

# 國立交通大學

資訊科學與工程研究所

## 碩士論文

以大眾分類法為基礎之網頁應用程式合作式測試方法

Applying Folksonomy-Based Approach to Support Collaborative  
Testing of Web Applications

研究生：黃國彰  
指導教授：曾憲雄 教授

中華民國 九十九年七月

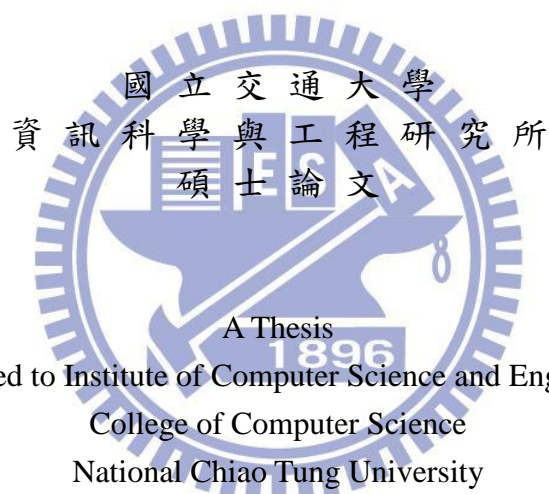
以大眾分類法為基礎之網頁應用程式合作式測試方法  
Applying Folksonomy-Based Approach to Support Collaborative  
Testing of Web Applications

研究生：黃國彰

Student : Kuo-Chang Huang

指導教授：曾憲雄

Advisor : Dr. Shian-Shyong Tseng



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

July 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

# 以大眾分類法為基礎之網頁應用程式合作式測試方法

研究生：黃國彰

指導教授：曾憲雄博士

國立交通大學資訊學院

資訊科學與工程研究所

## 摘 要

現今，以網頁為基礎的軟體系統成長快速，確保軟體的品質以及可依賴性變成一個非常重要的議題。在一般的軟體測試程序中，測試程序是非常密集且需要在測試人員、開發者以及一般使用者間互相合作配合。近幾年來，在軟體工程的領域上，許多針對網路應用程式以及合作式測試的方法不斷的被提出。然而，大部分的合作式測試工具往往只針對在測試程序以及臭蟲回報這方面。並沒有任何一個工具能夠考慮到測試需求間的彼此溝通以及測試資源的限制這一類型的合作式測試。網際網路上存在著大量各形各色且有經驗的人力資源，而這些資源非常適合應用在合作式測試上。針對這些網路上各形各色的測試人員，我們提出了一個以大眾化為基礎的測試方法支援合作式測試並且根據我們提出的方法建構出相對應的原型工具。在這篇研究中，一個以大眾化為基礎的兩階段式原型工具將被提出。且根據實驗結果顯示我們所提出的方法在測試方面是相當有效果且有效率的。實驗顯示平均的減少比率可達到將近 90%。

**關鍵字：**大眾化分類、合作式測試、網路應用程式測試、測試收斂、功能性測試、程式相依性圖表

# Applying Folksonomy-Based Approach to Support Collaborative Testing of Web Applications

**Student: Kuo-Chang Huang**

**Advisor: Dr. Shian-Shyong Tseng**

**Institute of Computer Science and Engineering  
Nation Chiao Tung University**

## Abstract

As the quantity and breadth of Web-based software systems continue to grow rapidly, assuring the quality and reliability of this software domain is becoming critical. Software testing processes, in general, are labor-intensive processes and involve substantial collaboration between testers, developers, and even users. Recently, many approaches have been proposed to address Web application testing and collaborative testing in software engineering domain. However, most collaborative testing tools just focus on testing processes and generating bug report. There are no tools supporting collaboration test with the consideration of communication requirement and testing resource constraints under test scheduling. Under Internet environment, there are a large amount of various and experienced human resource. And it is appropriate for collaborative testing of software functionality and usability by utilizing folk resource on the Internet. To test Web application with folk testers on Internet, we proposed a two-phase folksonomy-based approach to support collaboration and then constructed a prototype tool accordingly. The experimental results show that our approach is effective, and that the average reduction rate for testing effort is almost 90%.

**Keywords:** folksonomy-based approach, collaborative testing, web application testing, test convergence, functional testing, program dependence diagram.

## 致謝

碩士生涯的兩年求學過程，絕對是一生中值得的難忘回憶。本篇論文的完成，首先最要感謝我的指導教授 曾憲雄老師，老師總是不厭其煩的引領著我一步步的思考、探究並從而學習；跳脫出以往的思考框架限制，建構出正確的問題解決思路以及清楚表達的自我論述能力。此外，也要特別感謝在口試時給我許多寶貴意見的口試委員們：黃國禎教授、楊鎮華教授和彭文志教授，感謝你們賦予了此篇論文更豐富的內涵及意義。

再來要感謝在此研究過程中，給我許多鼓勵及建議的學長們：元昕學長、瑞鋒學長及宗儒學長，感謝你們三位的付出讓我得以順利完成此篇論文。也感謝實驗室的其他學長姐：喚宇學長、怡利學姐、俊銘學長及哲青學長，謝謝你們曾經給過的任何指導和幫助。再來是最親愛的同學們：杰峰、紹宜、佳榕、嘉祥和金龍，兩年來我們一路互相扶持、成長；笑過、哭過，這些回憶我都會一輩子珍藏，永遠都要當好朋友喔!!

最後我要感謝我的家人及朋友，感謝爸媽從小到大的栽培及撫育，因為你們最無私的關懷及愛才能有現在的我，感謝姊姊的照顧和關心，雖然平常很少跟你聊到心裡話，但我們彼此知道我們是最關心對方的，若我日後能有任何成就，榮耀都將歸屬於你們，我的家人。也謝謝我的女朋友伶蕙，很開心我的碩士生涯能夠有你陪伴在旁。

感謝所有經歷過的一切，不論好的壞的，都使我成長，使我銳變成更好的人。

# Table of Content

摘要.....	iii
Abstract.....	iv
致謝.....	v
Table of Content.....	vi
List of Figures.....	viii
List of Tables.....	ix
Chapter 1. Introduction.....	1
Chapter 2. Related Work.....	4
2.1 Web application testing approach .....	4
2.2 Collaborated-Based approach in Software Engineering.....	6
2.3 Folksonomy-Based approach.....	7
2.4 Program slicing and program dependence diagram .....	9
Chapter 3. Folksonomy-based approach for collaborative testing .....	11
3.1 Motivation Example.....	12
3.2 Problem definition: job assignment in collaborative testing.....	16
3.2.1 Problem definition .....	16
3.2.2 NP-Complete problem .....	19
3.3 Dependence graphs and Folksonomy-base approach.....	22
3.3.1 Dependence graph.....	22
3.3.2 Folksonomy-based approach for Web application testing .....	24
Chapter 4. Folksonomy-Based Collaborative Testing Algorithm .....	25
4.1 System Architecture of Proposed Model.....	26
4.2 Dependence graph definition .....	27
4.3 Construction Phase .....	29
4.3.1 The Example of Construction Phase .....	31
4.3.2 Construction Phase Algorithm.....	33
4.4. Testing Phase .....	36
4.4.1 Collaborative Testing Example.....	36
4.4.2 Collaborative Testing Algorithm .....	38
Chapter 5. Implementation and Experiment .....	42
5.1 System Implementation .....	42

<b>5.2 Experiments and Results</b> .....	<b>45</b>
<b>5.2.1 Experiment I - Dependence graph construction</b> .....	<b>45</b>
<b>5.2.2 Experiment II – Performance Evaluate Simulation</b> .....	<b>47</b>
<b>5.2.3 Experiment III – Performance Evaluation for Folksonomy-based Collaborative Testing</b> .....	<b>51</b>
<b>Chapter 6. Conclusion</b> .....	<b>53</b>
<b>References</b> .....	<b>54</b>



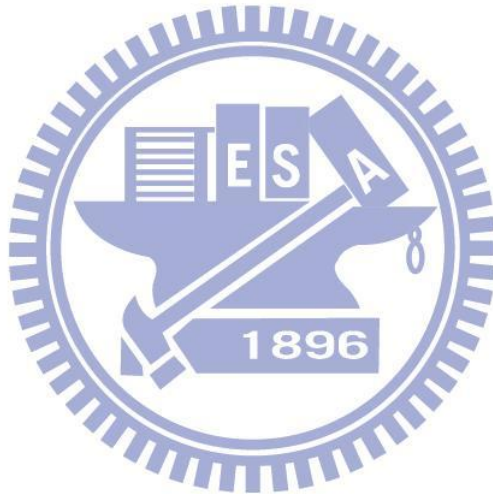
## List of Figures

Figure 1. Folksonomy testing scenario .....	12
Figure 2. Three-level dependence graph example:”BookStore” web application..	23
Figure 3. Collaborative testing example .....	24
Figure 4. System architecture .....	26
Figure 5. Dependence graph construction example: Node Detection .....	32
Figure 6. Dependence graph construction example: Edge Construction .....	32
Figure 7. Collaborative testing example .....	37
Figure 8. Collaborative testing system module architecture.....	43
Figure 9. Administration windows for User session logs .....	43
Figure 10. Testing flows of collaborative testing system.....	44
Figure 11. Dependence graph statistic result.....	46
Figure 12. “BookStore” web application with 10 support threshold .....	48
Figure 13. “BookStore” web application with 20 support threshold .....	48
Figure 14. “BookStore” web application with 30 support threshold .....	48
Figure 15. “BookStore” web application with Non-Guiding testing approach .....	49
Figure 16. “BookStore” web application with Guiding testing approach .....	49
Figure 17. Costing time comparing with different web application.....	50
Figure 18. Statistics of tester quantity and bug report about two kind of approach .....	51
Figure 19. Costing time of all testers spend on each page.....	52
Figure 20. Total completion time comparing.....	52



## List of Tables

<b>Table 1. Job-Page covered matrix.....</b>	<b>13</b>
<b>Table 2. Job execution time matrix.....</b>	<b>13</b>
<b>Table 3. Page confirmed matrix.....</b>	<b>14</b>
<b>Table 4. Tester trustworthy matrix.....</b>	<b>14</b>
<b>Table 5. Notations of Job assignment problem definition .....</b>	<b>17</b>
<b>Table 6. Property of Patterns .....</b>	<b>30</b>
<b>Table 7. Notations of Folksonomy-Based Testing Algorithm .....</b>	<b>39</b>
<b>Table 8. Dependence graph statistic result .....</b>	<b>45</b>



## Chapter 1. Introduction

As the quantity and breadth of Web-based software systems continue to grow rapidly, assuring the quality and reliability of this software domain is becoming critical. Low reliability of software can result in serious detrimental effects for businesses, consumers, and the government as they increasingly depend on the Internet for routine daily operations. A major impediment to producing reliable software is the labor and resource-intensive nature of software testing. Software testing processes, in general, are labor-intensive processes and involve substantial collaboration between testers, developers, and even users. In testing, testers are requested to assess software in restricted time, and report bugs back to the development team. In collaborative testing, testers need to share the test result and test status to the others. Since the test results are needed to report to developer and users, the communication is necessary under test processes. In some online collaborative software testing, such as Web application testing, game beta testing, open-source software testing, how to utilize a large amount of human resources on the Internet to perform test is interesting, so a collaborative tool to support collaborative testing is necessary. Under the situation of a complex architecture of Web application, the problem of how to perform testing rapidly and support collaborative work automatically should be considered for Web applications testing.

Recently, many approaches have been proposed to address Web application testing and software engineering collaboration. In [31], Whitehead classified collaboration tools of software engineering into four categories, model-based collaboration tools, processes support tools, awareness tools, and collaboration infrastructure. However, to the best of our knowledge all existing collaborative testing tools just focus on testing processes and generating bug report. There are no tools supporting collaboration test

with the consideration of communication requirement and testing resource constraints under test scheduling. Under Internet environment, there are a large amount of various and experienced human resource. And it is appropriate for collaborative testing of software functionality and usability by utilizing folk resource on the Internet. To test Web application with folk testers on Internet, we proposed a folksonomy-based approach to support collaboration and constructed a prototype tool with proposed approach.

In collaborative testing, we are concerned with the following issues [18][24][34][38]. (1) Software testing are inherently cooperative, requiring many testing engineers to coordinate their efforts to perform a software system testing. To coordinate this effort, we guide test activities and share information each other over the entire testing process. How to propose a collaboration technology within testing process is an interesting issue in collaborative testing. (2) In Web application testing, the combination of path executions and input parameters may cause test case explosion problem. With a large amount of test cases and restricted resources, how to assign the required test cases or jobs to corresponding testers becomes an important research topic. (3) Web applications typically involve complex, multi-tiered, heterogeneous architectures including Web sites, applications, database servers, and clients. How to perform Web application modeling for testing and consider a variety of situations to be able to handle the testing of various components in these architectures is also a challenging issue.

In our research, we propose a folksonomy-based testing approach that addresses these issues by analyzing user sessions to guide folk testers during testing period, potentially reducing the effort involved in testing. The two-phase folksonomy-based testing approach is proposed to support collaborative testing. Phase 1 is Web application program dependence graph construction. We represent program

constructions and information flows of Web application with the defined three levels program dependence graphs, code-level dependence graph, function-level dependence graph, and page-level dependence graph. Phase 2 contains folksonomy-based collaborative testing. We implement a prototype system to evaluate our proposed approach. With our collaborative testing tool, we can collect user sessions logs of participated testers during testing processes. To support collaborative testing, we further analyze the user session logs and guide the test activities. The results will be analyzed and utilized to improve the way of collaborative testing. In the experiment, we simulate the collaborative test activities with our proposed model. The experimental results show that our approach is effective and well performance in testing and defects revealing. And the average reduction rate for testing effort is almost 90%.

The main contributions of this research are as follows:

1. Model Web application with the dependencies, data dependence and control dependence, and propose the formulation of the Web application for collaborative testing.
2. Define the collaborative testing problem as ILP-formulation and prove the problem is NP-Complete.
3. Propose a folksonomy-based approach to support collaborative test of Web applications and implement a prototype tool based upon the approach.
4. Evaluate the experiments of the effectiveness of the proposed folksonomy-based approach.

The remainder of this research is organized as follows: Chapter 2 discusses related works. Chapter 3 gives an overview of the proposed approach. Chapter 4 explains how the test approach is performed in test scenarios. Chapter 5 explains how the experimental design and experimental result. Section 6 presents the conclusion and proposes future work.

