Research Article

Test Case Based Selection for the Process of Software Maintenance

Adtha Lawanna

Department of Information Technology, Faculty of Science and Technology, Assumption University, Bangkok, Thailand
Corresponding author. E-mail address: adtha@scitech.au.edu

Received March 29, 2013; Accepted August 7, 2013

Abstract

Software maintenance is the special process in the software-development life cycle. Particularly, the programmers have tried to reduce the size of testing and maintaining new software while fixing bugs is also realized. The large amounts of tests may cause time consuming, especially execution and operation. In response to this, many specialists propose the techniques for test case selection such as random selection and safe selection relying on the concept of regression testing. However, the ability of the new software is still required to be improved. Therefore, the test case based control-path is preferred to increase the performance of the program by creating and selecting the least test case as well as the faultless rate is preserved.

Key Words: Software Development Life Cycle; Software Maintenance; Regression testing

Introduction

Due to the body knowledge of software engineering, this becomes an issue in developing programs. Up to now, most of the development teams are still creating new software for responding to the needs of users to support business objectives in their organizations (Carmel, 1995). In response to this, the software-development life cycle (SDLC) is powerful methodology that helps the programmers to produce the specific software, e.g., waterfall, iterative, prototyping, and spiral model (Larman and Basili, 2003). As we know, SDLC comprises the phase of user requirement, analysis, coding, testing, implementation, and maintenance (Cohen, 2010). User requirement is the phase for gathering the needs and wants from users in details. After this is the step of analyzing the specific problem as well as preparing a good design, including data flow diagram, entity relationship, and database. Later, the programmers write the software, which is good or not depending on their skills and experiences. Next, is to monitor and correct the program, e.g., testing quality of software, entire system, and user satisfaction. Finally, in the process of SDLC is the software maintenance. The maintenance process is one of the most important phases in SDLC; particularly, it is designed to plan and control the new program in the entire system after adapting the existing software (Leau et al., 2012). The term maintenance includes fixing bugs, modifying, updating deleting, or adding some piece of the software. Modifying software may be required for adapting the system to the changes, e.g., technology, environment, or trend of customer life style. This paper considers those changes, including specification requirement, line of codes, and bugs. Sometime modification easily can be done, whereas there are few bugs occurred within the modified program. In general, bugs will be occurred, whenever the programmers do coding. On the other hand, modification may fail, in which the skills and experiences of the experts are involved e.g., the knowledge of programming languages, merging the difference codes, and the configuration (Chapin et al., 2001). As well as updating the software, the programmers may consider all functions and types of the program for selecting the important modules in order to improve its ability. The most difficulties of updating software are to change the previous programming language to the new and integrate the structure of the difference codes, including testing the entire system.

The research area of software maintenance concerns the test suite selection, minimization, and prioritization. The techniques of test suite selection can be used to determine the numbers of the test cases from a test suite (Harrold et al., 1993 ; Harrold, 1999). Particularly, any test suite contains a set of test case, which is created relying on the specific factors e.g., requirements, codes, and bugs. More specifically, another purpose of software maintenance is to preserve the faultless within the changed program (Niessink, 2000). Those factors affect the entire software system in terms of executing and running the modified software. The execution time is a major problem when all test cases are audited as well as the running time (Musa, 1993). As we know, the higher numbers of the test cases often show the better abilities of the new software, whereas the faults are small. However, the maximum numbers of the test cases increase the execution time. Therefore, many researches propose the techniques that select the minimum numbers of test cases as well as fixing bugs within the new software.

This paper presents the technique that can solve the remained problems by selecting the lower numbers of test cases while the faultless rate is preserved than the traditional technique. The proposed model concerns the subject programs, specifically used in the area of selecting test cases and decreasing the faults of the software. In addition, this paper shows some of the traditional techniques, which are used to compare their abilities.

Materials and Methods

Data set

Preparing the experiment is one of the most important methods. Accordingly, the data set is required. In Table 1, the seven subject programs are required whereas the program name, numbers of function (F), lines of code (C), faulty versions (V) and the test suites (T) are available (Rothermel and Harrld, 1998). To manage a test suite and automate test execution, a test database management system is created, and playback tools are captured (Rothermel, 1996). Those subject programs are written by the developers of the Siemens suite of programs with manually fixing bugs or faults. The artifacts of all seven programs have consequently, been revised and extended by other agents. These programs are preferred because of the development of the related artifacts as well as the historical significance. Numerous high-quality experimental software engineering researchers have used the Siemens suite (Ostrand, 1998).

