CIT. Journal of Computing and Information Technology, Vol. 28, No. 3, September 2020, 147–164
doi: 10.20532/cit.2020.1005155

A Review of Software Reliability Testing Techniques

Zhouxian Jiang, Honghui Li, Dalin Zhang, Rui Wang, Junwen Zhang, Xiuru Li, Meng Zhang and Penghao Wang

Beijing Jiaotong University, China

In the era of intelligent systems, the safety and reliability of software have received more attention. Software reliability testing is a significant method to ensure reliability, safety and quality of software. The intelligent software technology has not only offered new opportunities but also posed challenges to software reliability technology.The focus of this paper is to explore the software reliability testing technology under the impact of intelligent software technology. In this study, the basic theories of traditional software and intelligent software reliability testing were investigated via related previous works, and a general software reliability testing framework was established. Then, the technologies of software reliability testing were analyzed, including reliability modeling, test case generation, reliability evaluation, testing criteria and testing methods. Finally, the challenges and opportunities of software reliability testing technology were discussed at the end of this paper.

ACM CCS (2012) Classification: Software and its engineering → Software creation and management → Software verification and validation → Software defect analysis → Software testing and debugging

Keywords: software reliability, intelligent software, test framework, test methods, evaluation methods

1. Introduction

In recent years, many large-scale complex systems (LSCSs) have been applied in safety-critical areas, such as aerospace, rail transportation, finance, and military [1]. Software systems based on big data and artificial intelligence (AI) technologies are also widely applied to such areas as image recognition [2], speech recognition [3], medical diagnosis [4], machine translation [5], and natural language reasoning [5]. Besides, they are increasingly being deployed in such security-critical areas as autopilot [6] and malware detection [7]. As software becomes more complex, intelligent and deployed extensively, its security and reliability are facing greater challenges, and the predictable and assessable behavior are of great significance to ensure its security and safety.

Software reliability engineering (SRE) is defined by J. D. Musa [18] as "a series of technical and management activities of an engineering nature carried out to achieve software reliability". The ultimate goal of software reliability testing (SRT) is to verify whether software requirements are met by predicting and evaluating reliability of the system under test.

There are two reasons that motivate us to compose this review.

As first, traditional software has presently been developed to a relatively mature degree due to various testing methods and model technologies. However there are still problems with test model application, evaluation standard design, and automation difficulties for large-scale software in different fields. It is expected that some inspirations for solving these problems could be provided via conducting a summary of the current research status.

Secondly, although intelligent software based on big data and artificial intelligence has achieved remarkable acomplishments in machine trans-

lation and natural language inference, it could induce unexpected conditions that may lead to accidents in safety-critical systems, such as autonomous driving systems. Therefore, provision of some inspirations for a reliability test of intelligent software in combination with traditional software engineering is expected.

In this paper, the background of SRT will be introduced in Chapter 2; traditional SRT will be summarized in Chapter 3; intelligent SRT will be summarized in Chapter 4; traditional software and intelligent SRT technology will be compared in Chapter 5; the challenges and opportunities will be proposed in Chapter 6; a summary will be given in Chapter 7.

2. Background

Software reliability is closely related to system input and system application.The IEEE Computer Society of America has provided a definition of "software reliability" as follows:

1. probability that the software will not cause failure in the systems under specified conditions and for a specified period of time;
2. the ability of a program to perform the required functions under the conditions described within a specified period of time.

The main process of SRT is to conduct a test as per provisions in a real operating environment or in a simulated test environment, and then adequately collect the test data and evaluate software reliability [19]. However, there are significant differences in logic and composition between traditional and intelligent software, which can be specifically described as follows.

- Traditional software is mainly to construct a usage model, which has a relatively clear content and can express the structure and behavior of the software. Intelligent software is often composed of neural networks and intelligent algorithms. Therefore, the components of intelligent software under test are also different from traditional software.
- The failure of traditional software is mainly related to the function, structure, and internal logic, while the failure of intelligent software is more related to datasets, algorithms and program frameworks.
- For traditional reliability evaluation, the reliability evaluation models are generally modeled for analysis, evaluation and prediction. However, as for intelligent software current research still stops at test methods and coverage criteria, it is obvious that more attention should be paid to the testing and evaluation of intelligent software.

In addition, a general SRT framework has been proposed by combining the characteristics of traditional software with intelligent software, which presents the correlation between SRT and software life cycle, as shown in Figure 1. The upper part of the figure is the life cycle of software and the dotted line connection with the software reliability test process at the bottom of the figure indicates that the life cycle process is related to the process of the software reliability test.

Figure 1 shows that software reliability testing starts with the establishment of reliability goals, then requirements are analyzed and a model is built. Traditional software builds a usage model, while intelligent software builds a deep learning algorithmic model. Intelligent software has more steps of model training and verification before software deployment than traditional software. After the model is deployed, we need to design test cases for execution and then collect the failure data. In the evaluation stage, it is necessary to design evaluation criteria for reliability. Besides, it should be analyzed whether the result meets the reliability goals. If not, the fault shall be repaired and regression testing shall be modified until the reliability goals are reached. Finally, a report shall be generated when the test is finished.

Due to limited space, the requirement analysis part will not be introduced here, and we will focus on the technologies of modeling, testing, and evaluation.

3. Traditional Software Reliability Testing

In 2012, Li *et al.* [14] summarized SRT as reliability testing, reliability evaluation and prediction models. In 2015, Sharma *et al.* [15]

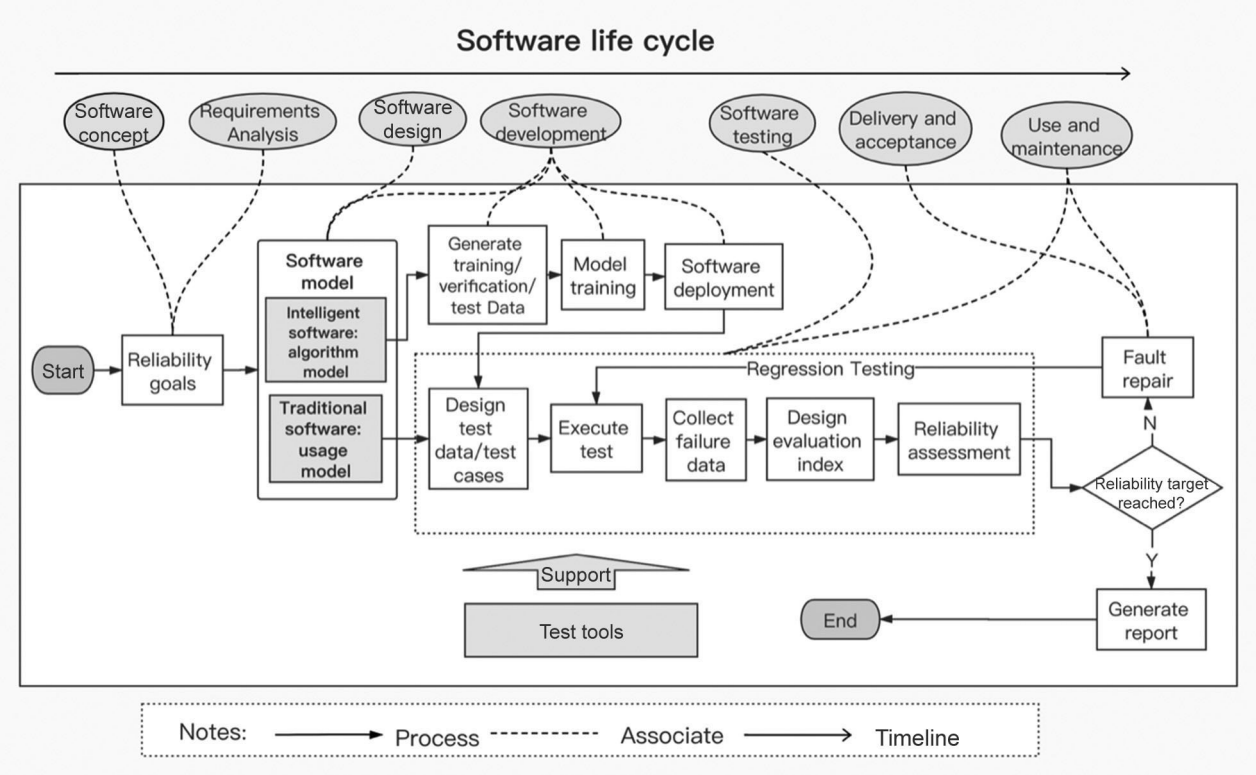

Figure 1. Universal SRT framework.

summarized the *software reliability growth model* (SRGM) and the tools, explaining various software failures, performance testing, fault tolerance detection, and software system reliability evaluation methods. In 2016, Kumar *et al.* [16] summarized SRGM, the tools and datasets, which divided SRGM into parametric and non-parametric models, and provided some reference datasets and software reliability tools.

A thorough study on traditional SRT and related evaluation models has been conducted in the above mentioned papers, and especially on SRGM. At present, the research direction of traditional SRT is becoming increasingly multidimensional. Therefore, it is necessary to fully understand the development of this field. In this chapter, we focus on the construction of usage models, generation of test cases, reliability evaluation technologies and so on.

