國立交通大學

多媒體工程研究所

碩 士 論 文

標籤式電腦程式同儕互評

Programming Peer Assessment Using Tagging Approach

研 究 生：劉怡利

指導教授：曾憲雄　教授

中 華 民 國 九 十 七 年 六 月

標籤式電腦程式同儕互評

Programming Peer Assessment Using Tagging Approach

研 究 生：劉怡利　　　　Student：Yi-Li Liu

指導教授：曾憲雄　　　　Advisor：Shian-Shyong Tseng

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis
Submitted to Institute of Multimedia Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

標籤式電腦程式同儕互評

學生：劉怡利　　　　　　　　　　指導教授：曾憲雄博士

國立交通大學資訊學院
多媒體工程研究所

摘　　要

程式技能包含，程式撰寫、程式追蹤、測試及除錯，是程式設計師應該具備重要的能力。為了讓程式的初學者於初學階段能知道自己的程式迷失概念，程式設計的課程中會有程式的評量活動來幫助學生學習。但是因為傳統的程式評量活動，例如、上機考試，以程式執行的對錯來評量評分，這使得程式有錯卻又不知錯誤所在的學生來說成效不佳；並且學生程式追蹤、測試及除錯的能力在傳統的評量中也不易被評量。在本篇論文提出一個新的程式評量方法，用同儕互評的方式讓學生在互評活動中找尋同學的程式錯誤，使得撰寫程式的同學可以得到除了執行結果外的評論；而評論的同學除了在評量的過程中練習到程式追蹤、測試及除錯外，評論的資料也會被記錄及分析，因此學生的程式技能可以在此評量活動中多方的得到檢測，學生的學習狀態及結果也可以作為老師實施補救教學的參考。由我們的實驗得知，學生的程式迷失概念可以被我們的評量方法找出來，並且我們於評量活動中提出的追蹤指引可以幫助學生學習如何追蹤、測試及除錯，讓學生的程式技能得到改善。

關鍵字：電腦程式同儕互評、同儕互評、電腦程式學習

Programming Peer Assessment Using Tagging Approach

Student: Yi-Li Liu Advisor: Dr. Shian-Shyong Tseng

Institute of Multimedia Engineering
College of Computer Science
National Chiao Tung University

ABSTRACT

Most of the programming novices feel hard to learn programming skills, such as coding, tracing, debugging and testing. To help them learn programming well, computer program assessments which can find out the novices' programming misconception have been proposed. Although traditional computer program assessment can score the student's program by executing the program with test cases, it is ineffective because the novices' programming misconception in the coding mistakes and tracing mistakes can not be found out. In the peer assessment activity, the learners can discuss their works with others to get the comments about his/her work together with the score. This thesis proposes a new computer program peer assessment to help the novices easily find out their programming misconception in the coding mistakes and tracing mistakes. An experiment has been done to evaluate the effectiveness of the new assessment. We may conclude that the process is useful to find out the learner's coding and tracing problems, and that the tracing guidance is useful to direct the novice to trace peer's program.

Keywords: computer program peer assessment, peer assessment, computer program learning

誌　　謝

首先要感謝的是我的指導教授，曾憲雄博士。碩士班期間，曾教授相當細心與耐心的指導我研究的方法；從老師的身上學習到許多研究的方法與態度，寶貴的經驗讓我獲益良多開闊學習的視野，不甚感激！因此也讓我能順利的完成此篇論文。同時也感謝我的口試委員，孫春在教授、黃國禎教授以及楊鎮華教授所給予的寶貴意見，讓我的論文研究能夠更有價值。

接下來要感謝感謝翁瑞鋒、林喚宇、曲衍旭學長們不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑。在這短短兩年中讓我學會許多理論知識及實務技巧，並給予我許多論文上的寶貴意見，協助我論文上的修改工作，使得這篇論文能夠順利的完成，也感謝念主、立晧同學的幫忙及互相勉勵，陪伴我渡過這充實的碩士生涯。

另外要感謝我的父母親在背後默默地支持我完成我的學業，讓我可以安心的做研究而不需要煩惱生活上的問題。最後要特別感謝我的男朋友喚宇，在我身邊不時地關心我，並在我感到挫折時給我支持的力量，也讓我能夠自信地、樂觀地面對一切難題。日後，我會更加努力的繼續向前進！

Table of Content

摘　　要 I
ABSTRACT II
誌　　謝 III
Table of Content IV
List of Figures V
Chapter 1.　Introduction 1
Chapter 2.　Related Work 4
　2.1.　Computer Program Peer Assessment with Text-based comment 4
　2.2.　Computer Program Peer Assessment with Questionnaire-based comment 4
Chapter 3.　Computer Program Peer Assessment Architecture 6
Chapter 4.　Computer Program Peer Assessment Tag Schema 8
　4.1.　The Reviewer Tag Schema 8
　　4.1.1.　Structure Tag Schema 8
　　4.1.2.　Test Case Tag Schema 10
　　4.1.3.　Mistake Tag Schema 11
　4.2.　The Programmer Tag Schema 14
　　4.2.1.　The Feedback Tag Schema 14
Chapter 5.　Experiment 16
　5.1.　Programming Assignment 16
　5.2.　Computer Program Peer Assessment System 18
　　5.1.1.　The Web System for Exchanging Document among Peers 18
　　5.1.2.　The Microsoft Word for Making Tag-based Comments 19
Chapter 6.　The Misconception Pattern Analysis 22
　6.1.　The distribution of four kind of tag-based comment 22
　6.2.　The evaluation of the learners’ tracing ability 23
　6.3.　The evaluation of the learners’ testing ability 25
　6.4.　The evaluation of the learners’ debugging ability 28
　6.5.　Discussion 30
Chapter 7.　Conclusion 31
Reference 32

List of Figures

Figure 1: Computer Program Peer Assessment 2

Figure 2: Marking criteria in Unix programming module........ 5

Figure 3: Computer Program Peer Assessment Architecture 7

Figure 4: Structure Tag Schema 9

Figure 5: Using structure tag schema to mark the program's non-sequential statements 10

Figure 6: Test Case Tag Schema 10

Figure 7: Using test case tags to record the execution status of program 11

Figure 8: Mistake Tag Schema 12

Figure 9: Using the mistake tags to denote the program mistake of initial values........ 13

Figure 10: Using mistake tag to indicate the mistake on the test case tags 13

Figure 11: Feedback Tag Schema 14

Figure 12: The example of accepting the reviewer's comments. 15

Figure 13: The GUI of the program assignment........ 17

Figure 14: The coding area for the novice programmers 17

Figure 15: The image of success on the GUI 18

Figure 16: The image of failure on the GUI........ 18

Figure 17: Computer Program Peer Assessment System 19

Figure 18: Review tags predefined in Microsoft Word document. 20

Figure 19: The predefined execution table in Microsoft Word document........ 20

Figure 20: Feedback tags predefined in Microsoft Word document 20

Figure 21: The tagged program 21

Figure 22: The distribution of the novice programming learner's tag-based comment 23

Figure 23: The distribution of correctness of the novice reviewers' structure tag comments.. 24

Figure 24: An example of the wrong structure tag comments made on tagging the nested conditional statements. 24

Figure 25: The distribution of correctness of the novice reviewers' test case tag comments .. 25

Figure 26: An example of tagged program made by the novice reviewer........ 26

Figure 27: An example of the execution table made by the novice reviewer........ 26

Figure 28: The watch point made by the novice reviewer 27

Figure 29: An example of the execution table made by the novice reviewer 28

Figure 30: The distribution of different level of mistake tags used by the novice reviewers... 29

Figure 31: An example of the syntax error program tagged by the mistake tag 29
Figure 32: An example of the execution table annotated the syntax error of the program 30

Chapter 1. Introduction

The basic programming skills, such as coding, tracing, debugging, testing, etc., are of most importance for a programmer to implement, maintain, and extend computer programs. However, it is hard for a novice to learn these programming skills. In order to find out the misconception of the novice programmers, a computer program assessment is needed [1-4]. As we know, the traditional computer program assessment uses test cases to evaluate the correctness of the learners' program assignments. When the program can not be correctly executed for the test cases, the learner's coding ability still can not be enhanced without being taught the knowledge about the mistakes in the program. Furthermore, the traditional assessment does not take the tracing ability of the learner into account. Therefore, how to evaluate the learners' coding and tracing ability is our concern.

According to the pedagogical theory [5], we know that peer assessment [6, 7] is an effective learning method for learners to understand others' work [8-11]. In the peer assessment activity, the learners usually discuss their works with others, and every learner can get the comments about his/her work together with the score. Using the peer assessment for learning programming, the learner can get the comments about their programming mistakes from others, so that more information rather than the result of program execution can be obtained to help learner understand his/her programming misconception in the coding mistakes and tracing mistakes. Therefore, computer program peer assessment is a good approach to improve the novices' programming skills.

In the computer program peer assessment activity as shown in Figure 1, a learner, who plays the role of a programmer, implements the program to satisfy the requirements and solve the problem, and then another learner, who plays the role of a reviewer, reviews the programmer's program and gives comments about the coding mistakes. According to the reviewer's comments, the programmer may modify himself/herself program or discuss the comments with the reviewer to improve the coding ability. Besides, the reviewer also can improve the tracing ability by reviewing the programmer's program and being notified whether the comments have been adopted or not after the discussion with the programmer. Furthermore, all of the comments and programmer's feedbacks can be collected as a data source from which data analysis or data mining can be done to find out the frequent programming misconception pattern of the learners to help the teacher to design the remedial instruction.

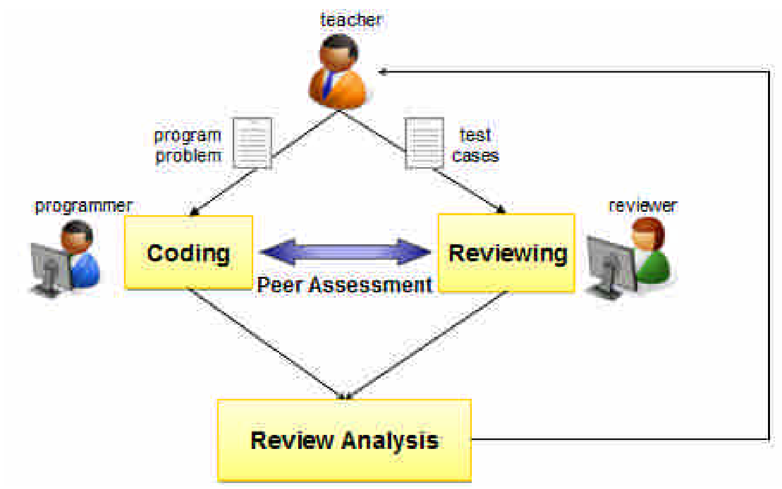


Figure 1: Computer Program Peer Assessment

In previous research about computer program peer assessment [12-14], the text-based comments [15-17] and the questionnaire comments [16, 18] are usually used to communicate among peers. The former is hard to be used to precisely point out the mistakes of the peer's program, and also hard to be dealt with by the computer program to provide teachers some information about the activity. The latter is based on the research using the questionnaire of 5-point Likert scale to help the reviewer make comments, so the reviewer can easily make the comments by only choosing the question, but only the comments, which are covered by the questions in the questionnaire, can be expressed.

As mentioned above, it is hard for a reviewer to give precise comments for the concerned problems, where the reviewer's programming skills should be good enough to give the correct comments, and how to make good use of the comments, the important resource about the learners' learning status, to help finding the frequent misconception of the learners is also an interesting issue.

Therefore, our idea is to develop a tag-based comment environment for the learner to comment using appropriate and predefined tags. We propose the tracing guidance to guide the novice learner tracing code and finding the coding mistake by providing the proper demonstration. We also propose the feedback mechanism for the programmer to express his/her acceptance or rejection about the reviewer comments. When the programmer does not agree with the reviewer's comment, the feedback mechanism becomes the peer assessment of the reviewer's reviewing ability or tracing ability.