Table 1 The Subject programs

Program Name	Numbers of function(F)	Line of codes(C)	Faulty Versions (V)	Test Suite (T)
print-tokens	18	402	7	4,130
print-tokens2	19	483	10	4,115
replace	21	516	32	5,542
schedule	18	299	9	2,650
schedule2	16	297	10	2,710
tcas	9	148	41	1,608
totinfo	7	346	23	1,052

Regression Testing (RT)

Regression testing is the method of testing changes within software or programs to ensure that the existing system still work with the new changes (Agrawal et al., 1993). Regression testing is a basic part of the SDLC, especially in the software maintenance. For the large companies, RT is done by software testing specialists or programmers. The typical steps of RT are described as follows:

(1) Select the test cases from a test suite.

A test suite is the set of the test cases, which can be constructed automatically by the test case generator.

$$T = \{t_1, t_2, t_3, ..., t_n\} \quad (1)$$

Where: T is a test suite and t is the test case.

(2) Test the program (P) with the selected test cases.

In general, a test suite contains huge amounts of the test cases. Therefore, the developers select some of the test cases for the process of software maintenance, e.g., bugs and run time.

$$t^* = \{t_1, t_2, t_3, ..., t_m\} \quad (2)$$

Where: t^* is a set of selected test cases regarding the specifications requirements from users and developers.

(3) If necessary, create new test cases for the program.

If the selected test cases by (2) cannot cover all the specifics requirements, then the new test cases should be chosen for fixing this problem.

$$t^{**} = \{t_1, t_2, t_3, ..., t_v\} \quad (3)$$

Therefore, the total selected test cases equal $t^* + t^{**}$

These test cases form what becomes the test bucket. Before releasing a new version of a software product, the old test cases are also run against the modified version in order to make sure that all the exist capabilities still run. The reason that they might not work is because modifying or adding new code to a program can easily produce bugs into code that may not have intended to be made. Test department coders do program test scenarios and exercises that will test new modules of code after they have been written.

Researchers have tried to perform regression testing more efficient and more effective by preparing regression test selection (RTS) techniques, but many problem remain, such as: RTS techniques may save time and money, however they sometimes may select most or all of the original test cases (Leung and White, 1991). Therefore, specialists using RTS techniques can find themselves worse off for having done so

(Ball, 1998). Testing time is often limited, e.g., must be finished overnight. RTS techniques do not focus such problems and, hence, can select more test cases than can be work. RTS techniques can maximize the average regression testing ability rather than optimize aggregate ability over many parts of testing software.

Random Selection (RS)

Random Selection is the technique that is created after the Testing All-Selection (AS) is applied. The major benefit of AS is the minimum numbers of faultless rate. However, it may cause a big problem of time consuming. Therefore, many development turns to the RS because it is simplest and to avoid the high cost, including timeless (Grave et al., 2001). The steps of RS are explained as follows:

(1) Due to using the RTS, a test suite is given.

(2) Randomly select the test cases from a test suite.

This step can be done by a spreadsheet Microsoft Excel^{TM}. For example, if there is a test suite (T), the numbers of the selected test cases can be computed by "=T(RAND())".

Note: RS technique gives the least execution time for module testing, but it may not guarantee the ability of the program in terms of producing the new bugs whereas the entire software is not tested. More details can be found in the article of Grave and team, 2001.

Safe Selection (SS)

Safe Selection is the technique an efficient regression selection (Hutchins et al., 1994; Rothermel and Harrld, 1996; Rothermel and Harrold, 1997; Rothermel and Harrold, 1998). It is one of the regression test selections implemented as a tool called DejaVu. Specifically, this technique provides smaller numbers of test cases compared with AS, and RS. Another reason of using the SS is the least of bugs are performed. This technique can provide the better results in lower cost and timeless in the process of execution and testing (Wong et al., 1997). The steps of SS are explained as follows:

(1) Due to using the RTS, a test suit is given.

(2) Create a control flow graph (CFG) for the program.