## Chapter 2. Related Work

### 2.1 Web application testing approach

Recently, many approaches have been proposed to address Web application testing. We briefly describe relevant studies as follows. Ricca et al. [17] described an approach based on UML model for high level abstraction Web application. Similar to their approach, Jia et al. [48] presented a technique based on formal specification instead of UML model. These approaches are not appropriate to cope with dynamic behavior of modern Web application. Kung et al. [12] presented an approach based on multiple models of Web application under test. Andrews et al. [1] proposed a system-level testing technique that combines test generation based on finite state machines with constraints. These approaches have state space explosion problem if the Web application is very complicated.

Deng et al. [49] proposed an approach based on static analysis for recovering the model of Web application. In their approach, the model of the Web application is built by scanning its source code to identify links and names of input parameters. However, their approach only identifies the names of input parameters without considering its domain information, such as parameter types, relevant values of the parameters, so it might not be applied directly to generate test case. William et al. [47] presented a technique for automatically discovering Web application interfaces based on a two-phrase static analysis algorithm. To generate test cases, the problem of how to model the behavior of Web application should be considered in Web applications testing.

User-sessions-based approach is a convenient way to collect test cases. Elbaum et al. [42] proposed the user-session-based approach to test Web application by transforming user sessions into test cases, where different strategies can be applied to

construct test cases for the collected user sessions [42], [30]. The promising results demonstrated the fault detection capabilities and cost effectiveness of user-session-based testing. In addition, they observed that the effectiveness of user-session-based testing can be improved as the number of collected sessions increases; however, the cost of collecting, analyzing, and replaying test cases also increases. User-session-based testing techniques are complementary to the testing performed during the development phase of the application [8], [20], [17], [29], [7], [49], [34].

It is a serious problem that how to select test cases from a large number of test cases when testing the application. Since test suite reduction has several advantages, such as reducing the cost of executing, validating, and managing test suites as the application evolves, the test suite reduction is important for Web application testing. Several test suite reduction techniques have been proposed [37], [45], [28], [26], [13], [41], [14], [15]. Harrold et al. [37] developed a test suite reduction technique that employs a heuristic based on the minimum cardinality hitting set to select a representative set of test cases that satisfies a set of testing requirements. Harder et al. [36] proposed an operational-abstraction-based minimization technique that can be executed incrementally, but dynamically generating operational abstractions can be costly. In this research, we propose a novel approach to support collaborative testing for Web application and construct a new model with folksonomy-based approach.

## 2.2 Collaborated-Based approach in Software Engineering

In software engineering, participants have adopted a wide range of communication and collaboration technologies to assist in the coordination of project work[31]. Engineers have developed a wide range of model oriented technologies to support collaborative work on their projects. These technologies span the entire lifecycle, including collaborative requirements tools [3], collaborative UML diagram creation, software configuration management systems and bug tracking systems. Process modeling and enactment systems have been created to help manage the entire lifecycle, supporting managers and developers in assignment of work, monitoring current progress, and improving processes [19] [5] . In the commercial sphere, there are many examples of project management software, including Microsoft Project and Rational Method Composer. Several efforts have created standard interfaces or repositories for software project artifacts, including WebDAV/DeltaV [32] [16] and PCTE [33]. Web-based integrated development environments serve to integrate a range of model-based (SCM, bug tracking systems) and unstructured (discussion list, web pages) collaboration technologies.

There are many researches related to collaborative testing. However, most collaborative testing tools above just focus on testing processes and generating bug report. There were no tools supporting collaboration test with considering communication requirement and testing resource constraints under test scheduling in fact. To test Web application with folk testers on Internet, we proposed a folksonomy-based approach to support collaboration and constructed a prototype tool with proposed approach.

## 2.3 Folksonomy-Based approach

A folksonomy is a system of classification derived from the practice and method of collaboratively creating and managing tags to annotate and categorize content; this practice is also known as collaborative tagging, social classification, social indexing, and social tagging. Folksonomy, a term coined by Thomas Vander Wal, is a portmanteau of folk and taxonomy.

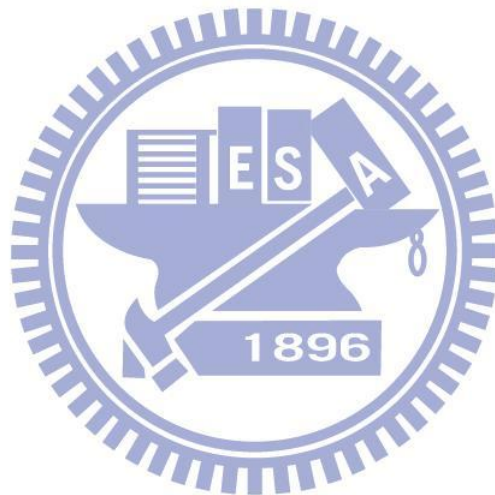
General overviews on folksonomy systems and their strengths and weaknesses are given in [46][4]. In [40], Mika defined a model of semantic social networks for extracting lightweight ontologies from del.icio.us. Recently, work on more specialized topics such as structure mining on folksonomies—e. g. to visualize trends [35] and patterns [9] in users' tagging behavior—as well as ranking of folksonomy contents [2], analyzing the semiotic dynamics of the tagging vocabulary, or the dynamics and semantics [23] have been presented.

In the literatures, the existing approaches usually focus on the collaborative filtering and information retrieval areas. AutoTag [21], e.g., is a tool that suggests tags for weblog posts using information retrieval techniques. Xu et al. [50] introduced a collaborative tag suggestion approach based on the HITS algorithm [27]. A goodness measure for tags, derived from collective user authorities, is iteratively adjusted by a reward-penalty algorithm. Benz et al. [11] introduced a collaborative approach for bookmark classification based on a combination of nearest-neighbor classifiers. A keyword recommender plays the role of a collaborative tag recommender, but it is just a component of the overall algorithm. Besides, the standard tag recommenders, in practice, are services that provide the most-popular tags used for a particular resource. This is usually done by means of tag clouds where the most frequent used tags are depicted in a larger font or otherwise emphasized.

The approaches described above address important aspects of the folksonomy,



collaborative intelligence. Collective intelligence is a shared or group intelligence that emerges from the collaboration and competition of many individuals and appears in consensus decision making in bacteria, animals, humans and computer networks.



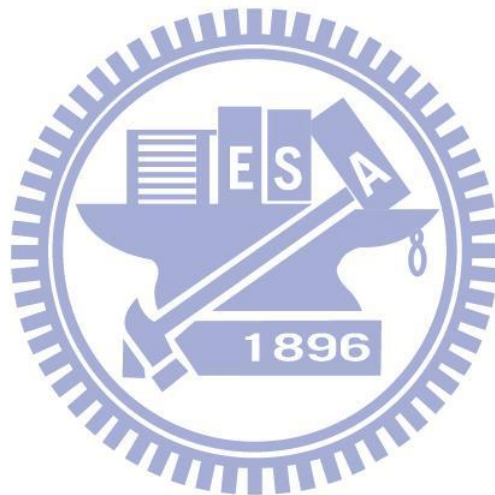
## 2.4 Program slicing and program dependence diagram

Much related work has been performed over the past ten years in the area of dependence-based program representations. The traditional syntactic view of software dependency had its origins in compiler optimizations, and focused on control and dataflow relationships [25]. This approach extracts relational information between specific units of analysis such as statements, functions or methods, and source-code files. Dependencies are discovered, typically, by analysis of source code or from an intermediate representation such as bytecodes or abstract syntax trees. These relationships can be represented either by a data-related dependency (e.g., a particular data structure modified by a function and used in another function) or by a functional dependency (e.g., method A calls method B).

The work by Hutchens and Basili [22] and Selby and Basili [43] represent of the first use of dependency data in the context of a system's propensity for failure. Based on the concepts of coupling and cohesion proposed by Stevens et al. [44], Hutchens and Basili [22] presented metrics to assess the structure of a system in terms of data and functional relationships, which were called bindings. The authors used clustering methods to evaluate the modularization of a particular system. Selby and Basili [43] used the data binding measure to relate system structure to errors and failures. They found that routines and subsystems with lower coupling were less likely to exhibit defects than those with higher levels of coupling. Similar results have been reported in object-oriented systems. Chidamber and Kemerer [10] proposed a set of measures that captures different aspects of the system of relationships between classes. Briand et al. [6] found that the measures of coupling proposed by Chidamber and Kemerer were positively associated with failure proneness of classes of objects.

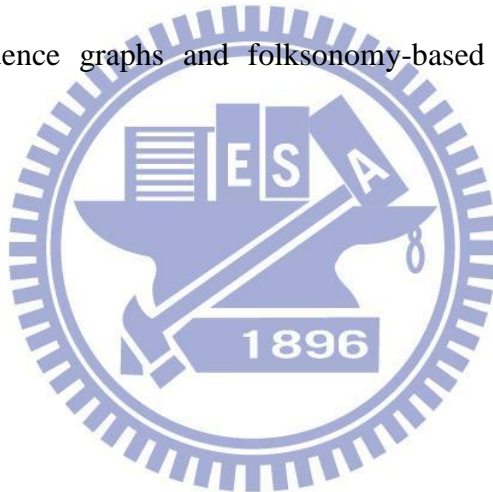
In this research, we use program slicing techniques to construct dependence graphs. To represent Web application, the three-level dependence graphs are constructed in

Chapter 3. And different graphs stand for different semantic meanings of software testing.



## Chapter 3. Folksonomy-based approach for collaborative testing

In this chapter, we will introduce our proposed approach to support collaborative testing for Web application. First, we describe a motivating example of job assignment problem. Then, we define the job assignment problem based upon mathematical definition. We construct our problem model as ILP-formulation by considering resources constrains and job assignment under testing. The problem is proved as an NP-Complete problem. We reduce the problem to minimal representative set problem and prove the problem is NP-Complete. In the last section of Chapter 3, we propose the heuristic ideas, dependence graphs and folksonomy-based approach, to solve the problem.



### 3.1 Motivation Example

Under Internet environment, there are a large amount of various and experienced human resource. And it is appropriate for collaborative testing of software functionality and usability by utilizing folk resource on the Internet. However, people usually execute some specific popular function of the application when they face a new system without interact each other. When participated testers focus on specific popular functions, the resources are wasted in overlapped and duplicated testing. And the lack of testing of the other pages will result in unequal distribution of resources and slow test convergence. In this situation, it seems that the problem is associated with job scheduling and resource allocation.

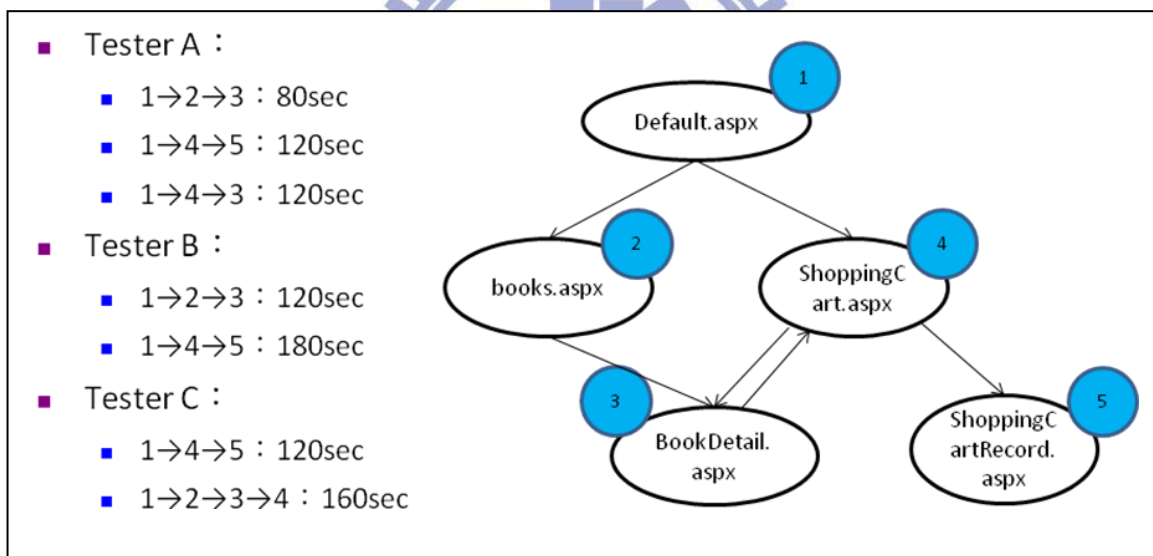


Figure 1. Folksonomy testing scenario

Table 1. Job-Page covered matrix

	Page 1	Page 2	Page 3	Page 4	Page 5
Job 1	1	1	1	0	0
Job 2	1	0	0	1	1
Job 3	1	0	1	1	0
Job 4	1	1	1	1	0

Table 2. Job execution time matrix

	Tester A	Tester B	Tester C
Job 1	80 sec	120 sec	80 sec
Job 2	120 sec	180 sec	120 sec
Job 3	120 sec	180 sec	120 sec
Job 4	160 sec	240 sec	160 sec

As shown in Figure 1, a simple application is excerpted from the online open-source project, “Bookstore”[<http://www.gotocode.com/>], which contains five Web pages and 4 jobs, where each job consists of several pages and possible executed paths. For example, the Job 1 in Figure 1 can be represented as (P1→P2→P3). As a result, the job-page covered matrix is formed as Table 1. The testing finishes when all of pages are executed at least one time. There are four testers participated in this testing and the job executing time for each tester can be estimated by summarizing corresponding page executing times. Tester A executes Job 1, consisted of page 1, 2, and 3, with 80 seconds in this testing. The Job execution time matrix is shown in Table 2 by summarizing execution times of testers.

We want to find out the minimal job quantity which can cover the whole page and make the minimal execution time of the maximum execution time of the user. When

testing begins, tester A, tester B, and tester C execute pages respectively. The testing is terminated after all of the pages can be executed at least one time. For the sake of testing, the testers should coordinate the jobs respectively and cooperate to get work done quickly. One of the solutions of this case is shown below:

Job 1: Tester A spends 80 Seconds.

Job 2: Tester A spends 120 Seconds.

However, we should further consider other factors in our testing according to the real environment. First, what is “testing done” under testing? Each page should have same standard or not? We create the confirmed criteria for each page based on different complexity. In the page confirmed matrix, each page has he own threshold of testing quantity substitute for test once. Second, should we trust the testing results of testers? Based on folksonomy-based approach, we create the tester trustworthy matrix with testers. By considering these constraints, the problem is more complex than original one.