This paper proposes a new Computer Program Peer Assessment Activity using tag-based comment to help the learners to improve their programming skill effectively. In the scheme, after the programmer implementing the program given by the teacher, the reviewers can make tag-based comments according to the running status of the test cases using the Authoring Tool with Program Tagging Schema, where three tasks for different tracing purposes are proposed to guide the reviewer to trace the program. The first one is to trace the structure of the program, the second one is to trace the execution of the program, and the third one is to point out the mistakes in the program. After the reviewers finish the review, the Review Analysis Process will analyze the tag based comments and provide learners' learning status for teachers to help the learner learning programming well.

Finally, an experiment has been done to evaluate the effectiveness of the Computer Program Peer Assessment Activity. We can conclude that the process is useful to find out the learner's tracing problem. The tracing guidance is useful to direct the novice learners to trace peer's program.

Chapter 2. Related Work

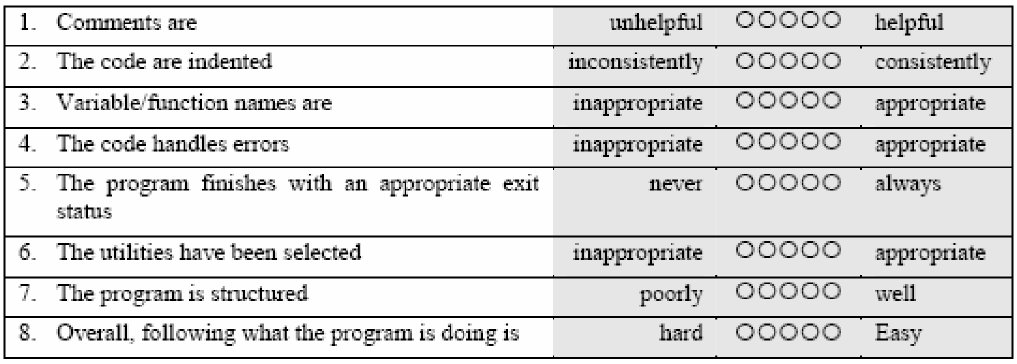
In previous research about computer program peer assessment, the text-based comments [15, 16] and the questionnaire comments [16, 18] were used to communicate among peers.

2.1. Computer Program Peer Assessment with Text-based comment

The research of text-based comments shows that the novices can not understand what comments are necessary and the comments usually give some useless comments. Besides, the text-based comments are difficult to be analyzed automatically to provide teachers information about the learning status in the assessment.

2.2. Computer Program Peer Assessment with Questionnaire-based comment

The other research [18] describes the experiences with a novel web-based peer assessment system deployed on a large class of undergraduate computer science students studying computer programming. In this research, they developed the software containing three components, which are automatic test results, strict marking guideline, and tutor support, to assist the peer assessment activity. The strict marking guideline is the questionnaire of 5-point Likert scale, shown in Figure 2, which was set by the teacher, used by the students to give comments. They found that most of the students understood the marking guideline and could give the comments easily. However, the students can not give the comments that the questionnaire doesn’t cover. The questionnaire is difficult to be designed to fulfill different kinds of program assignments. Therefore, the questionnaire is hard to be used to give precise comments on finding out the programming mistake.



1. Comments are	unhelpful	○○○○○	helpful
2. The code are indented	inconsistently	○○○○○	consistently
3. Variable/function names are	inappropriate	○○○○○	appropriate
4. The code handles errors	inappropriate	○○○○○	appropriate
5. The program finishes with an appropriate exit status	never	○○○○○	always
6. The utilities have been selected	inappropriate	○○○○○	appropriate
7. The program is structured	poorly	○○○○○	well
8. Overall, following what the program is doing is	hard	○○○○○	Easy

Figure 2: Marking criteria in Unix programming module

Chapter 3. Computer Program Peer Assessment Architecture

In a computer program peer assessment, we need to overcome three issues to improve the learning effectiveness of programmers, reviewing, and teaching effectiveness of teachers.

- How to help reviewers give precise comments, focusing on concerned problems?
- How to evaluate reviewers' tracing ability in the activity?
- How to mine the learners' learning status from data in the peer assessment?

Thus, we propose a tag-based computer program peer assessment, where tags are defined to guide reviewers to give precise comments. The programmers, who receive the tag-based comments, can challenge the comment tags to give reviewers precise feedback. Besides, the predefined tags are easy to be analyzed automatically to help teachers understanding the learners' learning status.

Accordingly, we propose a Computer Program Peer Assessment Architecture (CPPAA), shown in Figure 3.

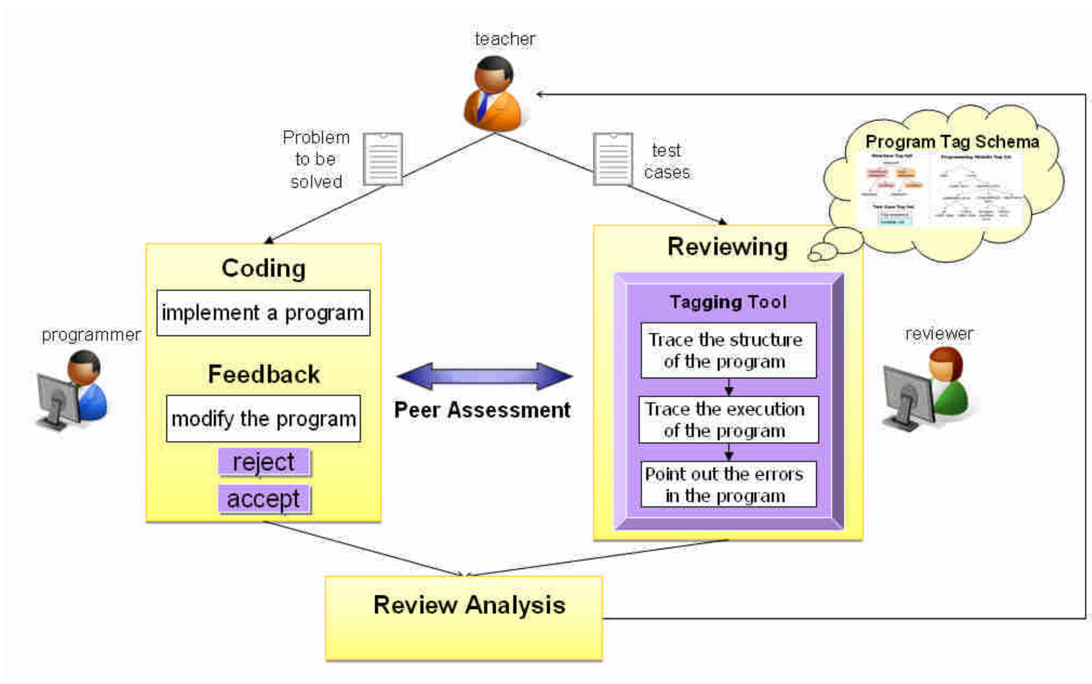


Figure 3: Computer Program Peer Assessment Architecture

In the beginning of the peer assessment, teachers give learners a programming assignment with a set of test cases. After the first round of coding activity, each completed program will be reviewed by other two programmers, who have the similar ability of programming. In the reviewing activity, each learner needs to trace others programs to annotate the programs' structures, test cases running status, and program mistakes with predefined tag schemas. In the second round of coding activity, the reviewing results will be returned to the programmers to help them enhance their codes, if the programmers do agree with the comments, given by reviewers; otherwise, they can reject. Finally, the programmers' feedback will be given to the reviewers to let reviewers know the effectiveness of their comments. After the peer assessment, in the review analysis, the frequent patterns of coding and tracing mistakes can be mined automatically and can be referred by teachers to design the supplementary instructions.

Because our targeted learners are novice programmers, the scope of the programming language is limited to the basic concept of structured programming language, including conditional statement, repetition statement, and the build-in data type. Thus, not only the advanced programming language concepts, such as *"function"* and *"object"*, but also the *"goto statement"* which is easy to break the execution flow of a program, are excluded in our programming language scope.

Chapter 4. Computer Program Peer Assessment Tag Schema

In CPPAA, reviewers will use the predefined reviewer tag schema to annotate their reviewing results, and programmers can use the programmer tag schema to give feedback to reviewers.

4.1. The Reviewer Tag Schema

The reviewer tag schema includes structure tag schema, case tag schema, and mistake tag schema to guide learners to recognize the structure of the traced program, use the test cases to evaluate the correctness of the program, and find out the mistakes in the programs.

4.1.1. Structure Tag Schema

The structure tag is used to describe the structure of the program. In general, the statements in the program code are executed sequentially if there is no *"conditional statement"* or *"repetition statement"* in the program code; otherwise, the program may not be executed sequentially, and may result in increasing the difficulty for the novices to trace the program. Therefore, we guide the learner to mark the scope of non-sequential statements, such as conditional statements and repetition statements, firstly.

As shown in Figure 4, the structure schema set is a hierarchical structure, where the tag *"conditional statement"* is used to tag the conditional block and its child tag *"condition"* is used to tag the condition statement in the conditional statement block. The tag *"loop statement"* is used to tag the loop block in the program and its child tag *"condition"* is used to tag the condition statement in the loop block. Since the structure tag set is also a composite structure, the nesting loop or nesting condition can also be tagged.

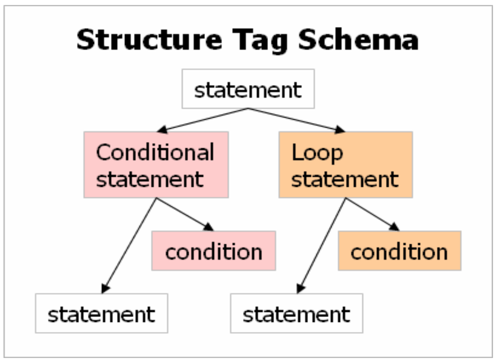
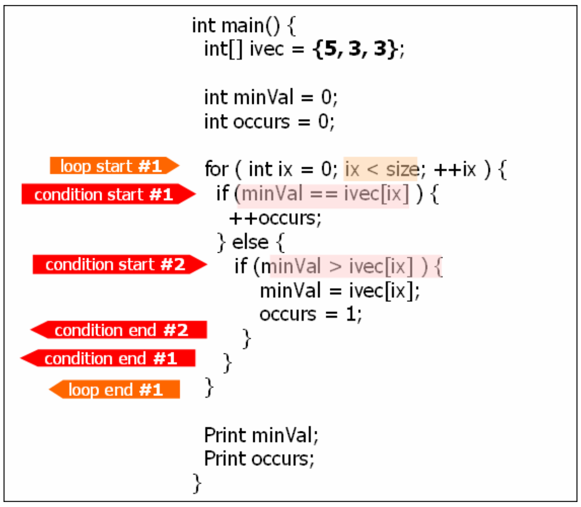


Figure 4: Structure Tag Schema

If the learner can mark the scope and the condition of the non-sequential statements correctly, it means s/he can understand the execution flow of the program. Otherwise, the missing or wrong tags are useful for the teacher to know the learner's misconception and then give him/her appropriate help.

Example 1: Using Structure Tags to Represent Nested Program Structure

As shown in Figure 5, the program, designed to find out the minimal number and the count of this minimal number in a given integer array *"5, 3, 3"*, contains a nested structure of a *for-loop* statement and two *if* statements, so learners need to identify each statement's scope with the loop or branch conditions. The scope of the most outside *for-loop* statement is marked from tag "*loop start #1*" to tag "*loop end #1*" and the loop condition is marked with the light-orange background color to annotate the constraint of the *for-loop* statement. The internal nested *if* statements can be marked by the tags "*condition start*" and "*condition end*" with different tag numbers to denote scopes of different statements. Thus, learners have to mark the scope of the outer *if* statement from tag "*condition start #1*" to tag "*condition end #1*", and the scope of the inner *if* statement from tag "*condition start #2*" to tag "*condition end #2*" to clearly denote the nested program structures. Similar to *loop* statement, learners need to use *"light pink background color"* to identify the constraint of the conditional statement.



```
int main() {
  int[] ivec = {5, 3, 3};

  int minVal = 0;
  int occurs = 0;

  for ( int ix = 0; ix < size; ++ix ) {
    if (minVal == ivec[ix] ) {
      ++occurs;
    } else {
      if (minVal > ivec[ix] ) {
          minVal = ivec[ix];
          occurs = 1;
      }
    }
  }

  Print minVal;
  Print occurs;
}
```

Figure 5: Using structure tag schema to mark the program's non-sequential statements

4.1.2. Test Case Tag Schema

When learners evaluate the correctness of others' program, it is important to use critical test cases and monitor the values of variable to find out the mistakes. In order to help learners trace the test case execution, we want the learner to record the variation of variables. Thus, we define test case tag schema, which support the learner to record certain (critical) variation of certain (critical) variables.