In a program, control flows are created from variables and procedures, e.g. such as if- and while-constructs in the programming language. It is a representation of its possible control flows through the whole program. Particularly, all nodes correspond to statements and decisions, including edges are used to represent the flow of control in a code. Statement coverage is constructed in a test suite that can execute every statement at least once of a whole program.

(3) Test execution profiles and choose all test cases in a test suite that, when executed through the program.

(4) Exercise the program at least on statement that is deleted from the program, or that, when executed on the modified version.

(5) Exercise the modified program at least on statement that becomes a new or modified in the latest version.

The statement that does not exist in the program cannot be executed. Therefore, the selected test cases can be provided by exercising the program or the modified version, which is created to be safe.

The conceptual overview of the proposed model namely Lawanna Selection (LS)

The activities in the process of software maintenance and Lawanna Selection are shown in Figure 1. The details of the whole steps are described as follows;

(1) After the process of software-development life cycle is reached, the software will be released to the users.

(2) When the time goes by, users may need the modified software. Therefore, the development team needs the specification requirements from the users in details.

(3) After this, the programmers will modify the previous program regarding (2).

(4) Testers test the modified program, e.g., checking inputs, functions, and outputs of the code, including fixing bugs.

(5) When faults or bugs are produced, it is necessary to redo (4) again.

(6) Check the side effect of a whole program, e.g., the relationship of variables, functions, and the expected results such as the ability of running the program, execution time, and user acceptance. Particularly, this activity refers using the regression testing.

(7) A test suite is given by (6), there are the large amounts of the test cases are generated. This causes the complexity of testing the software, the consequence of corresponding failures, difficulty of solving errors, and debugging the test cases. In this research, the LS is proposed to solve the problem about the increasing size of a test suite by selecting few test cases, which can preserve the competency of the whole program. However, one problem can be occurred in LS, which is producing the irrelevant selected test cases. These test cases cannot be run properly or there are bugs found. Therefore, to fix this problem is necessary to regenerate the test cases again. Besides this, LS technique needs to use the outcome of the regression testing by realizing the major variables, which are the value of F and C from the general subject program. Come to this point, the modified program will be released to the users, whereas the whole processes are done.

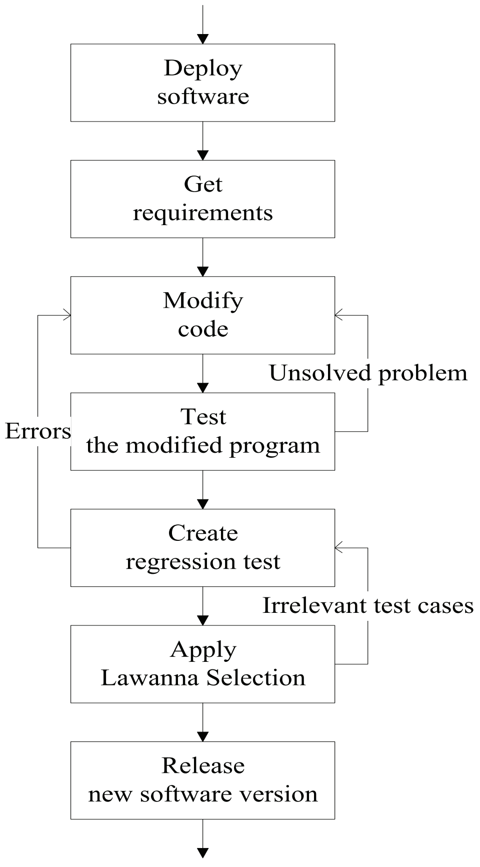

Figure 1 The process of software maintenance and Lawanna Selection

Lawanna Selection (LS) concerns the relationship of the numbers of function (F), line of codes (C), and the faulty versions (V). Eq. 4 to Eq. 6 are shown as follows;

$$F = \{1,2,3,\ldots,n\} \quad (4)$$

$$C = \{1,2,3,\ldots,m\} \quad (5)$$

$$V = \{1,2,3,\ldots,r\} \quad (6)$$

In particular, not only the value of F, C, and V are required by developers, but it includes the user requirements and test case generator. This is because they affect the size and quality of a test suite and the competency of a whole program.

A set of the selected test cases is generated and located in a test suite. One of the main objectives is to get the small test cases by avoiding the problem of run time. Next, LS can prepare the higher reduction rate, including providing the least faults or bugs in the modified program.