Table 3. Page confirmed matrix

	Page 1	Page 2	Page 3	Page 4	Page 5
Confirm condition	10	10	10	8	8

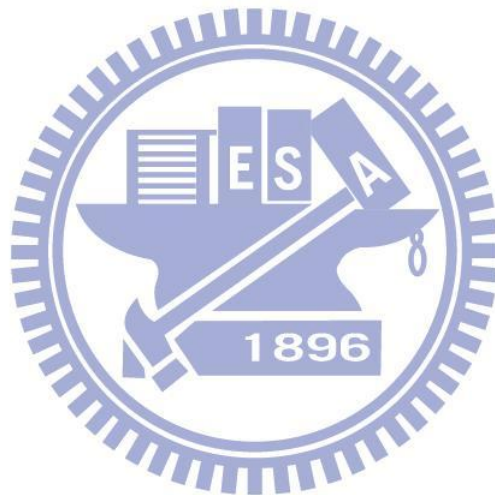
Table 4. Tester trustworthy matrix

	Tester A	Tester B	Tester C
Trustworthy	1	0.5	0.7

Based on description above, we discover that the purpose of the proposed testing is to find the “maximal” completion time of the participated testers. And the key point is how to assign jobs to testers and guide testers to the required regions to achieve the

“minimal” total completion time. It is a job assignment problem with resource constraints. And we can conclude some proposed problems below:

- (1) Testing resource constraints should be considered in testing.
- (2) What is the stop criteria of testing?
- (3) How can we trust the testing results of participated testers?





## 3.2 Problem definition: job assignment in collaborative testing

### 3.2.1 Problem definition

In this section, we define the proposed problem we mentioned above with mathematical formulation. In Table 5, we show the notation definition used in the formulation. The model of proposed problem, job assignment problem, is formulated as ILP formulation in Definition 1.

We adopt three variables to represent jobs, testers, and Web pages. The variable  $i$  represents the  $i_{th}$  job sequence, the variable  $j$  represents the  $j_{th}$  tester, and the variable  $k$  represents the  $k_{th}$  page of the web application. We use three binary decision variables to represent the status of variables  $i$ ,  $j$ , and  $k$ . The variable  $\delta_{ij}$  equals 1 if the  $i_{th}$  job assign to tester  $j$  and response successfully. The variable  $\ell_i$  equals 1 if job  $i$  is legal. And the variable  $d_{nm}$  equals 1 if page  $n$  has a dependence edge link to page  $m$ . Some variables are defined according to the constraints of above variable. The variable  $H_j$  stands for testers' total working hours, and is used to represent the total completion time limit of  $j_{th}$  tester. The function  $T(P_i, j)$  stands for the  $i_{th}$  job completion time of tester  $j$ . The variable  $W_j$  stands for the trustworthy weight of tester  $j$ . And each tester has his own trustworthy weight when he executes testing works. The variable  $S_k$  stands for the support threshold of  $k_{th}$  page, each page has his own testing terminate condition.

Table 5. Notations of Job assignment problem definition

$i$  : ith job sequence ;  $j$  : jth tester ;  $k$  : kth page

$\delta_{ij}$  = ith job assign to tester j and response successfully

$T(P_i, j)$  = ith job costing time of tester j

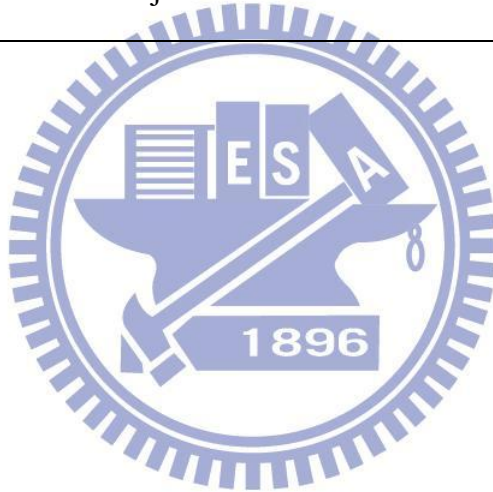
$W_{ij}$  = trustworthy weight of tester j to do ith job

$S_k$  = support threshold of kth page

$\ell_i$  = job i is legal

$d_{nm}$  = page n has a dependence edge to page m

$H_j$  = total testing time limit of jth tester



Definition 1: Job assignment problem definition

Objective function:

$$1. \min_{time} \max_j \sum_i \delta_{ij} \ell_i T(P_i, j)$$

Subject to:

$$2. \delta_{ij} \in \{0,1\}$$

$$3. T(P_i, j) \in R^+$$

$$4. \sum_j \delta_{ij} \ell_i T(P_i, j) \leq H_j$$

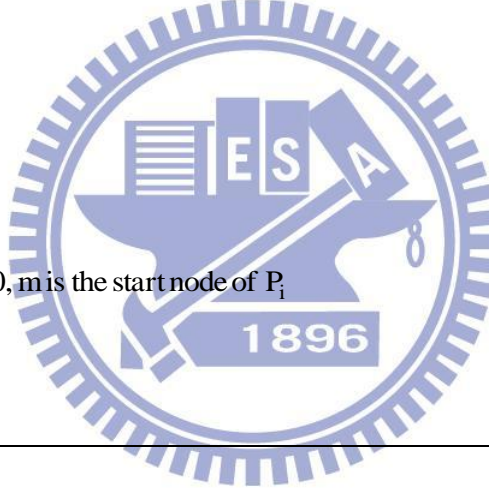
$$5. \sum_j \sum_{\{i|k \in P_i\}} \delta_{ij} \times W_j \geq S_K$$

$$6. S_K > 0$$

$$7. d_{nm} \in \{0,1\}$$

$$8. \ell_i = \begin{cases} 0, & \text{If } d_{nm} \neq 0, m \text{ is the start node of } P_i \\ 1, & \text{otherwise} \end{cases}$$

$$9. 0 < W_{ij} \leq 1$$



In Definition 1, the Job assignment problem can be modeled as ILP-formulation with these variables. The objective function (1) minimizes the total completion time. The constraint (2) ensures the testing job is assigned. The constraint (3) ensures the job costing time is a positive real number. The constraint (4) ensures each tester has reasonable total testing time, avoid the overloading. Constraints (5) and (6) ensure each page has enough testing quantity and it is a positive number. Constraints (7) and (8) ensure the job should be illegal, and the starting page should be in-degree dependence edge. Finally, constraint (9) ensures the value of user trustworthy weight should from zero to one.

### 3.2.2 NP-Complete problem

This section shows the job assignment problem in this study domain is a NP-Complete problem. First, we will show this problem is an NP-problem. Second, we reduce the “Optimum representative set”[39] NP-Hard problem to our problem to show it is an NP-Complete problem.

The corresponding decision problem of job assignment problem:

JAP= $\{ \langle G, J, U, D, S, W, H, T, k \rangle \}$

$G=(V, E)$  is a complete directed graph.

$J$  is a set of walks in  $G$ .

$U$  is a set of testers.

$D$  is a function from  $V \times V \rightarrow Z_2$

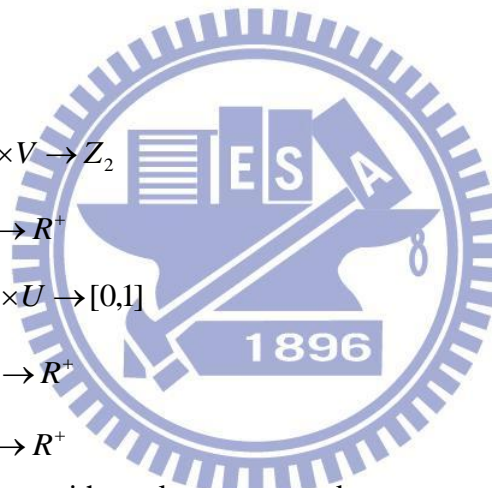
$S$  is a function from  $V \rightarrow R^+$

$W$  is a function from  $J \times U \rightarrow [0, 1]$

$H$  is a function from  $U \rightarrow R^+$

$T$  is a function from  $J \rightarrow R^+$

And there is an assignment with total cost at most  $k$ .



Thm Job Assignment Problem (JAP) is NP-Complete

Proof:

We first show that JAP belongs to NP. Given an instance of the problem, the verification algorithm checks that the sum of the trustworthy  $W_i$  of assigned legal test paths of every tester  $I$  of each page  $k$  exceeds the threshold  $S_K$ , the sum of the execution time of assigned legal test paths of each tester  $I$  does not exceed the threshold  $H_i$ , and checks whether the maximum of the sum of the execution time of assigned legal test paths of each tester is at most  $k$ . This process can certainly be done in polynomial time.

To prove that JAP is NP-Hard, we show that  $RS \leq_p JAP$ . Let  $R = \{R_i | i = 1, \dots, n\}$ ,  $T = \{T_i | T_i \subseteq R, i = 1, \dots, m\}$  and  $k$  be an instance of RS. We construct an instance of JAP as follows. We form the complete directed graph  $G=(V,E)$  where  $V=R$  and  $E = \{(i, j) | i, j \in V, i \neq j\}$ , and we define the test job set  $J = \{P_i | V(P_i) = T_i, P_i \text{ is a path and } V(P_i) \neq V(P_j), \text{ if } i \neq j\}$ , the tester set  $U = \{1\}$ , the dependence function  $d$  by  $d(i, j) = 0 \quad \forall i, j$ , the confirm function  $S$  by  $S(i) = 1 \quad \forall i \in U$  the trustworthy function  $w$  by  $w(i, j) = 1 \quad \forall i \in U, j \in T$  the human resource function  $H$  by  $H(i) = |V| \cdot |T|, \forall i \in U$  the execution time function  $t$  by  $t(P_i, j) = 1, \forall P_i \in T, j \in U$ .

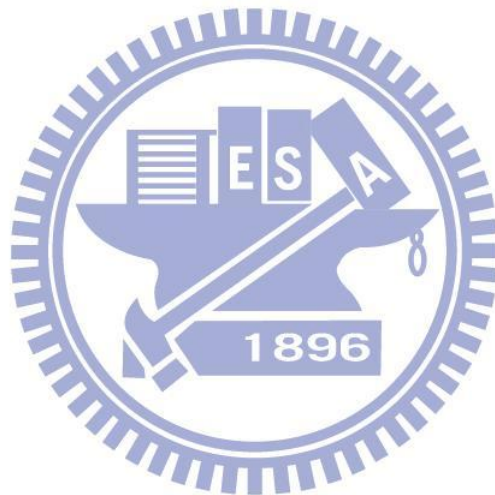
The instance of JAP is then  $\langle G, J, U, d, S, W, H, T, k \rangle$ , which is easily formed in polynomial time.

We now show that there is a representative set  $T'$  with  $|T'| = k$  if and only if graph  $G$  has an assignment  $\delta$  of the maximum total cost at most  $k$ . Suppose that there is a representative set  $T'$  with size  $k$ . Therefore, there exists an assignment  $\delta$  such that  $\delta_{i1} = 1$  if  $T_i \in T'$ , otherwise  $\delta_{i1} = 0$  because  $U\{T_i | T_i \in T'\} = R = V$ , the trustworthy of tester is 1, the confirm threshold of each page is 1, and the human resource of tester is  $|R| \cdot |T|$ . Thus, this assignment  $\delta$  is a feasible solution and the maximum total cost is  $k$ . Conversely, suppose that there is an assignment  $\delta$  with the maximum total cost is  $k$ . Then, there existed a subset of  $T$ ,  $T' = \{T_i | T_i = V(P_i), \delta_{i1} = 1\}$ . Because  $\ell_i = T(P_i, 1) = 1, \forall i$ , and  $\max_j \sum_i \delta_{ij} \ell_i T(P_i, j) = \sum_i \delta_{i1} = k$ ,  $|T'| = k$ .

$\sum_j \delta_{ij} \sum_{\{i|k \in V(P_i)\}} W_{kj} \geq S_k = 1, \forall k \in V$  , this implies that for each requirement

$r \in R = V, \exists T_i \in T'$  such that  $r \in T_i$ . Therefore,  $T'$  is a representative set with size.

Hence, JAP is NP-Complete #



### 3.3 Dependence graphs and Folksonomy-base approach

#### 3.3.1 Dependence graph

A Web application composed of Web pages. Pages interact with each other with some methods, such as: Web submit form, transfer data through URL, transfer data through sessions and cookies, and access database. The methods construct the dependence relationships in Web applications. According to these dependence relationships, we can represent the Web application with a dependence graph with different testing level, page level, function level, and code level. We construct the page-level dependence graph, function-level dependence graph, and code-level dependence graph with the idea of program structures. The dependence graphs can cover different program semantic levels that stand for the different testing level. The different dependence graph means different usability testing, page-level testing, function-level testing, and code-level testing. In code level, the dependence graph is constructed by code block. In function level, the dependence graph is constructed by functional statements. And in page level, the dependence graph is constructed by Web pages.

We represent Web application with dependence graphs. The dependence graphs are shown in Figure 2. Nodes in the page level dependence graph stand for Web pages. The arrow means control flow of Web application. The page is composed of several functions and function is composed of code block. For example, the page “ShoppingCartRecord.aspx” contains 8 functions, “page\_init()”, “page\_load()”, “page\_how()”, “page\_nload()”, “update\_click()”, “delete\_click()”, and “cancel\_click()”. And the “update\_click()” function contains 6 code blocks. Nodes in the code-level dependence graph stand for compound statements in functions, example show compound statements contains in the “update\_click()” function.

