# iTest: Testing Software with Mobile Crowdsourcing

Minzhi Yan, Hailong Sun, Xudong Liu
School of Computer Science and Engineering
Beihang University, Beijing, China
{yanmz,sunhl,liuxd}@act.buaa.edu.cn

## ABSTRACT

In recent years, a lot of crowdsourcing systems have emerged and lead to many successful crowdsourcing systems like Wikipedia, Amazon Mechanical Turk and Waze. In the field of software engineering, crowdtesting has acquired increased interest and adoption, especially among personal developers and smaller companies. In this paper, we present iTest which combines mobile crowdsourcing and software testing together to support the testing of mobile application and web services. iTest is a framework for software developers to submit their software and conveniently get the test results from the crowd testers. Firstly, we analyze the key problems need to be solved in a mobile crowdtesting platform; Secondly, we present the architecture of iTest framework; Thirdly, we introduce the workflow of testing web service in iTest and propose an algorithm for solving the tester selection problem mentioned in Section 2; Then the development kit to support testing mobile application is explained; Finally, we perform two experiments to illustrate that both the way to access network and tester's location influence the performance of web service.

## Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/Machine Systems; D.2.5 [**Software Engineering**]: Testing and Debugging

## General Terms

Human Factors

## Keywords

Software testing, mobile crowdsourcing, web service, mobile application

## 1. INTRODUCTION

Crowdsourcing[1][2] has emerged in recent years and draws lots of attention. Wike-pedia define crowdsourcing as: "the process of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, and especially from an online community, rather than from traditional employees or suppliers"[3]. Notable examples of this novel model include Amazon Mechanical Turk[1], Threadless[2], InnoCentive[3], Wiki-pedia[4], Waze[5] and user-generated advertising contests. All of these systems are very successful which illustrates that crowdsoucing can take advantage of the wisdom of crowd with a low cost and get great outputs.

As crowdsourcing has meet great success in many fields, can we adopt this model into software engineering area? Actually, open source software has adopted crowdsourcing for software development for years. In the development process of open source software, software's design or blueprint is universally accessible via free license and anyone can provide improvements to it which helps make the software robust and stable. However, in the open source model, every participant is a professional programmer or an amateur one, which means you need certain knowledge to take part in. If we adopt crowdsourcing into software testing, the threshold is significantly lower and much more testers and their devices can be employed for testing. Crowdtesting enables software developers to get their products tested on all major platforms, devices, system configurations and country or region-specific aspects under real-world conditions. With crowdtesting, the developers can get test results on functional testing, security testing, load & performance testing, usability testing and localization testing with reduced costs in shorter time. For instance, Raphael Pham et al.[4] propose an approach utilizing the phenomenon of drive-by commits in social coding sites that capable users quickly and easily build test suits in others' projects and move on. Di Liu et al. explored crowdsourcing as an alternative way to conduct remote usability testing and found that some important usability problems can be identified via crowdsourced usability tests[5]. In brief, crowdsourcing is a great opportunity for making software testing cheaper, quicker and comprehensive.

Thanks to the improved, technological smartphone features in the recent years, including reliable GPS, very good cameras and high-speed network environment, mobile phone users can work on crowdsourcing tasks without any further difficulties. Thus, mobile crowdsourcing arises. Mobile



*CrowdSoft'14*, November 17, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3224-8/14/11...$15.00
http://dx.doi.org/10.1145/2666539.2666569

[1]https://www.mturk.com/mturk/welcome
[2]https://www.threadless.com/
[3]http://www.innocentive.com/
[4]http://www.wikipedia.org/
[5]https://www.waze.com/

crowdsourcing [6] is a term that describes crowdsourcing activities that are processed on smartphones or other mobile devices. Adding software testing tasks onto mobile crowdsourcing would make it more convenient for testers finishing crowdtesting tasks anywhere and anytime. With mobile crowdtesting, mobile applications can be crowdtested and the web services used by these mobile applications can also be tested under diverse hardware environment, network situation and locations.