The conceptaul model of Lawanna Selection (LS)

The most important issue in the process of software maintenance is to preserve faultless rate of the minimal selected test cases in order to avoid the bugs that can be occurred. In response to this, the model of LS is proposed to build the concept of selecting the relevant test cases in any test suite of the program through the process of maintaining software.

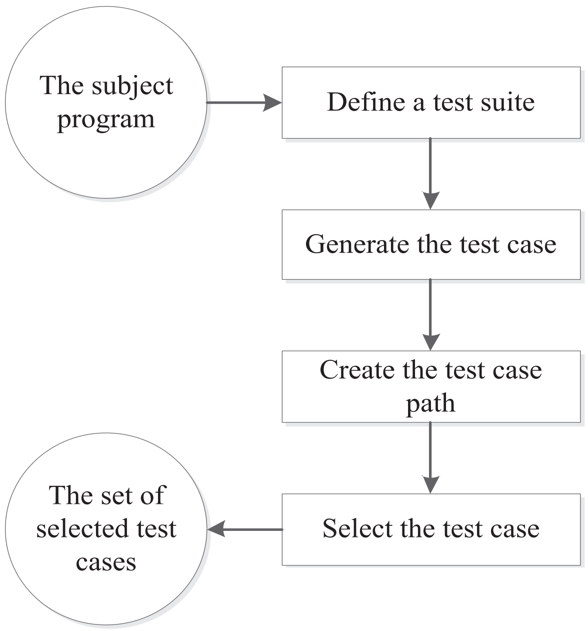

Figure 2 The Lawanna Selection (LS)

Figure 2 shows the conceptual model of the Lawanna Selection (LS). The process of LS starts when the users require the updated program for their purposes. Accordingly, the requirements relate directly to the numbers of function (F), line of codes (C), and numbers of bug (B). The general problem of LS and many traditional selection techniques involve creating a test suite. The reason is the complexity of a program, which combines all conditions of writing source code, e.g., user requirements, numbers of function, inputs, and expected outputs. Therefore, many researches import a given test suite which is automatically created by a specific test case generator. In the LS, a test suite also can be generated by the software named Reactis Tester. The test cases can be generated by importing inputs and clearly steps of testing in order to provide the appropriate output. A relevant test case will give “pass” not “fail” at the expected output. The most complicate part of LS is to create the test case path or control-path. Accordingly, the control-path shows the steps constructed in each test case. The first assumption by applying LS is that the generated test cases have their own steps of dealing with the different inputs for checking the outputs (pass or fail). The second assumption is the outputs of all generated test cases are “pass”. The reason is that 100% coverage is required. If “fail” is found, then that test case will be rejected. Besides this, one of the most important steps of the LS is to select the appropriate test cases. Of course, each test case may take the same or different steps for testing the specific inputs. For LS, it needs the shortest steps to be the representative. Surely, the expectation of proposing LS is to select small amounts of the test cases with 100 % coverage to avoid technical errors and keep the specification requirements.

The experimental steps of LS

(1) Define a given test suite (TS)

This step is created in order to define a test suite, which can be generated by Reactis Tester. A test suite will be constructed by executing the subject program, which the input and output values are recorded at each step.

(2) Generate the test cases.

The test case template is created as shown in

Table 2. It comprises the test case number, Input, output, and step of testing. The test cases can be generated regarding the percentage of covering the specifics requirements. Due to defining a test suite, it gives a test suite that contains the test cases with different % coverage. In the experiment, only the test cases with 100% coverage will be generated in order to avoid producing the irrelevant test cases (the coverage value is less than 100%). For example, if there are x steps in test case (1), which is constructed to handle y inputs, and all y outputs can be produced and reported. This means the test case (1) has 100% coverage. Therefore, test case (1) is usable, if it is not 100%, it will be rejected from a test suite.

Table 2 The test case template

Test Case (No.)	Input (1)	Input (2)	…	Input (y)	Output (1)	Output (2)	…	Output (y)
Step (1)								
Step (2)								
…								
Step (x)								

(3) Create the control-path.