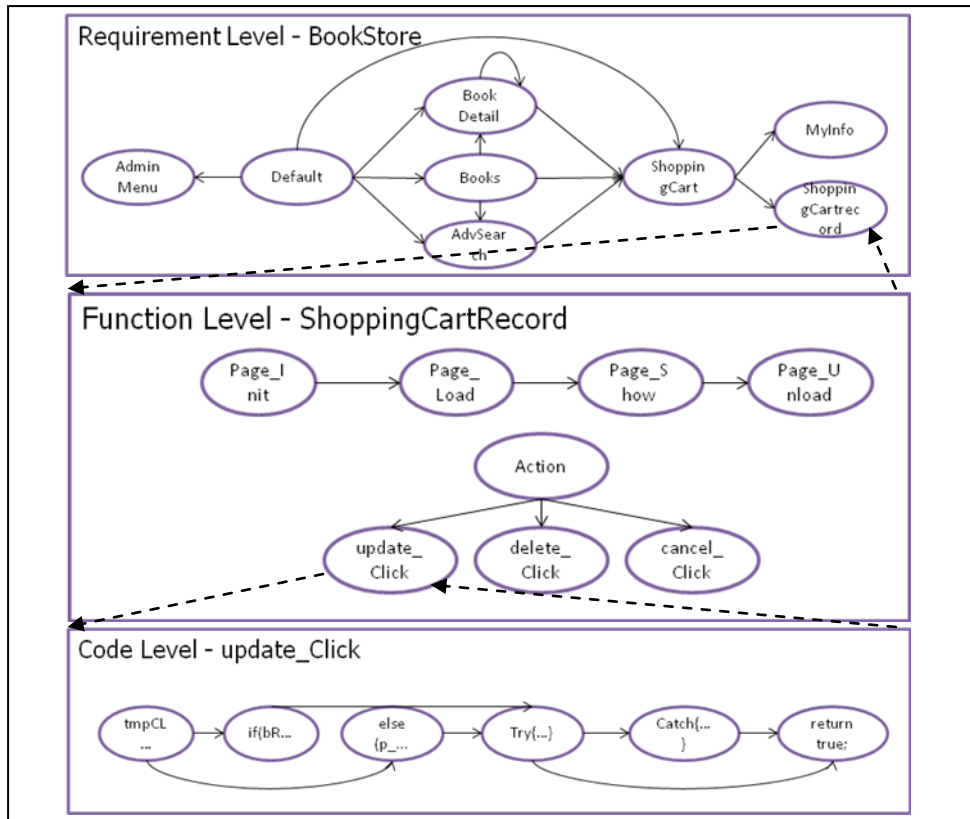
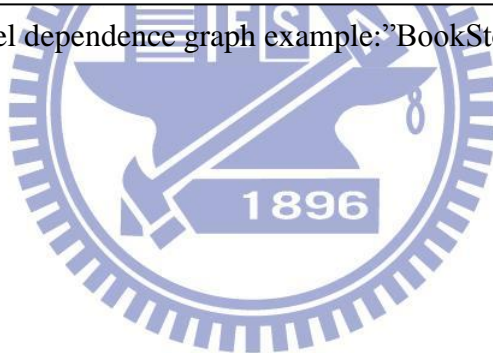


Figure 2. Three-level dependence graph example: "BookStore" web application





### 3.3.2 Folksonomy-based approach for Web application testing

In collaborative testing, the test result will be affected by each participated tester. And it is interesting that how to incorporate testers' contributions to finish the testing. In this research, we try to incorporate the testing results of testers and finish the testing with folksonomy-based approach. With the idea of folksonomy-based approach, we incorporate the folk testers in collaborative testing. We collect the session logs of testers and incorporate the testing result to conduct current collaborating situation iteratively. Then, we compute and assign the new testing objective to testers. When a testing block has enough simple testing quantity, and do not exist any obvious mistake until now, we want to guide tester to other testing block which have not enough simple testing quantity. Finally, the test finish until all testing blocks is assigned or done. Figure 3 shows the collaborative testing example with folksonomy-based approach, when testing block 1 achieve the confirm condition, we will guide tester to execute testing block 2.

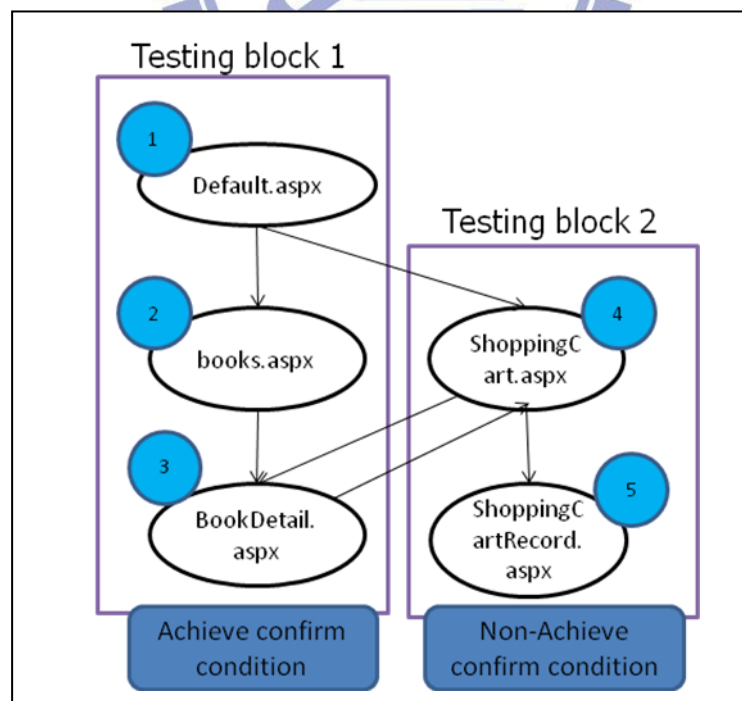
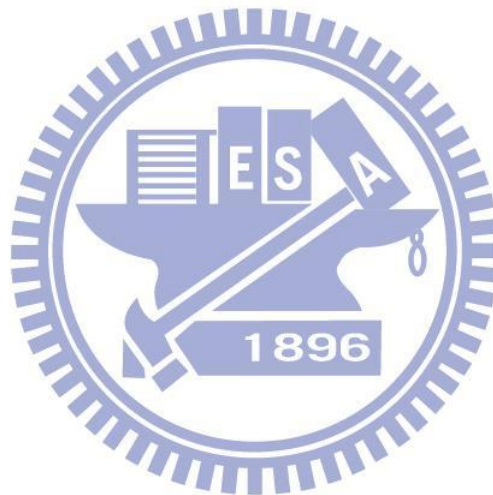


Figure 3. Collaborative testing example

## Chapter 4. Folksonomy-Based Collaborative Testing

### Algorithm

In this chapter, the folksonomy-based approach we proposed is used to support collaborative test. To perform test in Web application, there are two main phases in our proposed algorithm as shown in Figure 4. In Phase 1, we construct the dependence graphs by program slicing technique. We transform the source code into PDG, program dependence graphs. In phase 2, the testing phase contains collaborative testing algorithm.



## 4.1 System Architecture of Proposed Model

Figure 4 shows the two phases approach of our proposed methodology. In phase 1, we transform web application to corresponding dependence graph by graph construction algorithm. The Web application is represented as dependence graphs. It helps us to realize that the dependence relationships in a web application. In the testing phase, a testing job will be proposed for each tester according to computed results of the collaborative testing algorithm. The collaborative testing algorithm refers dependence graphs, user profiles, and user session logs, and computes the next test targets. After testing of each round, system will assign a new job to tester until all of the testing will be finished.

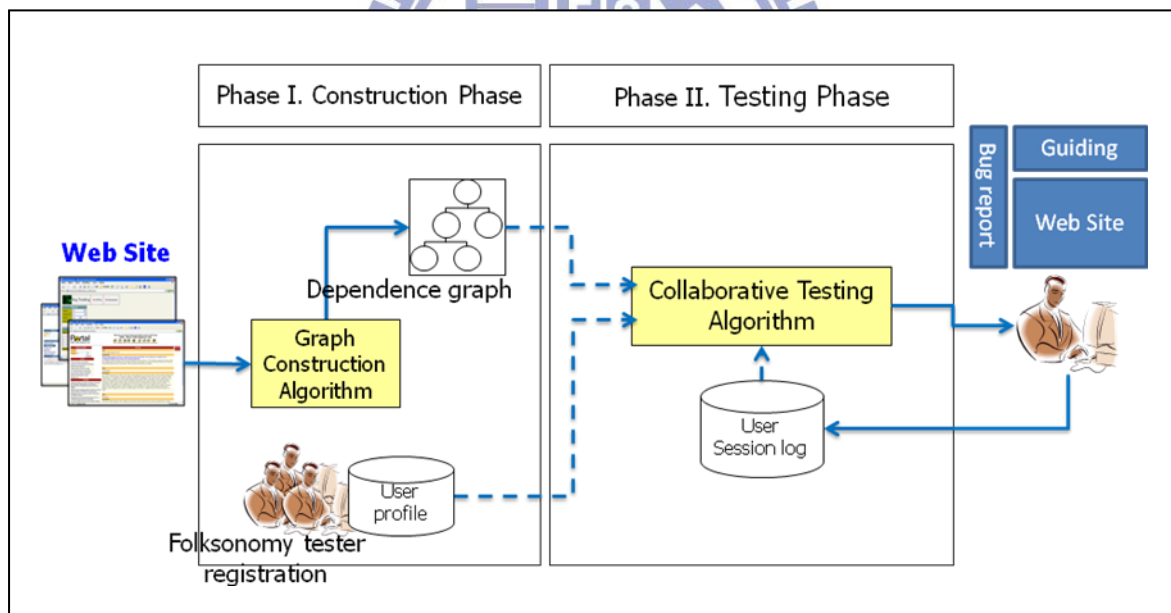
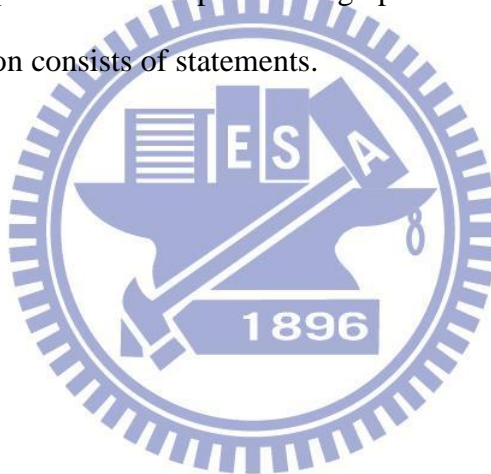


Figure 4. System architecture

## 4.2 Dependence graph definition

In this section, we define dependence graph based upon graph theory. There are three dependence graphs in our methodology. Each graph has its own vertexes and edges. We define the vertexes of dependence graphs with  $V_P$ ,  $V_F$  and  $V_C$  which means page, function and compound statement respectively. And there are five types of edges in the dependence graph,  $E_P$ ,  $E_F$ ,  $E_C$ ,  $E_{PF}$ , and  $E_{FC}$ .  $E_P$ ,  $E_F$ , and  $E_C$  stand for the edges between  $V_P$ ,  $V_F$  and  $V_C$ .  $E_{PF}$  represents the edges between  $V_P$  and  $V_F$ , and  $E_{FC}$  represents the edges between  $V_F$  and  $V_C$ . The edges  $E_{PF}$  and  $E_{FC}$  are used to connect different level graph of dependence graphs. And we use the edges to represent the hierarchical relationships between dependence graphs. The page contains several functions and the function consists of statements.



## Definition2: Dependence graph definition

Dependence directed graph

$$G = (V, E)$$

$$V = V_P \cup V_F \cup V_C \text{ where}$$

$$V_P = \{P_i | P_i \text{ is a page}\} \text{ is the set of page}$$

$$V_F = \{F_i | F_i \text{ is a function}\} \text{ is the set of function}$$

$$V_C = \{C_i | C_i \text{ is a codestatement}\} \text{ is the set of code statement}$$

$$E = E_P \cup E_F \cup E_C \cup E_{PF} \cup E_{FC} \text{ where}$$

$$E_P = \{(P_i, P_j, I_i) | P_i, P_j \in V_P, I_i \subseteq I\}$$

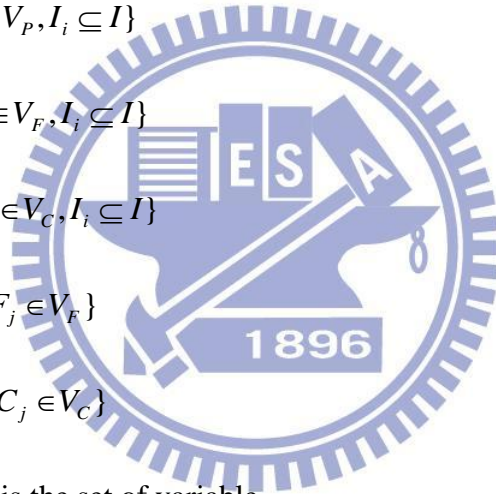
$$E_F = \{(F_i, F_j, I_i) | F_i, F_j \in V_F, I_i \subseteq I\}$$

$$E_C = \{(C_i, C_j, I_i) | C_i, C_j \in V_C, I_i \subseteq I\}$$

$$E_{PF} = \{(P_i, F_j) | P_i \in V_P, F_j \in V_F\}$$

$$E_{FC} = \{(F_i, C_j) | F_i \in V_F, C_j \in V_C\}$$

$$I = \{V_i | V_i \text{ is a variable}\} \text{ is the set of variable}$$





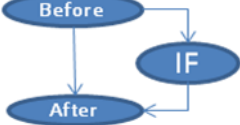
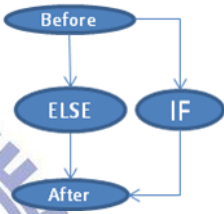



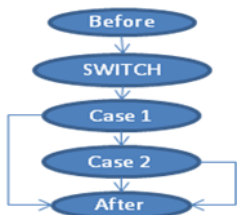



### 4.3 Construction Phase

To construct the dependence graphs from program codes, we analyze the program structures of Web application according to the properties of the C# .net language. We arrange Table 6 to record the information of each program structure patterns which will affect graph construction. In Table 6, we classify nodes, edges, and patterns according to different level, page, function, and code. The slot “pattern” shows the logic relationship between nodes and edges. Based on the syntax of each pattern, we can detect nodes and edges from Web application and form the dependence graphs.



Table 6. Property of Patterns

Level	Syntax	Pattern
None	None	
Page	NavigateUrl	
Page	Response.Redirect	
Function	Function caller	
Code	If	
Code	If...else...	
Code	While	
Code	For	
Code	Do...while	
Code	Switch	
Code	Try...Catch...Final	

### 4.3.1 The Example of Construction Phase

After defining the dependence graph, we present an example of dependence graph construction in this section. As shown in Figure 5 and Figure 6, the construction phase contains six main steps.

#### Step 1: Detect the nodes in the page level

According to the “Property of Patterns” in Table 6, each page file in the web application can be constructed as nodes in the page level.

#### Step 2: Detect the nodes in the function level

According to the “Property of Patterns” in Table 6, we can detect functional nodes that declaring in source codes. We construct nodes in the function level with detection.

#### Step 3: Detect the nodes in the code level

According to the “Property of Patterns” in Table 6, we can detect the nodes of code level dependence graph from source code. We construct nodes in the code level, and each node in the code level dependence graph is a compound statement.

#### Step 4: Construct the edges

According to the “Property of Patterns” in Table 6, we can connect nodes in dependence graphs with corresponding edges. And we connect different dependence graphs with edges to represent hierarchical relationships.



