International Journal of Scientific Research in Science, Engineering and Technology
© 2019 IJSRSET | Volume 6 | Issue 2 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
doi : https://doi.org/10.32628/IJSRSET2182134

Automated Software Testing Analysis on Test Case Optimization and Test Case Levels on Java and Python

Ranjeet Kumar[1], Prof. Dr. Ramdip Prasad[2], Arif Md. Sattar[3*]

[1]Department of Computer Science, Magadh University, Bodh Gaya, India

[2]Department of Mathematics, Danapur College, Patna, India

[3]Department of Computer Science, Anugrah Memorial College, Gaya, India

ABSTRACT

In today's world, object-oriented programming (OOP) languages are commonly employed in software development. The architecture and design of the Integrated Development Environment (IDE) models is constantly updated and modified to facilitate programming practice, keeping in mind the popularity of OOP languages. However, a huge percentage of syntactic and logical errors are caused by complicated programming structures and the inherent complexity of underlying applications. Although syntax issues are easier to identify during compilation, the majority of IDEs fail to detect logical flaws, resulting in not just debugging costs but also milestone delays. The goal of this study is to provide a new design model for IDEs that will allow them to detect logical mistakes. This will make it easier for developers to spot logical problems sooner in the development process, resulting in reduced development time and on-time milestone delivery. The proposed model will be implemented as a working Python IDE application and compared to other current state-of-the-art products on the market. The amount of logical mistakes found will be used to assess the proposed model's correctness. Python was chosen because of its broad application in Artificial Intelligence and Data Science. Machine learning approaches have been used to the process of student modelling, particularly in the development of tutors to help students learn to programme. These were created for a variety of languages (Pascal, Prolog, Lisp, C++), as well as programming paradigms (procedural and declarative), but never for Java object-oriented programming. Using inconsistencies between a student and the right software, JavaBugs automatically creates a bug library. While other studies look at code snippets or UML diagrams to infer student knowledge of object-oriented design and programming, JavaBugs looks at a complete Java programme and finds the most similar correct programme to the student's solution among a collection of correct solutions, then uses similarity measures and background knowledge to build trees of misconceptions. Experiments demonstrate that JavaBugs can find the most comparable correct programme 97 percent of the time, as well as discover and detect 61.4 percent of expert-identified student mistakes.

Keywords : Automated Software Testing, Test Case Optimization, Object Oriented Programming, Integrated Development Environment. Logical Flaws, Inherent Complexity, Syntax Issues.

I. INTRODUCTION

Python has grown in popularity as a tool for implementing artificial intelligence and data science in a variety of disciplines. As a result, many efforts have been made to make it easier for Python developers to code their programmes. One of the most important stages toward assisting Python developers, especially when it comes to detecting problems, is the development of user-friendly IDEs. Computer scientists are developing automated error recovery systems that can detect and correct programming faults during development.

As a result, clever compilers that can detect and highlight syntax problems are now on the market. B. K. Daniel, B. K. Xie, B. K. Xie, B. K. Xie, B. K. et al., (2018). The key issue, however, is to discover logical faults in the coding technique (Colapinto, C., 2017) and to predict the present instructions' input and output. In Python, the intelligent recovery unit is used to detect logical flaws. It will be a programming language environment upgrade that will help detect logical problems in addition to syntactic faults (Beltramelli, et al.,2018). The major goal is to detect logical problems early in the development process rather than later. The benefits of identifying logical faults were highlighted by an intelligent unit. New programmers are often unaware of the logical faults that might occur during the development of an application, and identifying and correcting logical errors is a difficult task. G. Samara is the author of this article et al., (2017). They devote time and money to learn the fundamentals of mistake detection strategies. T. J. Ketschau, T. J. Ketschau, T. J. Ketschau et al., (2019). Every year, millions of dollars are squandered in the search for the same logical flaw. Every existing computer language contains logical mistakes, which are corrected after detection improves (Mongkhonvanit et al., 2018). Even Nevertheless, there is a lot of discussion on logical error codes in various communities. However, they are difficult to comprehend and are occasionally located in a separate programming language, which has an impact on the entire technological development process. E. D. Berger, C. Hollenbeck, P. Maj, O. Vitek, and J. Vitek et al., (2019).