Assume that all inputs and output of a test case are active. The numbers of step are realized for creating the control-path. The good control-path will show the least steps of testing. On the other side, if the control-path (H) takes many steps, the control-path may not be appropriate. The control-path can be written as Eq. 7.

$$H_x = Step(x) \qquad (7)$$

Where; x is the number of step for testing a test case. The shortest control-path has minimum steps of testing the test case. Accordingly, a test case with the minimum step is required.

Algorithm of creating the control-path

```
if $H = Step(1)$ then
  create $H_1$
  else if $H = Step(2)$ then
  create $H_2$
        else if $H = Step(x)$ then
        create $H_x$
    end if
```

Therefore, a set of the control-path, H = $\{H_1, H_2, H_3,...,H_x\}$

(4) Select the appropriate test cases.

According to the algorithm of creating control-path, a set of $H_{\min}$ is constructed. In fact, the result of $H_{\min}$ can show the test cases number that has the minimum steps. For example, if $H_{\min} = \{t(1), t(10), t(45), t(100), t(124)\}$, then $t^* = \{t(1), t(10), t(45), t(100), t(124)\}$. Therefore, the numbers of the selected test cases equal to 5 or $t^* = 5$.

(5) Find the reduction rate (RR) can be written as Eq. 8.

$$RR = \frac{T - t^*}{T} \qquad (8)$$

The reduction rate is the ratio of the remained test cases and a test suite.

(6) Determine the faultless rate (FR) can be written as Eq. 9.

$$FR = 1 - \frac{t^* - B}{t^*} \qquad (9)$$

The faultless rate refers to the possibility of finding bugs in a set of the selected test cases. Eq. 9 is required for evaluating the ability of the comparative studies, e.g., RS, SS, and LS. However, the value of *B* is assumed to be 1 for the computation. This means every selected test case should avoid the numbers of bug. In worst situation, the only bug is allowed in a set of selected test case. However, the bug needs to be fixed by the programmer before the deployment.

Results and Discussions

In facts, there are many selection techniques are developed for improving the performance of eliminating the size of a test suite. In this paper, there are three comparative studies, which are RS, SS, and LS. This is because RS is the well-known and simplest technique that is used in the part of evaluating the ability of the comparative studies. Another is SS, which is the powerful technique in the field software maintenance. As we can see, Table 3 shows the numbers of the selected test cases by RS, SS, and LS. The numbers of the selected test cases by LS are lower than others. Therefore, one of the benefits of using LS is to provide the smallest size of a test suite.

Table 3 The numbers selected test cases of the comparative studies

Program Name	RS	SS	LS
print-tokens	382	318	67
print-okens2	299	389	76
replace	426	398	73
schedule	483	225	50
schedule2	57	234	56
tcas	203	83	50
totinfo	214	199	148

Table 4 presents the reduction rate of all programs that can be computed by using Eq. 8. The example of computing the reduction rate of LS on the program named print-token is shown as; .

$$RR = \frac{4130 - 67}{4130} = 0.9838$$

The results of finding the reduction rate due to the same computation of RS and SS are 0.9075 and 0,9230 respectively. According to this result, the reduction rate of RS is lower than SS and LS. This is because the numbers of the selected test cases are higher than others. Therefore, the second contribution of LS is to provide the higher reduction rate compared with the traditional techniques.

Table 4 Reduction Rate of the comparative studies

Program Name	RS	SS	LS
print-tokens	0.9075	0.9230	0.9838
print-okens2	0.9273	0.9055	0.9815
replace	0.9231	0.9282	0.9868
schedule	0.8177	0.9151	0.9811
schedule2	0.9790	0.9137	0.9793
tcas	0.8738	0.9484	0.9689
totinfo	0.7966	0.8108	0.8593

Table 5 shows the value of faultless rate of the comparative studies, which can be calculated by using Eq. 9. The example of computing the faultless rate at least one bug found of LS on the program named print-token is shown as;

$$FR = 1 - \frac{67 - 1}{67} = 0.0149\,.$$

The results of finding the faultless rate due to the same computation of RS and SS are 0.0026 and 0,0031 respectively. Regarding to Table 5, we can summarize that the results of finding the faultless rate of LS are higher than others. This means that the probability of producing bugs in a whole set of the selected test cases by using LS is less than RS and SS.