For engaging more crowdtesters into software testing, especially mobile application and web service testing, with mobile phone, we present iTest: A mobile crowdtesting platform for software testing. With iTest, testers can easily take part in web service and mobile application testing tasks anywhere and anytime. Thus, more useful test results from diverse test environment are collected in a short time. Major contributions of this work are listed asl followsčž

(1) We identify the key problems in mobile crowdtesting platform, including incentive mechanism, tester selection, tester management and test result aggregation.

(2) We present a framework for mobile crowdtesting: iTest and introduce the functions of each component in it.

(3) We propose a greedy algorithm for solving the tester selection problem when crowdtesting web services which can significantly reduce test task number and then save test costs for service developers.

(4) We provide a preliminary development kit to log and snapshot the running process of mobile applications and gather the logs and snapshots to our iTest server.

The remainder of this paper is organized as follows. Section 2 provides analysis to the key problems of mobile crowdtesting. The framework of iTest is introduced in Section 3. Section 4 describes how we test web services in iTest and the tester selection algorithm. The tools for mobile application testing are introduced in Section 5. Then, Section 6 presents two experiments to evaluate the influence of network type and location of the test device. In Section 7, we provide the introduction to related work. Finally, we conclude this work and propose the direction for future work.

# 2. PROBLEM ANALYSIS

We are now in a world where almost everyone has one or more devices and the hardware-software combinations are almost countless. This means software developers have to make sure their applications perform correctly and look good on a host of different devices. Significant quality improvement on the application or service would be achieved when testing is done across a wide set of platforms, locations and approaches etc. Mobile crowdtesting is the best way to cover these needs and execute test tasks anytime and anywhere with network access. However, we still need to face the challenges mobile crowdsourcing brings.

To make a mobile crowdtesting platform work well, we believe that these problems should be properly solved: tester management, tester selection, incentive mechanism, and test result aggregation. The details of these problems are described in the next subsections.

## 2.1 Tester Management

Tester management in mobile crowdtesting platform, just like the user management in many other systems, of course need to manage the basic information of each tester. However, this is not its only responsibility. In a mobile crowdtesting platform, tester management subsystem should also store the historical information of when and where did a tester participate which test task, the quality of each test result submitted by a tester. These historical information can be used to analyze a tester's test habit like in which time interval he likes to take a test task, which kind of task he is interested in. The test habit can then be adopted for tester selection which is explained in the next subsection. How to get the test habit of a tester precisely is a challenge.

## 2.2 Tester Selection

The target of tester selection is to ensure that each test task is taken by the right tester who can complete it very well, meanwhile, reduce the number of total test tasks for saving cost. Selecting testers that mirror target user demographics is more crucial. If one application or service is designed for a specific group of people, a man who dose not belong to the group isn't the ideal tester. We want tester who will use the application like your end users. If the software under test is intended for more general consumption, testers that represent all types of consumer should be included. How to meet the test requirements while cost the least is also a great challenge in mobile crwodtesting.

## 2.3 Incentive Mechanism

In crowdsourcing systems, incentive mechanism is very important to excite the works participate in micro tasks. The incentive in these systems can be money, fun, socialize, earn prestige, altruism, learning something new, unintended by-product, create self-serving resource and the combination of them[7]. Among these incentives, money is the most popular way to reward the workers. If the money reward is too low then few testers would take the test task, but if the money reward is set too high, many unskilled tester would swarm into for the money and the quality can not be guaranteed. How to price a test task properly is a great challenge in a mobile crowdtesting platform.

## 2.4 Test Result Aggregation

The test results generated by all the testers for a test task would be submitted to the crowdtesting platform. In mobile crowdtesting platform , the test results are mostly textual logs and sometimes the screen snapshots of the application's exception. The test results from different testers could conflict, and it is difficult to solve this conflict automatically. How to aggregate the test results together without conflict and provide useful analysis for the application or service developer is also a challenge to a mobile crowdtesting platform.