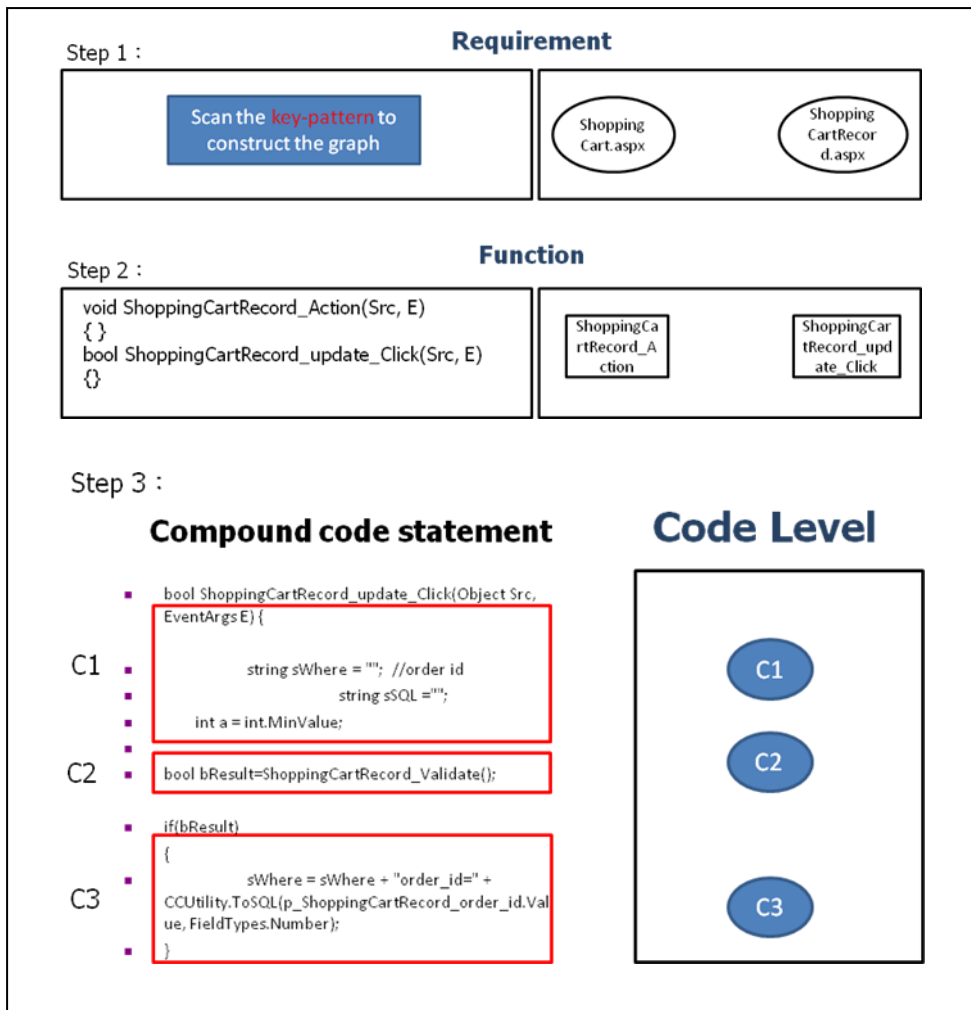


Figure 5. Dependence graph construction example: Node Detection

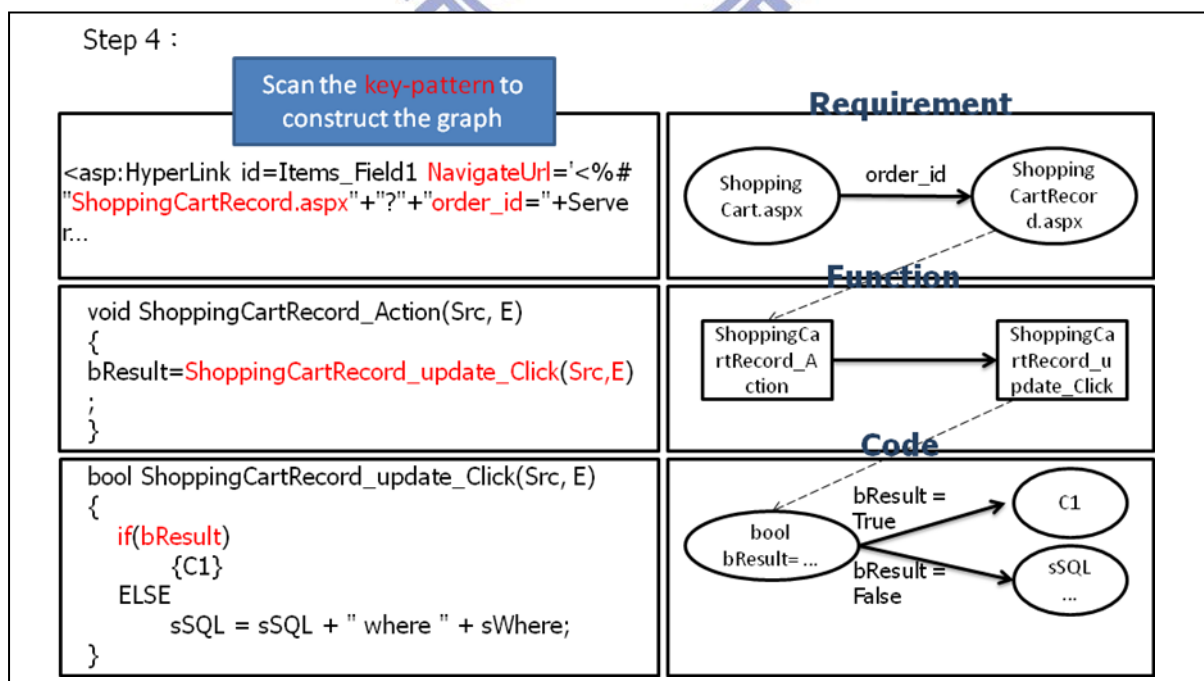


Figure 6. Dependence graph construction example: Edge Construction

### 4.3.2 Construction Phase Algorithm

Algorithm I shows the constructing steps of dependence graphs. First, it will detect all of the nodes which exist in the graph, and then construct edges by each node like the example in chapter 4.3.1. The function “SlicePageToFunction” will slice page into functions which contains in that page, and the function “SliceFunctionToCompoundStatement” will slice function into compound statements which belong to the function. The subroutine “EdgeConstruction” will construct the edges based on the property of each pattern in the nodes.



### Algorithm I. Three-Level Program Dependence Graph Construction Algorithm

Input:

Page code blocks  $\{B_P^1, \dots, B_P^n\}$

Pattern set of declare function  $P_F$

Pattern set of declare code  $P_C$

$T_P$  where  $T_P = P_R \cup P_F \cup P_C$

Output : Dependence graph ,  $G=(V, E)$  where

$V = V_R \cup V_F \cup V_C, E = E_R \cup E_F \cup E_C \cup E_{RF} \cup E_{FC}$

Method:

initial:

Step1: Add  $B_{Pi}$  into  $V_R$

Step2: For each  $B_{Pi}$  in  $V_R$

2.1: let  $F_i = B_{Pi}$  ,  $F = \{F_1, \dots, F_n\}$

2.2:  $F' = \text{SlicePageToFunction}(F, P_F)$

2.3: IF  $|F'| \neq |F|$  THEN Return  $F'$  to 2.2

2.4: Add  $F'$  into  $V_F$  , Add CNET into  $E_F$

Step3: for each  $B_{Fi}$  in  $V_F$   $C_i = (L_1, \dots, L_n)$

3.1: let  $C_i = B_{Fi}$  where

3.2:  $C' = \text{SliceFunctionToCompoundStatement}(C, P_C)$

3.3: IF  $|C'| \neq |C|$  THEN Return  $C'$  to 3.2

3.4: Add  $C'$  into  $V_C$

Step4: for each  $B_{Fi}$  in  $V_F$

4.1: EdgeConstruction ( $B_{Fi}$  ,  $P$ )

Step5: for each  $B_{Pi}$  in  $V_R$

5.1: EdgeConstruction ( $B_{Pi}$  ,  $P$ )

Step6: for each  $B_{Ri}$  in Web application

6.1: EdgeConstruction ( $B_{Ri}$  ,  $P$ )

### Subroutine of Algorithm I: SlicePageToFunction

Input: Code block B ,  $P_F$

Output: Code block B'

Method:

Step1: According to the  $P_F$  find the sub code block starting line of B.

Step2: According to the  $P_F$  find the sub code block ending line of B.

Step3: Add sub code block to B'.

Step4: Return B'.

### Subroutine of Algorithm I: SliceFunctionToCompoundStatement

Input: Code block B ,  $P_C$

Output: Code block B'

Method:

Step1: According to the  $P_C$  find the sub code block starting line of B.

Step2: According to the  $P_C$  find the sub code block ending line of B.

Step3: Add sub code block to B'.

Step4: Return B'.

### Subroutine of Algorithm I: EdgeConstruction

Input: Code block B , P, where  $P = P_R \cup P_F \cup P_C$

Output: Dependence graph of each level.

Method:

Step1: According to the different pattern in P, construct dependence edge of nodes.

Step2: Return Dependence graph of each level.

## 4.4. Testing Phase

### 4.4.1 Collaborative Testing Example

Figure 7 presents a collaborative testing example. We can split a dependence graph into testing blocks as shown below.

First, we pick up a minimal value of support/support-threshold which stands for the testing node that needed to be tested. In this example, the testing node is page “MemberInfo”. Then system will decide the testing block 3, node “MemberInfo”, is the most needed testing block. Based on computing result of our algorithm, the page “AdminMenu” will be assigned to tester A based on dependence graphs.

Finally, we collect the user session log and trustworthy weight of tester. The system will change the states of nodes by analyzing user session logs. In our example the pages “AdminMenu”, “MemberGrid”, “MemberInfo”, and “MemberRecord” are contained in the user session log of tester A. And these pages will change their support value according the trustworthy value of tester A. The support value of page “AdminMenu” will increase by one, same as remain pages. We adjust the support values for each page iteratively. Pages will confirm if the support and support threshold is equal to or bigger than the support threshold.

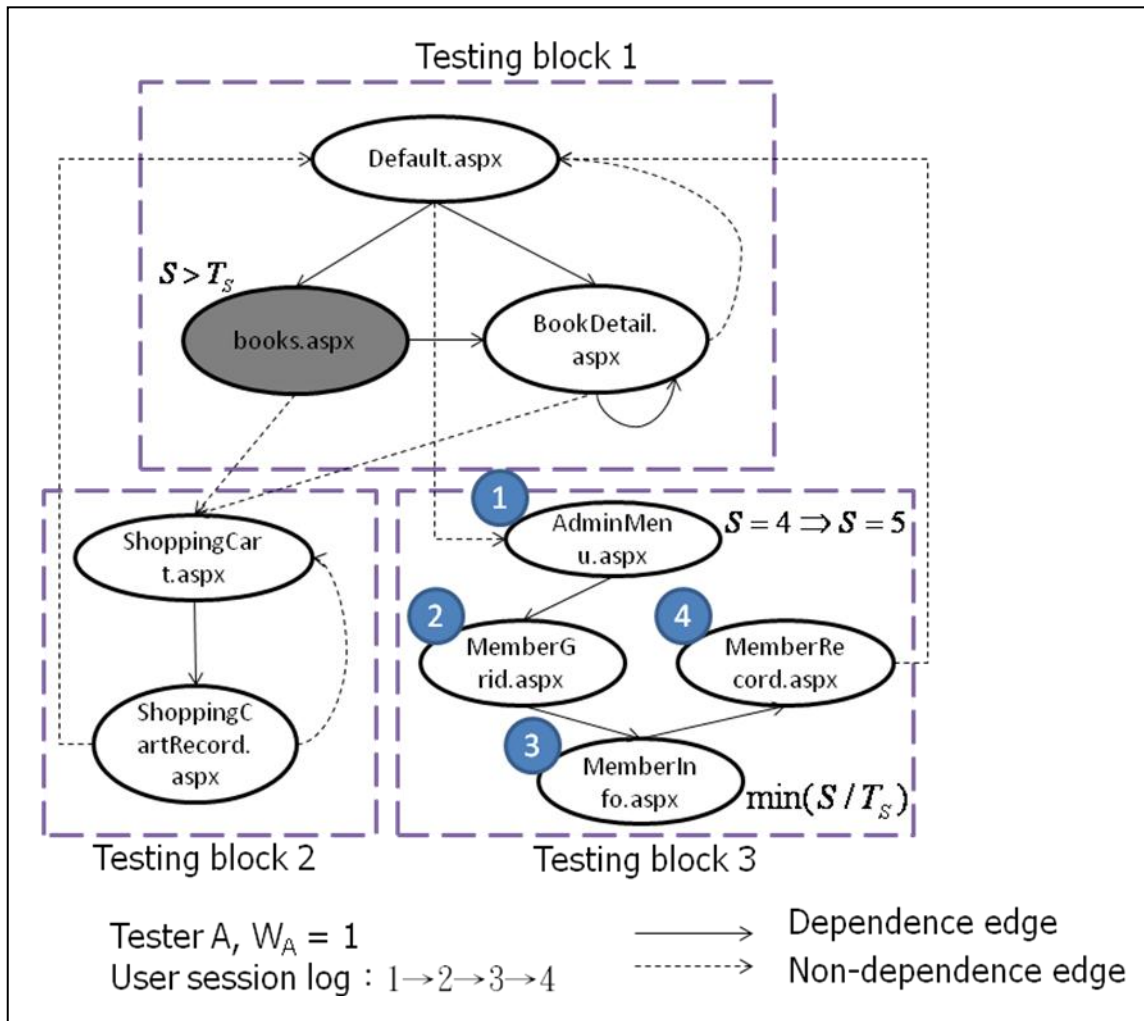


Figure 7. Collaborative testing example

## 4.4.2 Collaborative Testing Algorithm

In this section, we introduce our proposed collaborative testing algorithm based on the idea in Chapter 3.3. First, we define the notations according to the problem definition which considering the attributes in testing. The notation  $U$  means the set of folksonomy tester, and the notation  $L_u$  means the user session log, a user session log is a collect set of pages access logs and user id. The notation  $W_i$  means the trustworthy weight of user,  $L_u$ . The notation  $F_i$  means the fail ratio of node  $i$ , and the formulation is shown below. The notation  $T_F$  stands for the fail ratio threshold of node, the formulation  $F_i \geq R_F^i$  means that the node  $i$  is unreliable and need to be tested. We used the notation  $M_i$  to represent the complexity that based on line of code. The notation  $S_i$  means the testing quantity of node  $i$ , and the notation  $T_S^i$  stands for the support threshold of node  $i$ . When  $S_i \geq T_S^i$ , the node  $i$  is a confirmed node and it needs no more test.