Table 5 Faultless Rate of the comparative studies

Program Name	RS	SS	LS
print-tokens	0.0026	0.0031	0.0149
print-okens2	0.0033	0.0026	0.0132
replace	0.0023	0.0025	0.0137
schedule	0.0021	0.0044	0.0200
schedule2	0.0175	0.0043	0.0179
tcas	0.0049	0.0120	0.0200
totinfo	0.0047	0.0050	0.0068

Conclusion

The Lawanna Selection model is the alternative technique for the process of software maintenance by using the concept of regression test selection. It provides the process of selecting the minimum numbers of the test cases while the performance of the program is preserved. Particularly, when compare LS with the traditional techniques such as RS and SS. There are three benefits of using LS. First, the size of the selected test cases by using the LS is smaller than applying the RS and SS. Second, it gives the higher reduction rate than the traditional techniques. Third, LS gives lower numbers of producing the new bugs than RS and SS technique.

Reference

Agrawal, H., Horgan, J., Krauser, E., and Londo, S. (1993) Incremental regression testing. In *Proceedings of the Conference on Software Maintenance*, 348-357.

Ball, T. (1998) The limit of control-flow analysis for regression testing. In *International Symposium on Software Testing and Analysis*, 143-242.

Carmel, E. (1995) Cycle Time in Packaged Software Firms. *Journal of Product Innovation* 12:110-123.

Chapin, N., Hale, J. E, Ramil, J. F., and Tan, W. (2001) Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution* 13(1): 3-30.

Cohen, S. (2010) A Software System Development Life Cycle Model for Improved Stakeholders' Communication and Collaboration. *International. Journal of Computers, Communications and Control* 5(1): 20-24.

Harrold, M. J., Gupta, R., and Soffa, M. L. (1993) A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology* 2(3): 270-285.

Harrold, M. J. (1999) Testing Evolving Software. *Journal of Systems and Software* 47(2): 173-181.

Hutchins, M., Foster, H., Goradia, T., and Ostrand, T. (1994) Experiments on the effectiveness of dataflow- and control flow-based test adequacy criteria. *International of Software Engineering* 16(1): 191-200.

Larman, C. and Basili, V. R. (2003) Iterative and Incremental Development: A Brief History. *Journal of Computer* 36(6): 47-56.

Leau, Y. B., Loo W. K., Tham, W. Y., and Tan S. F. (2012) Software Development Life Cycle AGILE vs Traditional Approaches. *International Conference on Information and Network Technology* 37(1): 162-167.

Leung, H. K. N. and White, L. J. (1991) A cost model to compare regression test strategies. In *Proceedings of the Conference on Software Maintenance*, 201-208.

Niessink, F. and Van, V. H. (2000) Software maintenance from a service perspective. *Journal of Software Maintenance and Evolution : Research and Practice* 12(1): 103-120.

Musa, J. (1993) Operational profiles in software reliability engineering. *IEEE Software* 10(2): 14-32.

Ostrand, T. and Balcer, M. (1988) The category-partition method for specifying and generating functional tests. *ACM Transactions on Software Engineering and Methodology* 31(6): 676 - 686.

Grave, T. D., Harrold M. J., Kim, J. M., Porter, A., and Rothermel, G. (2001) An empirical comparison of regression test selection techniques. *ACM Transactions on Software Engineering and Methodology* 10(2): 184-208.

Rothermel, G. (1996) Efficient, effective regression testing using safe test selection techniques. *Technical Report Clemson University* 96-101.

Rothermel, G. and Harrld, M. J. (1996) Analyzing regression test selection techniques. *IEEE Transaction of Software Engineering.* 22(8): 529-551.

Rothermel, G. and Harrld, M. J. (1997) A safe, efficient regression test selection technique. *ACM Transactin of Software Engineering* 6(2): 173-210.

Rothermel, G. and Harrld, M. J. (1998) Empirical studies of a safe regression test selection technique. *IEEE Transaction of Software Engineering* 24(6): 401-419.

Wong, W., Horgan, R., London, S., and Mathur A. (1997) A study of effective regression testing in practice. In *8th International Symposium on Software Reliability Engineering*, 264-275.

Wong, W. E., Horgan, J. R., Mathur, A. P., and Pasquini, A. (1997) Test set size minimization and fault detection effectiveness: A case study in a space application. In *Proceedings of the 21st Annual International Conference on Computer Software and Applications.* 522-528.