Among all these problems we analyzed, the tester selection problem for reducing duplicate test tasks is solved in this paper with a greedy algorithm introduced in Section 4.2. The other problems could be our future research directions.

# 3. ITEST ARCHITECTURE

Based on the understanding to the key problems of mobile crowdtesting, we present a framework for mobile crowdtesting: iTest, and implement a prototype of iTest which supports web service and mobile application testing. Figure 1 shows the architecture of iTest which contains the iTestClient, iTestServer and the target web service and software on the internet. In iTestServer, there are two repositories

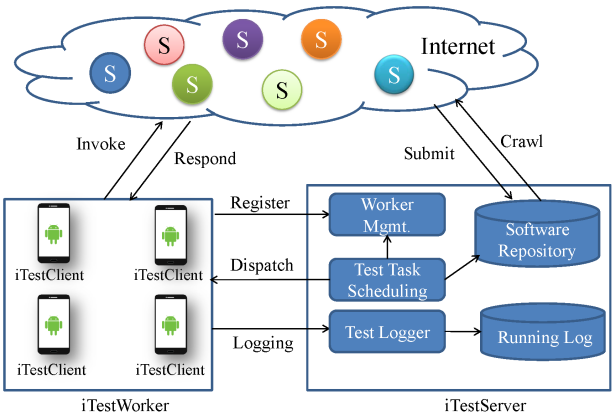


**Figure 1: The Framework of iTest**

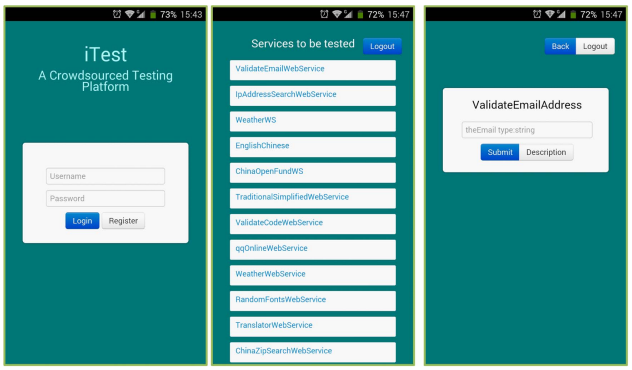


**Figure 2: Screenshots of iTestClient**


**Figure 3: Workflow of Testing Web Service**

which stores software information and the running log (or test results), respectively.

We have collected over 30,000 Web services. The WSDL addresses and corresponding information about these services are stored in the software repository in Figure 1. This repository maintains all the services and software needed to be tested. On the client side, we have developed an Android App based on PhoneGap(`http://phonegap.com/`) framework that provides support for developing mobile Apps with standard Web technologies. Currently, we only implement iTestClient on Android platform, as shown in Figure 2. Once iTestClient is installed on an Android mobile phone, the mobile phone will be registered to the server side as a mobile software testing volunteer called iTestWorker. The iTestServer is responsible for managing all the registered volunteers, scheduling testing tasks and maintaining the test results. When an iTestWorker goes online, it will notify iTestServer and the latter will respond with a list of Web services to be tested. Then the user can select a Web service to test. Meanwhile, the download links of mobile applications to be tested are also pushed to the iTestWorker. These applications are developed with our development kit (Please see Section 5 for detail) and the running log of them would be collected and transmitted to the running log repository in iTestServer.

# 4. WEB SERVICE TESTING

Considering that Web service invocation is highly related to invocation context like time and location, we choose to leverage mobile phones to collect the Service QoS data since the use of mobile phone is largely diversified in terms of geo-location, time and client environments. Invoking a Web service usually involves a lot of programming work to construct, send, receive and parse SOAP messages, therefore we must hide the programming complexity so as to let users easily fulfill the Web service testing task without knowing too much technical details.

In this section, we introduce how iTest works when testing a web service by describing the testing workflow and the algorithm for tester selection.

