Research and design of program complexity measurement technology based on OINK framework

Liping Qiao[1], Xuejun Zou[2], Rui Duan[3], Xueting Jia[4]
[1, 4]Department of Information Engineering, Xingtai Polytechnic College, Xingtai, Hebei, China
[2, 3]Wuhan Electricity Services Department of Wuhan Railway Administration, Wuhan, Hubei, China
[1]Corresponding author
E-mail: [1]*s976811080@163.com*, [2]*397818434@qq.com*, [3]*whdr1984@163.com*, [4]*594837257@qq.com*

Received 11 January 2023; accepted 24 March 2023; published online 4 April 2023
DOI https://doi.org/10.21595/mme.2023.23162

Abstract. With the expansion of software system scale, the study of software complexity has become a hot topic in software engineering. However, the domestic research on software complexity analysis technology is not mature, especially the measurement and evaluation methods of software complexity are not perfect. In order to solve the problem of prediction and evaluation of program structure complexity in software engineering more effectively, this paper proposed a program complexity measurement technique based on OINK framework. The technology uses the data sharing interface design to analysis target program by extracting the complex relationship between OINK components. On this basis, the technology adopts the layered software architecture to realize the automatic design of the function of the measurement data acquisition module, the complexity measurement module and the data management module of measurement results, thus, the structure complexity of the target program can be analyzed more clearly and accurately. At the same time, this technique applies multiple measurement methods to quantify the complexity of program structure, such as McCabe, HalStead, and Line Count. Experimental results show that this method can effectively measure the complexity of program structure. The solution on software complexity based on the open source ONIK framework will be open up worldwide, and will be continuously supported and improved by global communities and teams under the constraints of common driving forces.

Keywords: complexity, OINK, static analysis, McCabe.

1. Introduction

Complexity measurement analysis assesses the complexity of a program's source code in terms of its structure, length, size, etc. so in a sense complexity measurement analysis also belongs to the category of program understanding, and is a key part of the software static analysis technology. Complexity analysis can reflect the error rate in the process of software development and reduce the possibility of software failure, so as to improve the quality and maintainability of software [1]. The research on program complexity analysis technology should eventually be implemented in the research on one or more complexity measurement algorithms, so the most concerned problem is the data measurement element problem [2]. Based on program understanding, these metrics come directly or indirectly from the abstract syntax tree. Therefore, how to ensure that the data information on the nodes of the abstract syntax tree is used as measure element of software complexity after correct and reasonable analysis and statistics is the problem we need to solve.

Program complexity measurement analysis technology belongs to the category of code static analysis, and the static analysis technology of foreign source code has reached the third generation [3]. For example, Klocwork Insight, a source code analysis tool belonging to Klocwork, is the first tool that allows developers to control the entire analysis process and benefit from the accuracy of centralized analysis. Klocwork Insight allows developers to perform local analysis in the development environment they are familiar with, and can also achieve the consistency and analysis accuracy obtained by doing the same analysis steps in the integration verification phase. The tool for software complexity measurement mainly includes McCabe complexity, Halstead

program measurement, number of code lines, number of inheritance, number of loops and other basic metrics [4-6]. At present, China's research on this technology is still in the initial development stage, the scale is relatively small, and few scientific and technological personnel participate in the research. From the perspective of technical research, some technologies are still in the second stage of source code analysis, namely the centralized analysis stage. However, there are also some excellent tools of China, such as Safe Pro C/C++ developed by the Beihang University Software Engineering Institute. It provides user working environment with multi-selection window and single-driven; and supports fast association analysis of several test information. From the current situation of domestic and foreign research, the software complexity analysis technology of programs has not yet reached the mature stage [7]. The research of software complexity analysis technology also needs to be continuously improved. However, these traditional measurement algorithms can also reflect the complexity of software development to a certain extent. The key is how to use these measurement methods to evaluate the complexity of software reasonably and stably, reduce the possibility of software failure and improve the maintainability of software, so as to achieve the purpose of enhancing software quality.

Based on the above point of view, according to the current software testing requirements, this paper proposes a program complexity measurement technology based on OINK static analysis framework by using the principles of semantic analysis, syntax analysis, abstract syntax tree, local analysis and global analysis in static analysis. This technology can effectively combine the complexity measurement parameters and static analysis tools, and will be applied to the program understanding platform, so that the program complexity analysis results are more accurate and reliable. At the end of the article, the correctness of this technology is proved by designing the test environment and process, and verifying the results by various measurement methods.