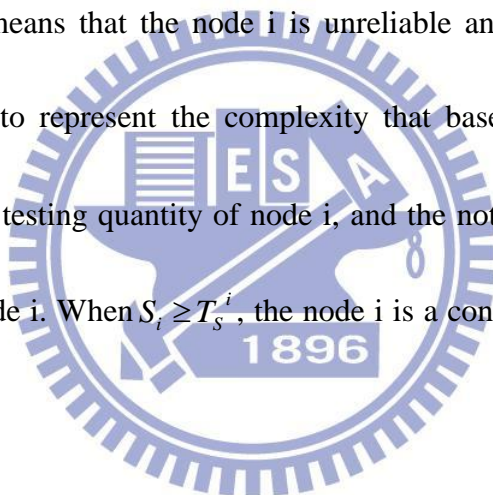
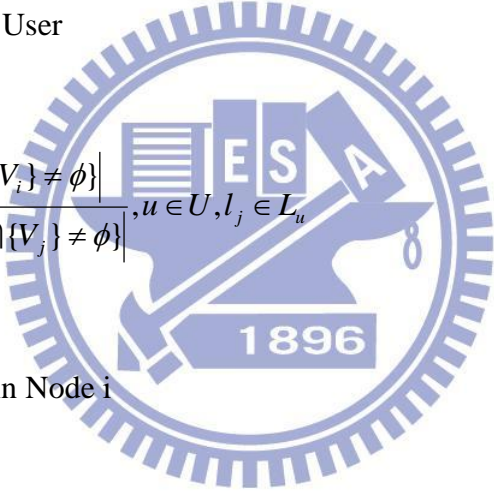


Table 7. Notations of Folksonomy-Based Testing Algorithm

<p>Folksonomy user:</p> <p><math>U = \{u_i   u_i \text{ is a folksonomy user}\}</math> is a set of user</p> <p>User session log of User:</p> <p><math>L_u = \{l_j   \forall j\}</math></p> <p><math>l_j = (P_j   F_j)</math> where <math>P_j = (V_1, \dots, V_K)</math></p> <p><math>F_j = \{V_i   V_i \text{ is reported as fail, } V_i \in V(P_j)\}</math></p> <p>Trustworthy weight of User:</p> <p><math>W_u \in [0,1]</math> ; initial by User</p> <p>Fail ratio of node i:</p> $F_i = \frac{\sum W_u  \{l_j   P_j \cap \{V_i\} \neq \phi\} }{\sum W_u  \{l_j   V(P_j) \cap \{V_i\} \neq \phi\} }, u \in U, l_j \in L_u$ <p>Complexity:</p> <p><math>M_i = \# \text{line of code in Node } i</math></p> <p>Support of Node i:</p> $S_i = \sum W_u  \{l_j   V(P_j) \cap \{V_i\} \neq \phi\} , \text{ where } u \in U, l_j \in L_u$ <p>Fail ratio threshold is <math>T_F</math></p> <p>Support threshold of Node i:</p> $T_S^i = c \times M_i$ <p>A node i is confirm IF <math>S_i &gt; T_S^i</math></p>	
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

The folksonomy-based testing algorithm is shown in the algorithm II. We try to confirm all nodes in dependence graphs to finish the test. In step 1, we initialed four



type nodes in the dependence graphs according. In step 2, for each node in the graph, we pick up a needed testing node by ranking the support value, and compute a proposed node in step 3 according our dependence graphs. We start guiding tester with the subroutine “GuidingTester” to find out the most suitable starting node and assign to tester. The starting node is computed by our proposed algorithm with dependence graphs. In step 4, the collecting user session logs are used to change the current testing status, page support threshold. Finally, we check the whole system state in dependence graphs. The testing is done if all nodes confirmed, or return to step 2.

#### Algorithm II. Folksonomy-Based Testing Algorithm

Input:

User Profile (personal trustworthiness)

Dependence Graph  $G=(V,E)$

Output:

Guided test component: the start node with inputs

Method:

initial:  $V' = \phi, E' = E$ , for each node  $N_i$  in  $V$  where  $V = V_R \cup V_F \cup V_C$ ,

InitialMetadata ( $N_i$ )

Step1:for each node in  $V'$

1.1: $N_i = \text{minimal}(S/TS)$  of node $i$

Step2:Guiding

2.1: GuidingTester ( $N_i$ )

Step3:Mapping

3.1:for each node  $n$  include in  $L_u$ ,  $n.S = n.S + W_u$

Step4:IF ( $S_i > T_s^i, \forall i$ ) THEN (Testing Finish) ELSE (return to step1)

### Subroutine of Algorithm II: InitialMetadata

Input:Code block B, P

Output:Code block B'

Method:

Step1:Set up the code block B' where X and Y is the TF and TS of B which compute by the code block complexity of B and the different pattern in P

B'.FailRatio  $\leftarrow$  0

B'.Support  $\leftarrow$  0

B'.TF  $\leftarrow$  X

B'.Ts  $\leftarrow$  Y

Step2:return B'

### Subroutine of Algorithm II: GuidingTester

Input:Node N'

Output:User Session Log L<sub>U</sub>

Method:

initial:starting page P = N' ; testing sequence stack S, push P to S

Step1:IF (exist d<sub>MP</sub>) AND (P isn't a Default page)

THEN (P = M) AND (push P to S) AND (Repeat Step1)

Step2:(assign S[top] to Tester) AND (pop S)

Step3:IF (S isn't Null)

THEN (guiding S[top] for tester) AND (pop S) AND (Repeat Step2)

Step4:According to the suggested page of tester, hiding other page link button of confirm page until testing finishes(user interface).

## Chapter 5. Implementation and Experiment

### 5.1 System Implementation

In this section, we implement the software system to support collaborative testing with our proposed methodology, so called collaborative testing system. We evaluate the performance of our approach by three experiments. The architecture of our collaborative testing system is shown in Figure 8. The system contains six main modules, “User Interface”, “Test Guiding Module”, “User Session Module”, “Bug Report System”, “Admin Module”, and “Graph Construction”, and three databases, “User Profile Database”, “User Session Log Database”, and “Dependence Graph Database”. The “User Interface” provides interact interface to participants. The module “Graph Construction” transforms web application into a corresponding dependence graph. The “Admin Module” is the administration functions provided to system administrator to monitor the testing, as shown in Figure 9. The “Test Guiding Module” provides a testing objective for testers. In Figure 10, we describe the testing flows for collaborative testing system. The module “Bug Report System” provides bug report function for tester. The “User Session Module” records user session log and change the testing state immediately.

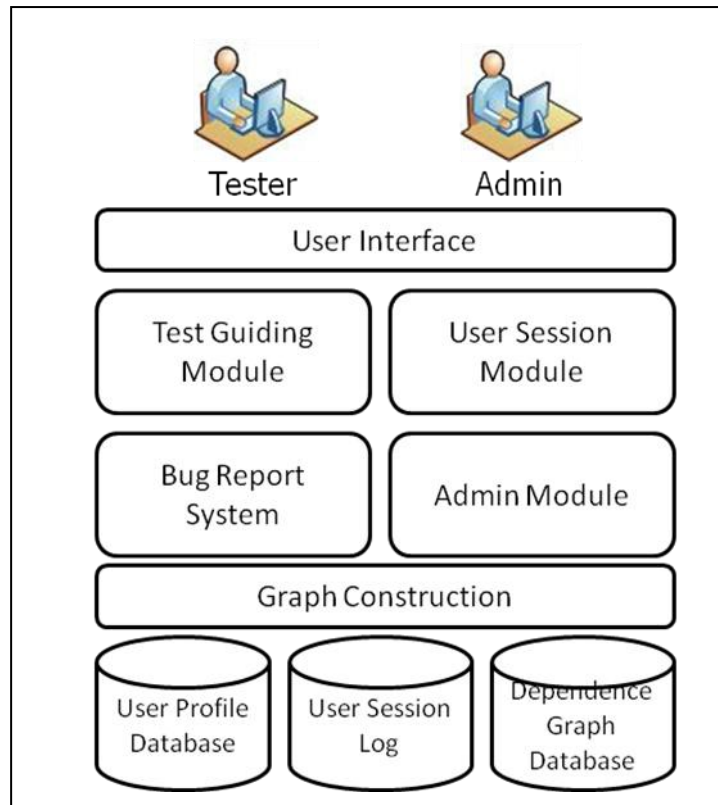


Figure 8. Collaborative testing system module architecture

The screenshot shows the 'Guiding Frame' administration window for 'Admin'. It displays a table of test results with columns for PageName, Support Ts, FailRatio, Tf, Confirm, Extend, FunctionName, and another set of Support Ts, FailRatio, Tf, Confirm, Extend. A red 'Extend' label with an arrow points to the 'Extend' button in the table.

PageName	Support Ts	FailRatio	Tf	Confirm	Extend	FunctionName	Support Ts	FailRatio	Tf	Confirm	Extend
AdvSearch	31	0	0%	0% Fail	Extend	Books	1	0	0%	0% Fail	Extend
BookDetail	0	0	0%	0%	Extend	ValidateNumeric	0	0	0%	0%	Extend
Books	279	0	0%	0% Fail	Extend	Page_Load	3	0	0%	0% Fail	Extend
CCPager	273	0	0%	0% Fail	Extend	Page_Unload	3	0	0%	0% Fail	Extend
Default	1356	0	0%	0% Fail	Extend	Page_Init	1	0	0%	0% Fail	Extend
Footer	353	0	0%	0% Fail	Extend	InitializeComponent	1	0	0%	0% Fail	Extend
Header	353	0	0%	0% Fail	Extend	Page_Show	1	0	0%	0% Fail	Extend
Login	105	0	0%	0% Fail	Extend	Results_Repeater_ItemDataBound	244	0	0%	0% Fail	Extend
MyInfo	0	0	0%	0%	Extend	Results_page_navigate_completed	1	0	0%	0% Fail	Extend
Registration	46	0	0%	0% Fail	Extend	Results_Bind	2	0	0%	0% Fail	Extend
ShoppingCart	150	0	0%	0% Fail	Extend	Search_Show	1	0	0%	0% Fail	Extend
ShoppingCartRecord0	0	0	0%	0%	Extend	Search_search_Click	0	0	0%	0%	Extend
						AdvMenu_Show	1	0	0%	0% Fail	Extend
						Total_Repeater_ItemDataBound	0	0	0%	0%	Extend
						Total_Open	19	0	0%	0% Fail	Extend
						Total_Bind	1	0	0%	0% Fail	Extend

Figure 9. Administration windows for User session logs



Figure 10. Testing flows of collaborative testing system

## 5.2 Experiments and Results

### 5.2.1 Experiment I - Dependence graph construction

In this experiment, we transform three web applications “BookStore” , “Classifieds”[<http://www.gotocode.com/>] and an e-learning platform called “OOLAYScience” into dependence graph respectively according to our proposed approach, and the statistic result is shown in Table 8 and Figure 11. According to these result, it is found out that the complexity of dependence relationships of Page-Level is much higher than Function-Level and Code-Level.

Table 8. Dependence graph statistic result

Application Name & Level	Type	Quantity
<b>BookStore</b>		
Page-Level	Node	25
	Edge	152
Function-Level	Node	352
	Edge	350
Code-Level	Node	836
	Edge	1409
<b>Classifieds</b>		
Page-Level	Node	16
	Edge	101
Function-Level	Node	253
	Edge	293
Code-Level	Node	541
	Edge	912

<b>OOLAYScience</b>		
Page-Level	Node	43
	Edge	284
Function-Level	Node	74
	Edge	174
Code-Level	Node	396
	Edge	732

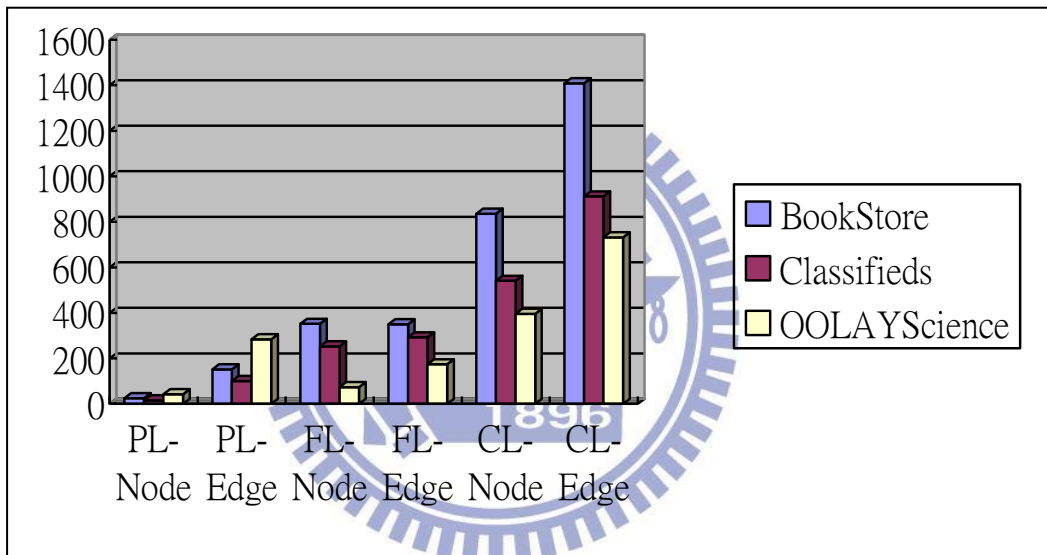


Figure 11. Dependence graph statistic result

## 5.2.2 Experiment II – Performance Evaluate Simulation

In general, folksonomy testing needs a large amount of human resource to perform test. In this paper, we implement a simulating system to evaluate the performance of our proposed approach. We propose three experiments with different parameter value to show the relationship between parameter and result. In our simulation each virtual testers has own trustworthy weight from 0.1 to 1, and accesses pages with different execution time. We simulate the testing activities of testers and collect testing session logs to perform our folksonomy-base approach. We guiding tester with our own method.

In the first experiment, we try to find out the relation between total completion time and tester quantity by using the same web application “BookStore”. Figure 12、Figure 13 and Figure 14 show the charts with different support threshold value with 10、20 and 30, that means a page called a confirm page if and only if every nodes in the Code-Level can be execute by 10、20 and 30 times. Observing these charts we can conclude some conclusions below.

- (1) The experimental results show that the evaluation of our approach is more efficient and well performance than traditional one. The average reduction rate for testing effort is almost 90%.
- (2) The more testers participate in testing, the shorter completion time we get. But the reduce rate also grow much slowly with more testers.