## 4.1 Testing Workflow

As seen in Figure 3, the testing workflow of web service is divided into 5 steps:

(1) Login & Probing: Firstly, the tester need to login with their username and password. While doing this, the time of communicating with iTestServer is probed by calculating the round trip time cost of the login request.

(2) Information Uploading: After probing the time cost of communicating with iTestServer, iTestClient uploads this time cost with other basic information of this tester and device, including username, location, network type etc.

(3) Tester Selection: In this step, iTestServer select proper testers from the online users for each web service based on the testers information and existing test results. The selection algorithm would be introduced in the next subsection.

(4) Invoking Web Service: After tester selection, the services for each tester are determined and pushed to the corresponding iTestClient. Then a tester can select a web service from the list, read its description and input the necessary parameters after the client fetches the WSDL file of the service and parses the need parameters, then invoke this web service.

(5) Test Results Uploading: The test result information, including username, current location of the test device, WSDL address of the web service under test, input parameters, invoking time, returned results and round trip time cost, are all submitted to iTestServer to store into the test result repository. Then these test results can be provided to the service developer to improve their web services, or used for service recommendation based on the QoS information in the results.

## 4.2 Tester Selection Algorithm

For avoiding to execute a web service test task under similar running environment repeatedly so that reducing the test cost. We need to design an efficient tester selection algorithm which only dispatch a web service test task to one tester out of those with similar running environment, including same network type and nearby location. Before stating our algorithm, we provide two definitions here:

DEFINITION 1. *We assume that two testers' positions on the map whose distance is smaller than a certain number, say R, would provide similar test results and we just need to*

*obtain one of the results. Then R is defined as the Effective Radius of a test result.*

R is an empirical value which could be adjusted in practice. Here we set it as 5 km.

DEFINITION 2. *For each online tester T whose location is P, other online testers whose distances to P are smaller than Effective Radius R are called the homogeneous points of P, and P is called the central point of all these points.*

The basic idea of our tester selection algorithm is to select the minimum number of *central points*, each of which stands for all its *homogeneous points*. And the union of these central points and their *homogeneous points* can exactly cover the set of online users' positions. It is clear that this problem equals to the typical NP-Complete problem: Set Cover Problem[8] which doesn't admit a polynomial time algorithm. Fortunately, we can find an efficient greedy algorithm, which always finds the optimal or an acceptable approximate solution, for solving this problem[9].

As shown in Algorithm 1, we present a greedy algorithm for tester selection (GAFTS). Firstly, we need to filter out the online testers which access Internet with the same network type (Line 15-19) . Secondly, the testers nearby the positions of existing test result to a specific service under the same network type should be removed from the candidates (Line 22-26). Then, every left candidates is grouped with its homogeneous points and thus we get |P'| subsets of all the candidates (Line 28-34). And then goes the key step, our algorithm greedily finds the biggest subset obtained in the last step, and select the central point(corresponding to a tester) as a tester for the current web service. If there are two candidate subsets while finding the biggest subset, the one whose central point performs better on probing time is selected as the biggest because . This step is repeated after removing the central point and its homogeneous points from the candidates until there is no candidate any more(Line 35- 49). Lastly, we can get the selected testers for each web service and dispatch these test tasks to the corresponding testers.

The complexity of Algorithm 1 is $O(MN^2)$ where $M$ is the number of services to be tested.

