KKU Res. J. 2013; 18(5): 739-748
http : //resjournal.kku.ac.th

The Effective Selection Model for the Process of Test Case Selection

Adtha Lawanna[1]

[1] *Lecturer, Department of Information Technology, Faculty of Science and Technology, Assumption University, Bangkok 10240, Thailand.*
* *Correspondent author: adtha@scitech.au.edu*

Abstract

This article presents the effective selection model for choosing the small numbers of test cases, relying on the concept of software testing. Currently, the traditional models that are the random and regression test selection are being used in the process of software testing. These models can create the test suite that contains the set of test cases. The test cases are used to fix bugs before deploying the new software version to the users. The main problem is the size of test cases.This can increase programming times, including in the new software may produce any error, which can fail the entire software system. Therefore, this paper offers the alternative techniques to improve the ability of design test suite and select the small numbers of test cases, while avoiding the fault is also considered. The algorithm of determining function modified, lines of code changes are created. Moreover, the algorithm of finding test suite and test cases are explained in details. For instance, numbers of the selected test case in tcas program by using random, regression selection and the proposed model are 266, 208, and 184 respectively. According to this, the proposed model gives the smallest size of a test suite compared with the traditional model.

Keywords : regression test selection, software testing, test suite, and test case.

1. Introduction

The software-development life cycle (SDLC) is one of the parts for creating the program. SDLC comprises getting requirement, analysis, design, coding, testing, and maintenance into the whole processes (1).

Getting requirement involves collecting the needs of users before the entire development starts. The team uses the requirement for producing the functions (*F*) inside the program.

After this, the analysis is prepared in order to identify the related problems with the set of solutions. Software analysis is a process of designing a program by equating pattern or abstract processes relevant to the workings of the software being developed. The method is to compare some type of the abstract processes that work effectively in some function, and then use the same logic and the terminology of the pattern to the software being created (2).

Next, software design is a method of providing a software solution. After the objective and requirements of the program are determined, programmers will prepare or employ designers to build a plan for a software solution. It combines the algorithm implementation

problems as well as the overview of the architectural software system.

Coding is the next process in SDLC, it concerns many facets of developing program and, while they do not affect the functionality of the request, they give the improved ability of source code. For the determination of this section, all lines of code (*L*) are considered.

Then, software testing is applied for checking reliability of the program.It is the process of executing software or the entire system with the intent of discovering reliability. Particularly, it focusses the activities aimed at evaluating an attribute of a program and defining that it meets its vital results. Software is different from other physical practices where inputs are established and outputs are formed (3).

Following this, software maintenance is provided. It is the process of modifying a software product after deployment, e.g., fixing bugs, improving performance and other product attributes, or adapting the product to a new version as well as changing environment.

The selection of test cases (*TC*) for regression testing, it requires the knowledge of fixing bugs fixes and how it affects the whole software system (4). Particularly, it involves the space of frequent faults, and the extent which has undergone many or recent lines of code changes (5). The mandatory requirements of the user selection of test cases for the methods of regression testing depends more on fixing bugs than the defect itself. Moreover, a minor defect can produce the major side effect and a bug fix for an extreme defect can have no a minor side effect. Therefore, the test engineer requires balancing these characteristics for choosing the test cases for regression testing (6).The process of selecting the test cases is shown in Figure 1.

Step 1: Retrieving and analyzing the program.

This is the first step in the process of selection.A test suite will be designed relying on the functions, lines of code and detecting faults, and the modified program.

Step 2: Defining test case.

A set of test case can be defined by two activities. First, it can be done manually by developers. Second, using test case generator creates the test cases automatically.

Step 3: Selecting test case.

The set of test cases are selected from the test suite of the modified program. In this step, many techniques are applied, e.g., retest all, random, regression test selection, minimization, prioritization, and others.

Step 4: Testing the new program.

This step can be done due to the concept of software testing. The main objective is to check and to fix bugs in the lines of code change, while some functions in the program are modified. If bugs are found in some test cases, then that test cases need to be reintroduced until the bugs are removed. But the test cases without bugs will be considered for the selection process.