3.1. Model-Based Reliability Test Case Generation

Given the fact that SRT requires as much data as possible with high coverage for statistical analysis, the testing process may be repetitive to a certain degree. Therefore, the realization of generating test cases and evaluating test results are mostly based on the model or derived model of the object under test, *i.e.* test models [21], [23], [24]. This type of testing, in which the test model is employed to formalize or semi-formalize the testing process, is also known as model-based testing. The software usage model is also a kind of testing model, which is an abstract representation of software behavior and structure, and can be used as a driving model for the generation of test cases and automating test. After the reliability goals are determined, the software usage model can be constructed and the test cases can be generated through the model.

3.1.1. Usage Models

In 1997, the unified modeling language (UML) was recognized as the standard modeling language by the Object Management Group (OMG) and since then has been widely used to construct and optimize software usage models

[23]. In addition, Markov Chain Model [25] [26], Finite State Machine Model [27], UML Model [28] [29], *etc.* are employed to construct software usage models. According to the different modeling methods, usage models can be broadly divided into four categories as follows.

UML-based usage models. Software systems are visualized and built from different angles through the four-level metamodel architecture to form multiple views of the system. Common modeling scenarios include the library system [35], online store [36], web server [37], and industrial projects [136].

Markov-chain-based usage models. A construction model based on Markov chain can statistically reflect the structural and behavioral characteristics of software, and perform statistical calculations and analysis [22]. Common modeling scenarios include ATM [26], and MaTeLo [142].

Operation-profile-based usage models [30]. The operation profile of software can be used to quantitatively describe the way that users actually use the software, and it is necessary to consider the using habit for the software in various modes and functions and the probability of use in the construction of the profile model. Common modeling scenarios include Phone follower software [41], and the space program [42].

Formal-language-based usage models. Functional structure and scenario path of the system are described through explicit formal mathematical language, and usage models are constructed, such as finite state machines (FSM) [27], Petri network [31], Z language [32], SOFL language [33], *etc.* Common modeling scenarios are Web app [43], Embedded systems [137], Mobile app [33], TWIN elevator system [138], Multilift system [139] and so on.

In addition, there are other modeling methods, such as grammar models [34], which can be employed to describe program syntax.

3.1.2. Test Case Generation

After a usage model is constructed, test cases can be automatically generated using that model. When the testset complies with the coverage criteria, the generation of test cases can be terminated [48]. In this paper, the usage models, common modeling methods, and corresponding methods of automatically generating test cases have been sorted out, as shown in Figure 2.

It can be seen that there are many methods of using the model to generate test cases. An appropriate usage model shall be selected based on the characteristics of the test goals, and then an appropriate method shall be selected to generate test cases.

3.1.3. Research Trends in the Current Decade

In software engineering, software usage models serve as an expression of formal ideas, which is of great significance. In 2015, Saurabh *et al.* [94] published a systematic literature review of the use case specification methods, which provides a reference for the use of model-based test case specification. In 2016, Utting *et al.* [95] summarized the results of the model-based testing field from 2006 to 2016, including the modeling process, modeling language, techniques for generating test cases from models, and application examples.

Normalization and improvement of the construction methods for software usage models. In 2010, Rita *et al.* [27] proposed a method to develop test suites with formal specification in the form of FSMs which is of some reference value for FSM modeling. In the same year, Liu *et al.* [47] extended the reliability information in the UML model and automatically converted the UML model into a Markov chain-based model.

Optimization of the test case generation method. In 2010, Vahid *et al.* [49] proposed a method based on UML 2.0 to supplement information to the sequence diagram for the construction of the model and to apply a genetic algorithm for the automatic generation of test cases that can meet the task timing protocol. Nayak *et al.* [28] transformed the UML activity diagram into an intermediate testable model and generated test scenarios to identify all possible scenarios and check defects in the use case scenarios.

Research on modeling tools. In 2015, Yue *et al.* [96] proposed an automation framework for deriving UML analysis models from use case models, and the aToucan tool. Due to the fact

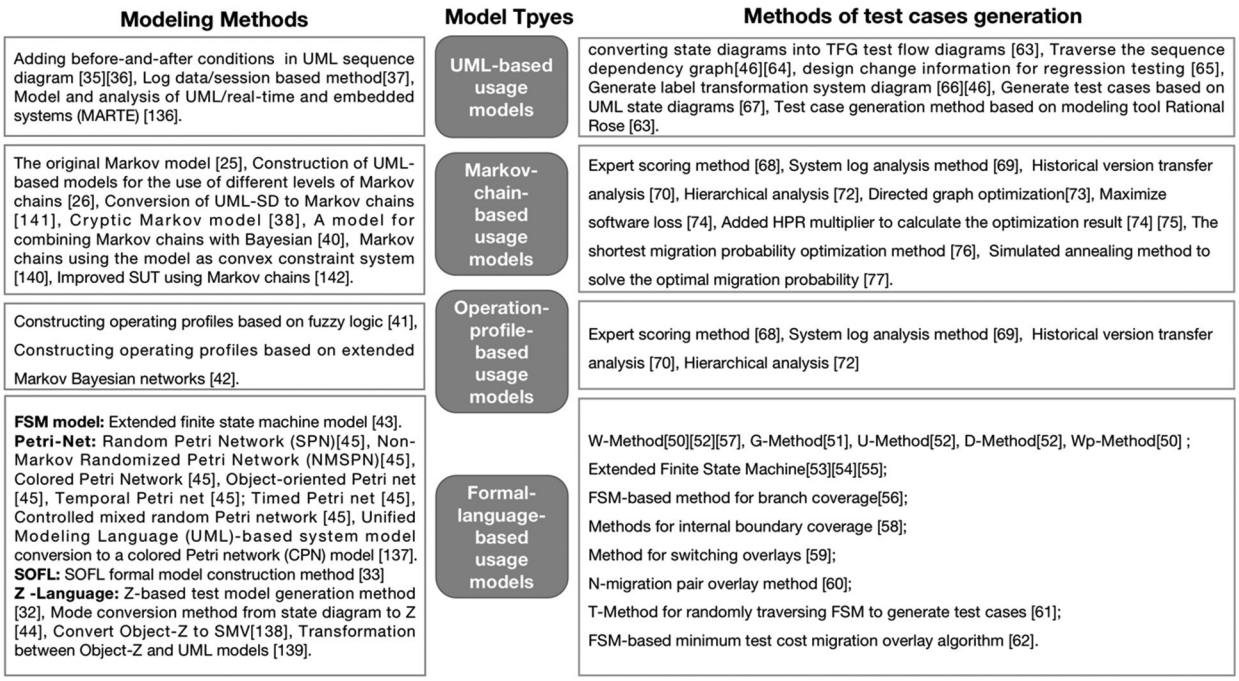

Figure 2. Usage models, modeling methods, and test cases generation methods.

that most use case models are constructed manually, there is a possibility of errors that could omit certain system use behaviors, and the generated test cases may ignore these behaviors. For that reason, in 2017, Gebizli *et al.* [97] introduced a method and the ARME toolset for automatic refinement of the system models based on test activities recorded by modeling engineers. In 2018, Liu *et al.* [98] introduced a new model-based test tool MTTool to simulate complex software behaviors and generate test cases from models.

The test case selection technique. In 2013, Hadi *et al.* [99] introduced 320 different similarity-based test case selection (STCS) techniques and drew a comparison among them with the aim of addressing the problem of scalability in model-based testing. Afterwards, Hadi proposed a method to determine the optimal trade-off between the number of test cases in operation and fault detection.

Application of usage models. In 2012, Wan *et al.* [24] proposed an evaluation method to assess the accuracy of the model used for various reliability tests based on the Markov usage model for web applications. In 2013, Wan *et al.* [100] designed a model-based approach to generate scenario test cases for nuclear digital security systems by converting scenarios. In 2016, in order to test large software products, Gebizli *et al.* [101] proposed a method for system reuse of hierarchical Markov chain usage models.

To sum up, the research on software usage models and automatic generation of test cases have emerged in the last decade, and a certain specification has been formulated. However, more thorough studies are needed to be conducted regarding how to choose the appropriate method to build the usage model, how to improve the coverage of test cases, how to verify the feasibility and correctness of testing models in more fields, and regarding the development and refinement of related tools.

3.2. Software Reliability Evaluation

After test execution is completed, the next step is to evaluate software reliability [96], which is generally calculated through the software reliability model. According to the standards IEEE1633-201[20] and GJBZ161-2012 (In Chinese) [19], the reliability evaluation process can be divided into six steps, *i.e.* failure

definition, failure trend analysis, selection of software reliability evaluation model, model parameter estimation, model verification, and reliability estimation and prediction.