**Algorithm 1** GAFTS

```
 1: Input N, R, P[N], G[N], S[ ], E[ ], T[ ];
 2: /*
 3:  N: The number of online tester,
 4:  R: Effective Radius,
 5:  P[N]: Testers' positions,
 6:  G[N]: Testers' network types,
 7:  S[ ]: Services under test;
 8:  E[ ]: Existing test results;
 9:  T[ ]: Time cost on probing an online user's device;
10: */
11: D[][]=0; //D[s][i]=1 means service s should be pushed
    to tester i for test
12: G'[]=removeDupli(G); //Get all the network types in G

13: for all (network type t in G') do
14:   P'[]={null};
15:   for (i=0; i<N; i++) do
16:     if (G[i]==t) then
17:       P'[i]=P[i];
18:     end if
19:   end for
20:   for all (service s in S) do
21:     C[][]=0;
22:     for (i=0; i<N; i++) do
23:       if (isExist(P'[i], E, s, R)) then
24:         P'[i]=null;
25:       end if
26:     end for
27:     total=getValidCount(P');//number of total posi-
        tions in P'
28:     for (i=0; i<N; i++) do
29:       for (j=0; j<N; j++) do
30:         if (P'[i]!= null && P'[j]!= null &&
            isHomo(P'[i], P'[j] R)) then
31:           C[i][j]=1;
32:         end if
33:       end for
34:     end for
35:     while (total>0) do
36:       i=getPopularCentral(C, T);//find i with the
          biggest \sum_{j=0}^{N-1} C[i][j]
37:       D[s][i]= 1;
38:       total = total - 1;
39:       for (j=0; j<N; j++) do
40:         C[j][i]=0;
41:         if (C[i][j]==1) then
42:           total = total - 1;
43:           for (k=0; k<N; k++) do
44:             C[k][j]=0;
45:             C[j][k]=0;
46:           end for
47:         end if
48:       end for
49:     end while
50:   end for
51: end for
52: Output D;
```

## 5. MOBILE APPLICATION TESTING

Aside from supporting web service testing, now iTest also provide a preliminary development kit in the form of .jar file to support testing mobile Android application. This development kit mainly consists of a Log4j-based[10] logging tool and a visual debugging info collector.

### 5.1 Logging Tool

The logging tool is the Log4j library wrapped with our adapter for running on Android phones and communicating with iTestServer. We keep the original advantages of Log4j which is possible to enable logging at runtime without modifying the application binary. Just like using Log4j, logging behavior can be controlled by editing a configuration file, without touching the application binary. The basic information of the device, system running states, application name and the logs generated while running the application would be all stored in the local storage of Android phone and periodically submitted to the iTestServer.

Submitting logs to iTestServer as soon as they are generated would be a great communication burden. But submitting batched logs in a long time may lead to the loss

of logs because the application could crash anytime. Thus, it's an important problem to design efficient log submitting mechanism to make a tradeoff.

## 5.2 Visual Info Collector

Our visual debugging info collector is designed to support the Android application developers collecting the snapshots when fatal errors occur. It provides interfaces for developers for take snapshots and submit them to iTestServer. Like the former Logging tool, this collector firstly stores the snapshots of the application into the local storage of the device, and then upload them to iTestServer.

As a snapshot may be a large file which need long time network transmission, the transmission is likely to be interrupted because of the crush of application or the shutdown operation from the tester. For solving this problem, we can retransmit the interrupted snapshot when the application under test is launched again. The other problem is how to make the testers fully trust our collector and believe that we are not stealing their privacy. We also need to study strategies for preventing some unreal developers from doing illegal things.

# 6. EXPERIMENTS

Many web services perform diversely under different running environment. For example, invoking a service from different location and under different network situation and access approach usually gets various performance. In this section, we develop a demo web service with three replica which are deployed in Shanghai(East China), Qingdao( North China) and Hangzhou (South China), respectively. Then we design two experiments to evaluate the influence of network type and geographical location to the performance of our demo web service. The experiment results are shown below.

## 6.1 Network Type Influence on Service Performance

In this experiment, we employ 18 volunteer testers in Beijing whose mobile network type is HSPA, HSPA+, HSDPA, UMTS, EDGE and LTE (3 volunteers for each type). These volunteers are asked to install iTestClient and invoke our demo service replicas with the same input for 10 times both under Wifi and mobile network.