Our technology will be developed based on the open source framework, open up solutions, and have extremely reliable results, which is a very powerful outstanding advantage. Through mass discussion and supervision, it will produce stronger and more reliable results, benefit more enterprises and communities, and promote the vigorous development of process complexity measurement technology.

2. Program complexity research

Complexity is relative to simplicity, which refers to the characteristics of single, certainty, fixation, invariance; the complexity refers to the characteristics of diversity, uncertainty, randomness and variability [8]. Program complexity is a fundamental feature of program, but there is no accepted precise definition. Most researchers believe that program complexity is the difficulty of analyzing, designing, testing, maintaining and modifying software, and the difficulty will increase day by day; and some researchers believe that program complexity is mainly structural complexity and algorithmic complexity, which gradually occur in the software life process, especially in the design and coding phase [9]. Therefore, in order to study the program complexity, the analysis of the process of program structural complexity is the first step, and on this basis, it determines what kind of measurement method is used to quantify the complexity:

$$M: (C, R) \to (N, P). \tag{1}$$

The process of analyzing the generation of program complexity is to generate the mapping relationship M, from software code C and the relationship between codes R, to the complexity measurement results N, and the numerical relationship between measurement results P.

Take structured program SP as an example:

$$SP = \langle S, Seq, Sel, Rep \rangle. \tag{2}$$

S includes basic statements (empty statements, assignment statements, procedure call

statements), sequence control statements Seq, structure control statements Sel, and loop control statements Rep.

The measurement of program complexity is to give an expression to each program, and use this expression to describe various characteristics of program complexity. If $S_1 \in SP$, $S_2 \in SP$:

$$M\big(Seq(S_1, S_2)\big) = M(S_1) \times M(S_2), \tag{3}$$

where × represents the multiplication of expressions.

$$M\big(Sel(S_1, S_2)\big) = M(S_1) + M(S_2), \tag{4}$$

where + represents the addition of expressions.

$$M\big(Rep(S_1)\big) = [M(S_1, 1)]_{r:=r^c}, \tag{5}$$

where $r := r^c$ represents substituting r with r^c in the expressions.

2.1. Performance behavior of procedural complexity

The main reasons for program complexity are: the complexity of operating environment, software requirements, data model, design process, software project management, software architecture, project testing and non-formal methods [10]. These reasons are ultimately manifested in the complexity of program code data structure and control structure, where both control flow and data flow generation are derived from abstract syntax trees in program understanding techniques. Abstract syntax tree is generated by the input source program through lexical analysis, syntax analysis and semantic analysis, and transformed and filtered through the basis of the parse tree. Fig. 1 shows the process of program complexity of the static analysis.

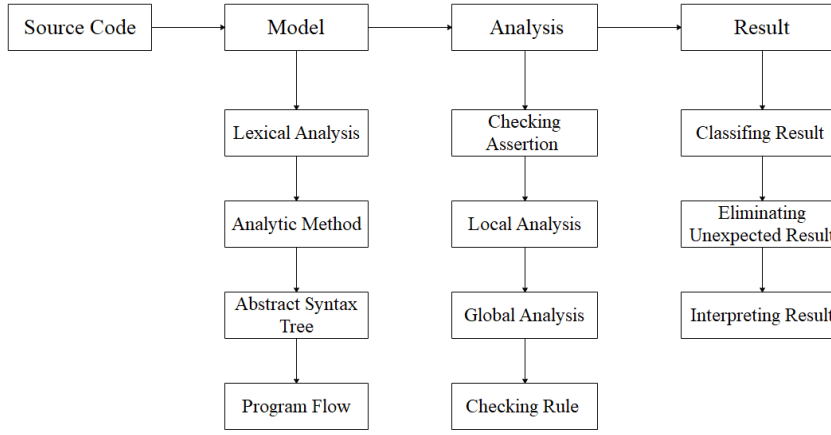


Fig. 1. Program static analysis process

2.2. Program complexity measurement method

Measurement of program complexity is an important work in program understanding and maintenance. The existing measurement methods include quantitative and qualitative methods. Quantitative analysis can be divided into four types: program scale measurement, data complexity measurement, computational complexity measurement, and control complexity measurement. Qualitative analysis work attempts to explain the root of complexity rather than simply giving the metric formula [11].