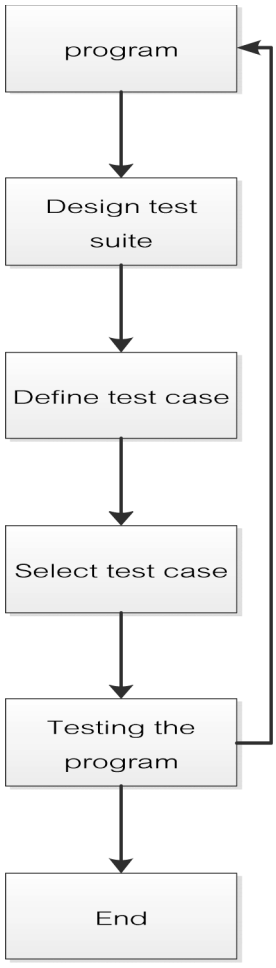

Figure 1. The process of the test cases selection

When a new version of software is released to the users, the programmers need to perform a full run of all the existing test cases. According to this, it is very time consuming. The question, it is possible or not to determine the modified parts of the code base and then execute the test cases related to those parts. However, the regression test is affected by those lines of code and functions modified (7).

Regression testing is denoted as rerunning test cases from existing test suites (*TS*) to build confidence that software changes have no unplanned side-effects. In general, the process is to create the test suite and run it after every single change (8). Unluckily, for numerous developments this is just incredible because test suites are huge, since the changes arise too fast and manyhumans are involved in the testing loop, and highly requirements of simulation laboratories are requested (9). In addition, testing must be completed on many different hardware and operating system platforms.Particularly, the regression testing supports the constancy of the changed program by discovering errors in the modified software, and confirming the continued operation of the new version. This method is a costly and uses significant expenses of resources. During the method, an already given test suite is available for reprocess.

A regression test selection technique can be applied to select an appropriate number of test cases from the given test suite. The simplest and oldest technique is to run all test cases (AT)in the modified source code. This is the useful technique, but it is not practical when the size of a test suite is too large. Therefore, we may choose a set of the test cases by random technique (RT) to decrease the size of the designed test suite. However, many test cases selected by this technique may not be relevant to the modified program. Another technique is called regression test section (RTS) proposes the selection of test cases that perform the changed portions of the software.

This paper focuses Rothermel and Harrold's regression test selection tool because their results are better than retest-all selection and random/ad-hoc selection. This technique constructs the control flow graphs for a program or procedure and its modified program and uses the flow graphs to select test cases that execute modified code from the original test suite. They describe that, under certain conditions, the set of test cases their technique selects includes every test case from the original test suite that can expose faults in the modified program or procedure. Particularly, although their algorithms may choose some test case that cannot

expose faults, they are at least as accurate as other safe regression test selection techniques. Unlike many other regression test selection techniques, their algorithms can handle all types of program modifications and all language constructs. They have implemented their algorithms; initial empirical studies prove that their technique can significantly reduce the cost of regression testing modified program (10).

Particularly, the portions can be affected by these modifications(11). These test cases are known as modification enlightening test cases. According to this, those test cases that reveal bugs in the modified software are known as fault enlightening test cases. Unfortunately, there is no efficient selection technique to discover fault enlightening. We may also specify the preference with which a test case may be prepared during the process of regression testing(12). Mostly, a noble process of software testing depends on the techniques of test case selection. Accordingly, the small size of the test cases can be studied without time consuming. Therefore, the effective test case generator needs to be used very in order by the development team. This is because, the functions modified and lines of code changes can affect the capability of the new software version, whereas the bugs are produced. Hence, not only the small size is required, but the percent faultless should be protected. Moreover, the test team should work collaboratively through the whole process of the software-development life cycle, particularly in each step of software testing.This article focuses the part of software testing(13),because there are many researchers proposed to improve the capabilities of the software, including the program modification. Those techniques also present the methods of designing test suite of the software, which contain large amounts of test cases. This causes the testing time and cost increase(14). Although, the techniques of test case selection are created for solving these problems,but the new issue is produced that is the rising of bugs in the new software version after modifying the source code(15).So, this paper proposes the effective selection methods (ESM) to solve the problems mentioned above. The expected contributions of this research are selecting the smaller amounts of test cases thanthe traditional technique.