All the invoking results are submitted to iTestServer, thus we can obtain service replicas' performance (average response time) under different network access fashion as shown in Figure 4. It can be seen that under different network access fashion, the performance of each replica may change substantially. And the replica with the best performance differs, for example, under HSDPA replica 1 (Deployed in Shanghai) performs best, but replica 2 ( Deployed in Qingdao) wins out under EDGE. These performance information could be used for dispatching the service invoking requests, which come from different network type, to the proper replica for getting response quicker.

## 6.2 Location Influence on Service Performance

In this experiment, we ask the volunteers in Beijing, Xiamen, Xi'an and Hangzhou to invoke the three replicas of our demo service under Wifi network access fashion. Other experiment setups are just the same as those in the experiment above.

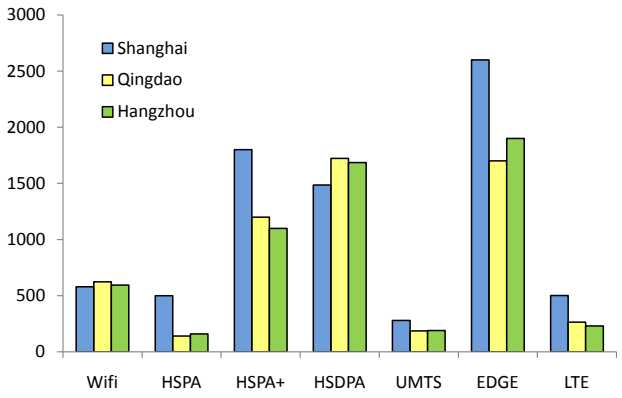


**Figure 4: The service replica performance under diverse network type**

From Figure 5, we can get similar conclusion with last experiment that the replica performs the best differs under different location. These information can also be used for recommending the perfect replica to invoke when a invoking request is launched from these cities.

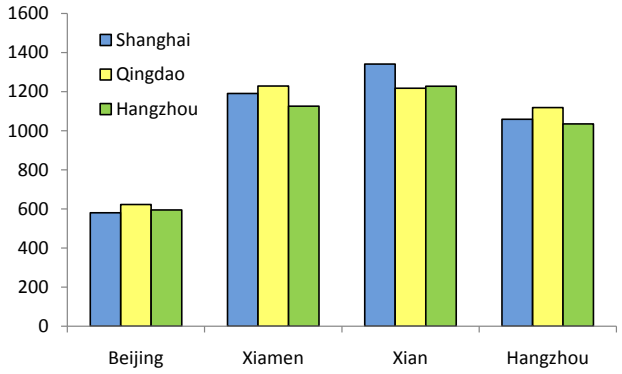


**Figure 5: The service replica performance under diverse location**

These two experiments illustrate that both the way to access network and tester's location influence the performance of web service. Thus, when testing a web service, it is very necessary to crowdtest it with diverse network type at different locations to get a full-scale evaluation.

# 7. RELATED WORK

The related works of iTest can be categorized into two types: Mobile Crowdsourcing and Crowdsourced Testing.

## 7.1 Mobile Crowdsourcing

The prevalent use of smartphones with powerful computing capability and various sensors provides a good opportunity to incorporate mobile computing into crowdsourcing, i.e. mobile crowdsourcing. For examples, Moboq[11] is a location-based real-time social question answering service, which sends a question to the selected Sina Weibo(a microblogging service in China:`http://www.weibo.com`) users who are around the place related to this question. Yao-Chung et al. propose schemes for effectively enabling mobile

participants to perform place name annotations and bridge the gap between Wi-Fi signals and human-defined places[12]. The common feature of these systems is to ask the crowd on site to help. iTest also make use of the position information of testers together with other test environment properties to get the on-site performance of web service or mobile application.

## 7.2 Crowdsourced Testing

Crowdsourced software testing has witnessed increased interest and adoption, meanwhile, many crowdsourced testing products have arose. uTest[13] is the world's largest marketplace for software testing services(`http://www.utest.com/`) with 100000+ QA professionals from more than 200 countries and territories. uTest supports the test for web sites, mobile applications and games. Pay4Bugs(`https://www.pay4bugs.com/`) is a self-serve product in which you can control testing from your account and test when you want on your schedule. There are also some other crowdtesting products, like Mob4Hire, Feedback Army, 99tests etc., which provide similar functions. The test results in these products are manually submitted. However, in iTest, all the test results or running logs are submitted automatically which can ease the burden of the testers and make the test process more efficient.