The *"test case"* tag set, shown in Figure 6, is composed of two kinds of tags: *"Watch point"* tag is used to indicate the program line, where the reviewer wants to know the variables' values, and *"variables set"* tag is used to record the value of variables at the line of *"watch point"*. If the *"watch point"* is in a loop block, then a variable may have different values at different iterations. So the *"variables set"* tag is a two dimensional table, where the first dimension is the watched variables, and the second one is the watched iterations.

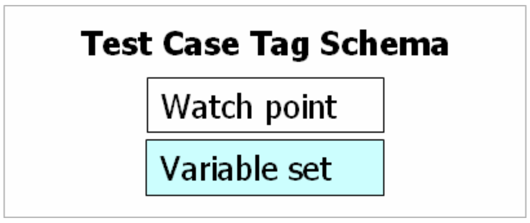
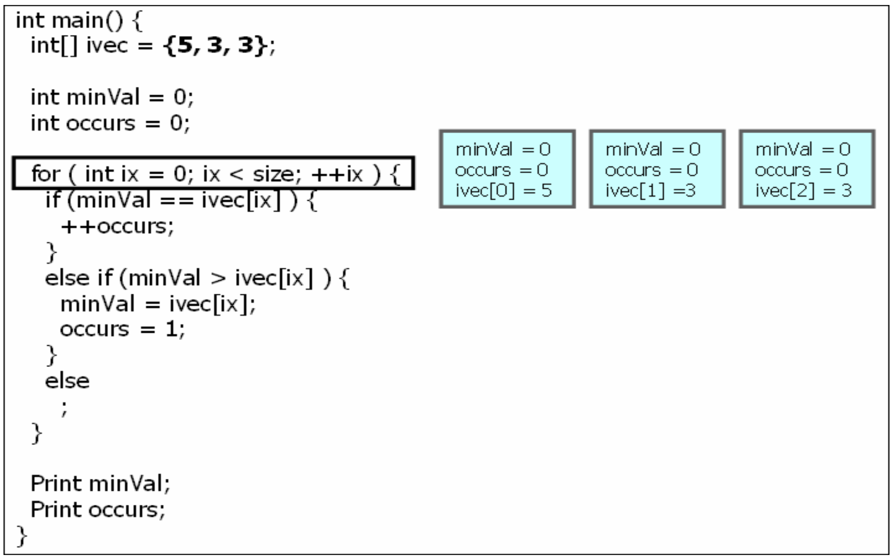


Figure 6: Test Case Tag Schema

With the test case tags, we can evaluate whether the learners can correctly identify the critical variation and trace the values or not. If the learner chooses the irrelevant variables or can not record the values correctly, teachers can easily capture the learners' misconceptions about recognizing the program's key points or tracing the values during the execution of the program.

Example 2: Using test case tags to record the execution status of program

For representing the execution status of a program, the watch points should be assigned by the novice reviewer to point out the critical position of the program. As shown in Figure 7, the for-loop statement is assigned with a watch point and three critical variables, *“minVal”*, *“occurs”*, and *“ivec[ix]”*, are defined as the watched variable set, where the variable *“minVal”* is used to record the current minimal number of the array, the variable *“occurs”* is used to record the current count of the minimal number in execution, and the variable *“ivec[ix]”* is used to represent the element in the array. Because the watch point is assigned in the loop structure, the value of each variable may be changed in different iterations. Thus, three variable set tags are used to record the values of variables in the three iterations, where *“minVal”*, *“occurs”* are assigned 0 in all iterations, and *”ivec[ix]”* is changed to *“5, 3, 3”*.



```
int main() {
  int[] ivec = {5, 3, 3};

  int minVal = 0;
  int occurs = 0;

  for ( int ix = 0; ix < size; ++ix ) {
    if (minVal == ivec[ix] ) {
      ++occurs;
    }
    else if (minVal > ivec[ix] ) {
      minVal = ivec[ix];
      occurs = 1;
    }
    else
      ;
  }

  Print minVal;
  Print occurs;
}
```

Figure 7: Using test case tags to record the execution status of program

4.1.3. Mistake Tag Schema

After tracing the programs with test cases, reviewers may find some mistakes in the programs. In order to guide reviewers to give more precise comments, we propose a mistake tag schema to hierarchically structure the mistake tags from general to specific, as shown in Figure 8. With the schema, reviewers are asked to find the correct and precise mistake tag, so reviewers’ debugging ability could be evaluated.

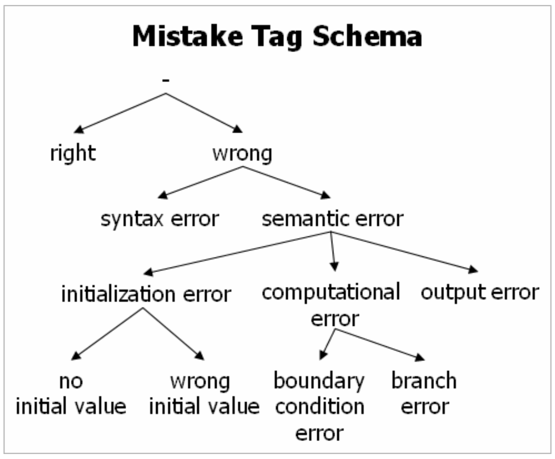
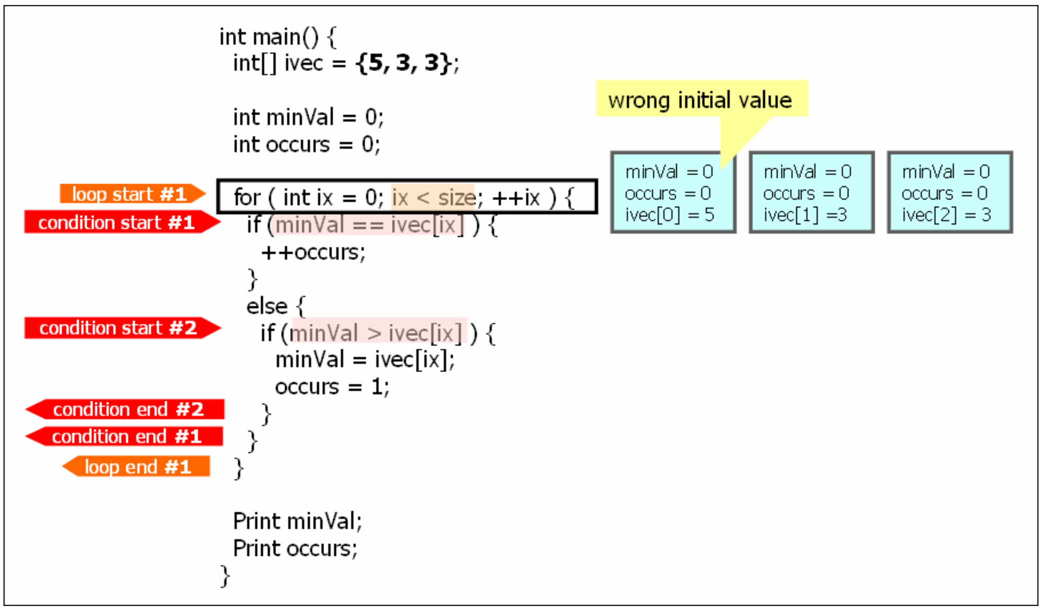


Figure 8: Mistake Tag Schema

With the mistake tag schema, reviewers can tag the test case tag with program mistakes to specify the program mistake and the faced situation. If the reviewer can tag the correct and precise comments, it means that the reviewer has good debugging ability about finding program mistakes. If the reviewer can not tag the correct or precise comments, then her/his programming misconception in debugging will be found.

Example 3: Using the mistake tags to denote the program mistake of initial values

Tracing the program and record the variation of the concerned variables may help to find out some mistakes of the program according to the tracing records. For example, when the learner records the values in the first iteration, s/he can find that *"minVal"* is not changed until the negative number appears in *"ivec[ix]"* because the initial value of *"minVal"* is 0. Thus, as shown in Figure 9, a mistake tag *"wrong initial value"* can be used to denote the found mistake in the first iteration of execution.

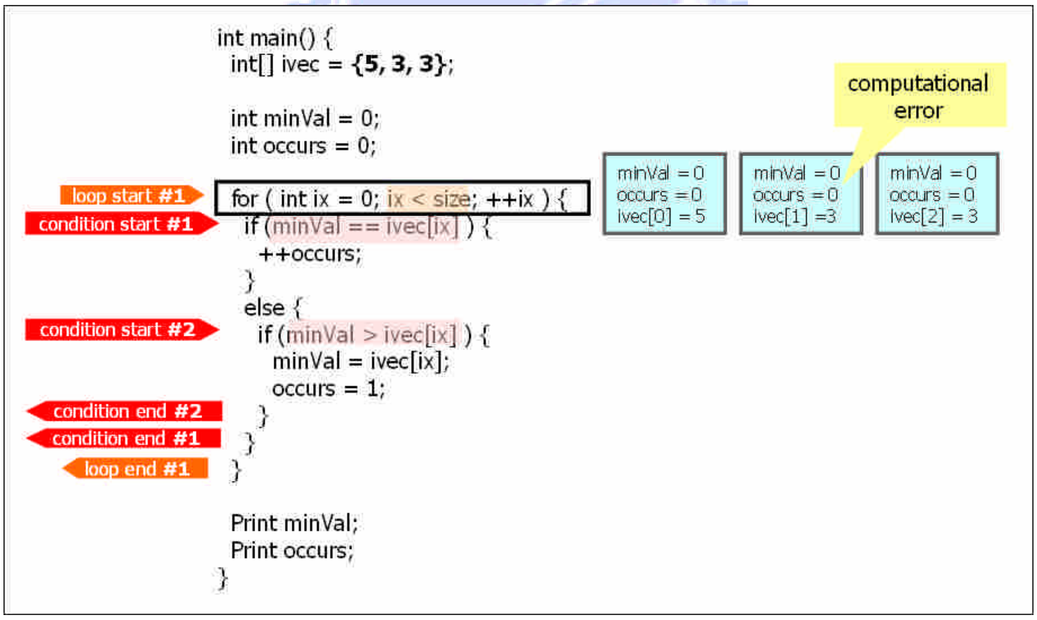


```
int main() {
  int[] ivec = {5, 3, 3};

  int minVal = 0;
  int occurs = 0;

  for ( int ix = 0; ix < size; ++ix ) {
    if (minVal == ivec[ix] ) {
      ++occurs;
    }
    else {
      if (minVal > ivec[ix] ) {
        minVal = ivec[ix];
        occurs = 1;
      }
    }
  }

  Print minVal;
  Print occurs;
}
```

Figure 9: Using the mistake tags to denote the program mistake of initial values

Example 4: Using mistake tag to indicate the mistake on the test case tags

If the learner can not identify the mistake of initial value, s/he may find that the truly minimal number of the array is 3, stored in *"ivec[1]"*, but the *"minVal"* is not equal to 3 after the second iteration. Thus, as shown in Figure 10, s/he can assign the more general mistake tag *"computational error"* in the variable sets in third iteration.



```
int main() {
  int[] ivec = {5, 3, 3};

  int minVal = 0;
  int occurs = 0;

  for ( int ix = 0; ix < size; ++ix ) {
    if (minVal == ivec[ix] ) {
      ++occurs;
    }
    else {
      if (minVal > ivec[ix] ) {
        minVal = ivec[ix];
        occurs = 1;
      }
    }
  }

  Print minVal;
  Print occurs;
}
```

Figure 10: Using mistake tag to indicate the mistake on the test case tags

4.2. The Programmer Tag Schema

The programmers, who receive the comments from the reviewers, have to return their feedback about the comments to the reviewers, and thus the reviewers' tracing ability can be evaluated.