2. Materials and Methods

This section explains the subject program sued for doing experiments and the methods of selecting the test case in detail.

2.1 The subject programs

The subject programs as shown in Table 1 are written in C language developed by the developers of the Siemens suite of programs with manually fixing bugs or faults(16). 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 (11). Table 2 is the example of modifying 10 versions of print-token2 program. In version 1, there are five functions that were modified and 120 lines of code are changed. Likewise, all programs have their own records, which are used in the experiments.

Table 1. The seven subject programs

Programname	numbers of functions	lines of code	number of version	test suite
replace	21	516	32	5542
print_token	18	402	7	4130
print_token2	19	483	10	4115
schedule2	16	297	10	2710
schedule	18	299	9	2650
totinfo	7	346	23	1054
tcas	9	138	41	1608

Note: the experiment is provided for determining a test suite for the next software version.

Table 2. The example of modifying ten versions of print-token2

version	functions modified	lines of code changed
1	5	120
2	14	153
3	13	185
4	9	127
5	18	166
6	9	170
7	19	164
8	13	190
9	19	101
10	2	191

2.2 The proposed model

This section describes the details of determining the selected test cases. The proposed methodology is called the effective selection model (ESM). There are four steps explained as follows;

Step 1: Determine the functions modified.

If $FM \propto F$

then $FM = \beta F$ (1)

or $\beta = \frac{FM}{F}$ (2)

else if $FM \propto \frac{1}{F}$ // $\propto$ refers to "varied" (3)

then $FM = \theta \frac{1}{F}$ (4)

EndIf

where

FM is the functions modified.

F is the number of functions.

C is the lines of code.

β is the estimated constant value, whereas FMis varied by C.

θ is the estimated constant value, whereas FM is undirected to C. This value is important C has not changed.

Step 2: Createthelines of code change.

If $LC \propto C$

then $LC = \omega C$ (5)

or $\omega = \frac{LC}{C}$ (6)

elseif $LC \propto \frac{1}{C}$

then $LC = \sigma \frac{1}{C}$ (7)

EndIf

where

LC is the lines of code changed.

ω is the estimated constant value, whereas LC is varied by C.

σ is the estimated constant value, whereas LC is undirected to C. This value C has not changed.

Step 3: Buildthe test suit.

If $V \propto FM$

then $V = \lambda FM$ (8)

esleif $TS \propto LC$

then $TS = \lambda LC$ (9)

elseIf $TS \propto \frac{1}{LC}$

then $TS = \mu \frac{1}{LC}$ (10)

EndIf

where

V is the faulty versions.

TS is the test suite.

λ is the estimated constant value, whereas V is varied by FM.

μ is the estimated constant value, whereas TS is varied by LC. This value is necessary, whereas FM does not exist.

Step 4: Select the test cases.

If $TC \propto TS$

then $TC = \kappa TS$ (11)

elseif $TC \propto \frac{1}{TS}$

then $TC = \varnothing \frac{1}{TS}$ (12)

EndIf

where

TC is the selected test case.

κ is the estimated constantvalue of selecting test case.

$\varnothing$ is the estimated constant value, whereas TC is undirected to TS. The value of $\varnothing$ will be applicable when the selected test case are very small (e.g., 2-3). Using this model, the value of $\varnothing$ is invalid. Therefore, it is not mentioned in the results.

3. Results and Discussions

The experiments are set and tested to predict the value of function modified, the lines of code change, and selected test cases for the next version of each program. The records in this section are useful for the programmers to get the guidelines to make a good plan to modify the software. According to the algorithm in Step 1, the value of β and FM can be computed. For example, there are 19 functions in print-token2 and the versions are modified for ten times. Then the software version 11 comprises two functions modified, while β equals 0.1. All results are shown in Table 3.

Table 3. The results of determining β and FM

program name	F	FM	β
replace	21	2	0.1
print_token	18	7	0.39
print_token2	19	6	0.32
schedule2	16	8	0.5
schedule	18	9	0.5
totinfo	7	6	0.86
tcas	9	6	0.67

The objective of finding β is to predict function modified for the next generation. In part of finding θ is not shown, this is because it does not relate to the functions in each program, including the results are consistency. However, this value will be applied,

whereas the lines of code are very short with none of the bugs. After this, is to measure the changes in the source code by 100 experiments. The values of ω that are shown in Table 3can be applied in order to find the value of *LC*, while σ gives the negative because most of the codes are changed.The value of σ is not determined because it doesn't affect this situation.