3.2.1. Software Reliability Evaluation Models

Between 1956, when Weiss *et al.* [83] proposed a hardware reliability evaluation method, and 1987, when Musa *et al.* [84] co-authored the book *Software Reliability – Evaluation, Prediction and Application*, the software reliability issue was getting more comprehensive and apt to more in-depth discussions. By the 1990s, reliability model technologies had developed to a relatively high level of maturity. In 1991, Cai *et al.* [102] reviewed the development of software reliability models and concluded that almost all previous software reliability models had been developed in a probabilistic context. Wang *et al.* [96] conducted a related review; Sharma *et al.*[103] proposed to select a model by model assumptions similarity; Musa *et al.* [104] proposed five software reliability model evaluation criteria, which have been more widely recognized currently. Moreover, there are also many researchers who have begun to study the universal model with integrated prediction framework, neural networks, artificial intelligence and other methods for software reliability integrated prediction [105] [106]. It is evident that after the 1990s, more attention is being paid to the applicability of the model.

In 2016, Kumar *et al.*[16] provided a summary of parametric and non-parametric modeling of software reliability, and listed some tools and datasets. In the next year, Zhang *et al.* [107] analyzed and summarized SRGM, comparing SRGMs typical at that time. At present, there are various types of software reliability evaluation models, such as the JM model [17], Musa model [90], GO model, LV model [143], Schneidewind model [87], universal exponential model [88], generalized exponential model [92], Shooman model [90], Yamada S model [91], Duane model [93] and other classic models, as well as the Rome model [85], Keene's development process model [86], which are all mainly used to predict failures. In addition to the classic model, there are models based on the architecture [118] which can be divided into path model, state model, component operation profile model and others.

In addition, according to different modeling methods, reliability models can be divided into analytical models and heuristic models. According to whether the internal structure of software is clear, evaluation models can be divided into white box models, black box models, *etc.*

3.2.2. Research Trends in Current Decade

In the last section, we present research work on software reliability evaluation and summarize the research direction over the past ten years.

In 2010, Zeng *et al.* [78] proposed a software reliability model selection method based on historical software project experience. Ullah *et al.* [80] [81] proposed a method to select the model that best predicts Open Source Software (OSS) residual defects. Vladimir *et al.* [107] proposed a reliability comparison method, but this method lacks more formal verification. Their work focuses on how to better choose the evaluation model. In addition, Huang *et al.* [108] are committed to studying the "change point" problem in software reliability assessment, and propose to incorporate multiple change points into software reliability modeling. Domenico *et al.* [115] studied optimization models. They proposed a new *reliability evaluation and improvement* (RELAI) testing technology, which can improve reliability through adaptive testing schemes.

As shown in Figure 3, we have summarized the software reliability evaluation research during the last ten years into 6 directions, including: model selection, solve the "change point" problem, imperfect debugging scheme, model optimization, novel models, and field application.

4. Intelligent Software Reliability Testing

With the advancement of intelligent software technology, the reliability of intelligent software has become a hot topic in this field. However, there are significant differences in the models and testing methods between intelligent software and traditional software. The methods

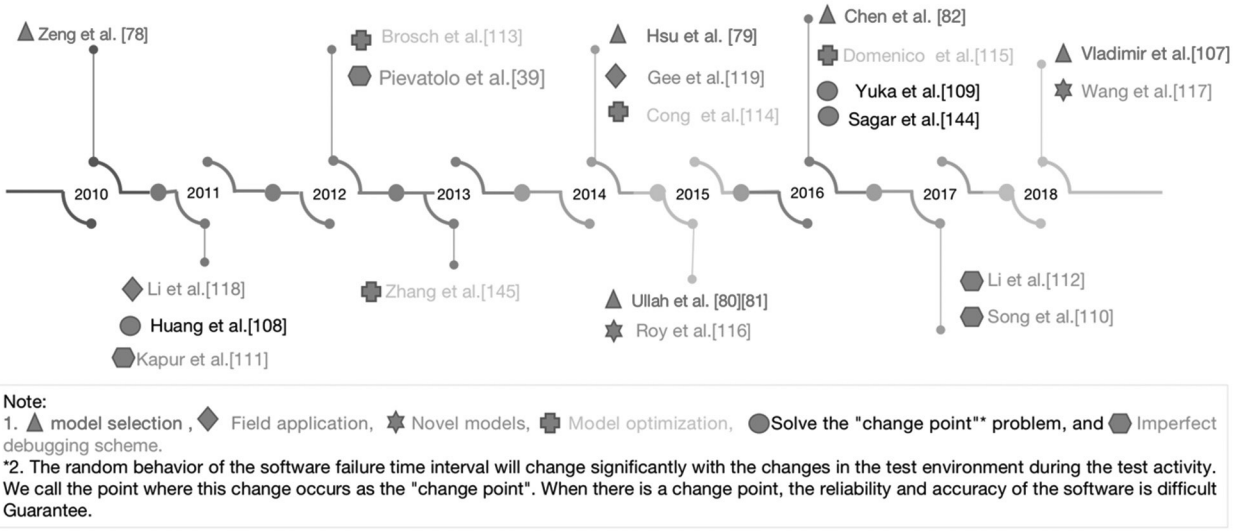

Figure 3. Research on software reliability evaluation technology over the past ten years.

of testing and evaluation of traditional software cannot be applied to intelligent software directly. Zhang *et al.* [120] proposed the framework of machine learning testing and summarized the testing content, processes, attributes, methods, coverage criteria, applications, and testing scenarios. However, they did not make a summary from the perspective of SRT. The current reliability research of intelligent software mainly focuses on the testing methods and coverage criteria. In this chapter, an overview of intelligent software and relevant reliability testing technologies will be discussed from the perspective of testing methods, frameworks, and testing coverage criteria.

4.1. Intelligent Software Testing Methods

In reliability testing of intelligent software, it is almost impossible to generate test cases manually. There are two problems to be solved:

- the Oracle problem [12] [13]: It is difficult or impossible to verify the test results under a given test case. This is a particularly common case with intelligent software, especially for unsupervised learning;
- the problem of reliable test sets [12] [13]: Since there is no possibility to make an exhaustive list of possible test cases, it is difficult to effectively select a subset of test cases that can verify procedural correctness. Intelligent software does not have the same explicit logical coverage criteria as traditional software, and the functions model in Deep Neural Networks (DNNs) are non-linear, so it would be more difficult to verify the adequacy and reliability of the test set. Also, finding inputs in the DNN that lead to high model coverage is hard.

As a result, traditional software testing methods are not necessarily applicable to intelligent software. In order to solve the above problems, more testing methods for intelligent software are proposed, such as Metamorphic Testing [121], cross-referencing as Oracles Testing [120], Concolic Testing [11], *etc.*

4.1.1. Metamorphic Testing

The method of *metamorphic testing* (MT) was proposed for the first time in 1998, T. Y. Chen [121] used the method to generate test cases. Relevant researchers were pleasantly surprised to find that MT can effectively relieve oracle problems, and it could also contribute to providing an effective and reliable test set. The MT timeline is shown in Figure 4.

2018 and 2019 studies showed that MT is effective in testing of intelligent software and especially in real scenarios, such as automatic driving systems.

4.1.2. Cross-Referencing as Test Oracles

Cross-referencing is a test oracle technique in intelligent software testing, which includes differential testing and *N* version programming. Differential testing is a traditional software testing technique that detects errors by observing whether similar applications may produce different outputs for the same input [146] [122]. Differential testing is one of the main testing methods for detecting compiler errors [123], and it is closely related to *N* version programming which aims to generate multiple functionally equivalent programs based on a single specification, thus resulting in a higher degree of fault tolerance and robustness for combinations of different versions [124].

Both DeepXplore [8] and DLFuzz [130] adopt differential testing as a test benchmark to find valid test inputs. *I.e.*, test inputs that lead to different behavior between algorithms or models are preferred during the test generation.

4.1.3. Concolic Testing

Concolic testing is a modern symbolic execution testing technique, which is an effective way to automatically generate test cases. Sun *et al.* [131] developed the first kind of concolic testing approach for DNN in 2018, where reasonable *quantitative linear arithmetic* (QLAR) was adopted to express test requirements, and test cases could be generated progressively for a given set of test requirements, to improve coverage by the shift between concrete execution and symbolic analysis. In 2019, they further introduced a DNN testing and debugging tool called DeepConcolic, which could detect errors with sufficient rigor to be suitable for testing DNN in security-related applications [132]. DeepConcolic was the first kind of tool to implement concolic testing techniques for DNN and to provide users with a test tool to investigate specific parts of DNN functionality.

4.2. Intelligent Software Testing Criteria

In order to evaluate the adequacy of intelligent software testing, scholars have also proposed a number of test coverage criteria when studying intelligent software testing. Currently, there are mainly three types of test coverage criteria: structure-based test criteria, semantic test criteria, and combination-based test criteria.

Structure-based test criteria. Such as neuron coverage [8], *k* multi-segment neuron coverage [133], boundary neuron coverage [133], Top-*k* neuron coverage [133], MC/DC criteria [135]. This type of test criteria mainly focuses on the internal structure of the neural network and considers the factors related to neuron coverage.