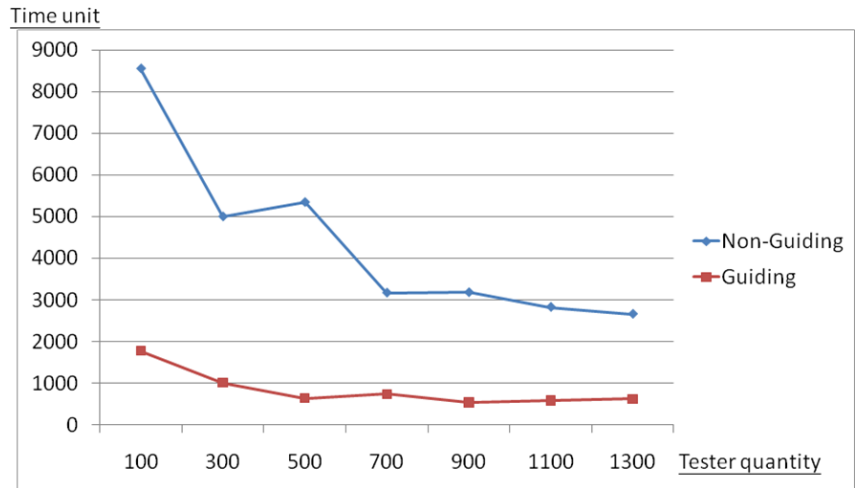


Figure 12. “BookStore” web application with 10 support threshold

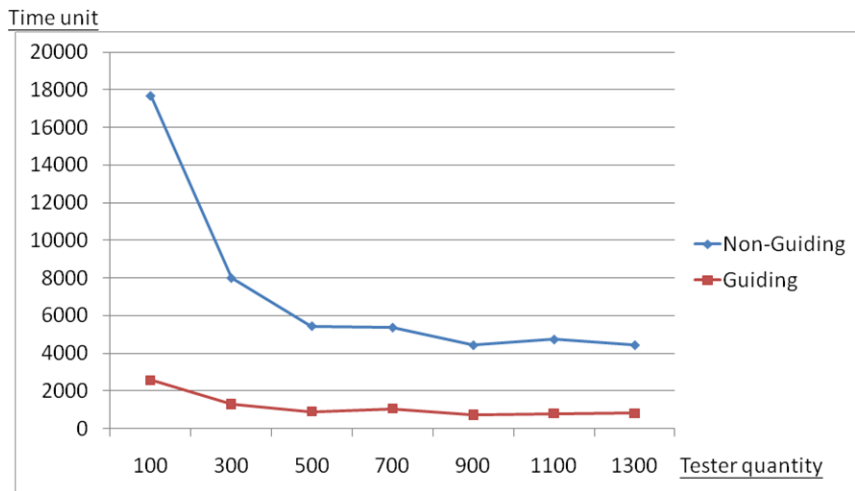


Figure 13. “BookStore” web application with 20 support threshold

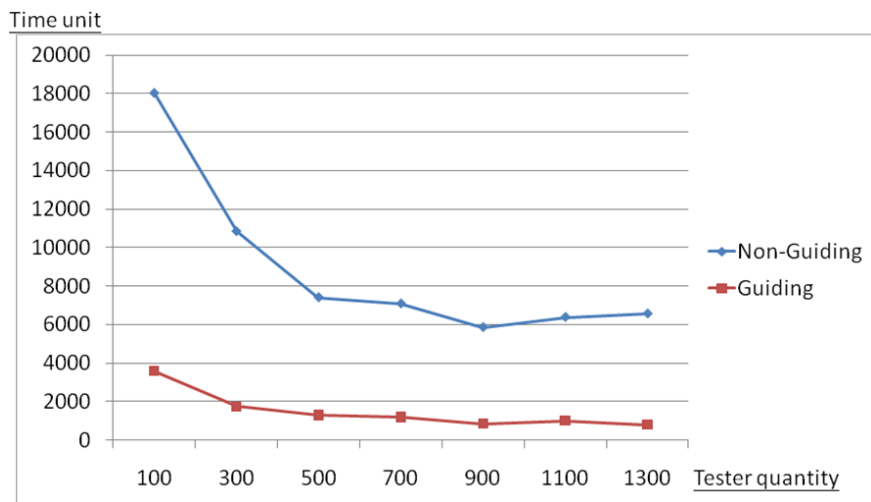


Figure 14. “BookStore” web application with 30 support threshold

In the second experiment, we try to find out the relation between total completion times with different support threshold by using the same web application “BookStore”. Figure 15 and Figure 16 show the charts with different testing approach, “Non-Guiding” and “Guiding” approach respectively. Observing these two charts we can conclude that total completion time is almost in direct ratio with different value of support threshold. The much higher threshold value, the higher completion time we get.

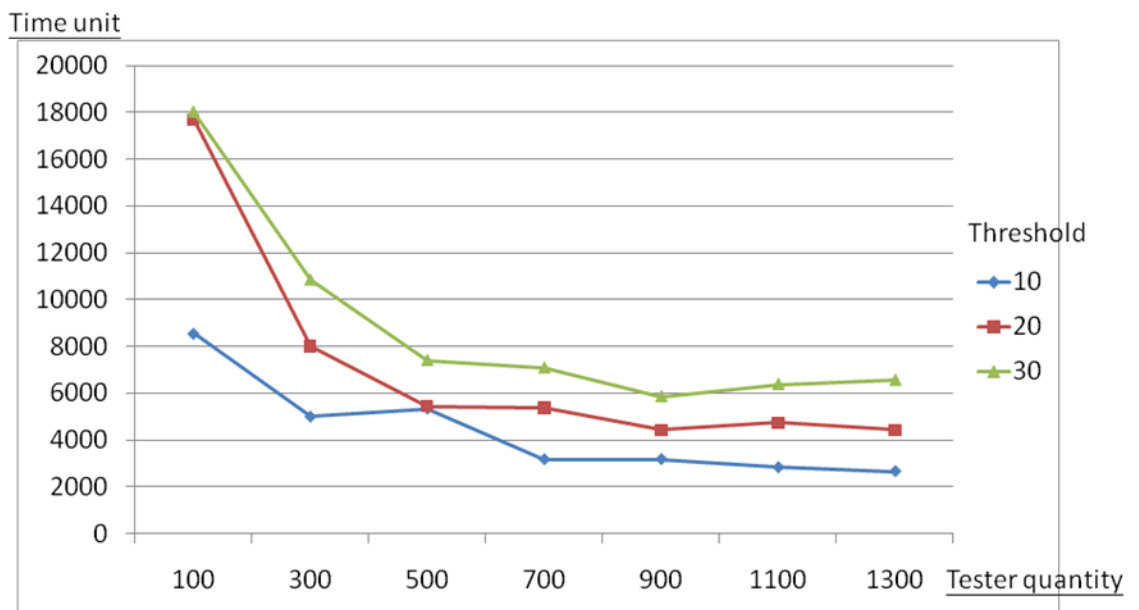


Figure 15. “BookStore” web application with Non-Guiding testing approach

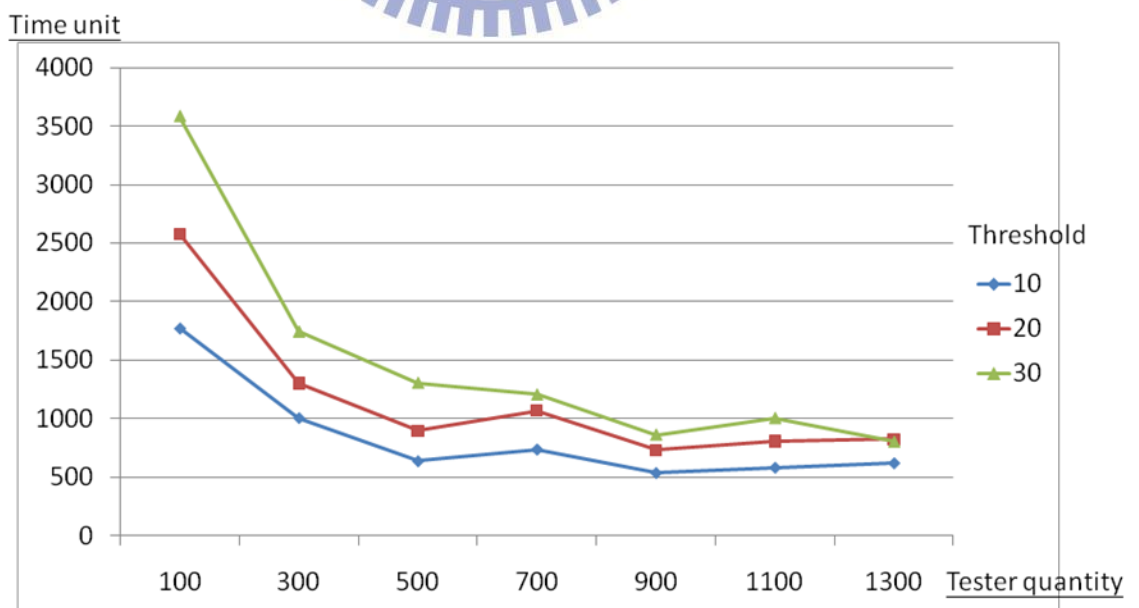


Figure 16. “BookStore” web application with Guiding testing approach

In the third experiment, we try to compare the total completion time with different web application execute by two kind of testing approach, and with the 100 testers and 10 support threshold show on Figure 17. According to Figure 17 we can find out that “OOLAYScience” web application has the biggest different between “Non-Guiding” and “Guiding” approach. We refer all these results to the complexity of web application. According to the result of “Experiment I - Dependence graph construction”, we can know “OOLAYScience” web application has the maximum of nodes in the Page-Level. Because of this property, some pages existed in the “OOLAYScience” web application will much harder to execute by tester because the execution path which include those pages is too long to execute from beginning to end. Therefore, the effect of “Guiding” approach will much obvious also.

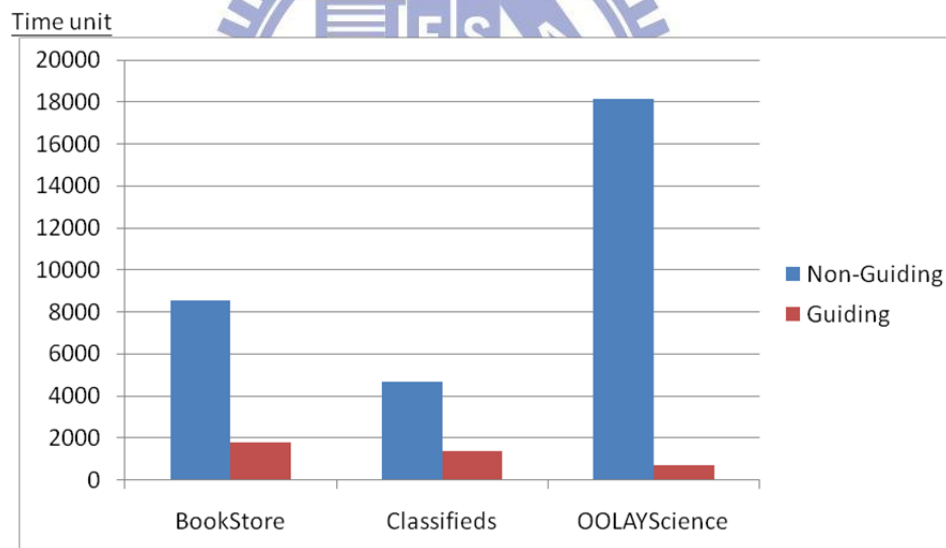


Figure 17. Costing time comparing with different web application

### 5.2.3 Experiment III – Performance Evaluation for Folksonomy-based Collaborative Testing

In this experiment, we create a real testing environment of “BookStore” web application provide for folksonomy testers which existed on the social network “Facebook” to participate our testing. This experiment cans division into two different testing approaches, “Non-Guiding” and “Guiding” respectively, but same as other conditions, and the value of support threshold in this experiment is 10. Figure 10 shows the testing flows of collaborative testing system with “Guiding” approach. The difference between “Non-Guiding” and “Guiding” is the guiding hint and the bug hint suggestions show on Figure 10, “Non-Guiding” approach non-exist these suggestions.

In this experiment, we hide three bugs in the “BookStore” web application, vote image error \ registration E-mail error and modify shopping quantity error. Figure 18 shows the information about this experiment, more testers will report more bug reports, but still existed much garbage bug reports, and in this experiment, “Non-Guiding” and “Guiding” approach has the same ability in the bug detection, “Non-Guiding” approach find out image error and modify shopping quantity error, “Guiding” approach find out image error and registration E-mail error.

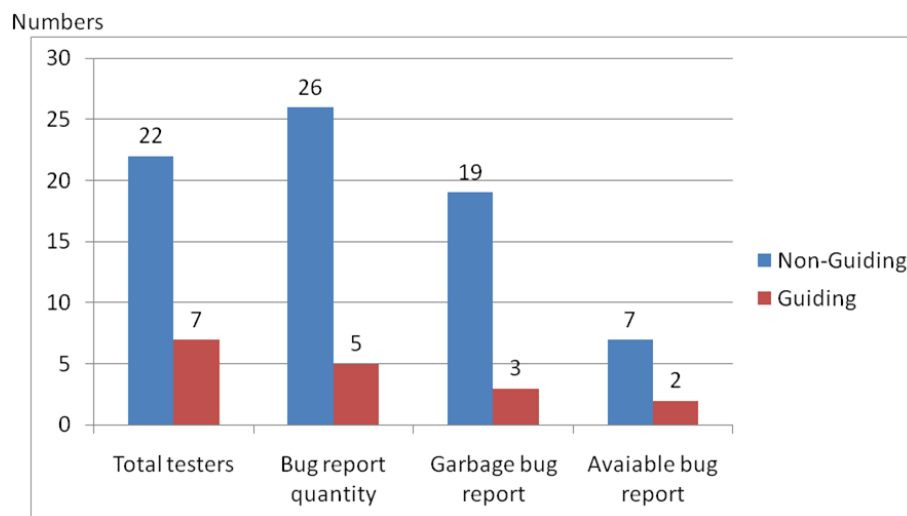


Figure 18. Statistics of tester quantity and bug report about two kind of approach

Figure 19 shows the costing time of all testers spend on each page, to observe this result of experiment we can understand “AdvSearch” and ”MyInfo” is the bottleneck in testing.

Figure 20 shows the total completion time comparing of two different guiding approaches, and the reduction rate for testing effort is almost 90% also.