Any other part of resolving logical errors is companies that have developed their own solutions and have not made them available to the general public. M. Wyrich et al., (2019). The intelligent unit will be open to any developer and will share its database for correcting logical flaws. As a result, any developer can benefit from its development. There will be no requirement for extensive coding skills. Every piece of code must adhere to the constraints imposed by the programming language. K. Deulkar, K. Deulkar, K. Deulkar, K et al., (2016). Machine language can be used to comprehend the code, but the logical error is more complicated than coding laws and regulations. The IT industry is facing major calamities as a result of logical mistakes. For example, NASA missions have frequently failed due to logical mistakes, and some space missions have been severely disrupted, resulting in human deaths. It is a logic flaw between software and hardware capacity that cannot be enforced on the system. Furthermore, by coding more than physical components, the logical error can be corrected. J. Lee et al., (2018).

As a result, it is a simple and inexpensive method of resolving logical mistakes. It has a low cost and saves time over seeking a solution to the same problem repeatedly. The intelligent unit, which emerges from the coding environment and executes independently, will be introduced. If the developer chooses to use the solution, it will issue a warning, make a suggestion, and automatically correct the fault. Intelligent units will be designed in accordance with the environment and architecture in which they will operate. However, just one IDE is currently under consideration.

Artificial Intelligence in Education (AIED) has progressed as a field of study over time. It has resulted in major, practical solutions to problems in the creation of intelligent software for educational support. The difficulty of the student modelling challenge appears to have piqued academics' curiosity among the various problems covered by the area [10]. Advances in artificial intelligence techniques, notably machine learning, have led to the development of self-improving student models ([1], [3], [4], [11]), that is, a student model that can automatically update its knowledge based on observed student behaviour. Building a student model can be done in a variety of ways. The goal of this project is to create a bug library for newbie Java programmers. A bug library is a collection of commonly made mistakes and misunderstandings. The complexity and cost of building a bug library are the two most significant challenges. By creating self-improving bug libraries, ASSERT [3] and MEDD [11] addressed these concerns. Machine learning techniques were utilised to generate and extend the bug library automatically. Automatic Bug Library Construction for Object-Oriented Novices 185 Only a few ITSs (hence, student modellers) have been built in the age of object-oriented programming. This could be because the challenging task of student modelling for programming is made even more difficult by the complex process of object-oriented programming. While there have been studies on Java programming faults ([5], [6], [12]), they have primarily focused on typical syntax errors and compilation behaviour, rather than learning object-oriented ideas and programming. In order to study objectoriented programming, it is necessary to go beyond reviewing student syntax errors. To accomplish this, programming solutions must be evaluated in light of the student's goals [7]. A student's intended strategy for solving any programming challenge is called an intention. Intention-based diagnosis also identifies one accurate solution against which the student's solution can be compared. Because there are many correct programming solutions, this is critical when evaluating them. This strategy is used in ([11] and [9]). The latter, in particular, assesses student object-oriented programming in Eiffel solutions. It determines the disparities between a student's purpose and their solution. The meaning of these differences is subsequently deduced by the human expert. JavaBugs is the subject of this paper. Using the multistrategy technique employed in MEDD, it automatically creates a bug library of Java rookie programmer errors in object-oriented programming. By automatic, we imply that the bug library can be organised without the assistance of a human expert. The following is how it's laid out: In Sections 2 and 3, the input and output are presented, followed by a discussion of the algorithms for intention identification and discrepancy extraction, as well as misperception detection and discovery. The findings of the tests, as well as their analysis, are reported in Section 4. The paper concludes with a discussion of the work's conclusion and future directions.

II. Literature Review

A method for generating code from an image has been developed by Beltramelli et al., (2018). Every image is not coded, and the biggest disadvantage is that an image design must be created before it can be transformed into code. The use of GUI programming style is a step toward expanding software development possibilities. It produces better results than coding, but coding will continue to be a part of programming. It reduces production time and costs. It makes use of an existing image of an application's GUI, and its best feature is that it reengineers the image and provides the code for customization. Using Android, iOS, and web-based platforms, it boosts productivity by 77 percent. All images are incompatible with the proposed GUI programming methodology. Accepting serious professional development takes a long time. However, a coding environment and a wide range of software development abilities are still required. The programming methodology hasn't changed much, and there are still a lot of errors in the code.

According to Parnianifard et al., (2018), appropriate admittance and procedure of techniques has become more complicated with the passage of time, and their required workforce and resources are quantitatively used to complete. The anonymous setting of various unexpected, sudden change, unmanageable circumstances, various framework changes at moments, handling different responses mixing with each other, contrasting data objects, understanding new languages, and the problem of updating technologies, among other things, are all contributing to a climate of more calculated ramification in the issue. Not all strategies are ineffective, but many of them are.