4.2.1. The Feedback Tag Schema

After receiving the reviewers' comments, the programmers can give feedbacks to express whether the comments can be accepted or not. As shown in Figure 11, the feedback tag schema includes two tags, *"reject"* and *"accept"*.

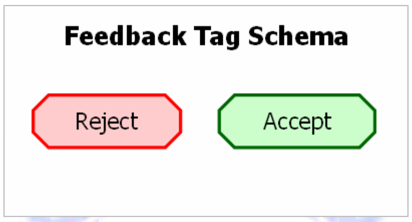
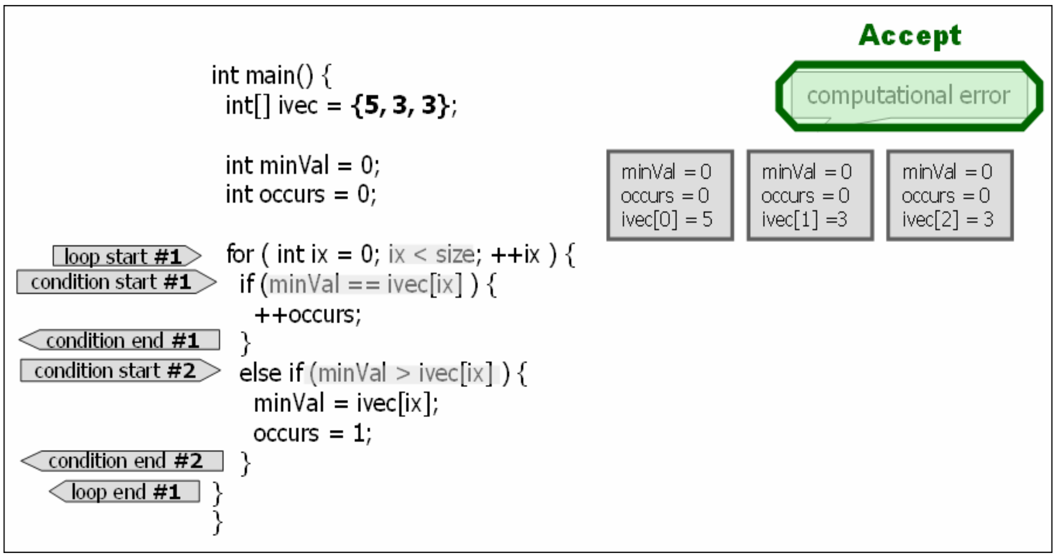


Figure 11: Feedback Tag Schema

Example 5: Using feedback tag to represent the rejection or acceptation of programmers

As shown in Figure 12, the programmer receives the comments from reviewers and agrees the comment of *"computational error"*, so s/he can refine the program and return the feedback comments with tag *"accept"* in the mistake tag to the reviewer.



```
int main() {
  int[] ivec = {5, 3, 3};

  int minVal = 0;
  int occurs = 0;

  for ( int ix = 0; ix < size; ++ix ) {
    if (minVal == ivec[ix] ) {
      ++occurs;
    }
    else if (minVal > ivec[ix] ) {
      minVal = ivec[ix];
      occurs = 1;
    }
  }
}
```

```
minVal = 0     minVal = 0     minVal = 0
occurs = 0     occurs = 0     occurs = 0
ivec[0] = 5    ivec[1] =3     ivec[2] = 3
```

Figure 12: The example of accepting the reviewer’s comments.

Chapter 5. Experiment

In order to evaluate the effectiveness of our computer program peer assessment, an experiment was performed. The target learners were around 19~20 years old sophomore undergraduate students in Department of Computer Science and Information Engineering, Minghsin University of Science and Technology, who have learned the basic VB language, including data type, expression, sequence statement, condition statement, and repetition statement, for one year. Each three learners, with similar programming ability, were assigned to a group, and all learners were asked to implement the programming assignment *"Determining the Winning Tiles of the Sixteen Tiles Mahjong"* [19]. This program assignment, designed to determine the win of the given set of seventeen tiles, is appropriate to evaluate novice programmers programming skill, because learners have to use a simple problem solving strategy with basic repetition, conditional statements in the determination function of the program. Moreover, the Sixteen Tiles Mahjong game is easy to motivate learners to accomplish the assignment.

5.1. Programming Assignment

In order to let learners easily check the execution results of their programs, a pre-constructed GUI, as shown in Figure 13, was provided in the assignment, and left the determination function, the core of the program, for novice programmers to design and implement.

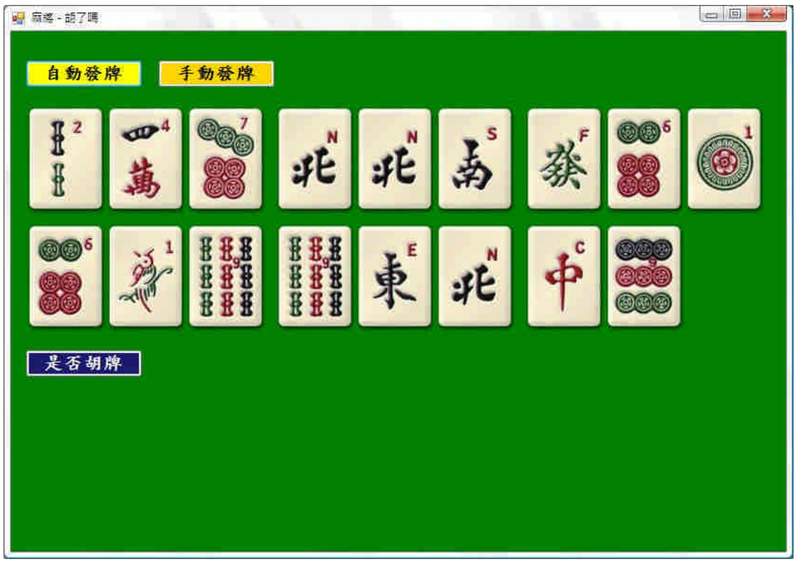
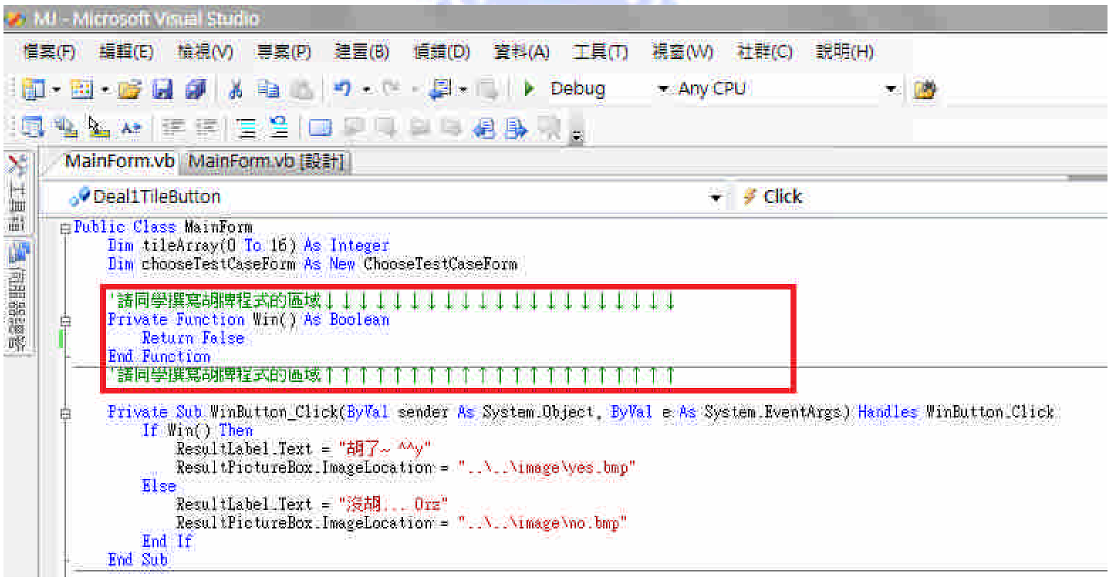


Figure 13: The GUI of the program assignment

As shown in Figure 14, the determination function, named *"Win()"*, is used to determine the given set of seventeen tiles, defined in *"tileArray"*, and output a Boolean value to indicate whether the set of seventeen tiles can win the game or not.



```
Public Class MainForm
    Dim tileArray(0 To 16) As Integer
    Dim chooseTestCaseForm As New ChooseTestCaseForm

    '請同學撰寫胡牌程式的區域↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
    Private Function Win() As Boolean
        Return False
    End Function
    '請同學撰寫胡牌程式的區域↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

    Private Sub WinButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles WinButton.Click
        If Win() Then
            ResultLabel.Text = "胡了~ ^^y"
            ResultPictureBox.ImageLocation = "..\..\image\yes.bmp"
        Else
            ResultLabel.Text = "沒胡... Orz"
            ResultPictureBox.ImageLocation = "..\..\image\no.bmp"
        End If
    End Sub
```

Figure 14: The coding area for the novice programmers

If the output of the determination function is *"true"*, the image of success, as shown in Figure 15, can be displayed on the GUI, or if the result is *"false"*, the image of failure, as shown in Figure 16, can be displayed. Thus, programmers can be easy to check the given set of seventeen tiles and the result of their program.

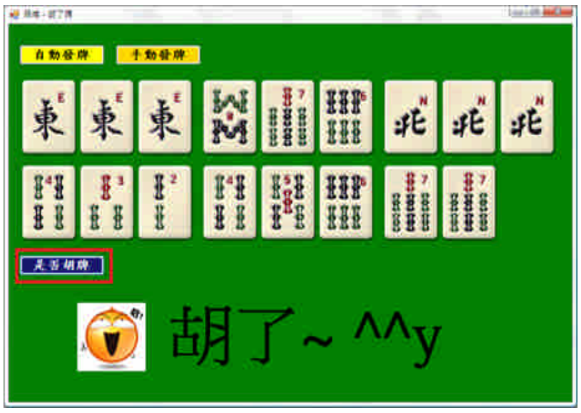


Figure 15: The image of success on the GUI

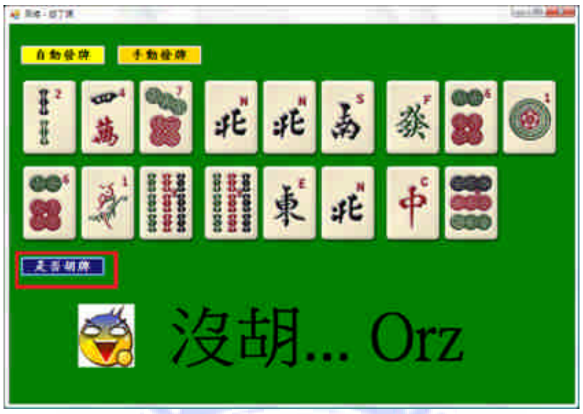


Figure 16: The image of failure on the GUI

5.2. Computer Program Peer Assessment System

5.1.1. The Web System for Exchanging Document among Peers

In order to facilitate the exchange of programs and comments, a Computer Program Peer Assessment System consisting of design and review region, as shown in Figure 17, was constructed, where a novice learner has to play the role of a programmer and a reviewer to upload an program code and reviewing reports. Therefore, after the learner logins to the system, s/he will see the web page. When the learner plays the role of the programmer, s/he uses the design region for uploading his/her program code, downloading the comments from the peer reviewers, and uploading his/her feedback of the reviewer's comment. When the learner plays the roles of the reviewer, s/he uses the review region for downloading the programmer's program, uploading his/her comments to the programmer and downloading the programmer’s feedback of his/her comments.