There are many measurements of software complexity algorithms that can be roughly divided

into two categories: object-oriented software complexity measurements and process-oriented software complexity measurements. The most active and fruitful research is process-oriented software complexity measurements. This paper mainly uses the McCabe structure complexity measurement method, HalStead software science measurement method and Line Count code line measurement method. The McCabe measurement method is essentially a measure of the complexity of the program topological structure, and the strict analysis and calculation of the control structure of the program. It clearly points out the complex part of the program task [12-13]. The HalStead measurement method is to take the operators and operands present in the program as the counting objects, with the number of times they appear as the counting target directly measured indicators, and then calculate the program length and workload according to the HalStead complexity formula. The Line Count measurement method is a static analysis of the physical scale of the program and the code characteristics.

According to Shannon's information theory, if the information source is based on probability P_j randomly send the j-th message from the total of i messages ($1 \leq j \leq i$). Entropy is defined as:

$$H = -\sum_{j=1}^{i} P_j \log_2 P_j. \tag{6}$$

If all messages are sent with equal probability, $P_j = 1/i$, so:

$$H = -\log_2 i. \tag{7}$$

In the program with length N, the vocabulary composition is as follows:

$$N = N_1 + N_2, \tag{8}$$

where N_1 is the total number of operators, N_2 is the total number of operands. These words can be selected from n with equal probability:

$$n = n_1 + n_2, \tag{9}$$

where n_1 is the number of operator types, n_2 is the number of operand types. Therefore:

$$P_j = \frac{1}{n_1 + n_2}. \tag{10}$$

Then for the program, its entropy is:

$$H = N_H \log_2(n_1 + n_2). \tag{11}$$

If the predicted vocabulary is defined as follows:

$$LE = N_H = n_1 \log_2 n_1 + n_2 \log_2 n_2. \tag{12}$$

The program volume V, that is, the minimum amount of information required to solve the problem, reflects the lexical complexity of the program:

$$V = H = N_H \log_2(n_1 + n_2). \tag{13}$$

Program level PL is the ratio of problem complexity to program complexity, which reflects the efficiency of the program. The program level of high-level language is close to or up to 1, where:

$$N_H = N = N_1 + N_2 = n_1 \log_2 n_1 + n_2 \log_2 n_2, \tag{14}$$

$$PL = \frac{2n_2}{n_1 N_2}. \tag{15}$$

Program language level LL is related to PL and V:

$$LL = PL^2 \times V. \tag{16}$$

From the predicted vocabulary and program level, it can be concluded that the workload E of the program is:

$$E = \frac{LE}{PL}. \tag{17}$$

According to the above formula, the errors in the program can also be calculated, and the number of program errors B is directly proportional to the program volume:

$$B = \frac{V}{3000} = \frac{(N_1 + N_2)\log_2(n_1 + n_2)}{3000}. \tag{18}$$

3. Analysis and design of OINK frame structure

OINK is an open source program understanding tool, which can complete lexical analysis, syntax analysis, abstract syntax tree and program flow diagram to realize software complexity analysis. Its core function is program understanding. OINK can perform many static analysis methods of C and C++ programs, mainly for data stream analysis. In the program understanding function, it includes analyzing and processing the data flow at the expression level and type level of the source program, and the control flow at the statement level (which is realized by the relevant functions of Elsa). It can also realize the output of the analysis results such as the data flow graph of program, the control flow graph, and the class inheritance relationship graph (C++).

3.1. Analysis of OINK frame structure

There are many source code files of OINK, which need to be analyzed according to the code files on which different functions depend. The objects analyzed include function interface definition, variable declaration and object dependency in the code.

OINK's source package (OINK-Stack) includes smbase, ast, elkhound, elsa, libregion, libqual, platform-model, and oink. OINK has three main functions, including staticprint, cfgprint and dfgprint. Staticprint analyzes the inheritance relationship of classes of object-oriented; cfgprint main analyzes program structure, and outputs program control flow graph; dfgprint analyzes the data transfer state of the program, and displays the analysis results from the form of data flow graph of the program. Fig. 2 shows the structure of dependency and reference relationships of OINK source code files to better help us develop software complexity measurement system.