Mongkhonvanit et al., (2018) have created a tactile interface that is intended to provide practise for new programmers. The computer programming language can be designed by any type of entity and in any context, and computing systems can bend it into a stable form. It uses pre-built functions like any programming languages, but the essential point is that Testudinata is a perfect example of physical inputs and outputs, which is unique. It demonstrates that computing interaction is no longer traditional, as it employs a real-world graphical user interface. However, Testudinata is still in progress and has flaws. It's a novel form of programming error that hasn't been specified before.

B. K. Daniel et al., (2018) highlights the new difficulty of methodology (Data Science) as it relates to the passage of time. Each data-related element at some point (input/output) indicates some new difficulties that are appearing often. These concerns necessitate a new mythos for performance as well as novel approaches to solve complex problems. Many difficulties with technique have been brought to light, prompting a search for a new methodology. With the help of online services, data collection is now relatively simple. To achieve better results, data collecting over the internet has been advocated. However, the nature of questionnaires evolves throughout time in order to obtain high-quality data. The question design is critical because the majority of the survey's focus is on determining the pure evaluation rather than applying filters by the collector. GUI programming methodology has been determined to be necessary for productivity. After some time, a new methodology is required to address all types of issues that arise as a result of the system's use. The most important idea is to update on a regular basis, however an unstable methodology poses a significant risk to the data volume. A poor methodology plan can result in data loss or have an effect on the outcomes. When data integrity is compromised, it is impossible to provide correct and required statistics for data analysis.

Berger, E. D., Hollenbeck, C., Maj, P., Vitek, O., and Vitek, J. et al., (2019) plan to use a programming language to regulate the flow of computation on computing machines. Computer Language assists programmers in archiving computing tasks in the form of software with a stated goal. This study is the result of analysing the characteristics and flaws of various computer languages in terms of their scopes. Even a survey of software development in several programming languages revealed costly research in the range of skills required to build. Each programming language specialist, according to the review, tries to fulfil language physiognomies. A single language code, on the other hand, can be designed for use in multiple programming languages. The git platform keeps track of code modifications for this reason. All languages fail to meet the statistical model's quality requirements.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H.,..., & Ko, A. J. et al., (2019). The most serious issue with programming languages is the version gap between learning and becoming a professional developer. However, the facts show that using code, as well as earlier work, to bond scoped writing codes for a precise answer is quite tough. Each programming language approaches the problem in its own unique way. This demonstrates how adaptable each language is. Most programming languages evolve over time, and they are no longer focused on syntax mistakes, warnings, or logical flaws.

C. Colapinto, R. Jayaraman, and S. Marsiglio The importance of goal programming models, which are characterised as research mechanisms in scientific management, is highlighted in et al., (2017)'s work. Due to the intricacy of model patterns and processes to imitate the goal of one solution, real-world problems cannot be solved. Instead of archiving one or two goals, the Goal Programming approach allows you to archive multiple goals in a comprehensive fashion. Goal programming, on the other hand, is not just employed in engineering but also in other fields of research and management. There is no model that is 100 percent implementable, that provides exact solutions, and that is ideal.

M. Wyrich, D. Graziotin, and S. Wagner's et al., (2019) research focused on simulation-based learning since virtual real-time practise provides greater experience than practise after reading theoretical material. The fundamental feature is that any situation may be projected precisely, therefore developed patterns are extremely useful for precisely determining object qualities. The issue with simulation is that it isn't standardised. Simulator software is developed by each software development organisation. Every business does not focus on everything and places limitations on their software. This work is an excellent example of maximising the use of coding while achieving the greatest results for obtaining substantial information value. However, the most serious simulation issue was highlighted.

T. J. Ketschau and J. Kleinhans et al., (2019) worked on an analytical study report, and the findings reveal that professional coders do not always overcome coding issues. The main reason for this is that firms use various technologies, and programming language updates and upgrades confuse programmers during programming tests. It simply assesses a programmer's skill, yet owing to a variety of issues, numerous excellent programmers have failed to pass. However, this article points to a problem with syntax capabilities, but debugging is more involved in this case.

The downloadable copy of the study paper by Samara, G. et al., (2017) underlined the importance of having a feature to monitor syntax for detecting logical mistakes in computer applications in a software development environment. However, if programming matches action behaviour that appropriately reflects class objects within a scope, it could be a simple task. Logical errors are nothing more than a programming error that is specified outside of its scope. Indirectly, it's a difficulty with undefined mathematical concepts, overloaded phrases, or logic problem terms derived from human perception. Furthermore, logical defects can be detected using error information, which can aid in the future correction of common logical problems. The main focus of the research is on the maintenance of developed applications and the development of software quality. The integrity of logical error-free software development is ensured by logical error principals.