Semantic-based test criteria. DeepImportance [148] uses semantics of the neuron's influence on the Deep Learning (DL) system as the test

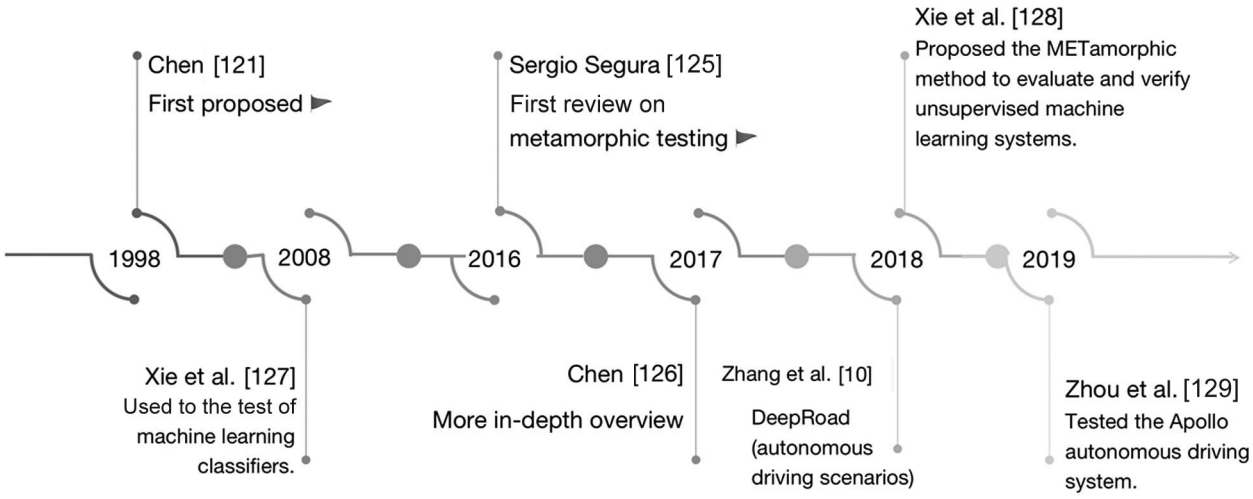

Figure 4. History of metamorphic testing.

sufficiency criterion. This type of method not only considers the internal structure of the neural network, but it also considers semantics of the neuron's influence on the DL system.

Combination-based testing criteria. DeepCT [147] proposes *t*-way combined sparse coverage, *t*-way combined dense coverage, and (*p*, *t*)-integrity coverage. This type of method mainly draws on the idea of combined testing and considers the adequacy of neural network testing from another perspective.

Based on the above, in 2019, Zhang *et al.* [134] carried out further research on the test criteria, conducting a large-scale investigation of the essential diversity of neural networks, and defined five indicators to quantify the diversity of neuronal activities.

5. Reliability Testing of Traditional Software vs. Intelligent Software

Based on the previous analysis, it becomes obvious that traditional software and intelligent software are very different in terms of constitutive logic, test methods and test criteria. The summary of comparison between traditional and intelligent software, is shown in Figure 5.

6. Challenges and Opportunities

SRT covers a wide range of fields. Intelligent software technologies have brought great opportunities and challenges to traditional software reliability. The traditional SRT becomes more intelligent, and reliability requirements for intelligent software becomes more urgent. Specifically, SRT techniques are facing the following challenges.

Challenges in test case generation. According to Utting *et al.* [95], although model-based testing and use of case techniques have made great progress in theoretical research, they are slow in industry application and the cost of application is not economic, which is caused by modeling activities and the ability of testers. The scale of traditional software continues to expand and intelligent software is often made up of large

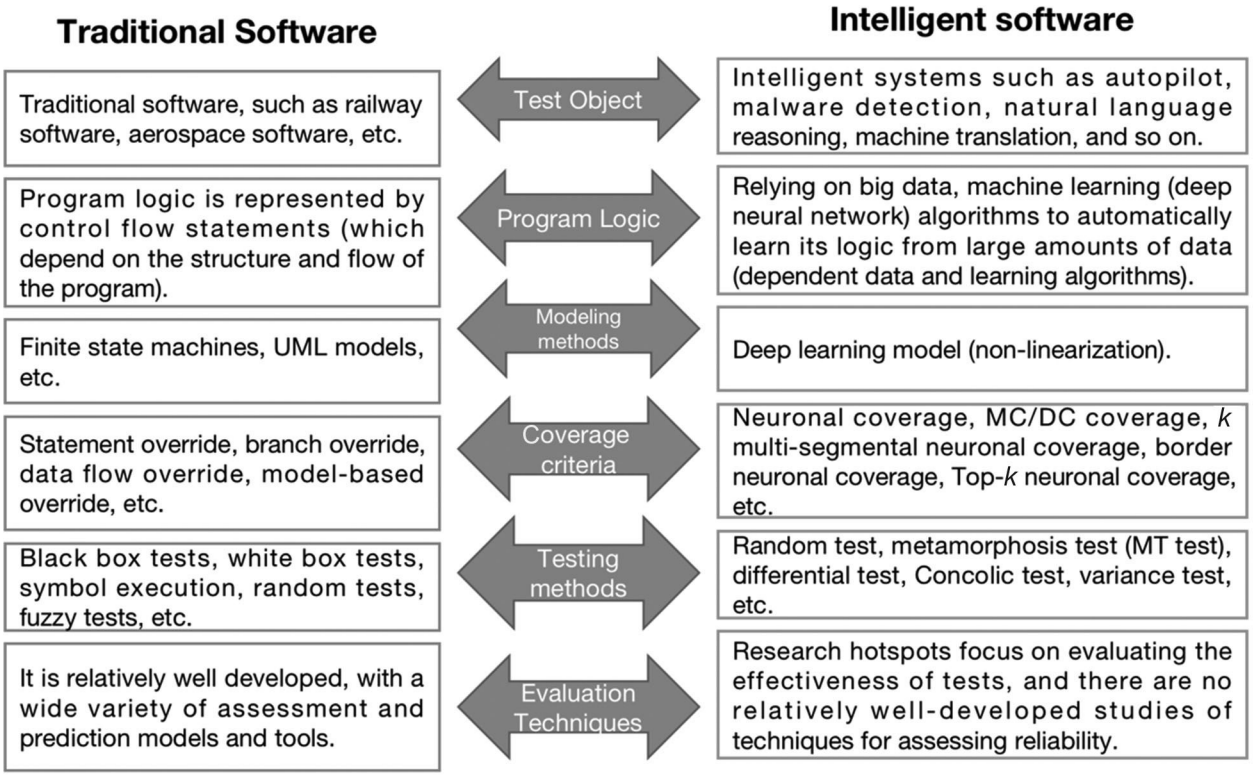

Figure 5. Comparison of traditional and intelligent software.

behavioral spaces. These factors have brought huge challenges to generate effective test sets.

Challenges in testing evaluation criteria. While traditional SRT has resulted in many criteria, the applicability of those criteria to intelligent software is a question that needs to be explored. At present, there is no uniform system of definitions and criteria for reliability testing of intelligent software. Besides, although much work has been done exploring the ways how to assess quality or adequacy of test data, there is still a lack of systematic evaluation of how different evaluation criteria relate to each other, or how they relate to the ability of tests to detect errors.

Challenges in evaluation model. As software becomes increasingly complex and large, evaluation models are difficult to construct. The construction of a broadly common set of evaluation models is also a challenge. Besides, currently there is no mature model for assessing the reliability of intelligent software, and there is an urgent demand for new evaluation methods.

Challenge in test oracle. Although DeepTest [9] can effectively mitigate the oracle problem by metamorphic testing, these metamorphic relationships are proposed mostly by humans and may contain false positives. Therefore, the automatic identification and construction a reliable oracle for SRT is still a huge challenge.

In addition to the above challenges, there are also many opportunities in the research of SRT, two of which are mentioned below.

Intelligent methodological study of traditional software reliability techniques. The development of intelligent software technology has brought more possibilities to ensure reliability of traditional software. We can apply intelligent software technology to improve automation and efficiency of testing and evaluation, as well as to reduce the cost of testing. Besides, the machine learning-based model can also be applied in software reliability evaluation.

Research on reliability evaluation methods for intelligent software. Intelligent software is usually based on DNN models. There are no mature methods and criteria for reliability evaluation. There is a series of mature theories and methods on reliability evaluation for traditional software, and applying them to reliability evaluation for intelligent software is also a direction worth studying.

7. Conclusion

In this paper, we introduced a software reliability testing technology, including traditional and intelligent software reliability testing. According to previous work, we proposed a general framework for SRT. Secondly, we introduced a usage model, test methods and evaluation methods of traditional SRT. Moreover, we also summarized the existing methods, frameworks and test coverage criteria of intelligent SRT. Finally, we compared reliability testing technologies of intelligent software and traditional software, and analyzed current challenges and opportunities of SRT technology. Hopefully, our study can help interested researchers understand the current technologies of SRT, and provide guidance for a more in-depth research.

Acknowledgment

This research was supported by Fundamental Research Funds for the Central Universities, No. 2019JBM026.