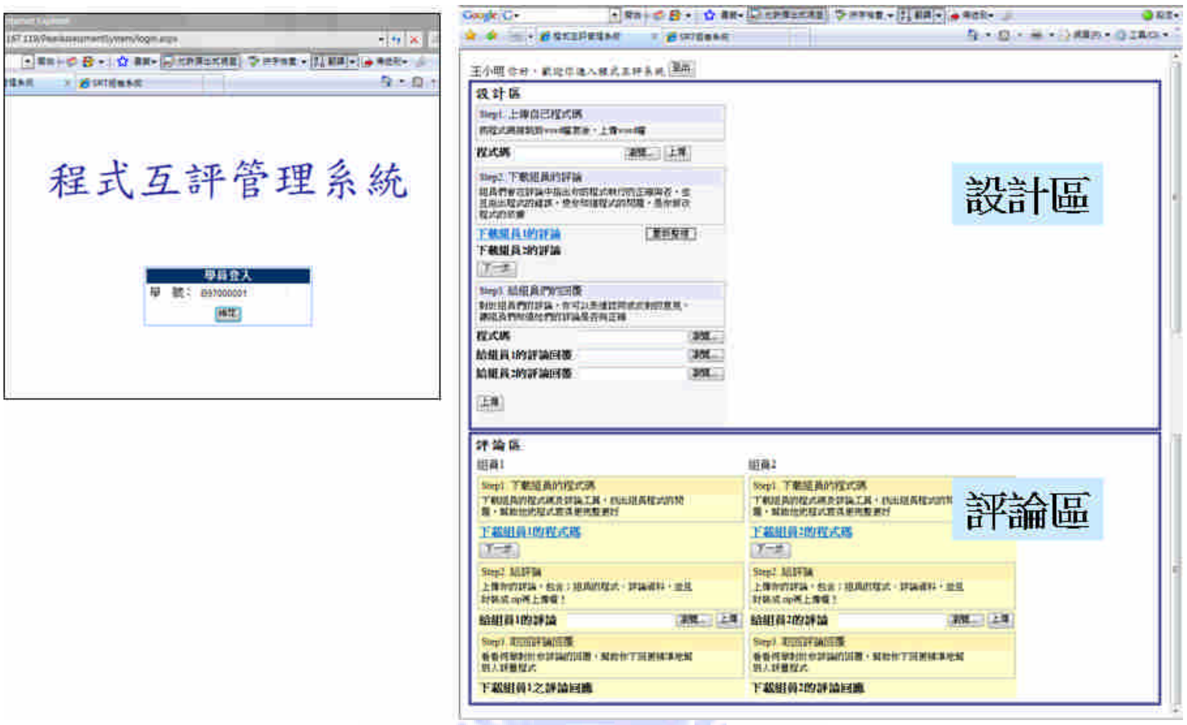
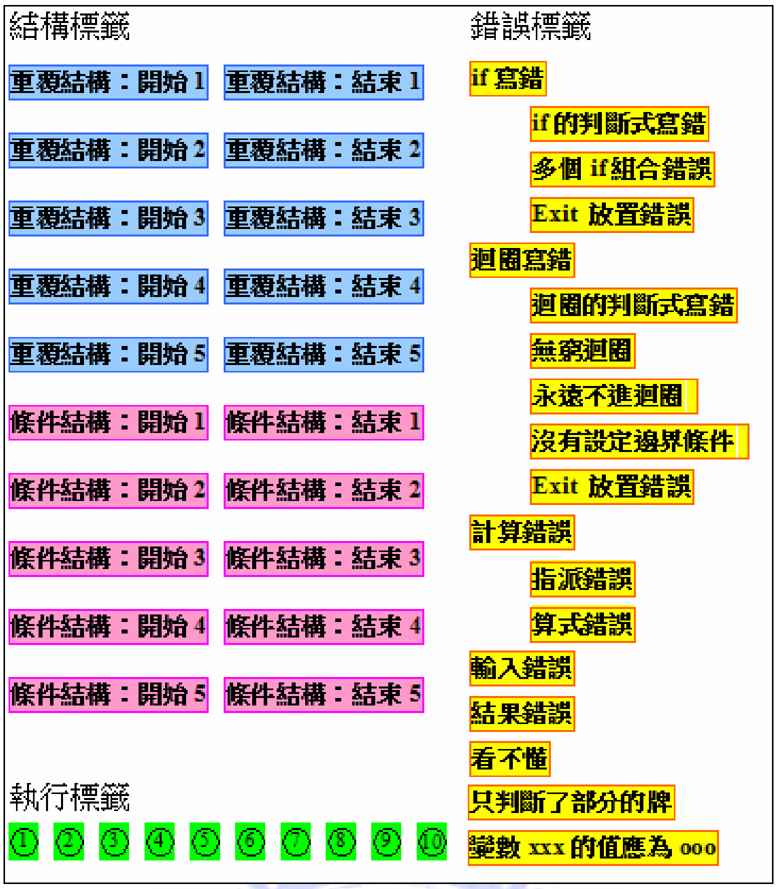


Figure 17: Computer Program Peer Assessment System

5.1.2. The Microsoft Word for Making Tag-based Comments

To ease the tagging process, the tagging tasks could be done using in Microsoft Word. We predefined review tags, as shown in Figure 18, the execution table, as shown in Figure 19, and the feedback tag, as shown in Figure 20, in the Microsoft Word document for reviewers and programmer. When making comments or feedback, the learner copies the peer’s program code in the Microsoft Word document and then copies and pastes the tag comment on the Word document of peer’s program code. They also used Microsoft Word document to record the variation of the program variables. The tagged programs are shown in Figure 21.



結構標籤

重覆結構：開始 1 重覆結構：結束 1
重覆結構：開始 2 重覆結構：結束 2
重覆結構：開始 3 重覆結構：結束 3
重覆結構：開始 4 重覆結構：結束 4
重覆結構：開始 5 重覆結構：結束 5
條件結構：開始 1 條件結構：結束 1
條件結構：開始 2 條件結構：結束 2
條件結構：開始 3 條件結構：結束 3
條件結構：開始 4 條件結構：結束 4
條件結構：開始 5 條件結構：結束 5

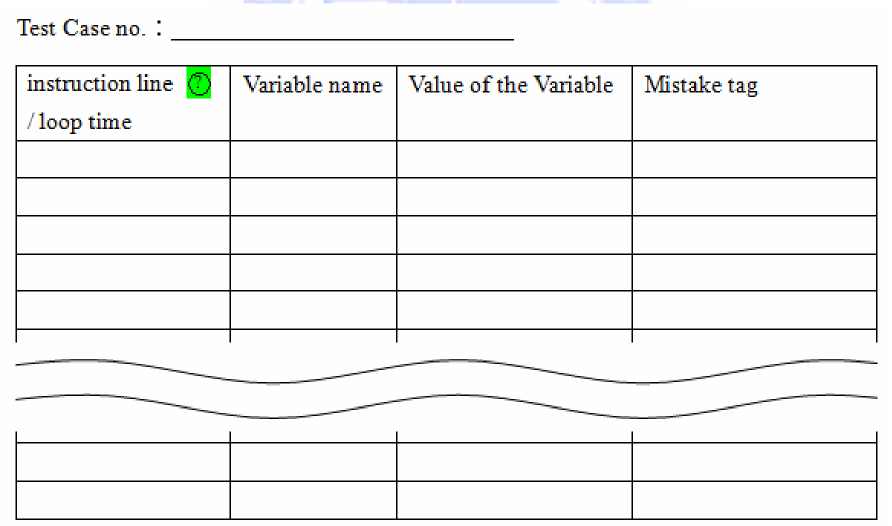
執行標籤

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

錯誤標籤

- if 寫錯
  - if 的判斷式寫錯
  - 多個 if 組合錯誤
  - Exit 放置錯誤
- 迴圈寫錯
  - 迴圈的判斷式寫錯
  - 無窮迴圈
  - 永遠不進迴圈
  - 沒有設定邊界條件
  - Exit 放置錯誤
- 計算錯誤
  - 指派錯誤
  - 算式錯誤
- 輸入錯誤
- 結果錯誤
- 看不懂
- 只判斷了部分的牌
- 變數 xxx 的值應為 ooo

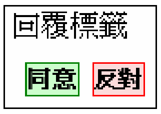
Figure 18: Review tags predefined in Microsoft Word document.



Test Case no.：________________________

instruction line ⑦ / loop time	Variable name	Value of the Variable	Mistake tag

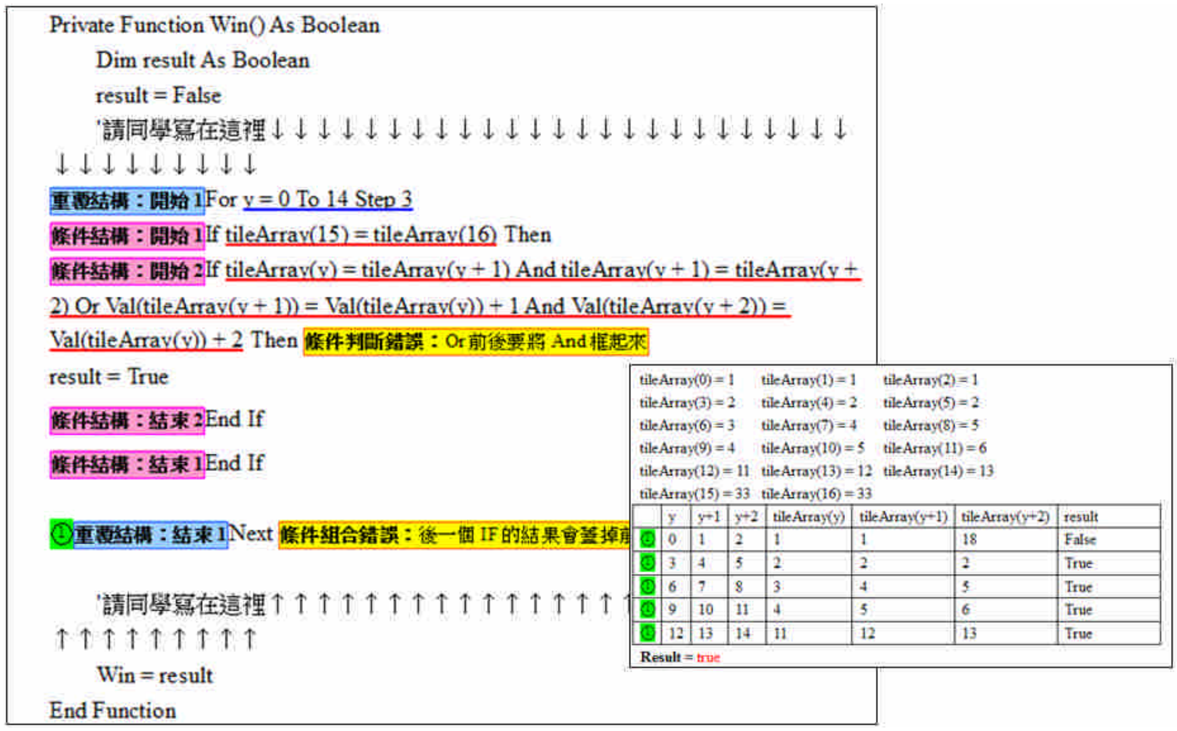
Figure 19: The predefined execution table in Microsoft Word document



回覆標籤

同意 反對

Figure 20: Feedback tags predefined in Microsoft Word document



```
Private Function Win() As Boolean
    Dim result As Boolean
    result = False
    '請同學寫在這裡↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
↓↓↓↓↓↓↓↓↓
重覆結構：開始 1For y = 0 To 14 Step 3
條件結構：開始 1If tileArray(15) = tileArray(16) Then
條件結構：開始 2If tileArray(y) = tileArray(y + 1) And tileArray(y + 1) = tileArray(y +
2) Or Val(tileArray(y + 1)) = Val(tileArray(y)) + 1 And Val(tileArray(y + 2)) =
Val(tileArray(y)) + 2 Then 條件判斷錯誤：Or 前後要將 And 框起來
result = True
條件結構：結束 2End If
條件結構：結束 1End If

①重覆結構：結束 1Next 條件組合錯誤：後一個 IF 的結果會蓋掉前

    '請同學寫在這裡↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
↑↑↑↑↑↑↑↑↑
    Win = result
End Function
```

```
tileArray(0) = 1    tileArray(1) = 1    tileArray(2) = 1
tileArray(3) = 2    tileArray(4) = 2    tileArray(5) = 2
tileArray(6) = 3    tileArray(7) = 4    tileArray(8) = 5
tileArray(9) = 4    tileArray(10) = 5   tileArray(11) = 6
tileArray(12) = 11  tileArray(13) = 12  tileArray(14) = 13
tileArray(15) = 33  tileArray(16) = 33
```

	y	y+1	y+2	tileArray(y)	tileArray(y+1)	tileArray(y+2)	result
①	0	1	2	1	1	18	False
①	3	4	5	2	2	2	True
①	6	7	8	3	4	5	True
①	9	10	11	4	5	6	True
①	12	13	14	11	12	13	True

Result = true

Figure 21: The tagged program

Chapter 6. The Misconception Pattern Analysis

According to the tag schema we defined, we can use them to evaluate the novice learners' programming misconception. In this chapter, we provide the misconception pattern analysis methods of the four kinds of the tags, and then the misconception of the novice programming learner will be found out.