According to Deulkar, K. et al., (2016), a programme cannot be turned into machine code without proper coding, regardless of how the code is written in a structure. With its syntactic standards and regulations, code quality is unimportant. Correcting the code, on the other hand, is an important aspect of software development in order to build authentic computation as a required duty. Software quality is determined by a number of factors, but a logical flaw in the code poses a significant difficulty. Furthermore, syntactical problems are detected earlier than logical faults. The nature of logical mistakes is dependent on the programming language, and there is currently no mature solution.

The black word on this soft copy, authored by Lee, J. et al., (2018), is projecting research on detecting and diagnosing logical faults. The concept is to use machine learning to correct code faults. To begin, use programming data sets to train a model to detect the programming problem. Which is gathered through the use of computer programmes. The study demonstrates a brand new error-correcting algorithm that categorises and corrects faults. However, this study is still in its early phases and will take time to evolve into a viable answer.

JavaBug:

In an intelligent tutoring system, student modelling is the process of approximating a student's knowledge of a lesson. Understanding student behaviour in the software environment is required to "approximate." It compares a student's final solution to a programming exercise to its library of frequent errors and misconceptions to infer the student's knowledge (bug library). Detecting the most similar correct programme (intention expressed as reference programmes), extracting the superficial differences (discrepancies) between the student's and the correct programme, and forming misconception definitions (error hierarchies) described by discrepancies based on similarity and causality heuristics are all part of the automatic bug library construction task.

Input:

A student's programme (in.java format), a knowledge base comprised of the bug library, a set of correct programmes known as reference programmes, and causality heuristics are all inputs to JavaBugs. The bug library contains the most typical mistakes students make when studying Java object-oriented programming. JavaBugs uses domain knowledge to sever nodes in the error hierarchy using causality heuristics. Discrepancies, for example, can exist.

III. Methodology/Research Design

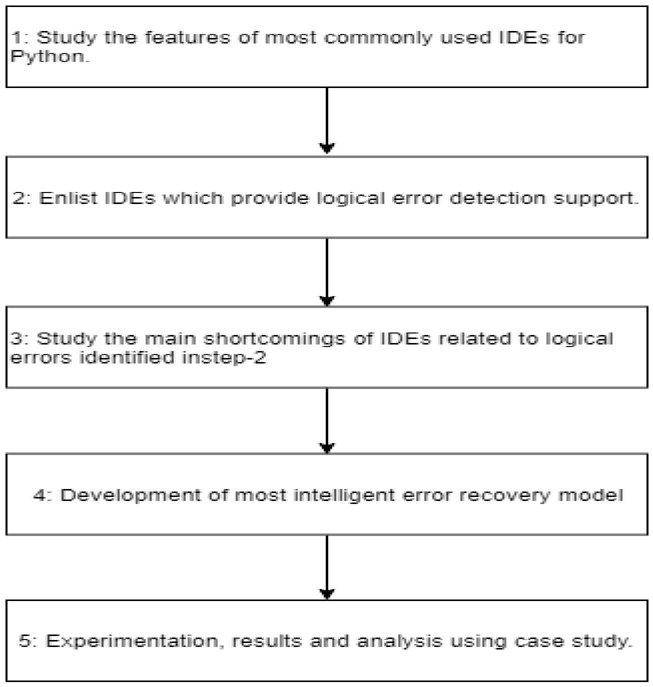
1: Study the features of most commonly used IDEs for Python.
2: Enlist IDEs which provide logical error detection support.
3: Study the main shortcomings of IDEs related to logical errors identified instep-2
4: Development of most intelligent error recovery model
5: Experimentation, results and analysis using case study.

Fig: 1 - Main steps of Research

Step 1: Research the features of the most popular Python IDEs.

In this phase, we'll look at the most important aspects of the most often used IDEs for Python application development. We do this by looking at websites where programmers discuss logical flaws, as well as code examples and setup configurations. We extract logical errors, code, and setup information in great detail. In the initial stage, the primary focus will be on gathering configuration information and identifying IDE names as well as their unique pattern aspects to aid software development. Obtaining information about their error-handling technique and the extent to which they can cover logical faults. Identifying vulnerabilities that prevent a recovery solution from being implemented. One Ide, for example, is very friendly but not smart when it comes to monitoring human programming faults, while another Ide is enhanced when it comes to detecting flaws but is not very user-friendly.

Step 2: Look for IDEs that provide logical error detection.