After understanding the organizational structure of OINK source code file, the OINK platform interface for developing measurement system can be realized. For example, McCabe complexity is the complexity measurement of program structure, and the cfgprint.cc file in OINK source code can output the program control flow graph, so the interface of McCabe complexity measurement system can be realized in this file.

3.2. Design of OINK data service interface

Through the previous analysis, it can be concluded that OINK can complete semantic analysis, lexical analysis and abstract syntax tree construction in the process of program understanding. On

this basis, the OINK data service interface is designed, which can provide data measurement element for program complexity measurement technology. Fig. 3 shows the OINK data service interface. In Fig. 3, the information statistical interface of edge nodes and arc nodes provides measurement data onto program control structure (i.e., McCabe standard); the information statistical interface of program operands and operators provides measurement data for the physical structure of the program (i.e., Halstead standard). These service interfaces are all deployed on the server in the form of Web Service [14].

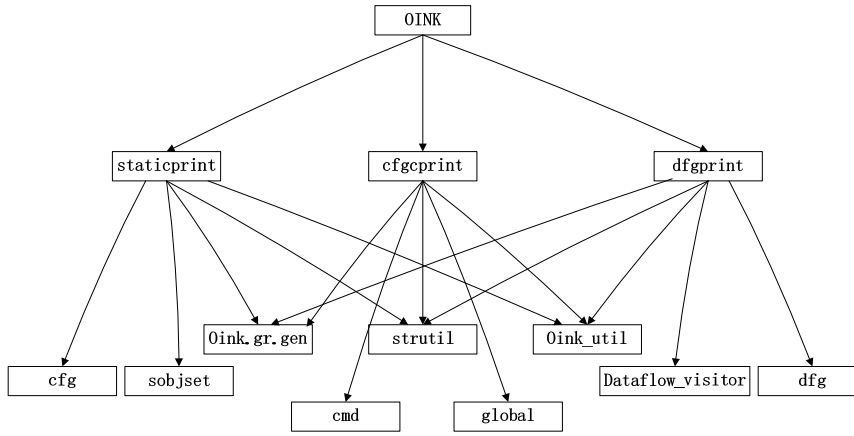


Fig. 2. OINK source code file structure

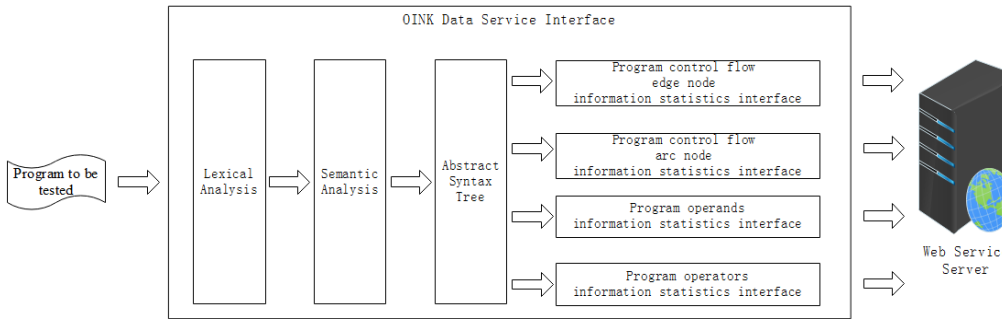


Fig. 3. Design of OINK data service interface

The operations performed by the data service interface can be defined as the following formula:

$$R = F(C). \tag{19}$$

F represents the operation done by the data service interface; C represents the incoming data, that is, the original code; R is the operation result:

$$F = \langle f_1, f_2, f_3, f_4, f_5 \rangle. \tag{20}$$

F is divided into f_1, f_2, f_3, f_4, f_5 five steps:

$$R = \langle R_e, R_a, R_{n_1}, R_{n_2} \rangle. \tag{21}$$

R includes edge node information, arc node information, operand information and operator information.