Table 4. The results of finding ω and *LC*

program name	*C*	*LC*	ω
replace	516	249	0.48
print_token	402	176	0.44
print_token2	483	161	0.33
schedule2	297	127	0.43
schedule	299	80	0.27
totinfo	346	41	0.12
tcas	138	118	0.86

In addition, using these results can help the programmers to check bugs easier than the entire lines of code. Next step is to provide the value of λ for constructing the test suite, as shown in Table 5. On the other hand, the values of μ are not recorded because there are the functions modified in each program. This means μ will be useful, when the functions are not changed, which not relate the properties of seven subject programs used in this paper.

Table 5. The results of finding λ and *TS*

program name	*TS*	λ
replace	516	0.48
print_token	402	0.44
print_token2	483	0.33
schedule2	297	0.43
schedule	299	0.27
totinfo	346	0.12
tcas	138	0.86

As shown in Table 6, finally, the values of κ are performed relying on λ. The reason is that λ is varied on *FM* and *LC*. There *TC* of each program is identified. In this Step, *TC* presents that $TC = \{t_1, t_2, t_3,...t_n\}$ For example, in the program namely replace shows that there 133 selected test cases. From the experiments, the test case numbers 1 to 133 are selected. Some ofthe examples; $t_1$provides Login function,$t_2$is for registration, and $t_3$is the method of verification. However, the abilities of the select test cases are effective for the program, when all of them are created properly. This may depend on the design from the programmers at the previous process.

Table 6. The results of finding κ and *TC*

program name	*TC*	κ
replace	133	0.03
print_token	120	0.03
print_token2	109	0.03
schedule2	48	0.02
schedule	143	0.06
totinfo	130	0.09
tcas	33	0.04

The comparative studies used in this paper are random (RT), regression test selection techniques and the proposed model.The reason of using RTis becausethe oldest and simplest methods. Besides this, the RST is applied as the well-known methods because it gives the good results for improving the ability ofverification and validation, while the entire processes are being developed (16).

There are five criteria for the evaluations explained as follows;

Criteria 1:The amounts of the selected test cases.

The results of the comparative studies indicate that the amount of test case inESM is the smallest as

shown in Figure 2. This can guarantee that the ability of ESM is better than RT and RST through the process of software testing. In the concept of software testing, the specialists tried to propose the techniques that can select the appropriate numbers of test cases as small as possible. According to this, the small sizes of test cases can help the programmers to avoid time consuming when executing the new version.