By analyzing the distribution of the tag-based comments, the structure tags, the test case tags, and the mistake tags could be helpful for evaluating the learner's different abilities of programming skills, the tracing ability, the testing ability, the debugging ability, respectively.

Analyzing the structure or sequence of the learner's structure tags, we can find the learner's misconception in various structure of the program.

According to the watch point marked by the novice reviewers, we can know what part of the program is the major concern. To observe what the variables are chosen, we can know whether the novice reviewer got the major meaning of the program or not. Furthermore, we check the correctness of the variation of the variables to understand the students' thought of the execution of the program.

The mistake tag schema is designed to evaluate whether the novice reviewer's comment of program mistakes are specific enough or not. So, we can map the novice reviewer's mistake tag with the hierarchical mistake tag schema to know whether the mistake tag is specific enough.

6.1. The distribution of four kind of tag-based comment

As shown in Figure 22, seventy-one percent of the novice reviewers can make the structure tags, so we know that most of the novice reviewers can understand the peer's structure of the program. Forty-nine percent of them can make the mistake tags, so half of them can find the mistake in the peer's program. But, only eleven percent of them can make the test case tags means that most of the novice reviewers feel difficult to trace the execution of peer's program. Finally, because the duration of the feedback process in this experiment is not long enough, only eleven percent of the novice programmers make the feedback of novice reviewer's comments.

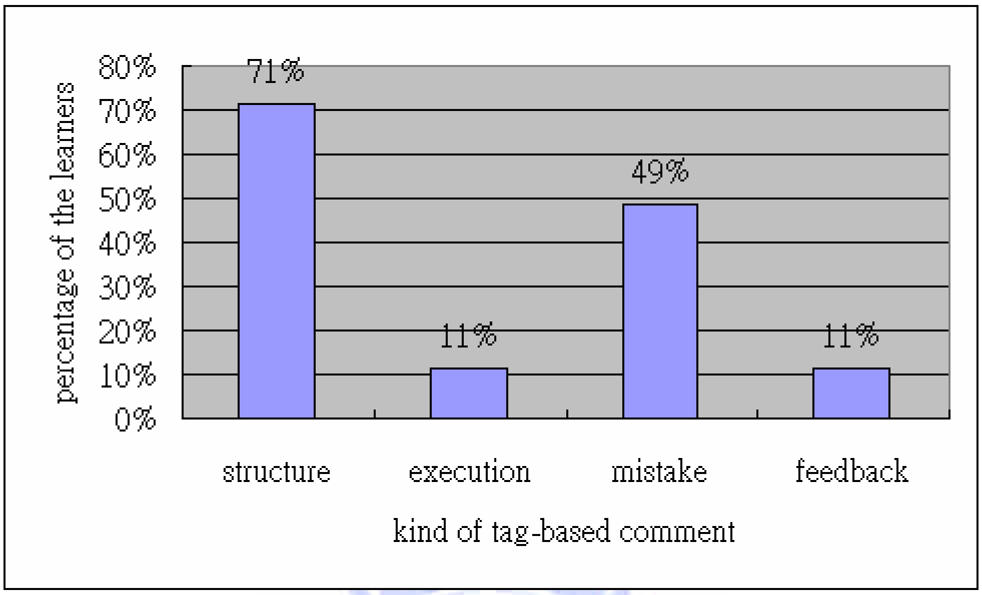


Figure 22: The distribution of the novice programming learner's tag-based comment

6.2. The evaluation of the learners' tracing ability

According to the structure tags tagged by the novice reviewers, we can evaluate their understanding about the structure of the program. As shown in Figure 23, twelve percent of the novice reviewers made the wrong structure tags. One third of the novice reviewers, who made the wrong structure tags, can not recognize the *"if statement"* is a conditional statement because s/he used the *"repetition statement"* tag to mark the scope of the *"if statement"*. Two thirds of the novice reviewers, who made the wrong structure tags, have the misconception of the nested structure because the scopes, defined by their structure tags, are not properly nested.

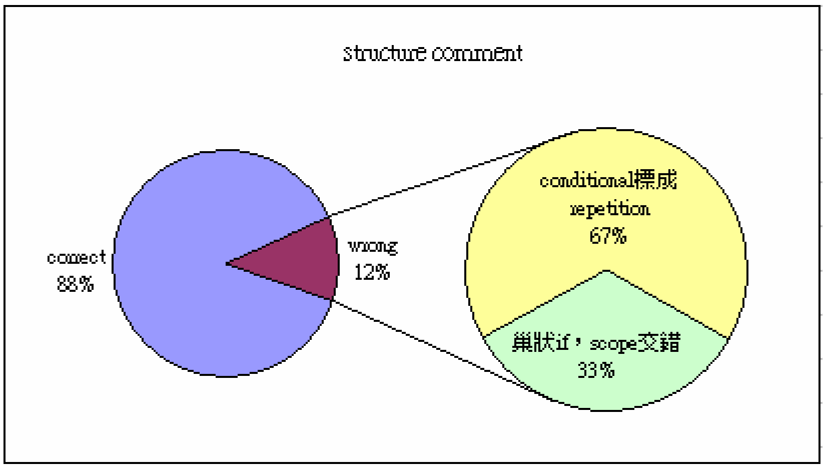
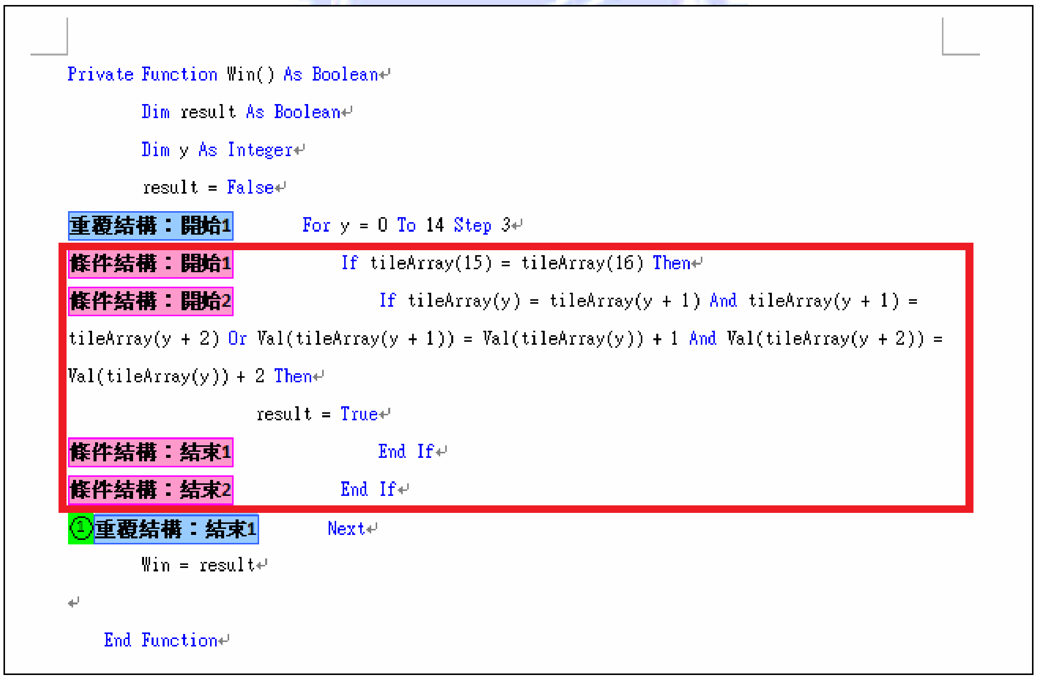


Figure 23: The distribution of correctness of the novice reviewers' structure tag comments

Example 6: The misconception in nested structure of the program

We can analyze the order of the structure tags tagged by the novice reviewer to find out the learner's programming misconception in the program structure. As shown in Figure 24, the learner tagged the wrong structure tag, because the tag pair of *"condition structure 2"* should be closed inside the tag pair *"condition structure 1"*, and then we can find this learner may have the misconception of the nested structure of the program.



```
Private Function Win() As Boolean
        Dim result As Boolean
        Dim y As Integer
        result = False
重覆結構：開始1        For y = 0 To 14 Step 3
條件結構：開始1            If tileArray(15) = tileArray(16) Then
條件結構：開始2                If tileArray(y) = tileArray(y + 1) And tileArray(y + 1) =
tileArray(y + 2) Or Val(tileArray(y + 1)) = Val(tileArray(y)) + 1 And Val(tileArray(y + 2)) =
Val(tileArray(y)) + 2 Then
                    result = True
條件結構：結束1                End If
條件結構：結束2            End If
①重覆結構：結束1        Next
        Win = result

    End Function
```

Figure 24: An example of the wrong structure tag comments made on tagging the nested conditional statements.

6.3. The evaluation of the learners' testing ability

Most of the novice reviewers can not make the test case tags of peer's program, shown as Figure 22, but three fourths of the novice reviewers, who can make the test case tag comments, can make the correct test case tags of comment, as shown in Figure 25.