# 8. CONCLUSION AND FUTURE WORK

Web service invocation is highly related with invocation context like network access type and location, we choose to leverage mobile phones to collect the Service QoS data since the use of mobile phone is largely diversified in terms of geo-location, time and client environments like the way it access the Internet. Meanwhile, mobile application developers are confused by the problem of adapting their application to thousands of types of mobile devices. Thus, in this paper, we firstly present the mobile crowdtesting framework: iTest and introduce the functions of each component; then analyze the key problems in mobile crowdtesting platform, including incentive mechanism, tester selection, tester management and test result aggregation. After mapping our tester selection problem into the typical set cover problem, we design a greedy algorithm for tester selection when mobile crowdtesting web services which can significantly reduce test task number and then save test costs; We also provide a development kit to log and snapshot the running process of mobile applications and gather the log and snapshots to our iTest platform. The two experiment results in Section 6 illustrate the influence of geographical location and network type on web service performance.

Future work can leads to two directions. First, our iTest framework is limited to a small scale of use due to lack of appropriate incentives. We will study an effective incentive mechanisms for mobile crowdsourced testing. Second, we would consider detailed user contexts, including the way their devices connect to the internet and their geographic position, in computing the similarity of users, and recommend the service with the best QoS property based on historical test result of similar users.

# 9. ACKNOWLEDGMENT

This work was supported by China 863 program (No. 2012AA 011203), the Fundamental Research Funds for the Central Universities(No. YWF-14-JSJXY-004), A Foundation for the Author of National Excellent Doctoral Dissertation of PR China (No. 201159), and Beijing Nova Program(2011022) and the Program for New Century Excellent Talents in University.

# 10. REFERENCES

[1] A. J. Quinn and B. B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of CHI* 2011, 1403-1412.

[2] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the World-Wide Web. *Commun. ACM* 54, 4 (April 2011), 86-96.

[3] Crowdsourcing. `http://en.wikipedia.org/wiki/Crowdsourcing`

[4] R. Pham, L. Singer and Kurt Schneider. Building Test Suites in Social Coding Sites by Leveraging Drive-By Commits. *35th International Conference on Software Engineering*, pp. 1209-1212, May, 2013.

[5] D. Liu, M. Lease, R. Kuipers and R. Bias. Crowdsourcing for Usability Testing. March 2012. arXiv 1203.1468. `http://arxiv.org/abs/1203.1468`

[6] Mobile Crowdsoucing, `http://www.clickworker.com/en/crowdsourcing-glossar/mobile-crowdsourcing/`

[7] L. Guoliang. Crowdsourcing: Challenges and Opportunities. *Tutorial on HotDB 2012*. February, 2012.

[8] R. M. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*. New York: Plenum. pp. 85ÍC103. 1972.

[9] V. V. Vazirani. Approximation Algorithms. *Springer*: New York, 2001. ISBN 3-540-65367-8

[10] Apache Log4j Project. `http://logging.apache.org/log4j/2.x/`

[11] Y. Liu, T. Alexandrova and T. Nakajima. Using stranger as sensors: temporal and geo-sensitive question answering via social media. In Proceedings of the 22nd international conference on World Wide Web (WWW '13), pp. 803-814.

[12] Y. Fan, W. H. Lee, C. T. Iam, G. H. Syu, Indoor Place Name Annotations with Mobile Crowd, *2013 International Conference on Parallel and Distributed Systems* (ICPADS) pp.546-551, 15-18 Dec. 2013

[13] uTest. White Paper: Crowdsourced Usability Testing. `http://alexcrockett.com/wp-content/uploads/downloads/Books/Crowdsourced_Usability_Testing.pdf`