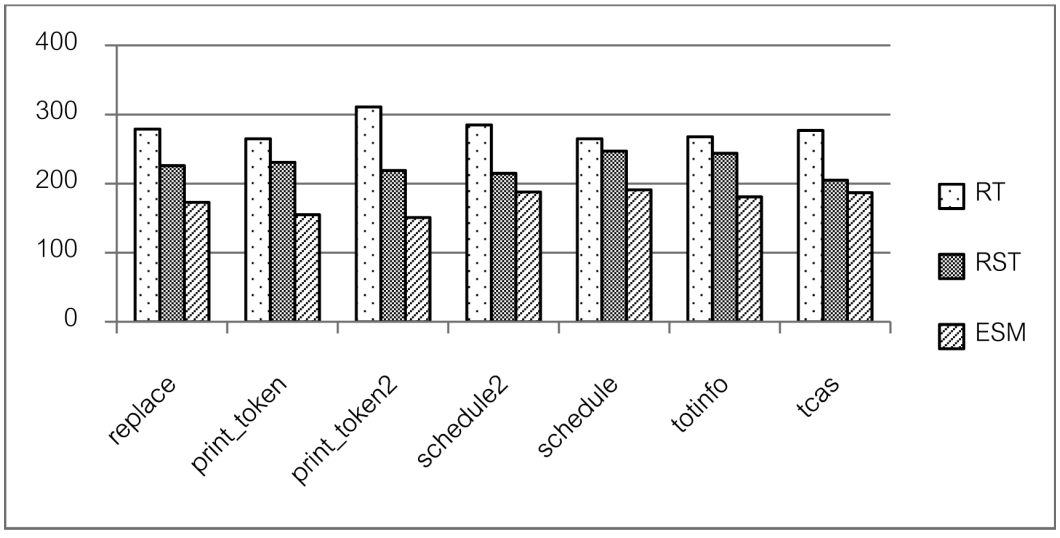

Figure 2. The size of the selected test case of the seven subject program regarding RT, RST, and ESM

Criteria 2: The ability of reduction.

The percent reductions of the comparative studies are shown in Figure 3. As we can see that the results of ESM for each subject program are higher than others. However, the results cannot prove it is the best technique for reducing the size of the program.

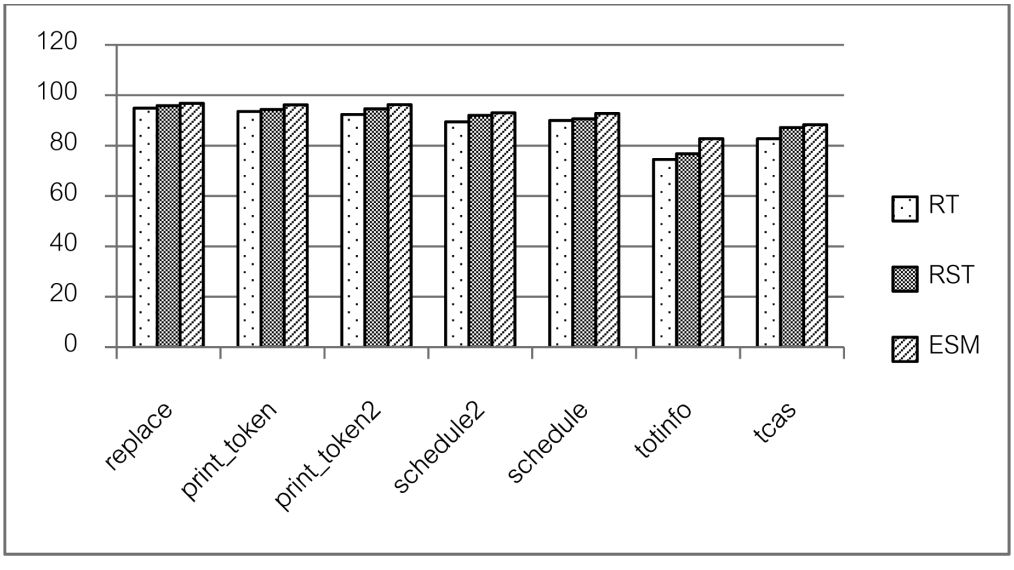

Figure 3. The ability of the reductions

Criteria 3:The amounts of the test cases at least one fault.

The amounts of the test cases at least one fault that can be occurred in the new source code, as shown in Figure 4. For example, the results from RT, RST, and ESM applying to print-token are 274, 215, and 198 respectively. This explains that the numbers of test cases by using ESM is the smallest, while only one bug is found in the program. Come to this point, in summary, the ability of ESM is a littlebit test caseslower than the comparative studies.

