proposed mutation-testing-based automatic evaluation framework **a** Clone Generation Phase **b** Clone Tools Evaluation Phase

Stored in Source Database

Randomly selected code segments are stored in the source database, and then, mutants will be created by using these randomly selected code segments.

Generate Mutant of Random Code Segment

The edit-based taxonomy of mutation operators is used for generating mutant versions of randomly or sequentially selected code fragments which are stored in the mutant source database. An example of mutant-version of original code base is shown in Fig. 4. A number of mutants can be generated for one code segment.

The mutated version of the code base after replacement will be fed as input to clone detection tool. The detection tool will be evaluated on the basis of how many clones it will detect accurately and how fast they detect clones.

5.2 Tools Evaluation Phase

In this phase, each of the mutated code is stored in the database, and then, each mutated code is fed into clone detection tools as an input for evaluating and comparing clone detection tools. The main key feature of this phase is threshold. We defined threshold for clone detection tools. If a tool does not fulfill the threshold limit, then that will be eradicated and a new tool will be selected for evaluation.

Replace Random Code Segment with Mutants
The mutated code is generated for each of the mutant code variants of the source code. The generated mutated version of the original code base is replaced with the randomly selected code segments in the original code base.

Select a Clone Detection Tool
In this phase, a tool will be selected, and then, mutated code base is given as input into the selected tool.

Run Injected Code Clones on Tool
The mutated version of code base after replacement will be given as input on clone detection tool. The detection tool will be evaluated on the basis of how much clone it will detect accurately and how fast they detect clone.

Evaluation Report of Tool
The evaluation report makes a decision for the tool. It analyzes the performance of the tool on the basis of precision (number of accurately detected clones) and recall (total number of clones), scalability (tool supports large database) and portability (language support).

Define Threshold
The key objective of this phase is to diminish the execution time and cost of this process. A threshold will be defined by a user for a clone detection tool, and the tool crossing the threshold will be eliminated, and it will select a new tool for evaluation. Further, the tool which has low precision and recall will be discarded and the high precision–recall tool will be evaluated. It will be useful for those users who are seeking high precision and recall-based tools.

Eradicate Inefficient Tool
Most of the tools have low accuracy in the sense that they did not detect all clones (recall), as they returned a large number of false positive (precision). This led to the cognizance which is a step of valuating the results of tools that should be saved at the end.

Statistical Analysis Report
Once the experimentation is accomplished, then evaluation database is used to calculate clone detection tools accuracy as precision, recall, portability and scalability for each type of clone.

6 Clone Terminology

The clone detection tool returns clones in the form of clone pairs (CP), clone classes (CC) or both. The similarity relation between two or more clone fragments is illustrated by these two terms. Further, the clone similarity relation between code segments is an equivalence relation which is described in [41]. The clone relation exists between two segments if they are similar structurally or semantically. A pair of code fragments is called clone pair if there exists a clone relation, while clone class is a union of all clone pairs [42].

Definition 1 (*Code Segment*) A code segment (CS) consists of any subsequence of code string. It is defined by any type of granularity as fixed (predefined syntactic boundary as function, begin-end block) or free (no syntactic boundary). A CS is detected on the basis of granularity in the original program, and it is implied as (CS.Filename, CS.BeginLine, CS.EndLine).

Let $P = \{0, 1, 2, \ldots)$ and $P{+}{=} \{1, 2, \ldots)$. For $p \in P$, denoted by $O\ (p)\ A$ the set of n operations on A and set $OA{:} = Up \in P{+}\ O(p)A$. A subset CS $\subseteq OA$.

Definition 2 (*Software Clone*) A code segment CS2 is a duplicated software clone of another source segment CS1 if they are identical on the basis of some given definition of similarity that is $f(\text{CS1}) = \text{CS2}$, where f is a similarity function (textual or semantic). Further, when two code segments are similar with each other, they are called clone pairs.

$$\text{CP} = (\text{CS1}, \text{CS2})$$

Definition 3 (*Software Clone Types*) Software clones are classified on the basis of their attributes as textual similarity-based or syntactic-based, and other is functional similarity-based or semantic-based [5].

Definition 4 (*Code Segment Encompassment*) If two or more code segments are contained within same file or the boundary of line numbers of CS1 is within the boundary of line numbers of CS2.