After conducting research, a list of IDEs will be compiled at this phase. Analyze them to see how capable they are at detecting and resolving errors. Determine the extent of the logical flaw and how it affects the solution. When they are chosen for research, they will proceed to the following round, where each Ide will testify and experiment with various logical error codes acquired during the first stage. Marking the level of ide's performance on each logical error. This will serve as a foundation for the next stage of the research, as well as aid in the definition of parameters for logical flaws in order to restore principals.

Step 3: Research the major flaws in IDEs that are connected to the logical faults discovered in step 2.

The success rate of IDEs that assist developers in resolving programming errors will be measured in this section of the study. They will be chosen based on their ability to detect faults, issue warnings, recover from errors, and provide solutions to problems. To further this research, a mature logical error handler IDE or IDEs will be ideal. That IDE or IDEs that work with logical errors are more useful, keep an eye out for new intelligent logical error recovery units to be created. The data gathered during this phase of study will serve as a foundation for the recovery model.

Step 4: Create the most sophisticated error recovery model possible.

The clever Unit will begin developing a logical error model in the fourth stage of study, which will be experimental. The model's elements and deciding parameters will be adjusted. Defining its theory and method of operation Its entire workflow and component integration. Its subunits, components, and the entire system are tested and validated. After all of the development procedures have been completed. The logical error recovery model will be released for Beta testing and distributed to developers for input.

Experiments, data, and analysis utilising a case study are the fifth and final steps.

After you've completed the logical error model, you can move on to the next step. The information from each step of the model-making process will be recorded. Its findings will be extremely beneficial in the study and enhancement of logical error detection, resolution, and solution suggestions. Comparing research findings to those of other similar case studies. The importance and evaluation of the intelligent logical error recovery unit will be ensured by matching marks. All IDEs that respect their development programming language will include an intelligent logical error unit.

IV. REFERENCES

[1]. Beltramelli, T. (2018, June). pix2code: Generating code from a graphical user interface screenshot. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (p. 3).ACM.

[2]. Parnianifard, A., Azfanizam, A., Ariffin, M. K. A. M., & Ismail, M. I. S. (2017). An overview on robust design hybrid metamodeling: Advanced methodology in process optimization under uncertainty.

[3]. Mongkhonvanit, K., Zau, C. J. Y., Proctor, C., & Blikstein, P. (2018, June). Testudinata: a tangible interface for exploring functional programming. In Proceedings of the 17th ACM Conference on Interaction Design and Children (pp. 493-496). ACM.

[4]. Daniel, B. K. (2018). Reimaging Research Methodology as Data Science. Big Data and Cognitive Computing, 2(1), 4.

[5]. Berger, E. D., Hollenbeck, C., Maj, P., Vitek, O., & Vitek, J. (2019). On the Impact of

Programming Languages on Code Quality. arXiv preprint arXiv:1901.10220.

[6]. Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., ... & Ko, A. J. (2019). A theory of instruction for introductory programming skills. Computer Science Education, 1-49.

[7]. Colapinto, C., Jayaraman, R., & Marsiglio, S. (2017). Multi-criteria decision analysis with goal programming in engineering, management and social sciences: a state-of-the art review. Annals of Operations Research, 251(1-2), 7-40.

[8]. Wyrich, M., Graziotin, D., & Wagner, S. (2019). A theory on individual characteristics of successful coding challenge solvers. PeerJ Computer Science, 5, e173.

[9]. Ketschau, T. J., & Kleinhans, J. (2019). Concept and Implementation of a Two-Stage Coding Scheme for the Development of Computer-Based Testing (CBT)-Items in Traditional Test Software. J, 2(1), 41-49.

[10]. Samara, G. (2017). A Practical Approach for Detecting Logical Error in Object Oriented Environment. arXiv preprint arXiv:1712.04189.

[11]. Deulkar, K., Kapoor, J., Gaud, P., & Gala, H. (2016). A novel approach to error detection and correction of c programs using machine learning and data mining. International Journal on Cybernetics & Informatics, 5(2), 31-39.

[12]. Lee, J., Song, D., So, S., & Oh, H. (2018). Automatic diagnosis and correction of logical errors for functional programming assignments. Proceedings of the ACM on Programming Languages, 2(OOPSLA), 158.

Cite this Article :

Dr. Ranjeet Kumar, Prof. Dr. Ramdip Prasad, Arif Md. Sattar, "Automated Software Testing Analysis on Test Case Optimization and Test Case Levels on Java and Python", International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Online ISSN : 2394-4099, Print ISSN : 2395-1990, Volume 6 Issue 2, pp. 812-819, March-April 2019.

doi : https://doi.org/10.32628/IJSRSET2182134

Journal URL : https://ijsrset.com/IJSRSET2182134