References

[1] M. Palviainen *et al.*, "The Reliability Estimation, Prediction and Measuring of Component-Based Software", *Journal of Systems and Software*, vol. 84, no. 6, pp. 1054–1070, 2011. http://dx.doi.org/10.1016/j.jss.2011.01.048

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ArXiv preprint, vol. abs/1409.1556, 2014. Available: http://arxiv.org/abs/1409.1556

[3] G. Hinton *et al.*, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups", *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012. http://dx.doi.org/10.1109/MSP.2012.2205597

[4] H. Greenspan *et al.*, "Guest Editorial Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique", *IEEE*

Transactions on Medical Imaging, vol. 35, no. 5, pp. 1153–1159, 2016. http://dx.doi.org/10.1109/TMI.2016.2553401

[5] O. Vinyals *et al.*, "Show and Tell: A Neural Image Caption Generator", in *Proc. of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2015, pp. 3156–3164. http://dx.doi.org/10.1109/CVPR.2015.7298935

[6] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars", arXiv e-prints, arXiv:1604.07316, 2016.

[7] K. Grosse *et al.*, "Adversarial Examples for Malware Detection", in *Proc. of the European Symposium on Research in Computer Security*, 2017. http://dx.doi.org/10.1007/978-3-319-66399-9_4

[8] K. Pei *et al.*, "Deepxplore: Automated Whitebox Testing of Deep Learning Systems", in *Proc. of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18. http://dx.doi.org/10.1145/3132747.3132785

[9] Y. Tian *et al.*, "DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars", in *Proc. of the International Conference on Software Engineering (ICSE)*, 2018.

[10] M. Zhang *et al.*, "DeepRoad: GAN-Based Metamorphic Autonomous Driving System Testing", ArXiv e-prints (Feb. 2018), 2018, arXiv:cs.SE/1802.02295

[11] Y. Sun *et al.*, "DeepConcolic: Testing and Debugging Deep Neural Networks", in *Proc. of the ICSE-Companion*, Montreal, Canada, 2019, pp. 111–114. http://dx.doi.org/10.1109/ICSE-Companion.2019.00051

[12] E. T. Barr *et al.*, "The Oracle Problem in Software Testing: A Survey", *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015. http://dx.doi.org/10.1109/TSE.2014.2372785

[13] T. Y. Chen *et al.*, "Fault-Based Testing Without the Need of Oracles", *Information & Software Technolog*, vol. 45, no. 1, pp. 1–9, 2003. http://dx.doi.org/10.1016/S0950-5849(02)00129-5

[14] Z. Li *et al.*, "Reliability Engineering, 2nd edition", *Journal of Quality Technology*, vol. 44, no. 4, pp. 394–395, 2012. http://dx.doi.org/10.1080/00224065.2012.11917908

[15] L. K. Sharma *et al.*, "Software Reliability Growth Models and Tools – A Review", in *Proc. of the 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2015, pp. 2057–2061.

[16] A. Kumar, "Software Reliability Growth Models, Tools and Data Sets A Review", in *Proc. of the 9th India Software Engineering Conference on – ISEC*, 2016, pp. 80–88. http://dx.doi.org/10.1145/2856636.2856648

[17] Z. Jelinski *et al.*, "Software Reliability Research", *Statistical Computer Performance Evaluation*, pp. 465–484, 1972. https://doi.org/10.1016/B978-0-12-266950-7.50028-1

[18] J. D. Musa, "Software Reliability Engineering", Wiley-IEEE Press, 1999. https://doi.org/10.1002/0471722324.ch22

[19] "Guide for Military Software Reliability Evaluation", in GJB/Z161-2012 (Revision of China Std GJB/Z161-2012), pp. 1–47, 2012. (In Chinese)

[20] "IEEE Recommended Practice on Software Reliability", in IEEE Std 1633-2016 (Revision of IEEE Std 1633-2008), pp. 1–261, 2017. http://dx.doi.org/10.1109/IEEESTD.2017.7827907

[21] R. Serfozoand, *Basics of Applied Stochastic Processes*, Springer, 2009.

[22] C. Zhenhua and W. Feng, "Research on Software Reliability Evaluation Method Based on Markov Chain Usage Model", *Computer Engineering and Design*, vol. 28, no. 12, 2007. (In Chinese)

[23] I. Schieferdecker, "Model-Based Testing", *IEEE Software*, vol. 29, no. 1, pp. 14–18, 2012. http://dx.doi.org/10.1109/MS.2012.13

[24] B. Wan *et al.*, "Evaluating Reliability-Testing Usage Models", in *Proc. of the 2012 IEEE 36th Annual Computer Software and Applications Conference*, 2012, pp. 129–137. http://dx.doi.org/10.1109/COMPSAC.2012.23

[25] J. A. Whittaker and M. G. Thomason, "A Markov Chain Model for Statistical Software Testing", *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994. http://dx.doi.org/10.1109/32.328991

[26] Y. Wang *et al.*, "A Method for Software Reliability Test Case Design Based on Markov Chain Usage Model", in *Proc. of the 2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, 2013, pp. 1207–1210. http://dx.doi.org/10.1109/QR2MSE.2013.6625785

[27] R. Dorofeeva *et al.*, "FSM-Based Conformance Testing Methods: A Survey Annotated with Experimental Evaluation", *Information and Software Technology*, vol. 52, no. 12, pp. 1286–1297, 2010. http://dx.doi.org/10.1016/j.infsof.2010.07.001

[28] A. Nayak and D. Samanta, "Synthesis of Test Scenarios Using UML Activity Diagrams", *Software and Systems Modeling*, vol. 10, no. 1, pp. 63–89, 2011. http://dx.doi.org/10.1007/s10270-009-0133-4

[29] A. K. Jena *et al.*, "A Novel Approach for Test Case Generation from UML Activity Diagram", in *Proc. of the 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014, pp. 621–629. http://dx.doi.org/10.1109/ICICICT.2014.6781352

[30] P. Runeson and B. Regnell, "Derivation of an Integrated Operational Profile and Use Case Model", in *Proc. of the Ninth International Symposium on Software Reliability Engineering*, 1998, pp. 70–79.
http://dx.doi.org/10.1109/ISSRE.1998.730843

[31] S. Garg *et al.*,"Analysis of Software Rejuvenation Using Markov Regenerative Stochastic Petri Net", in *Proc. of the International Symposium on Software Reliability Engineering*, 1995.
http://dx.doi.org/10.1109/ISSRE.1995.497656

[32] J. P. Bowen *et al.*, *ZUM '98: The Z Formal Specification Notation*, Springer Berlin Heidelberg, 1998.

[33] Z. Jiang *et al.*, *An Improved Reliability Testing Model Based on SOFL*, Springer, 2017.

[34] P. M. Maurer, "The Design and Implementation of a Grammar-Based Data Generator", *Software-Practice and Experience*, vol. 22, no. 3, pp. 223–244, 1992.

[35] X. Li *et al.*, "Consistency Checking of UML Requirements", in *Proc. of the 10th IEEE International Conference on Engineering of Complex Computer System*, 2005, pp. 411–420.
http://dx.doi.org/10.1109/ICECCS.2005.28

[36] M. Lohmann *et al.*, "Executable Visual Contracts", in *Proc. of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computin*, 2005, pp. 63–70.
http://dx.doi.org/10.1109/VLHCC.2005.35

[37] X. Tian *et al.*, "Web Service Reliability Test Method Based on Log Analysis", in *Proc. of the 2017 IEEE International Conference on Software Quality, Reliability and Security Companion*, 2017, pp. 195–199.
http://dx.doi.org/10.1109/QRS-C.2017.38

[38] M. Johansson and T. Olofsson, "Bayesian Model Selection for Markov, Hidden Markov, and Multinomial Models", *IEEE Signal Processing Letters*, vol. 14, no. 2, pp. 1291–132, 2007.
http://dx.doi.org/10.1109/LSP.2006.882094

[39] A. Pievatolo *et al.*, "A Bayesian Hidden Markov Model for Imperfect Debugging", *Reliability Engineering & System Safety*, vol. 103, pp. 11–21, 2012.
http://dx.doi.org/10.1016/j.ress.2012.03.003

[40] S. Assoudou and B. Essebbar, "A Bayesian Model for Markov Chains via Jeffrey's Prior", *Communications in Statistics - Theory and Methods*, vol. 32, no. 11, pp. 2163–2184, 2003.
http://dx.doi.org/10.1081/STA-120024474

[41] K. S. Kumar and R. B. Misra, "Software Operational Profile Based Test Case Allocation Using Fuzzy Logic", *International Journal of Automation and Computing*, vol. 4, no. 4, pp. 388–395, 2007.