File contained (CS CS1, CS CS2)

If((CS1.FileName==CS2.FileName) && (CS1.BeginLine>=CS2.BeginLine) && (CS1EndLine<=CS2.EndLine))

7 Measuring Recall

The main objective of our proposed framework is to automatically inject software clones in the source code which are generated by using mutation operators. Further, we evaluate clone detection tool's accuracy in terms of precision and recall. The proposed framework addresses recall for each type of software clones and for each

type of mutation operators for all the tools. In this proposed framework, the detectable clones are our injected mutant clone versions of the original source code, and that will be inserted into mutated source code base, randomly or automatically. The mutation testing-based techniques make recall simpler. Moreover, if the mutated code segment as moCS of original code segment oCS inserted into the mutated code base mioCB of original source code base oCB is "killed," oCScs and moCS are detected by clone detector and their threshold level value which returns 1 if detector's minimum threshold value is greater than 1 and maximum value less than 100, otherwise it will return to 0. The main objective of threshold value is to minimize the execution time and complexity. The threshold value depends on user's requirements, and the user can define threshold value between 1 and 100 because recall cannot be less than 1 and cannot be greater than 100.

$$\text{Recall} = (\text{Number of detected clones} * 100)/\text{Total number of clones}$$

R T (oCS, moCs) = {return 1, if (oCS, moCS) is detected by tool T in mioCB → THLV (minimum value>=1 & maximum value<=100) otherwise 0.

Where THLV means threshold level value. The similar code segments can be inserted or injected randomly, any number of times, in the original code base oCB and generate n different mutated variants of oCB as mioCB1, mioCB2, …, mioCBn. The proposed framework used mutation operators for creating various types of mutated versions of source code and then inserting them randomly several times to check the sensitivity of clone detector. However, the random segment selector selects m code base, and each of them will be mutated by mutation operators dmOP for generating mutated versions of code segments as moCS1, moCS2, …, moCSm.

Hence, the recall for mutation operator dmOP for clone detector is given as follows.

$$\text{RTdmOP} = \sum_{i=1} n * \text{mRT(oCSi)}/n * m$$

The type-1 software clones used four types of mutation operators (mCW, mCNWs, mCC, mCF, mCB) and their combination (mCW + mCNWs), (mCC + mCF), (mCF + mCB), (mCW + mCC), (mCW + mCF), (mCW + mCB), (mCNW + mCC), (mCNWs + mCF), (mCNW + mCB), (mCC + mCB) and mCF + mCB) can be applied to the m code segments (if we allow operator repetition, then a number of combination can be generated), and each of which is inserted n times into the code base. Therefore, the recall of clone detector tool T for type-1 can be defined as:

RTdmOP = $\sum_{i=1} n * m * (5 + 11)$ RT(oCSi, moCSi)/$m*n*(5+11)$ = {return H, if (n, m) is detected by clone detector tool T in mioCBTHLV (minimum value>=1 & maximum value<=100) otherwise L.

Where H means high recall, L means low recall and 5 indicates number of operators and 11 indicates number of combinations.

The overall recall of clone detectors can be defined as:

RTdmOP $= \sum_{i=1} n * m * (S + C)$ RT(oCSi, moCSi)$/m{*}n{*}(S{+}c) = \{$return H, if (n, m) is detected by tool T in mioCB $\rightarrow$ THLV (minimum value>=1 & maximum value<=100) otherwise L.

Where H means high recall, L means low recall and S indicates number clone mutation operators and C indicates number of combinations.

8 Measuring Precision

Precision measures irrelevant items which appeared in the results. Preferably, precision should be high when recall increases, but practically, it is difficult to accomplish. The precision definition is shown below.

Precision = (Number of correctly detected clones $*$ 100)/Total number of detected clones

The precision of a tool can be calculated as a mutated code segment moCS generated by using mutation operators dmOP, and clone detector tool T returning k clone pairs, (moCS, CS1), (moCS, CS2)…(moCS, CSK) in mutated code base mioCB.

PTdmOP = w.r.t. t. single insertion of moCS = a/k {return 1, if (oCS, moCS) is detected by tool T in mioCB $\rightarrow$ THLV (minimum value>=1 & maximum value<=100) otherwise 0.

Where a means accurate detection and k means number of clone pairs returned by a clone detector T.

The overall precision of clone detector tool in terms of number of mutation operators and number of combinations which is applied n times to m code segments is shown below.PTdmOP $= \sum_{i=1} n * m * (S + C) \Big/ \sum_{i=1} n * m * (S + C) = \{$return H, if (n, m) is detected by clone detector tool T in mioCBTHLV (minimum value>=1 & maximum value<=100) otherwise L. Where H means high precision and L means low precision.