The specific steps of data service interface are:

$$C' = f_1(C), \tag{22}$$

where f_1 stands for preprocessing operation, C is the original code, C' is the code after deleting preprocessing statements, comments, spaces and macro replacement:

$$y = \langle T, A \rangle = f_2(C'). \tag{23}$$

f_2 represents lexical analysis operation, the result y is lexical unit $\langle T, A \rangle$, (TokenName, AttributeValue).

$$I = f_3(C', y), \tag{24}$$

where f_3 represents semantic analysis operation, to match C' with y for type check.

$$z = AST = f_4(I), \tag{25}$$

where f_4 represents the operation of generating an abstract syntax tree, and the abstract syntax tree AST will be obtained.

$$R = f_5(z). \tag{26}$$

Finally, the abstract syntax tree is analyzed in f_5, and send to different interfaces to get R.

4. Design of program complexity measurement technology based on OINK framework

In order to complete the program complexity measurement technology, the data acquisition module and complexity analysis module are designed based on OINK data interface. The data acquisition module is used to collect the necessary data required for complexity calculation, and the complexity analysis module is used to calculate the complexity data results. Finally, the data can be stored and queried through data management.

4.1. Design of program measurement data acquisition module based on OINK framework

Data acquisition module needs to complete two tasks: obtaining data onto the intermediate results of program understanding platform; and storing data onto the form defined by the data structure. The function structure of the data acquisition module is shown in Fig. 4.

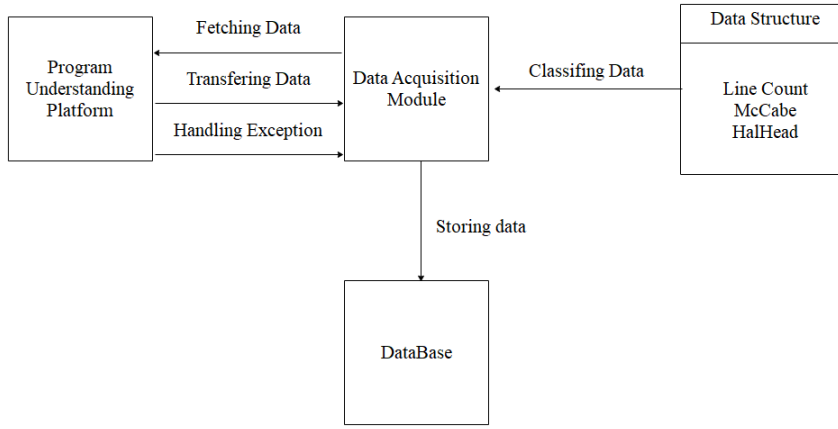


Fig. 4. Function structure of data acquisition module

The function of the data acquisition module shown in Fig. 4 includes: collecting data from the program understanding platform, classifying and storing data according to different types of data structures, and storing all data into the database. If the system is abnormal in the process of collecting data, the data acquisition module will receive abnormal information and send it to the system for processing. The algorithm designed in the data acquisition module is shown in Fig. 5.

Fig. 5 shows the commonly used interface name in the algorithm. The retrieve function judges that the data to be collected by the system is for Line Count complexity measurement, McCabe complexity measurement, or HalSteadcomplexity measurement, according to the value of parameters; then collects different data. The search function can traverse the number of operators and operands from the abstract syntax tree, and can also obtain the number of nodes and arcs and the number of branch judgments from the program control flow diagram. The Send function to send the obtained data onto the complexity measurement system.

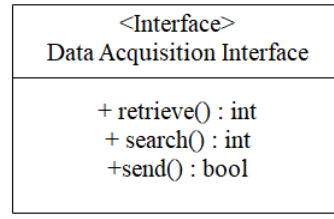


Fig. 5. Design of data acquisition interface

4.2. Design of complexity measurement analysis module for target program

The system can support the measurement of Line Count Complexity, McCabe complexity and HalStead complexity, which are the core function of the system. The functional structure of its implementation is shown in Fig. 6.