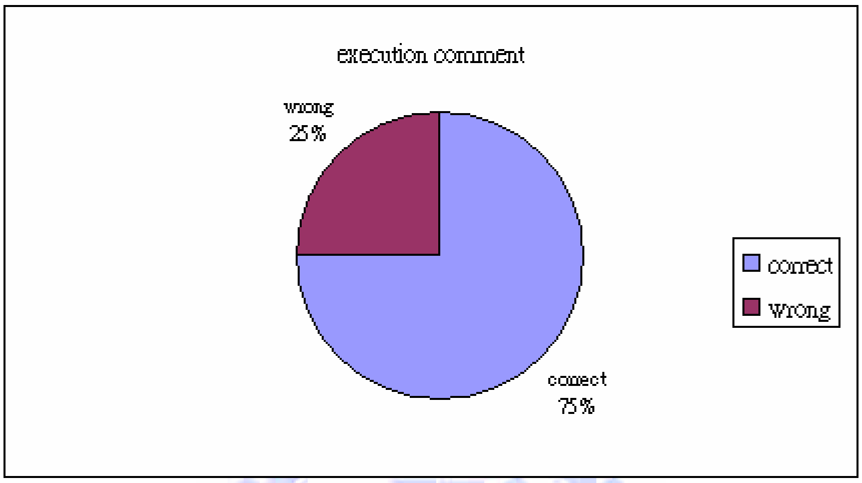
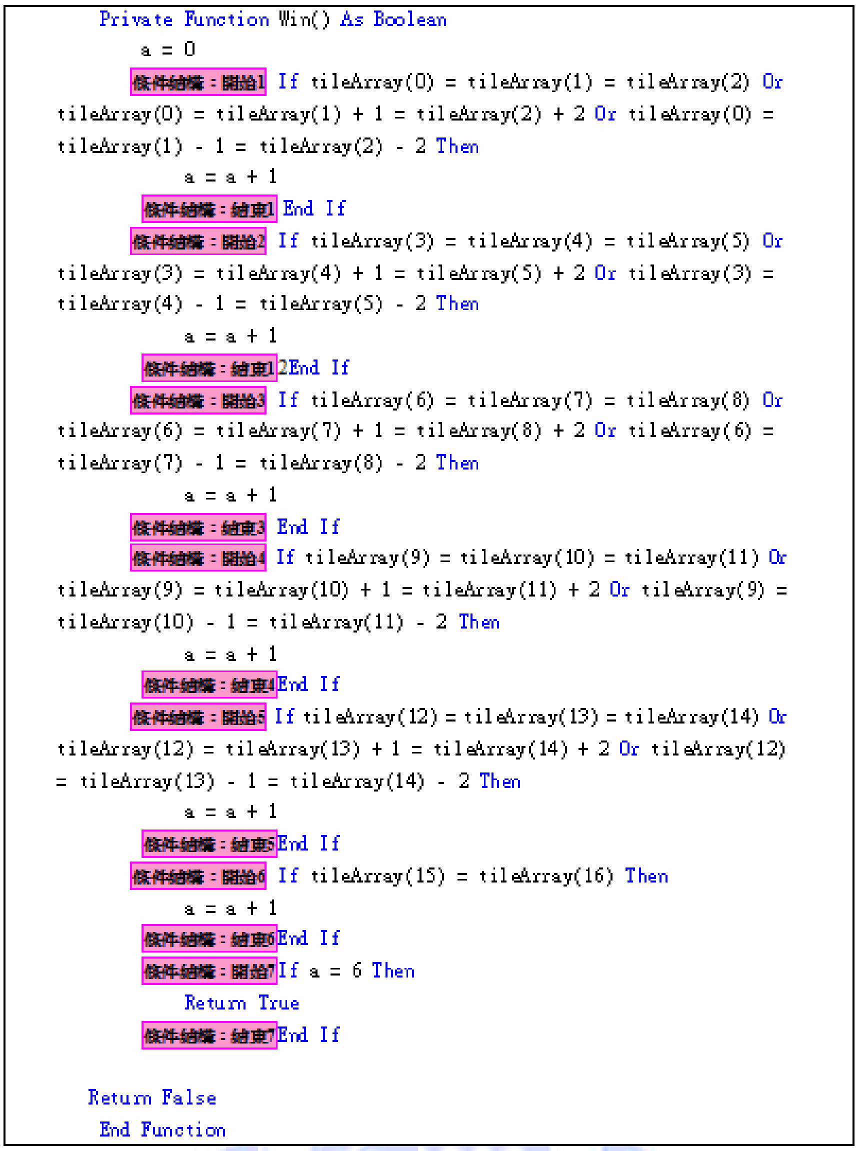


Figure 25: The distribution of correctness of the novice reviewers' test case tag comments

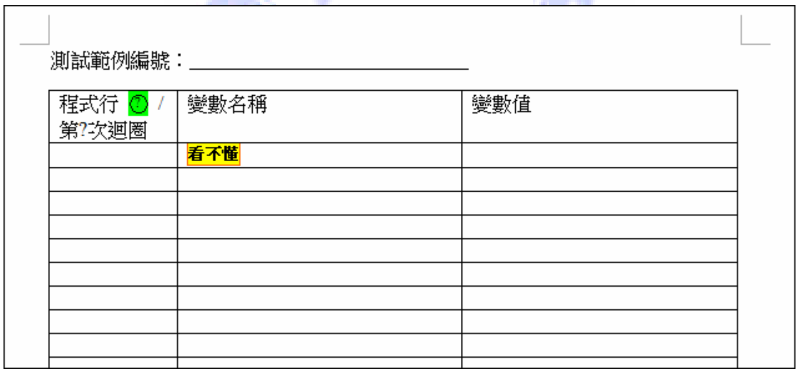
Example 7: The misconception in testing a long program

We can analyze the variables what the novice reviewer chose and the correctness of the variation of variable values recorded by the learner to find out the learner's programming misconception in the program execution. As shown in Figure 26 and Figure 27, the learner can tag the correct structure tags on the program, but s/he can not decide what variables were needed to be observed and then used the *"unknown(看不懂)"* tag to express him/her circumstance.



```
        Private Function Win() As Boolean
            a = 0
           條件結構：開始1 If tileArray(0) = tileArray(1) = tileArray(2) Or
    tileArray(0) = tileArray(1) + 1 = tileArray(2) + 2 Or tileArray(0) =
    tileArray(1) - 1 = tileArray(2) - 2 Then
                a = a + 1
            條件結構：結束1 End If
           條件結構：開始2 If tileArray(3) = tileArray(4) = tileArray(5) Or
    tileArray(3) = tileArray(4) + 1 = tileArray(5) + 2 Or tileArray(3) =
    tileArray(4) - 1 = tileArray(5) - 2 Then
                a = a + 1
            條件結構：結束12End If
           條件結構：開始3 If tileArray(6) = tileArray(7) = tileArray(8) Or
    tileArray(6) = tileArray(7) + 1 = tileArray(8) + 2 Or tileArray(6) =
    tileArray(7) - 1 = tileArray(8) - 2 Then
                a = a + 1
           條件結構：結束3 End If
           條件結構：開始4 If tileArray(9) = tileArray(10) = tileArray(11) Or
    tileArray(9) = tileArray(10) + 1 = tileArray(11) + 2 Or tileArray(9) =
    tileArray(10) - 1 = tileArray(11) - 2 Then
                a = a + 1
            條件結構：結束4End If
           條件結構：開始5 If tileArray(12) = tileArray(13) = tileArray(14) Or
    tileArray(12) = tileArray(13) + 1 = tileArray(14) + 2 Or tileArray(12)
    = tileArray(13) - 1 = tileArray(14) - 2 Then
                a = a + 1
            條件結構：結束5End If
           條件結構：開始6 If tileArray(15) = tileArray(16) Then
                a = a + 1
            條件結構：結束6End If
            條件結構：開始7If a = 6 Then
                Return True
            條件結構：結束7End If

       Return False
        End Function
```

Figure 26: An example of tagged program made by the novice reviewer



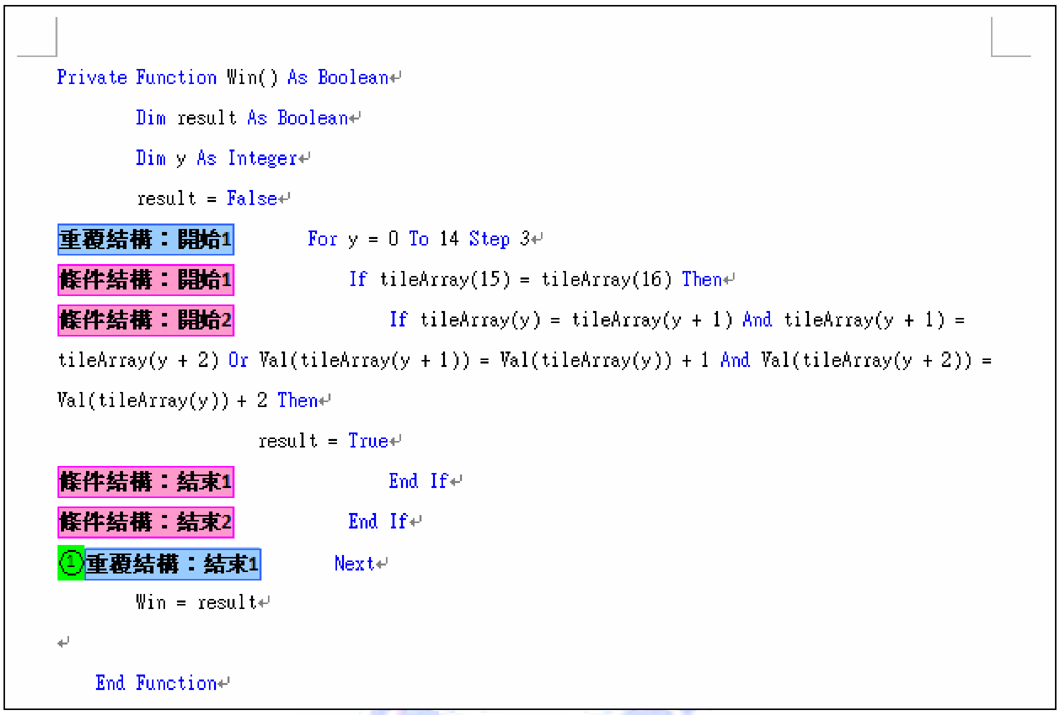
測試範例編號：______________________

程式行 ? / 第?次迴圈	變數名稱	變數值
	看不懂	

Figure 27: An example of the execution table made by the novice reviewer

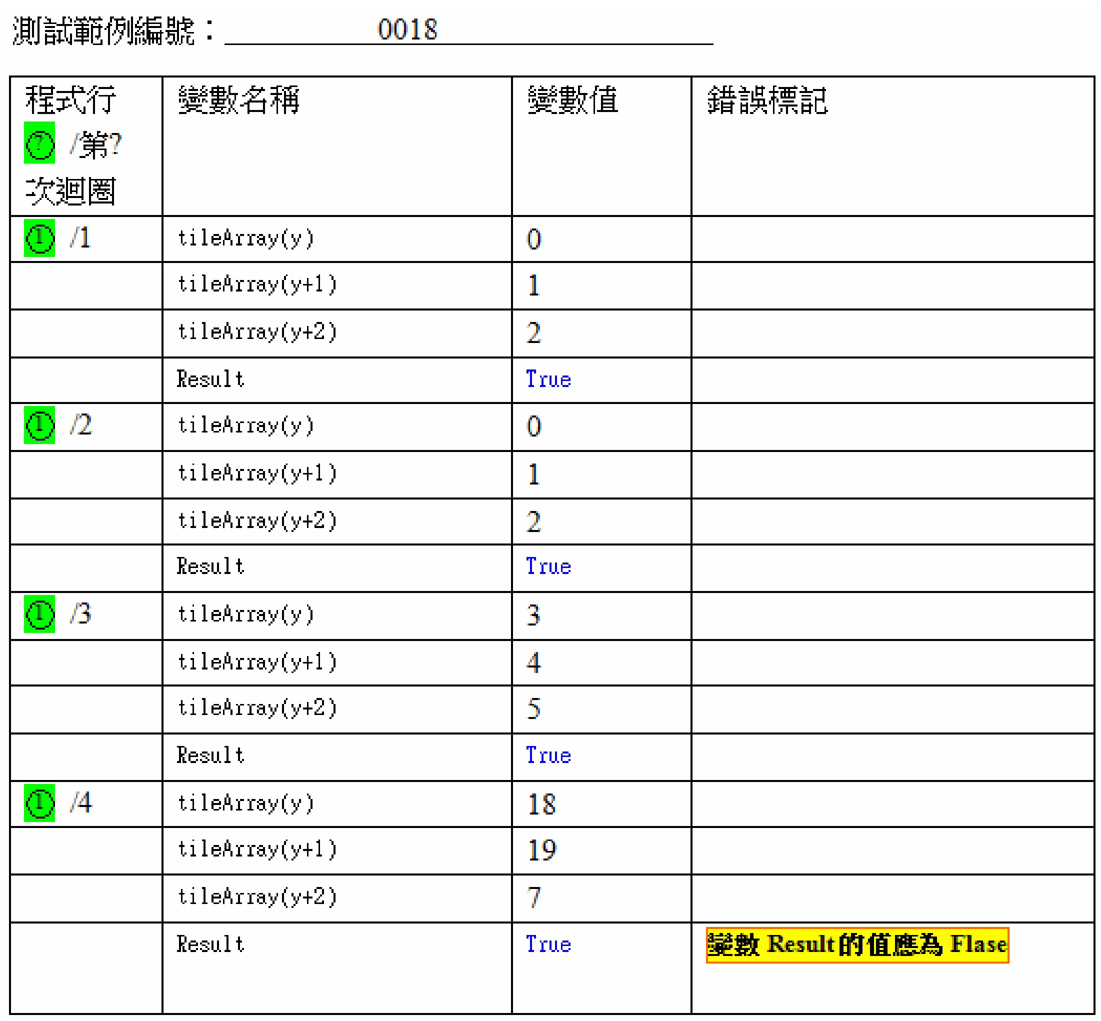
Example 8: The misconception of a novice reviewer in testing a program