9 Conclusion and Future Work

The valuation of code clone detection tools and techniques is an emerging issue in today's scenario. The previous studies for experimentally valuating clone detection tools and techniques had various constraints and hence, cannot present a persuasive comparable survey. Thus, in this paper, we proposed a peculiar mutation testing-based evaluation layout, for valuating code clone detection tools which are used by testing society over the past thirty years. Moreover, this paper encompasses mutation testing, operators and provides insight into related work on mutation testing. However, we have not accomplished the execution of the framework until now; we are assured that such a framework can present truthful analogous outcome for distinct tools, in finding deliberately generated software code clones. In the proposed structure,

18. Offutt J (2011) A mutation carol: past, present and future. Inf Softw Technol 53(10):1098–1107
19. Lipton RJ (1971) Fault diagnosis of computer programs. Student report, Carnegie Mellon University
20. Budd T, Sayward F (1977) Users guide to the Pilot mutation system. Technique report 114, Yale University, New Haven, CT
21. Hamlet RG (1977) Testing programs with the aid of a compiler. IEEE Trans Softw Eng 4:279–290
22. DeMillo RA, Lipton RJ, Sayward FG (1978) Hints on test data selection: help for the practicing programmer. Computer 11(4):34–41
23. DeMillo RA (1989) Completely validated software: test adequacy and program mutation (panel session). In: Proceedings of the 11th international conference on software engineering. ACM, Pittsburgh, PA, pp 355–356
24. Woodward MR (1990) Mutation testing—an evolving technique. In: IEE colloquium on software testing for critical systems, p 3-1
25. Woodward MR (1993) Mutation testing—its origin and evolution. Inf Softw Technol 35(3):163–169
26. Mathur AP (2013) Foundations of software testing, 2/e. Pearson Education India
27. Ammann P, Offutt J (2016) Introduction to software testing. Cambridge University Press, Cambridge
28. Offutt AJ, Untch RH (2001) Mutation 2000: uniting the orthogonal. In: Mutation testing for the new century. Springer, USA, pp 34–44
29. Jia Y, Harman M (2011) An analysis and survey of the development of mutation testing. IEEE Trans Softw Eng 37(5):649–678
30. Zhu Q, Panichella A, Zaidman A (2016) A systematic literature review of how mutation testing supports test activities (No. e2483v1). PeerJ Preprints, pp 1–57
31. Reales P, Polo M, Fernandez-Aleman JL, Toval A, Piattini M (2014) Mutation testing. IEEE Softw 31(3):30–35
32. Gligoric M, Jagannath V, Luo Q, Marinov D (2013) Efficient mutation testing of multithreaded code. Softw Test Verification Reliab 23(5):375–403
33. Baker R, Habli I (2013) An empirical evaluation of mutation testing for improving the test quality of safety-critical software. IEEE Trans Softw Eng 39(6):787–805
34. Mateo PR, Usaola MP (2015) Reducing mutation costs through uncovered mutants. Softw Test Verification Reliab 25(5–7):464–489
35. Namin AS, Xue X, Rosas O, Sharma P (2015) MuRanker: a mutant ranking tool. Softw Test Verification Reliab 25(5–7):572–604
36. Derezinska A, Hałas K (2015) Improving mutation testing process of python programs. In: Software engineering in intelligent systems. Springer, Cham, pp 233–242
37. Sun CA, Xue F, Liu H, Zhang X (2017) A path-aware approach to mutant reduction in mutation testing. Inf Softw Technol 81:65–81
38. Delgado-Pérez P, Medina-Bulo I, Palomo-Lozano F, García-Domínguez A, Domínguez-Jiménez JJ (2017) Assessment of class mutation operators for C++ with the MuCPP mutation system. Inf Softw Technol 81:169–184
39. Bashir MB, Nadeem A (2017) Improved genetic algorithm to reduce mutation testing cost. IEEE Access 5:3657–3674
40. Wet Lab (1989) Retrieved from http://ftp.gnu.org/gnu/wget/
41. Kamiya T, Kusumoto S, Inoue K (2002) CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Trans Softw Eng 28(7):654–670
42. Roy CK, Cordy JR (2007) A survey on software clone detection research. Queen's School of Computing TR, 541(115), pp 64–68