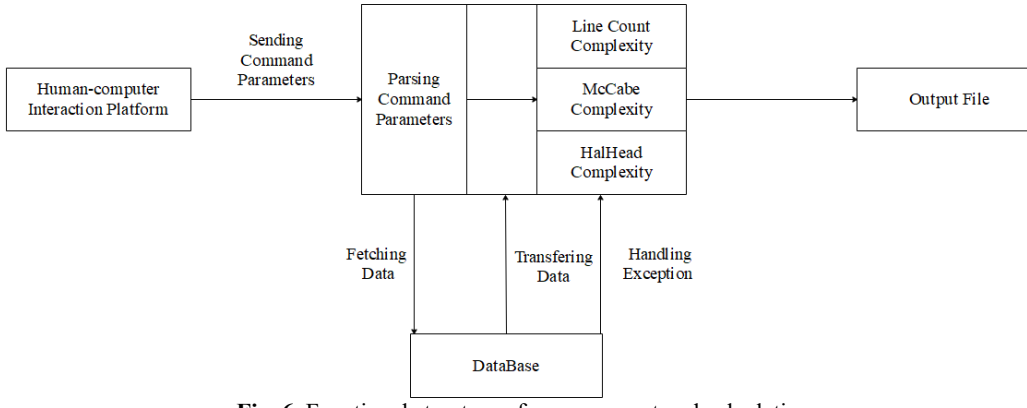


Fig. 6. Functional structure of measurement and calculation

It can be seen from the Fig. 6 that the functional principle of the complexity measurement system is to send the command parameters for executing complexity measurement of the human-computer interaction platform, and analyze the types of command parameters through the analysis command parameter interface of the measurement system. There are three main commands of complexity measurement: Line Count complexity command, McCabe complexity command and Halstead complexity command. The measurement system retrieved the corresponding data measurement element from the database and output the result after calculation. If the required data cannot be found from the database, the system issues a prompt message of command execution failure.

The measurement and calculation functions of the system mainly include the calculation

functions of Line Count complexity, McCabe complexity and Halstead complexity. Some specific differences between these three functions are shown in Table 1.

Due to the complexity of the algorithm structure, after describing the data acquisition algorithm of McCabe complexity measurement based on OINK platform, the next step is to analyze the algorithm of calculating cycle complexity measurement of McCabe complexity.

The algorithm mainly uses the cycle complexity to calculate the program control flow diagram. In formula, n_c stands for cycle complexity, n_a stands for the number of arcs, n_n stands for the number of nodes:

$$n_c = n_a - n_n + 2. \tag{27}$$

If the calculated cycle complexity is not equal to the result of judging the number of branches plus 1, it indicates that the input parameters of the algorithm are wrong, and the error information report is output. If the calculation is correct, the algorithm returns the value of cycle complexity.

Table 1. Parameter comparison table of complexity measurement method

Parameters measurement	Line count complexity	McCabe complexity	HalStead complexity
Command parameters	001	010	011
Input	Code file	Program diagram information	Types of operators and operands
Output	Code line, blank line, comment line	McCabe measurement element	HalStead measurement element
Performance measuring	Physical scale of the program	Structure of program	Length of program
Data storage mode	Database	Database	Database
Exception handling	Support	Support	Support

5. Experimental analysis

After completing the design of program complexity measurement technology based on OINK framework, the experimental test is carried out. Firstly, load the test codes AutoCode.c and AutoCode2.c, and then the abstract syntax tree is generated based on the OINK framework. After that, the metadata required for program complexity measurement is obtained through the data service interface. Based on these measurement data, the McCabe measurement results, Line Count measurement results and Halstead measurement results of test codes are obtained. Finally, the measurement results are compared with the actual complex measurement data of test codes, as shown in Table 2, Table 3, Table 4 and Table 5.

Table 2. Comparison table of experimental data in line count measurement

Object measured	Experimental data 1	Real data 1	Accuracy 1	Experimental result
Code line	643	643	100 %	Completely accurate
Blank line	27	27	100 %	Completely accurate
Comment line	15	15	100 %	Completely accurate
Object measured	Experimental data 2	Real data 2	Accuracy 2	Experimental Result
Code line	1086	1086	100 %	Completely accurate
Blank line	67	67	100 %	Completely accurate
Comment line	48	48	100 %	Completely accurate

Table 3. Comparison table of experimental data in McCabe measurement

Object measured	Experimental data 1	Real data 1	Accuracy 1	Experimental result
McCabe cycle complexity	8	8	100 %	Completely accurate
Object measured	Experimental data 2	Real data 2	Accuracy 2	Experimental result
McCabe cycle complexity	13	13	100 %	Completely accurate