The novice reviewer successfully made the comments of the execution of the peer's program. As shown in Figure 28, the novice reviewer used the green number tag, which is the watch point tag, to annotate the program line, which s/he will observe the variation of the variables. And then, as shown in Figure 29, the novice reviewer recorded the variation of the variables. S/he chose the four variables *"tileArray(y)"*, *"tileArray(y+1)"*, *"tileArray(y+2)"*, and *"Result"* and recorded four rounds of the variation. At the fourth round, s/he computed that the *"Result"* is true and the expected *"Result"* is false. Therefore, s/he made the mistake tag *"變數Result的值應爲False"* on the unexpected row. Actually, the value of the *"Result"* computed this program is false, so we found that the novice reviewer has the misconception of testing the program.



```
Private Function Win() As Boolean
        Dim result As Boolean
        Dim y As Integer
        result = False
重覆結構：開始1        For y = 0 To 14 Step 3
條件結構：開始1            If tileArray(15) = tileArray(16) Then
條件結構：開始2                If tileArray(y) = tileArray(y + 1) And tileArray(y + 1) =
tileArray(y + 2) Or Val(tileArray(y + 1)) = Val(tileArray(y)) + 1 And Val(tileArray(y + 2)) =
Val(tileArray(y)) + 2 Then
                    result = True
條件結構：結束1                End If
條件結構：結束2            End If
①重覆結構：結束1        Next
        Win = result

    End Function
```

Figure 28: The watch point made by the novice reviewer



測試範例編號：0018

程式行 ? /第?次迴圈	變數名稱	變數值	錯誤標記
① /1	tileArray(y)	0	
	tileArray(y+1)	1	
	tileArray(y+2)	2	
	Result	True	
① /2	tileArray(y)	0	
	tileArray(y+1)	1	
	tileArray(y+2)	2	
	Result	True	
① /3	tileArray(y)	3	
	tileArray(y+1)	4	
	tileArray(y+2)	5	
	Result	True	
① /4	tileArray(y)	18	
	tileArray(y+1)	19	
	tileArray(y+2)	7	
	Result	True	**變數 Result 的值應為 Flase**

Figure 29: An example of the execution table made by the novice reviewer

6.4. The evaluation of the learners' debugging ability

As shown in Figure 30, we conclude the distribution of different mistake tags, the novice reviewers made. The *"unknown(看不懂)"* tag and the *"output error(結果錯誤)"* tag were used frequently, which means that most of the novice reviewers have difficulty in finding out the program mistakes of peers or they just know the high level or more general mistakes of the program, such as *"只判斷部分牌" or "結果錯誤"*.

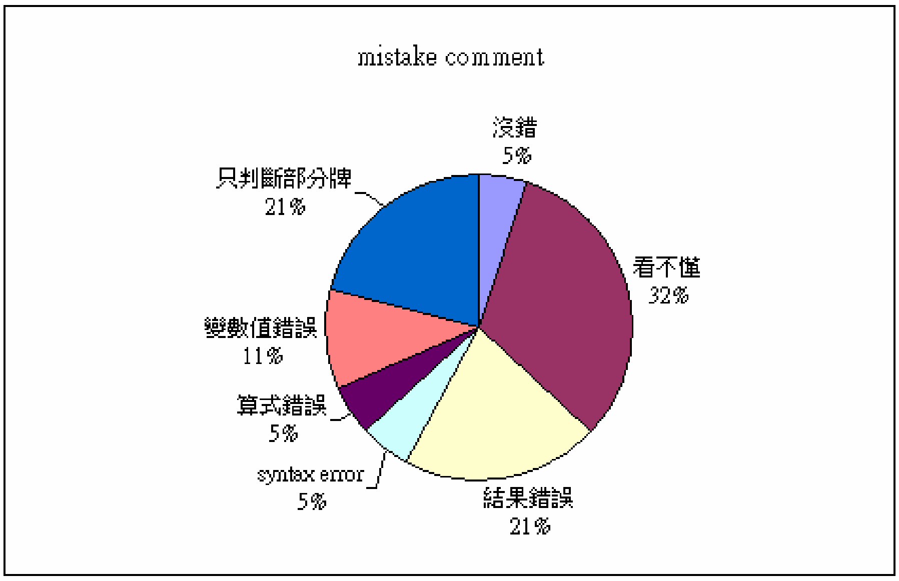
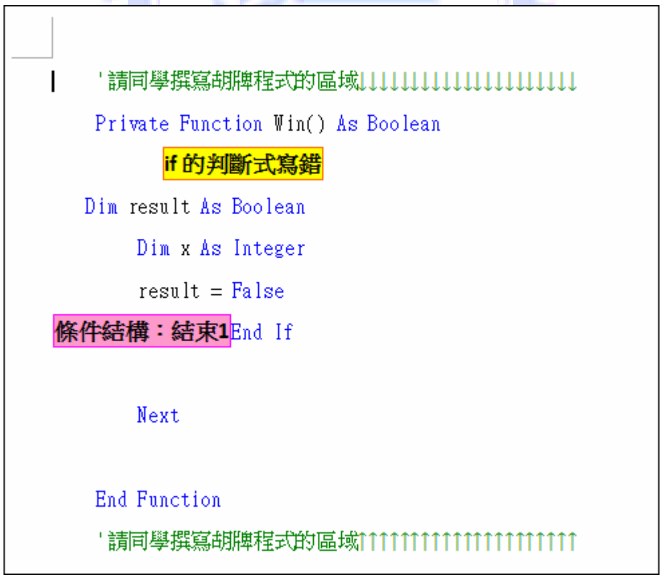


Figure 30: The distribution of different level of mistake tags used by the novice reviewers

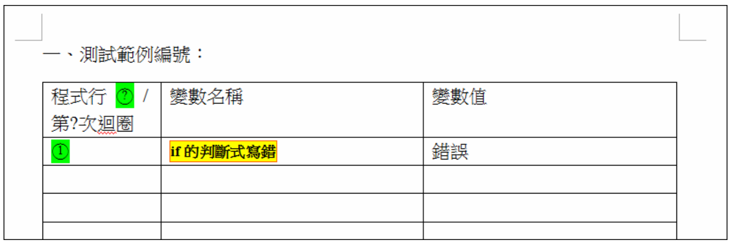
Example 9: A misconception in making a syntax error program

As shown in Figure 31, the novice reviewer used the mistake tag to annotate the peer's program has syntax error. To analyze the structure tags tagged by this novice reviewer, we can know this novice reviewer pointed out the mistake because s/he found the conditional statement does not have the start scope. Because of the syntax error, the peer's program can not be executed, as shown in Figure 32.



```
'請同學撰寫胡牌程式的區域↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
Private Function Win() As Boolean
        if 的判斷式寫錯
Dim result As Boolean
      Dim x As Integer
      result = False
條件結構：結束1End If

      Next

End Function
'請同學撰寫胡牌程式的區域↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑
```

Figure 31: An example of the syntax error program tagged by the mistake tag


一、測試範例編號：

程式行 ? / 第?次迴圈	變數名稱	變數值
①	**if 的判斷式寫錯**	錯誤

Figure 32: An example of the execution table annotated the syntax error of the program

6.5. Discussion

The results of experiment show that the structure tag is useful to evaluate the learners' understanding about program structure, and we find that twelve percent of learners have this kind of misconception. In order to guide learners to correctly trace the program and find the mistakes in the program, designing a critical test case is important, because learners are difficult to find out the mistakes by tracing the test case execution without a good test case. Besides, several learners can not choose the critical variables to observe, so they can not perform well in finding mistakes. Finally, the mistake tag schema for general purpose can only describe a part of program mistakes, so it is necessary to define a mistake tag schema for specific problem domain for reviewers to indicate more specific mistakes.

Chapter 7. Conclusion

In order to make the program assessment more effective to evaluate novices' coding and tracing abilities, we have proposed computer program peer assessment architecture, where the tagging approach is used to improve the communication between programmers and reviewers. In the reviewing activity of peer assessment, the proposed tracing guidance can guide learners to trace program step by step to give the more precise and critical comments. Moreover, tag-based comments are easier to be analyzed automatically to mine the novice's learning status for teachers. In the near future, the peer assessment can be extended to evaluate the more advanced programming skills.

Reference

[1]. Michael McCracken, V.A., Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, Tadeusz Wilusz, *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students.* ACM SIGCSE Bulletin, 2001. **33**(4): p. 125-180.

[2]. Riku Saikkonen, L.M., and Ari Korhonen, *Fully automatic assessment of programming exercises.* ACM SIGCSE Bulletin 2001. **33**(3): p. 133-136.

[3]. Dochy, F., McDowell, L., *Assessments as a tool for learning.* Studies in Educational Evaluation, 1997. **23**(4): p. 279-298.

[4]. Sluijsmans, D., Dochy, F., Moerkerke, G. *Creating a Learning Environment by Using Self-Peer-and Co-Assessment.* 1999 [cited.

[5]. Ennis, R. *Critical thinking: what is it?* in *Proceedings of the Forty-Eighth Annual Meeting of the Philosophy of Society Denver*. 1992. Colorado.

[6]. Dochy, F.S., M.; Sluijsmans, D., *The use of self-, peer and co-assessment in higher education: A review* Studies in Higher Education, 1999. **24**(3): p. 331-350.

[7]. Stefani, L.A.J., *Peer, self and tutor assessment: Relative reliabilities.* Studies in Higher Education. Vol. 19. 1994. 69-75.

[8]. Boud, D., Cohen, R., & Sampson, J., *Peer Learning and Assessment.* Assessment and Evaluation in Higher Education, 1999. **24**(4): p. 413-426.

[9]. Topping, K.J., & Ehly, S. E., *Peer-assisted learning.* Journal of Educational and Psychological Consultation, 2001. **12**(2): p. 113-132.

[10]. Falchikov, N., *Peer feedback marking: developing peer assessment.* Innovations in Education and Training International, 1995. **32**: p. 175-187.

[11]. Brindley, C., Scoffield, S., *Peer Assessment in Undergraduate Programmes.* Teaching in Higher Education, 1998. **3**(1): p. 79-89.

[12]. Davies, P., *Computerized peer assessment.* Innovations in Education and Training International, 2000. **37**(4): p. 346-355.

[13]. Sitthiworachart, J., and Joy, M., *Deepening Computer Programming Skills by Using Web-based Peer Assessment*, in *4th Annual Conference of the LTSN Centre for Information and Computer Sciences*. 2003: NUI Galway, Ireland.

[14]. Orsmond, P., Merry, S., and Reiling, K., *The use of Exemplars and Formative Feedback when Using Student Derived Marking Criteria in Peer and Self-assessment.* Assessment & Evaluation in Higher Education, 2002. **27**(4): p. 309-323.

[15]. Lin, S., Liu, E., and Yuan, S., *Web-based peer assessment: feedback for students with various thinking-styles.* Journal of Computer Assisted Learning, 2001. **17**: p. 420-432.

[16]. Bhalerao, A.a.W., A, *Towards Electronically Assisted Peer Assessment: A Case Study.*

Association for Learning Technology journal, 2001. **9**(1): p. 26-37.

[17]. Lin, S.S.J., Yang, K. H., Liu, E. Z. F. & Yuan, S.-M., *Peer Assessment in an Assembly Programming Course of a Vocational Industrial High School: A Case Study on Assessment Validity and Student Attitudes*. Journal of Technology, 2001. **16**(4): p. 613-623.

[18]. Jirarat Sitthiworachart, M.J., *Effective Peer Assessment for Learning Computer Programming*, in *Innovation and Technology in Computer Science Education*. 2004: Leeds, United Kingdom.

[19]. Willoughby, S. *Taiwan 16-Tile Mahjong Rules*. 1997 [cited; Available from: http://www.rag.com/~steve/mahjong/.