[42] C. G. Bai *et al.*, "Bayesian Network Based Software Reliability Prediction with an Operational Profile", *Journal of Systems & Software*, vol. 77, no. 2, pp. 103–112, 2005.
http://dx.doi.org/10.1016/j.jss.2004.11.034

[43] T. He and H. Miao, "Modeling and Composition of Web Application Components using Extended FSM", in *Proc. of the 2008 Fourth International Conference on Natural Computation*, 2008, pp. 363–368.
http://dx.doi.org/10.1109/ICNC.2008.889

[44] A. M. Mostafa *et al.*, "Toward a Formalization of UML2.0 Metamodel using Z Specifications", in *Proc. of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, 2007, pp. 694–701.
http://dx.doi.org/10.1109/SNPD.2007.508

[45] F. Huan *et al.*, "Review of Reliability Analysis Based on Petri Nets", *Computer Science*, 2014. (In Chinese)

[46] C. Xiaolin, "A Summary of the Automatic Generation of Test Cases Based on UML Model", *Modern Computer*, vol. 7, 2018. (In Chinese)

[47] L. Yi *et al.*, "A Conversion Method from UML Model to Reliability Analysis Model", *Journal of Software*, vol. 2, pp. 287–304, 2010. (In Chinese)

[48] J. Yan *et al.*, "Deriving Software Markov Chain Usage Model from UML Models", *Journal of Software*, vol. 16, no. 8, 2005.
http://dx.doi.org/10.1360/jos161386

[49] V. Garousi *et al.*, "A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation", *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 778–797, 2010.
http://dx.doi.org/10.1109/TSE.2010.5

[50] A. T. Endo *et al.*, "Evaluating Test Suite Characteristics, Cost, and Effectiveness of FSM-Based Testing Methods", *Information and Software Technology*, vol. 55, no. 6, pp. 1045–1062, 2013.
https://doi.org/10.1016/j.infsof.2013.01.001

[51] A. L. Bonifacio *et al.*, "Model Partitions and Compact Test Case Suites", *International Journal of Foundations of Computer Science*, vol. 23, no. 1, pp. 147–172, 2012.

[52] V. Santiago *et al.*, "An Environment for Automated Test Case Generation from Statechart-Based and Finite State Machine-Based Behavioral Models", in *Proc. of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, 2008, pp. 63–72.
http://dx.doi.org/10.1109/ICSTW.2008.7

[53] A. Kalaji *et al.*, "A Search-Based Approach for Automatic Test Generation from Extended Finite State Machine (EFSM)", in *Proc. of the 2009 Testing: Academic and Industrial Conference*

– *Practice and Research Techniques*, 2009, pp. 131–132.
http://dx.doi.org/10.1109/TAICPART.2009.19

[54] A. S. Kalaji *et al.*,"An Integrated Search-Based Approach for Automatic Testing from Extended Finite State Machine (EFSM) Models", *Information & Software Technology*, vol. 53, no. 12, pp. 1297–1318, 2011.
http://dx.doi.org/10.1016/j.infsof.2011.06.004

[55] J. J. Li and W. E. Wong, "Automatic Test Generation from Communicating Extended Finite State Machine (CEFSM)-Based Models", in *Proc. of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002*, 2002, pp. 181–185.
http://dx.doi.org/10.1109/ISORC.2002.1003693

[56] J. C. Huang, "An Approach to Program Testing", *ACM Computing Surveys*, vol. 7, no. 3, pp. 113–128, 1975.
http://dx.doi.org/10.1145/356651.356652

[57] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines", *IEEE Transactions on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, 1978.
http://dx.doi.org/10.1109/TSE.1978.231496

[58] W. E. Howden, "Methodology for the Generation of Program Test Data," *IEEE Transactions on Computers*, vol. C-24, no. 5, pp. 554–560, 1975.
http://dx.doi.org/10.1109/T-C.1975.224259

[59] S. Pimont and J. C. Rault, "A Software Reliability Evaluation Based on a Structural and Behavioral Analysis of Programs", in *Proc. of the 2nd Int'l Conf. on Software Engineering. San Francisco: IEEE Computer Society Press*, 1976, pp. 486–491.
http://www.informatik.uni-trier.de/~ley/db/conf/icse/icse76.html

[60] S. Fujiwara *et al.*, "Test Selection Based on Finite State Models", *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 591–603, 1991.
http://dx.doi.org/10.1109/32.87284

[61] D. P. Sidhu and T. Leung, "Formal Methods for Protocol Testing: A Detailed Study", *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 413–426, 1989.
http://dx.doi.org/10.1109/32.16602

[62] L. Pan *et al.*, "DFSM-Based Minimum Test Cost Transition Coverage Criterion", *Journal of Software*, vol. 22, no. 7, pp. 1457–1474, 2011.
http://dx.doi.org/10.3724/SP.J.1001.2011.03872

[63] K. Supaporn and R. Wanchai, "Automated-Generating Test Case Using UML State Chart Diagrams", in *Proc. of the SAICSIT*, 2003, pp. 296–300.

[64] P. Samuel and A. T. Joseph, "Test Sequence Generation from UML Sequence Diagrams", in *Proc. of the Ninth ACIS International Conference on Software Engineering*, 2008.

[65] J. Wen *et al.*, "Research on Regression Test Case Generation Based on UML Sequence Diagram", *Integration Technology*, vol. 2, no. 3, pp. 75–78, 2013. (In Chinese)

[66] E. G. Cartaxo *et al.*, "Test Case Generation by Means of UML Sequence Diagrams and Labeled Transition Systems", in *Proc. of the 2007 IEEE International Conference on Systems, Man and Cybernetics*, 2007, pp. 1292–1297.
http://dx.doi.org/10.1109/ICSMC.2007.4414060

[67] A. J. Offutt *et al.*, "Criteria for Generating Specification-Based Tests", in *Proc. of the Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'99) (Cat. No.PR00434)*, 1999, pp. 119–129.
http://dx.doi.org/10.1109/ICECCS.1999.802856

[68] H. Li *et al.*, "Software Reliability Metrics Selecting Method Based on Analytic Hierarchy Process", in *Proc. of the 2006 Sixth International Conference on Quality Software (QSIC'06)*, 2006, pp. 337–346.
http://dx.doi.org/10.1109/QSIC.2006.59

[69] S. He *et al.*, "Experience Report: System Log Analysis for Anomaly Detection", in *Proc. of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 207–218.
http://dx.doi.org/10.1109/ISSRE.2016.21

[70] K. D. Goseva-Popstojanova, "A New Markov Model of N Version Programming Systems", in *Proc. of the 1991 International Symposium on Software Reliability Engineering*, 1991, pp. 210–215.

[71] N. Yunhui *et al.*, "Research on Software Reliability Evaluation and Test Methods", *Reliability and Environmental Testing of Electronic Products*, 2009, 27 (z1). (In Chinese)

[72] C. Mengtian and Z. Yi-Cheng, "Application of Markov Chain Approach for Multi-Attributes Dynamic Software Reliability Evaluation Under Both AHP and Gray Correlation Methods", *International Journal of Modern Physics*, 2018.

[73] Y. Dai *et al.*, "Uncertainty Analysis in Software Reliability Modeling by Bayesian Analysis with Maximum-Entropy Principle", *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 781–795, 2007.
http://dx.doi.org/10.1109/TSE.2007.70739

[74] S. A. Sherer, "A Cost-Effective Approach to Testing", *IEEE Software*, vol. 8, no. 2, pp. 34–40, 1991.
http://dx.doi.org/10.1109/52.73747

[75] F. Wei, "An Automatic Generation Method of Transfer Probability of Markov Chain Model", Beijing: China Science and Technology Paper Online, 2007. (In Chinese)

[76] G. S. Semmel and D. G. Linton, "Determining Optimal Testing Times for Markov Chain Usage

Models [Software Testing]", in *Proc. of the IEEE Southeastcon '98 'Engineering for a New Era'*, 1998, pp. 1–4. http://dx.doi.org/10.1109/SECON.1998.673276

[77] Y. Jiong *et al.*, "Software Statistical Test Acceleration Based on Importance Sampling", *Computer Engineering and Science*, vol. 3, pp. 64–66, 2005. (In Chinese)

[78] J. Zeng *et al.*, "A Prototype System of Software Reliability Prediction and Estimation", in *Proc. of the 2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, 2010, pp. 558–561. http://dx.doi.org/10.1109/IITSI.2010.90

[79] C. Hsu and C. Huang, "Optimal Weighted Combinational Models for Software Reliability Estimation and Analysis", *IEEE Transactions on Reliability*, vol. 63, no. 3, pp. 731–749, 2014. http://dx.doi.org/10.1109/TR.2014.2315966

[80] N. Ullah *et al.*, "Selecting the Best Reliability Model to Predict Residual Defects in Open Source Software", *Computer*, vol. 48, no. 6, pp. 50–58, 2015.

[81] U. Najeeb, "A Method for Predicting Open Source Software Residual Defects", *Software Quality Journal*, vol. 23, no. 1, pp. 55–76, 2015.

[82] H. Chen, "Analysis and Comparison of Reliability Models Based on Software Architecture", in *Proc. of the 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, 2016, pp. 359–362. http://dx.doi.org/10.1109/ICOACS.2016.7563115

[83] H. K. Weiss, "Estimation of Reliability Growth in a Complex System with a Poisson-Type Failure", *Operations Research*, vol. 4, no. 5, pp. 532–545, 1956.