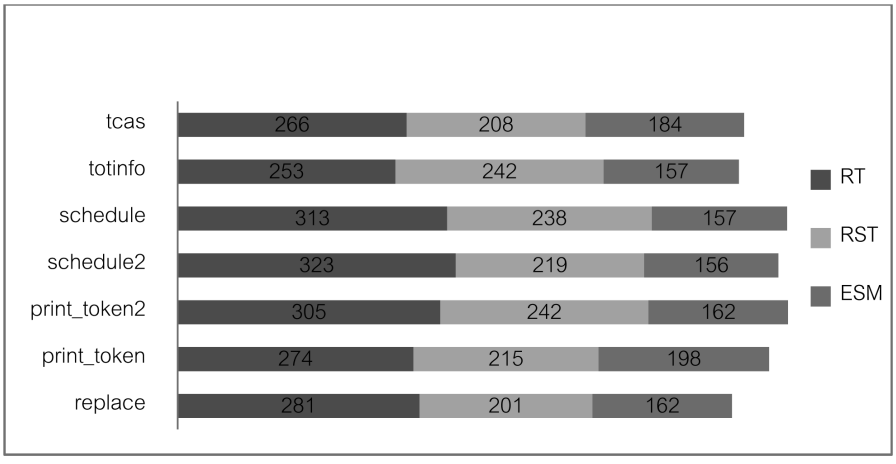

Figure 4. The amounts of the test cases at least one fault

Criteria 4: The evaluation of the programmer of the proposed model is assumed that each programmer has the same skill, knowledge, and experience. The reason is that this paper cannot explain some variables in term of mathematic model. According to the algorithms designed in this paper can show only some parts of the relationships.

Criteria 5: This paper is applicable for the object-oriented programing such the seven subject program used in the experiment is created by C programming.For the future work, the different programming should be considered.

4. Conclusion

Programmers in the development software system more seriously work hard on the process of software testing. One reason that may fail the entire software system is the unintended bugs, which can be occurred after modifying the new program. Moreover, some functions are modified in the new software version. This results in the size of source code. According to this, the lines of code may change that affect the quality of using test cases, including the size of the program may increase. Therefore, this paper proposes the selection methods that cover the issues mentioned above. Finally, the effective selection technique is created. And it shows the better results in terms of the smaller size of test, higher reduction than the comparative studies. Furthermore, the proposed model also gives the smaller numbers of test cases, while at least one fault is occurred in each program. However, there are many techniques proposed to select the appropriate numbers of the test cases, e.g., minimization and prioritization techniques. In fact, there is no best technique to guarantee the minimum numbers of test cases using in software testing without any happeningerror. Consequently, in the future works, the simulation and optimization technique should rely

on the concept of case-based reasoning be applied to thisbecauseit will generate, reuse, revise, and retain the relevant test cases with the competence preservation. This refers to the new technique should give the better result not only the size of test case but include the whole performance of the software system.

5. Acknowledgement

This research is supported by the software engineering laboratory (SEL), Faculty of Science and Technology, Assumption University, Bangkok, Thailand.

6. References

(1) Munassar1 NMA, Govardhan A. A Comparison Between Five Models Of Software Engineering. IJCSI. 2010; 1 7(5): 94-101.

(2) Royce W. Managing the development of large software systems.Proceedings of IEEE WESCON;1970 Aug 26; Los Alamitos, CA, USA; 1970. P.1-9.

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

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

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

(6) Barton J, Czeck E, Segall Z, Siewiorek D. Fault injection experiments using FIAT. IEEE Transactions on Computers. 1990; 39(4): 575–582.

(7) Mishra KK, Misra AK. Regression Testing: A Spectrum-based Approach. Journal of Computer Applications. 2010; 5(18): 35-42.

(8) Grave TD, Harrold MJ, Kim JM, Porter A, Rothermel. G. An empirical comparison of regression test selection techniques. ACM Transactions on Software Engineering and Methodology. 2001;10(2): 184-208.

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

(10) Rothermel G, Harrold MJ. Empirical studies of a safe regression test selection technique. IEEE Transactions on Software Engineering. 1998; 24(6): 401-419.

(11) Munson JC, Khoshgoftaar TM. The detection of Fault Prone Programs.IEEE Transaction on Software Engineering. 1992; 18(5): 348-357.

(12) Xie T, Notkin D. Checking inside black box: Regression testing by comparing value spectra. IEEE Transaction on Software Engineering. 2005; 31(10): 869-883.

(13) Rothermel G,Harrold MJ. Analyzing regression test selection techniques.IEEE Transactions on Software Engineering. 1996; 22(8): 529-551.

(14) Rothermel G,Harrold MJ. Experience with regression test selection.In International Workshop for Empirical Studies of Software Maintenance.1996: 178-188.

(15) Tao C, Li B, Gao J. A Systematic State-Based Approach to Regression Testing of Component Software.Journal of Software. 2013; 8(3): 560-571.

(16) Rothermel G,Harrold MJ. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology. 1997; 6(2): 173-210.