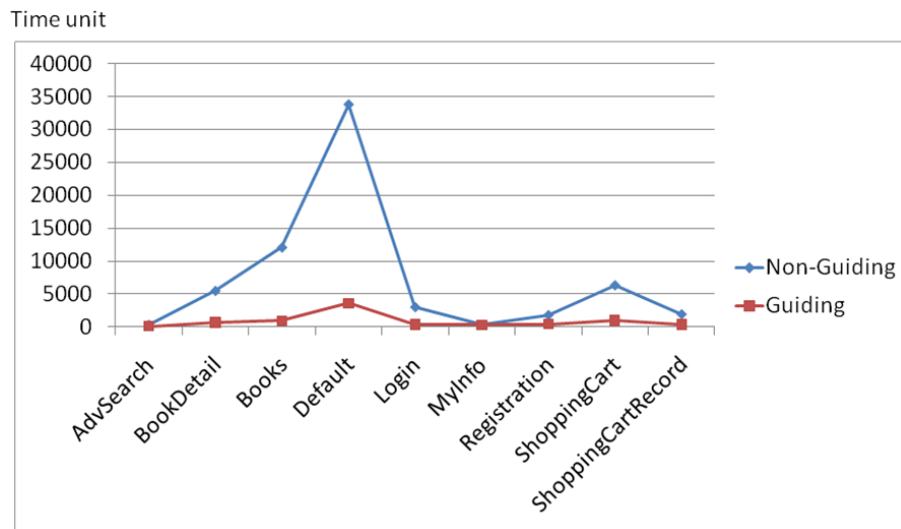


Figure 19. Costing time of all testers spend on each page

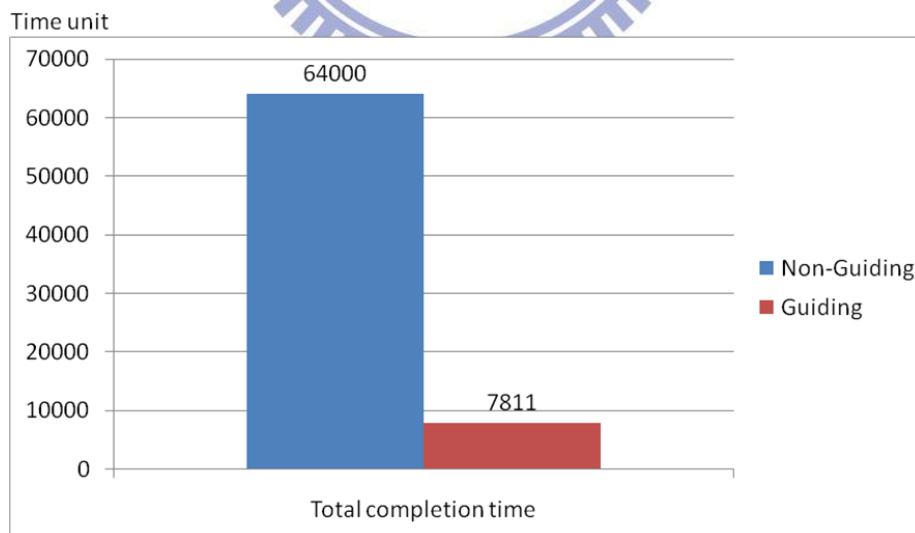
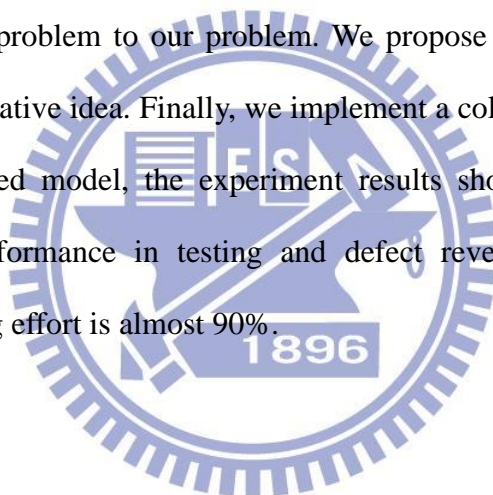


Figure 20. Total completion time comparing

## Chapter 6. Conclusion

In this thesis, we propose our folksonomy-based approach to support collaborative testing for Web application. In construction phase, we define the 3-level dependence graphs to represent the structures of web application. The different dependence graph means different usability testing, page-level testing、function-level testing and code-level testing. According to the dependence graphs, we can realize the structures of web application and help us assign jobs in testing. According to the real testing situation, we model the job assignment problem with a mathematics formulation, and prove that the problem is an NP-Complete problem. By reducing the “Optimum representative set”[39] problem to our problem. We propose our approach to support testing with the collaborative idea. Finally, we implement a collaborative testing system to evaluate our proposed model, the experiment results show that our approach is effective and well performance in testing and defect revealing. And the average reduction rate for testing effort is almost 90%.



## References

- [1] A. A. Andrews, J. Offutt, and R. T. Alexander, "Testing Web applications by modeling with FSMs", *Software Systems and Modeling*, Jul. 2005, pp. 326-345.
- [2] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in folksonomies: Search and ranking. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, p.411–426, Heidelberg, June 2006. Springer.
- [3] B. Boehm and A. Egyed, "Software Requirements Negotiation: Some Lessons Learned," in the 20th International Conference on Software Engineering(ICSE'98), Kyoto, Japan, 1998, pp.503-507.
- [4] Ben Lund, Tony Hammond, Martin Flack, and Timo Hannay. Social Bookmarking Tools (II): A Case Study - Connotea. *D-Lib Magazine*, April 2005.
- [5] B. S. Lerner, L. J. Osterweil, Stanley M. Sutton Jr., and A. Wise, "Programming Process Coordination in Little-JIL Toward the Harmonious Functioning of Parts for Effective Results," in *European Workshop on Software Process Technology*, 1998.
- [6] Briand, L.C., Wust, J., Daly, J.W. and Porter, D.V. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. *The Journal of Systems and Software*, 51, pp. 245-273, 2000.
- [7] C. Fu, B. Ryder, A. Milanova, and D. Wonnacott, "Testing of Java Web Services for Robustness," *Proc. ACM SIGSOFT Int'l Symp. Software Testing and Analysis*, pp. 23-34, 2004.
- [8] C.-H. Liu, D.C. Kung, and P. Hsia, "Object-Based Data Flow Testing of Web Applications," *Proc. First Asia-Pacific Conf. Quality Software*, pp. 7-16, 2000.
- [9] Christoph Schmitz, Andreas Hotho, Robert Jäschke, and Gerd Stumme. Mining association rules in folksonomies. In V. Batagelj, H.-H. Bock, A. Ferligoj, and A. Ziberna, editors, *Data Science and Classification: Proc. of the 10th IFCS Conf.*, *Studies in Classification, Data Analysis, and Knowledge Organization*, pp. 261–270, Berlin, Heidelberg, 2006. Springer.

- [10] Chidamber, S.R. and Kemerer, C.F. A Metrics Suite for Object-Oriented Design. IEEE Trans. on Soft. Eng., pp. 476-493, 1994.
- [11] D. Benz, K. Tso, and L. Schmidt-Thieme. Automatic bookmark classification: A collaborative approach. In Proceedings of the Second Workshop on Innovations in Web Infrastructure (IWI 2006), Edinburgh, Scotland, 2006.
- [12] D. C. Kung, C. H. Liu, P. Hsia, "An object-oriented Web test model for testing Web applications", in Proceedings of IEEE 24<sup>th</sup> Annual International Computer Software and Applications Conference, Taipei, Taiwan, Oct. 2000, pp. 537-542.
- [13] D. Jeffrey and N. Gupta, "Test Suite Reduction with Selective Redundancy," Proc. 21st IEEE Int'l Conf. Software Maintenance, pp. 549-558, 2005.
- [14] D. Leon, W. Masri, and A. Podgurski, "An Empirical Evaluation of Test Case Filtering Techniques Based on Exercising Complex Information Flows," Proc. 27th Int'l Conf. Software Eng., pp. 412-421, 2005.
- [15] D. Leon and A. Podgurski, "A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases," Proc. 14th Int'l Symp. Software Reliability Eng., pp. 442-453, 2003.
- [16] E. J. Whitehead, Jr. and Y. Y. Goland, "WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web," in 6th European Conference on Computer Supported Cooperative Work (ECSCW'99), Copenhagen, Denmark, 1999, pp. 291-310.
- [17] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications", In Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada, 2001, pp.25-34.
- [18] G. A. D. Lucca and A. R. Fasolino, "Testing Web-based Applications: The State of the Art and Future Trends", Information and Software Technology, 2006(48): 1172-1186.
- [19] G. A. Bolcer and R. N. Taylor, "Endeavors: a Process System Integration Infrastructure," in 4th International Conference on the Software Process (ICSP'96), Brighton, UK, 1996, pp. 76-89.
- [20] G. Di Lucca, A. Fasolino, F. Faralli, and U.D. Carlini, "Testing Web Applications,"



- Proc. 18th IEEE Int'l Conf. Software Maintenance, pp. 310-319, 2002.
- [21] Gilad Mishne. Autotag: a collaborative approach to automated tag assignment for weblog posts. In WWW '06: Proceedings of the 15th international conference on World Wide Web, pages 953–954, New York, NY, USA, 2006. ACM Press.
- [22] Hutchens, D.H. and Basili, V.R. System Structure Analysis: Clustering with Data Bindings. IEEE Trans. on Soft. Eng., 11, pp. 749-757,1985.
- [23] H. Halpin, V. Robu, and H. Shepard. The dynamics and semantics of collaborative tagging. In Proceedings of the 1st Semantic Authoring and Annotation Workshop (SAAW'06), 2006.
- [24] H. Miao, Z. Qian, and B. Song, “Towards automatically generating test paths for Web application Testing,” International Symposium on Theoretical Aspects of Software Engineering, 2nd IFIP/IEEE, 2008.
- [25] Horwitz, S., Reps, T., and Binkley, D. Interprocedural slicing using dependence graphs. ACM Trans. on Programming Languages and Systems, 22, pp. 26-60, 1990.
- [26] J. A. Jones and M. J. Harrold, “Test Suite Reduction and Prioritization for Modified Condition/Decision Coverage,” IEEE Trans. Software Eng., vol. 29, no. 3, pp. 195-209, Mar. 2003.
- [27] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. Journal of the ACM, 46(5):604–632, 1999.
- [28] J. Offutt, J. Pan, and J. Voas, “Procedures for Reducing the Size of Coverage-Based Test Sets,” Proc. 12th Int'l Conf. Testing Computer Software, pp. 111-123, 1995.
- [29] J. Offutt and W. Xu, “Generating Test Cases for Web Services Using Data Perturbation,” Proc. Workshop Testing, Analysis, and Verification of Web Services, 2004.
- [30] J. Sant, A. Souter, and L. Greenwald, “An Exploration of Statistical Models of Automated Test Case Generation,” Proc. Third Int'l Workshop Dynamic Analysis, May 2005.
- [31] J. Whitehead, “Collaboration in Software Engineering: A Roadmap,” Future of Software Engineering (FOSE' 07), 2007.

- [32] L. Dusseault, WebDAV: Next-Generation Collaborative Web Authoring, Prentice Hall PTR, 2003.
- [33] L. Wakeman and J. Jowett, PCTE: The Standard for Open Repositories: Prentice Hall, 1993.
- [34] M. Benedikt, J. Freire, and P. Godefroid, "VeriWeb:Automatically Testing Dynamic Web Sites", In Proceedings of 11th International World Wide Web Conference, Honolulu, HI, USA, May 2002, pp. 654-668.
- [35] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. In Proc. of the 15th International WWW Conference, Edinburgh, Scotland, 2006.
- [36] M. Harder, J. Mellen, and M.D. Ernst, "Improving Test Suites via Operational Abstraction," Proc. 25th Int'l Conf. Software Eng., pp. 60-71, 2003.
- [37] M. J. Harrold, R. Gupta, and M. L. Soffa, "A Methodology for Controlling the Size of a Test Suite," ACM Trans. Software Eng. and Methodology, vol. 2, no. 3, pp. 270-285, July 1993.
- [38] M. Wang, J. Yuan, H. Miao, and G. Tan, "A static analysis approach for automatic generating test cases for Web applications," International Conference on Computer Science and Software Engineering, 2008.
- [39] M.R. Garey , D.S. Johnson, In: V. Klee (Ed.), Computers and intractability, a guide to the theory of NP-completeness, Freeman, New York, 1979.
- [40] Peter Mika. Ontologies Are Us: A Unified Model of Social Networks and Semantics. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, ISWC 2005, volume 3729 of LNCS, pp.22–536, Berlin Heidelberg, November 2005. Springer-Verlag.
- [41] S. M. Master and A. Memon, "Call Stack Coverage for Test Suite Reduction," Proc. 21st IEEE Int'l Conf. Software Maintenance, pp. 539-548, 2005.
- [42] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher II, "Leveraging User Session Data to Support Web Application Testing," IEEE Trans. Software Eng., Vol. 31, No. 3, pp. 187-202, Mar. 2005.

- [43] Selby, R.W. and Basili, V.R. Analyzing Error-Prone System Structure. IEEE Trans. on Soft. Eng., pp. 141-152, 1991.
- [44] Stevens, W.P., Myers, G.J. and Constantine, L.L. Structure Design. IBM Systems Journal, 13, pp. 231-256, 1974.
- [45] T. Y. Chen and M. F. Lau, "Dividing Strategies for the Optimization of a Test Suite," Information Processing Letters, vol. 60, no. 3, pp. 135-141, Mar. 1996.
- [46] Tony Hammond, Timo Hannay, Ben Lund, and Joanna Scott. Social Bookmarking Tools (I): A General Review. D-Lib Magazine, April 2005.
- [47] William G. J. Halfond and Alessandro Orso, "Improving test case generation for Web applications using automated interface discovery", ESEC/FSE'07 Sep., 2007, pp. 145-154.
- [48] X. Jia and H. Liu, "Rigorous and automatic testing of Web applications", In 6<sup>th</sup> IASTED International Conference on Software Engineering and Applications, Nov. 2002, pp. 280-285.
- [49] Y. Deng, P. Frankl, and J. Wang, "Testing Web Database Applications," SIGSOFT Software Eng. Notes, Vol. 29, No. 5, pp. 1-10, 2004.
- [50] Zhichen Xu, Yun Fu, Jianchang Mao, and Difu Su. Towards the semantic web: Collaborative tag suggestions. In Proceedings of the Collaborative