Table 4. Comparison table of experimental data in Halstead measurement

Object measured	Experimental data 1	Real data 1	Accuracy 1	Experimental result
Program volume	209.50	210.68	99.44 %	Accurate
Program level	0.13	0.13	99.28 %	Accurate
Procedure difficulty	7.45	7.50	99.33 %	Accurate
Object measured	Experimental data 2	Real data 2	Accuracy 2	Experimental result
Program volume	438.23	442.21	99.10 %	Accurate
Program level	0.078	0.079	99.00 %	Accurate
Procedure difficulty	12.52	12.67	98.80 %	Accurate

Table 5. Comparison table of experimental data in Standardized measurement

Object measured	Experimental data 1	Real data 1	Accuracy 1	Experimental result
Standardization complexity	3.37	3.43	98.25 %	Accurate
Object measured	Experimental data 2	Real data 2	Accuracy 2	Experimental result
Standardization complexity	6.16	6.29	98.01 %	Accurate

It can be seen from Table 2 to Table 5 that the results calculated by the program complexity measurement technology based on the OINK framework are almost completely consistent with the real data, in which the Line Count measurement and McCabe measurement are completely consistent, and the accuracy of Halstead and standardized complexity measurement is close to 99.9 %. In order to more clearly illustrate the consistency of the two data, their histogram comparison and line chart comparison is shown in Fig. 7 and Fig. 8.

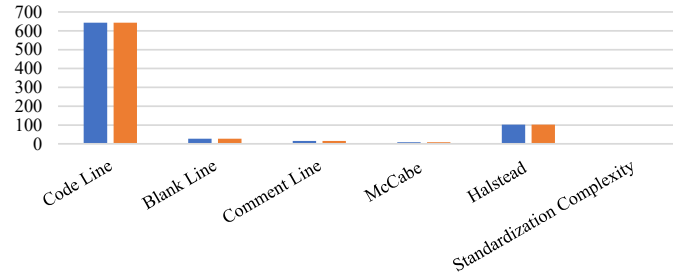


Fig. 7. Histogram of comparison between experimental data and real data

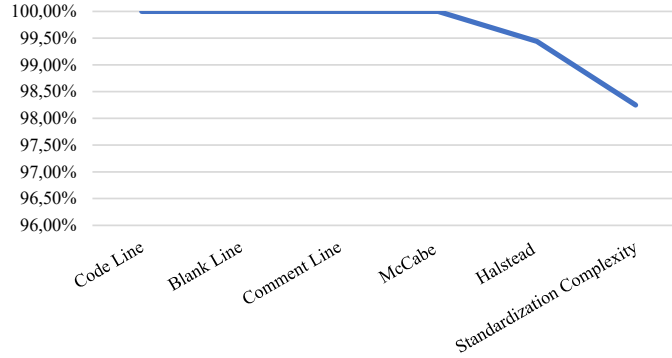


Fig. 8. Line chart of accuracy comparison between experimental data and real data

It is shown in the histogram that the experimental data are almost the same as the real data in the y axis, namely, the direction of the numerical axis. In Fig. 8 the curve direction of accuracy is almost parallel to 100 %. This further proves the rationality and accuracy of this technology.

6. Conclusions

The program complexity measurement technology studied in this paper uses OINK framework to statically analyze the program code. Firstly, based on the abstract syntax tree, the information statistics interface of edges and arcs of program control structure and the information statistics interface of program operators and operands are designed. Then, various types of measurement metadata are classified and summarized through the data acquisition algorithm, and the complexity results of the program are directly analyzed through the complexity measurement calculation module. Finally, the data is stored and queried through the measurement data management module.

This paper extracts the parameter factors required for code complexity analysis through abstract syntax tree, analyzes and calculates the complexity of software program structure by using Line Count measurement calculation method, McCabe measurement calculation method and Halstead measurement calculation method, and gives a comprehensive measurement report. At the end of this paper, the application testing environment is established, the application testing process is designed in detail, and then the complexity measurement test is carried out on the randomly selected program code. Through the analysis of the results, it shows the consistency between the system test results and the program complexity analysis results. Therefore, from the perspective of practical application, it proves the accuracy and feasibility of the program complexity analysis technology based on OINK proposed in this paper.

Acknowledgements

The authors have not disclosed any funding.