[84] J. D. Musa *et al.*, "Software Reliability – Measurement, Prediction, Application", Software Reliability: Measurement, Prediction, Application, 1987.

[85] J. McCall *et al.*, "Software Reliability Measurement and Testing Guidebook", Technical Report RLTR-92-52, Rome Laboratory USAF, 1992.

[86] L. Minyan, *Software Reliability Engineering*, Beijing: National Defense Industry Press, 2011, pp. 282–296. (In Chinese)

[87] N. F. Schneidewind, "Reliability Modeling for Safety-Critical Software", *IEEE Transactions on Reliability*, vol. 46, no. 1, pp. 88–98, 1997. http://dx.doi.org/10.1109/24.589933

[88] J. G. Shanthikumar, "A General Software Reliability Model for Performance Prediction", *Microelectronics Reliability*, vol. 21, no. 5, pp. 671–682, 1981.

[89] M. L. Shooman, "Structural Models forSoftware Reliability Prediction", in *Proc. of the International Conference on Software Engineering*, 1976.

[90] J. D. Musa and K. Okumoto, "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement", *International Conference on Software Engineering*, 1984.

[91] S. Yamada *et al.*, "S-Shaped Reliability Growth Modeling for Software Error Detection", *IEEE Transactions on Reliability*, vol. R-32, no. 5, pp. 475–484, 1983. http://dx.doi.org/10.1109/TR.1983.5221735

[92] K. M. Manjunatha and K. Harishchandra, "Modeling and Statistical Inference on Generalized Inverse Exponential Software Reliability Growth Model", *Far East Journal of Theoretical Statistics*, vol. 39, no. 1, pp. 67–77, 2012.

[93] J. T. Duane, "Learning Curve Approach to Reliability Monitoring", *IEEE Transactions on Aerospace*, vol. 2, no. 2, pp. 563–566, 1964. http://dx.doi.org/10.1109/TA.1964.4319640

[94] S. Tiwari and A. Gupta, "A Systematic Literature Review of Use Case Specifications Research", *Information & Software Technology*, vol. 67, no. 2, pp. 128–158, 2015.

[95] M. Utting *et al.*, "Chapter Two – Recent Advances in Model-Based Testing", *Advances in Computers*, vol. 101, pp. 53–120, 2016.

[96] W. Erwei, "Review of Software Reliability Model Research", *Software Engineering*, no. 2, pp. 1–2, 2016. (In Chinese)

[97] C. Ş. Gebizli and H. Sözer, "Automated Refinement of Models for Model-Based Testing Using Exploratory Testing", *Software Quality Journal*, vol. 25, no. 3, pp. 1–27, 2016.

[98] P. Liu and Z. Xu, "MTTool: A Tool for Software Modeling and Test Generation", *IEEE Access*, vol. 6, pp. 56222–56237, 2018. http://dx.doi.org/10.1109/ACCESS.2018.2872774

[99] H. Hemmati *et al.*, "Achieving Scalable Model-Based Testing Through Test Case Diversity", *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 1, pp. 1–42, 2013.

[100] W. H. Tseng and C. F. Fan, "Systematic Scenario Test Case Generation for Nuclear Safety Systems", *Information & Software Technology*, vol. 55, no. 2, pp. 344–356, 2013.

[101] C. S. Gebizli and H. Sözer, "Model-Based Software Product Line Testing by Coupling Feature Models with Hierarchical Markov Chain Usage Models", in *Proc. of the 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2016, pp. 278–283. http://dx.doi.org/10.1109/QRS-C.2016.42

[102] K. Y. Cai *et al.*, "A Critical Review on Software Reliability Modeling", *Reliability Engineering*

& System Safety, vol. 32, no. 3, pp. 357–371, 1991.

[103] Sharma *et al.*, *Computer Systems Engineering*, Springer, 2014.

[104] J. D. Musa and K. Okumoto, "Software Reliability Models: Concepts, Classification, Comparisons, and Practice", *Electronic Systems Effectiveness and Life Cycle Costing*, 1983.

[105] Q. P. Hu *et al.*, "Robust Recurrent Neural Network Modeling for Software Fault Detection and Correction Prediction", *Reliability Engineering and System Safety*, vol. 92, no. 3, pp. 332–340, 2007.

[106] P. Kumar and Y. Singh, "An Empirical Study of Software Reliability Prediction Using Machine Learning Techniques", *International Journal of System Assurance Engineering and Management*, vol. 3, no. 3, pp. 194–208, 2012.

[107] C. Zhang *et al.*, "A Review of Software Reliability Growth Models", *Journal of Software*, vol. 9, 2017. (In Chinese)

[108] C. Huang and M. R. Lyu, "Estimation and Analysis of Some Generalized Multiple Change-Point Software Reliability Models", *IEEE Transactions on Reliability*, vol. 60, no. 2, pp. 498–514, 2011. http://dx.doi.org/10.1109/TR.2011.2134350

[109] Y. Minamino *et al.*, "NHPP-Based Change-Point Modeling for Software Reliability Evaluation and its Application to Software Development Management", *Annals of Operations Research*, vol. 244, no. 1, pp. 85–101, 2016.

[110] K. Y. Song *et al.*, "An NHPP Software Reliability Model with S-Shaped Growth Curve Subject to Random Operating Environments and Optimal Release Time", *Applied Sciences*, vol. 7, no. 12, p. 1304, 2017.

[111] P. K. Kapur *et al.*, "A Unified Approach for Developing Software Reliability Growth Models in the Presence of Imperfect Debugging and Error Generation", *IEEE Transactions on Reliability*, vol. 60, no. 1, pp. 331–340, 2011. http://dx.doi.org/10.1109/TR.2010.2103590

[112] Q. Li and H. Pham, "NHPP Software Reliability Model Considering the Uncertainty of Operating Environments with Imperfect Debugging and Testing Coverage", *Applied Mathematical Modelling*, vol. 51, pp. 68–85, 2017.

[113] F. Brosch *et al.*, "Architecture-Based Reliability Prediction with the Palladio Component Model", *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1319–1339, 2012. http://dx.doi.org/10.1109/TSE.2011.94

[114] C. Jin and S. W. Jin, "Software Reliability Prediction Model Based on Support Vector Regression with Improved Estimation of Distribution Algorithms", *Applied Soft Computing Journal*, vol. 15, no. 2, pp. 113–120, 2014.

[115] D. Cotroneo *et al.*, "RELAI Testing: A Technique to Assess and Improve Software Reliability", *IEEE Transactions on Software Engineering*, vol. 42, no. 5, pp. 452–475, 2016. http://dx.doi.org/10.1109/TSE.2015.2491931

[116] P. Roy *et al.*, *Neuro-Genetic Approach on Logistic Model Based Software Reliability Prediction*, Pergamon Press, Inc. 2015.

[117] J. Wang and C. Zhang, "Software Reliability Prediction Using a Deep Learning Model Based on the RNN Encoder-Decoder", *Reliability Engineering & System Safety*, 2017.

[118] X. Li *et al.*, "Reliability Analysis and Optimal Version-Updating for Open Source Software", *Information and Software Technology*, vol. 53, no. 9, pp. 929–936, 2011.

[119] G. Y. Park and S. C. Jang, "A Software Reliability Estimation Method to Nuclear Safety Software", *Nuclear Engineering and Technology*, vol. 46, no. 1, pp. 55–62, 2014.

[120] J. M. Zhang *et al.*, "Machine Learning Testing: Survey, Landscapes and Horizons", 2019.

[121] T. Y. Chen *et al.*, "Metamorphic Testing: A New Approach for Generating Next Test Cases", Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology, Tech. Rep., 1998.

[122] W. M. McKeeman, "Differential Testing for Software", *Digital Technical Journal*, vol. 10, no. 1, pp. 100–107, 1998.

[123] V. Le *et al.*,"Compiler Validation Via Equivalence Modulo Inputs", *ACM SIGPLAN Notices*, vol. 49, pp. 216–226, 2014.

[124] A. Avizienis, "The Methodology of N-Version Programming", *Software Fault Tolerance*, vol. 3, pp. 23–46, 1995.

[125] S. Segura *et al.*, "A Survey on Metamorphic Testing", *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016. http://dx.doi.org/10.1109/TSE.2016.2532875

[126] T. Y. Chen *et al.*, "Metamorphic Testing: A Review of Challenges and Opportunities", *ACM Computing Surveys*, vol. 51, no. 1, pp. 4:1–4:27, 2018.

[127] X. Xie *et al.*, "Application of Metamorphic Testing to Supervised Classifiers", in *Proc. of the 2009 Ninth International Conference on Quality Software*, 2009, pp. 135–144. http://dx.doi.org/10.1109/QSIC.2009.26

[128] X. Xie *et al.*, "METTLE: A METamorphic Testing Approach to Assessing and Validating Unsupervised Machine Learning Systems", *IEEE*

Transactions on Reliability, vol. 69, no. 4, pp. 1293–1322, 2020. http://dx.doi.org/10.1109/TR.2020.2972266

[129] Z. Q. Zhou and L. Q. Sun, "Metamorphic Testing of Driverless Cars", *Communications of the ACM*, vol. 62, no. 3, pp. 61–67, 2019.