Data availability

The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Author contributions

Liping Qiao: writing – review and editing. Xuejun Zou: writing – original draft preparation. Rui Duan: prepare experimental materials and collect experimental data. Xueting Jia: data collation and analysis.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Y. W. Tang, "Algorithm for introducing test complexity to improve the efficiency of software test management," *Business Herald*, Vol. 21, pp. 29–30, 2015.

[2] W. Wang, "Large-scale software complexity metrics based on complex networks," *Software*, Vol. 36, No. 11, pp. 92–95, 2015.

[3] S. Nalinee, "Complexity measure of software composition framework," *Journal of Software Engineering and Applications*, No. 4, pp. 324–337, 2017.

[4] E. Pira, V. Rafe, and A. Nikanjam, "Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm," *Journal of Systems and Software*, Vol. 131, pp. 181–200, Sep. 2017, https://doi.org/10.1016/j.jss.2017.05.128

[5] B. Y. Wang, "Research on Complexity Measurement of software system architecture," *Software Guide*, Vol. 9, No. 10, pp. 7–9, 2010.

[6] Piqueira and José Roberto C., "Weighting order and disorder on complexity measures," *Journal of Taibah University for Science*, Vol. 11, No. 2, pp. 337–343, Mar. 2017, https://doi.org/10.1016/j.jtusci.2016.05.003

[7] H. Tian and H. Zhao, "Metrics for software structure complexity based on software weighted network," *Computer Science*, Vol. 43, pp. 506–508, 2016.

[8] C. L. Coyle and M. Peterson, "Learnability testing of a complex software application," in *Design, User Experience, and Usability: Novel User Experiences*, Vol. 9747, pp. 560–568, 2016, https://doi.org/10.1007/978-3-319-40355-7_53

[9] J. K. Chhabra and V. Gupta, "Evaluation of object-oriented spatial complexity measures," *ACM SIGSOFT Software Engineering Notes*, Vol. 34, No. 3, pp. 1–5, May 2009, https://doi.org/10.1145/1527202.1527208

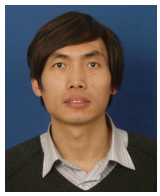
[10] N. Cai, "On quantitatively measuring controllability of complex networks," *Physica A: Statistical Mechanics and its Applications*, Vol. 474, pp. 282–292, May 2017, https://doi.org/10.1016/j.physa.2017.01.053

[11] Jerry Gao, "Complexity metrics for regression testing of component-based software," *Journal of Software*, Vol. 26, No. 12, pp. 3043–3061, 2015.

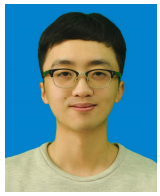
[12] I. H. Suh, S. H. Lee, N. J. Cho, and W. Y. Kwon, "Measuring motion significance and motion complexity," *Information Sciences*, Vol. 388-389, pp. 84–98, May 2017, https://doi.org/10.1016/j.ins.2017.01.027

[13] K. Kuramitsu, "Fast, flexible, and declarative construction of abstract syntax trees with PEGs," *Journal of Information Processing*, Vol. 24, No. 1, pp. 123–131, 2015, https://doi.org/10.48550/arxiv.1507.08610

[14] S. Z. Chen, Z. Y. Feng, C. Xu, and H. Liu, "Research on web services development based on service network," *Acta Scientiarum Naturalium Universitatis Nankaiensis*, Vol. 43, No. 5, pp. 60–66, 2010.



Qiao Liping received master's degree in computer application technology from Beijing University of Technology, Beijing, China, in 2010. Now he works in Hebei Vocational University of Technology and Engineering. His current research interests include Data Analysis, Software Technology and Application.



Zou Xuejun received bachelor's degree in railway signal from Lanzhou Jiaotong University, Lanzhou, China, in 2013. Now he works in Wuhan Electric Service Depot. His current research interests include Basic equipment information and Equipment operation safety.

Duan Rui received master's degree in railway signal from Lanzhou Jiaotong University, Lanzhou, China, in 2015. Now he works in Wuhan Electric Service Depot. His current research interests include Basic equipment information and Equipment operation safety

Jia Xueting received master's degree in software engineering from Beihang University, Beijing, China, in 2021. Now she works in Hebei Vocational University of Technology and Engineering. Her current research interests include Computer Software and Theory, Software Technology and Application.