[130] J. Guo *et al.*, "Coverage Guided Differential Adversarial Testing of Deep Learning Systems", *IEEE Transactions on Network Science and Engineering*. http://dx.doi.org/10.1109/TNSE.2020.2997359

[131] Y. Sun *et al.*, "Concolic Testing for Deep Neural Networks", in *Proc. of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering*, 2018, pp. 109–119. http://dx.doi.org/10.1145/3238147.3238172

[132] Y. Sun *et al.*, "DeepConcolic: Testing and Debugging Deep Neural Networks", in *Proc. of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings*, 2019, pp. 111–114. http://dx.doi.org/10.1109/ICSE-Companion.2019.00051

[133] M. Lei *et al.*, "DeepGauge: Comprehensive and Multi-Granularity Testing Criteria for Gauging the Robustness of Deep Learning Systems", 2018.

[134] Z. Zhang and X. Xie, "On the Investigation of Essential Diversities for Deep Learning Testing Criteria", in *Proc. of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, 2019, pp. 394–405. http://dx.doi.org/10.1109/QRS.2019.00056

[135] Y. Sun *et al.*, "Structural Test Coverage Criteria for Deep Neural Networks", *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5, pp. 1–23, 2019. https://doi.org/10.1109/ICSE-Companion.2019.00134

[136] M. Z. Iqbal *et al.*, "Applying UML/MARTE on Industrial Projects: Challenges, Experiences, and Guidelines", *Software & Systems Modeling*, 2015. http://dx.doi.org/10.1007/s10270-014-0405-5

[137] M. Shin *et al.*, "Analyzing Dynamic Behavior of Large-Scale Systems Through Model Transformation", *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 1, pp. 35–60, 2005. http://dx.doi.org/10.1142/S0218194005001896

[138] S. Preibusch and F. Kammüller, "Checking the TWIN Elevator System by Translating Object-Z to SMV", *International Workshop on Formal Methods for Industrial Critical Systems Springer*, Berlin, Heidelberg, 2007.

[139] A. Rasoolzadegan *et al*., "Reliable Yet Flexible Software Through Formal Model Transformation (Rule Definition)", *Knowledge and Information Systems*, vol. 40, pp. 79–126, 2014. https://doi.org/10.1007/s10115-013-0621-2

[140] J. H. Poore *et al.*, "A Constraint-Based Approach to the Representation of Software Usage Models", *Information and Software Technology*, vol. 42, no. 12, pp. 825–833, 2000. https://doi.org/10.1016/S0950-5849(00)00101-4

[141] F. Zhen and C. Peng, "A System Test Methodology Based on the Markov Chain Usage Model", in *Proc. of the 8th International Conference on Computer Supported Cooperative Work in Design*, Xiamen, China, 2004, pp. 160–165.

[142] H. Le Guen *et al.*, "Reliability Estimation for Statistical Usage Testing Using Markov Chains", in *Proc. of the 15th International Symposium on Software Reliability Engineering*, 2004, pp. 54–65. http://dx.doi.org/10.1109/ISSRE.2004.33

[143] B. Littlewood and J. L. Verrall, "A Bayesian Reliability Growth Model for Computer Software", *Journal of the Royal Statistical Society*, vol. 22, no. 3, pp. 332–346, 1973.

[144] B. B. Sagar *et al.*, "Exponentiated Weibull Distribution Approach Based Inflection S-Shaped Software Reliability Growth Model", *Ain Shams Engineering Journal*, 2015. https://doi.org/10.1016/j.asej.2015.05.009

[145] Z. Xiaonan *et al.*, "A New Method on Software Reliability Prediction", *Mathematical Problems in Engineering*, 2013. http://dx.doi.org/10.1155/2013/385372

[146] M. D. Davis and E. J. Weyuker, "Pseudo-Oracles for Non-Testable Programs", in *Proc. of the ACM 81 Conference*, pp. 254–257, 1981. http://dx.doi.org/10.1145/800175.809889

[147] L. Ma *et al.*, "DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems", in *Proc. of the 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 614–618. http://dx.doi.org/10.1109/SANER.2019.8668044

[148] S. Gerasimou *et al.*, "Importance-Driven Deep Learning System Testing", in *Proc. of the 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceeding*, 2020, pp. 322–323.

Received: December 2020
Revised: April 2021
Accepted: April 2021

Contact addresses:
Zhouxian Jiang
Beijing Jiaotong University
China
e-mail: zhouxianjiang@bjtu.edu.cn

Honghui Li
Beijing Jiaotong University
China
e-mail: hhli@bjtu.edu.cn

Dalin Zhang
Beijing Jiaotong University
China
e-mail: dalin@bjtu.edu.cn

Rui Wang
Beijing Jiaotong University
China
e-mail: rui.wang@bjtu.edu.cn

Junwen Zhang
Beijing Jiaotong University
China
e-mail: zjw@bjtu.edu.cn

Xiuru Li
Beijing Jiaotong University
China
e-mail: 17120462@bjtu.edu.cn

Meng Zhang
Beijing Jiaotong University
China
e-mail: 18120468@bjtu.edu.cn

Penghao Wang
Beijing Jiaotong University
China
e-mail: wangpenghao@bjtu.edu.cn

Zhouxian Jiang was born in Guangxi province, China in 1996. She received a BSc degree incomputer engineering from Beijing Jiaotong University, Beijing, China, in 2017. She is currently pursuing a PhD in software engineering at Beijing Jiaotong University, Beijing, China. Since 2017, she has worked in the Software Evaluation Laboratory of Beijing Jiaotong University. Her research interests include deep learning testing, software reliability testing, formal methods, and in particular the application of formal methods to deep learning testing. Ms Jiang has been engaged in software testing for 3 years, and has participated in more than 10 software evaluation projects, research projects, as well as National Key Research and Development Projects (China).

Honghui Li received her MSc degree in computer science from the Central South University, Changsha, China, in 1987. Her research interests include software testing technology and testautomation. Currently, she is a Professor at the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. She is currently the Deputy Director of the Engineering Research Center of Network Management Technology for High Speed Railway of MOE, Beijing, China. She has long been engaged in software quality assurance technology, high-reliability software, data mining analysis, and railway information technology. She has undertaken national-level provincial and ministerial research such as the National 863 Program, the Nuclear High-Level Project, and the China Railway Corporation.

Dalin Zhang received the B.E. degree in computer science, the MSc degree in science, and the PhD degree in computer science, all from the Beijing University of Posts and Telecommunications, in 2008, 2010, and 2014, respectively. In 2017, he was a postdoctoral researcher with the School of Electronics and Computer Engineering, Purdue University, USA. He is currently an Associat Professor of computer science and software engineering at Beijing Jiaotong University. His current research interests include railway information technology, software engineering, and information security. His research focuses on developing applications of program analysis and software testing for improving software reliability, security, and performance, as well as software engineering and data mining and programming languages.

Rui Wang received the B.E. degree (2011) and M.E. degree (2014) in electronic and information engineering from Beijing Jiaotong University, Beijing, China, and the PhD degree (2018) in computer science from the Institut National des Sciences Appliquées de Toulouse, France. She did her doctoral research work in the dependable computing and fault tolerance group of the Laboratory for Analysis and Architecture of Systems, French National Centre for Scientific Research, Toulouse, France. From 2018 to 2020, she was engaged in a postdoc research at Beijing Jiaotong University. Since 2020, she has been an Assistant Professor at the Computer and Information Technology School in the same university. Her main fields of interest include quality assurance and testing for artificial intelligence systems, robustness of machine learning algorithms, dependability assessment of safety critical systems, and quantitative measurement of uncertainty.

Junwen Zhang received the B.E. degree (1988) and M.E. degree (1997) from Beijing Jiaotong University, China, and the PhD degree (2000) in electronic and information engineering from the same institution. Currently, he is an Associate Professor at the School of Computer and Information Technology, Beijing Jiaotong University, China. Since 2000 Dr. Zhang has participated in a number of scientific research projects, including railway information sharing cloud computing application technology, rail transit network verification and testing laboratory platform construction. His main research directions are software testing, information security, artificial intelligence.

Xiuru Li received the MSc degree in software engineering from Beijing Jiaotong University, China in 2020. He has been engaged in software development for four years. His area of research is mainly in the application of machine learning in railway systems.

Meng Zhang received the BEng degree (2017) in computer engineering from Beijing Jiaotong University, China. He is currently an MSc student of software engineering at Beijing Jiaotong University, China. Since 2018, his research includes information security technology, software testing technology, especially in the field of test automation, and the automatic generation of test data.

Penghao Wang received an MSc (2020) degree in software engineering from Beijing Jiaotong University, China. He is now working as a test and development engineer at the China Electronics Technology Cloud. His work domain includes Platform-as-a-Service (PaaS) platform software automation testing and engineering efficiency improvement.