العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

Deployment of Software Engineering Tools as Cloud Services

Nawaf Almutairi
School of Computing Science, Newcastle University
n.m.m.al-mutairi@newcastle.ac.uk

Abstract. While much research has been carried out on the use of cloud computing, there has been little investigation of the cloud in software engineering. This study aims to demonstrate and evaluate software engineering tools in the cloud. In order to achieve this aim, technology in cloud computing and software engineering are reviewed and a workflow is created to facilitate the use of tools in the cloud for indirect and direct interaction approaches. Three differently sized case study scenarios are created to evaluate the use of the cloud in terms of cost, size and user efficiency. In addition, requirements of the service and users are explained and analysed. Consideration for the workflow design and interaction between users and the tools in the cloud are perceived as important aspects. This research also concerns the integration between the software engineering environment and the tools in the cloud and in particular how Scrum teams work with this integration. Evaluation of the results indicates that deployment of the cloud in the case study companies can be cost, time and user efficient.

Declaration: I declare that this dissertation represents my own work except where otherwise stated.

1. Introduction

Cloud computing has the potential to revolutionize computer technology, and because the cloud has several advantages, it has been used for many purposes in business and management [53]. Firstly, the cloud has qualities of elasticity and scalability, which refer to the ability to increase or decrease the number of resources automatically based on needs [4]. Moreover, it offers the ability to pay per unit of usage instead of buying, running and maintaining large servers, networks and storage [42]. In addition, developers can validate their programs through a number of standard mechanisms such as PDAs [4]. These benefits may attract developers to migrate their software engineering tools to cloud.

Providing an environment that is similar to a real environment needs a significant space for testing and verification to deal with the whole program. The results of verifications of large programs require servers with a large amount of space in order for these results to be displayed. This is clearly evident when trying to verify the small PROMELA code fragment (179 lines) (see Appendix A) in the Spin tool with a memory of 128 MB in the PC, it barely verifies, yielding the suggested action in Figure 1:

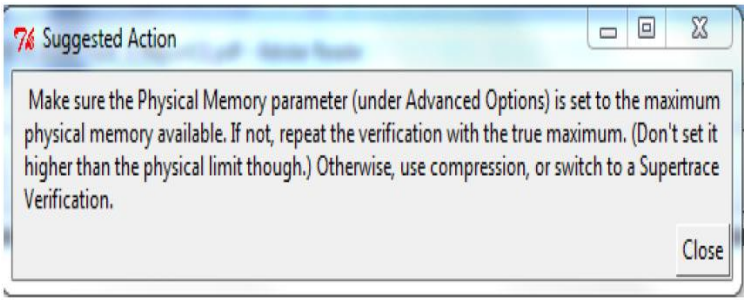

Fig. 1. Suggested action from Spin tool

The cloud can offer any infrastructure such as operating system [42]. Therefore, users can select the same infrastructure that would be found in the real environment. For example, some projects work in UNIX and the tester can select the same operating system in the cloud to test the projects.

The provision of big servers costs companies a significant amount of money, whereas cloud users only pay for what they use. These big servers also need maintenance and updates frequently. Nevertheless, the advantages of the cloud can handle all these difficulties.

The process of testing can be carried out by a large group of people [45]. Therefore, the software can be tested by many testers at the same time. This is very useful because it provides an outstanding opportunity to cover all the functions of the software. Developers can upload their programs in the cloud and ask testers from offshore locations to undertake testing and provide feedback on the programs.

Unfortunately, software engineering technology has not been widely deployed in cloud. Hence, the aim of this project is to demonstrate the deployment of this technology in the cloud. However, developers might use more than one tool in order to achieve high-quality results. Using a variety of tools requires them to be organized and connected together by workflow in order for them to be efficient and usable.

1.1. Objectives

1. **Review**: review the state-of-the-art technology in cloud computing and software engineering tools. The success criteria for this objective would be by providing a review of the stat-of-the-art technology in cloud computing and software engineering tools.
2. **Selection**: select cloud computing and software engineering tools for the project. The success criteria for this objective would be by selecting cloud provider and software engineering tools
3. **Define evaluation scenario and criteria:** assume scenario to discuss whether or not the deployment is helpful in this scenario. The success criteria for this objective would be by providing scenarios to discuss whether or not the deployment is helpful in these scenarios.

4. **Deployment**: demonstrate and evaluate the deployment of the software engineering tools in the cloud. The success criteria for this objective would be by providing demonstration and evaluation of the deployment of the software engineering tools in the cloud.
5. **Workflow**: create processes that could help the developers to validate and test their programs and move from tool to tool. The success criteria for this objective would be by creating processes that could help the developers to validate and test their programs and move from tool to tool.
6. **Evaluation**: investigate the scenario and evaluate the outputs and help decision-makers decide whether or not the deployment could be helpful. The success criteria for this objective would be by providing evaluation of the scenarios and make decision on whether or not the deployment is useful.

1.2. Methodology

This project will focus on the demonstration and evaluation of the deployment of software engineering tools in the cloud. The study provides three scenarios for three companies in different areas of business and with different development environments. These scenarios will be analysed before they used as the basis for evaluating the cloud deployment. A comparison will be made between different service models of cloud providers to select the appropriate cloud to suit the deployment. One of the important aspects which must be understood in the selection of the cloud is the development environment that allows the tools to be run. Next, the software engineering tools will be chosen for deployment.. Subsequently, the cloud will be used to deploy the selected tools and the process will then be created to allow the users to move from tool to tool. Finally, the project will be evaluated based on the scenarios which are assumed before and then investigating these scenarios to decide whether or not the deployment is helpful.

1.3. Structure of Dissertation

Section 2 provides the background to the study and reviews the technologies used. The scenarios and evaluation criteria motivating the work are described in detail in Section 3. The requirements needed to achieve the main project goal of effective cloud deployment of software engineering tools are discussed in Section 4 and the main technical approaches that have been implemented are described in Section 5. The evaluation (Section 6) is followed by conclusions and suggestions for further work (Section 7).

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

2. Background

This section provides definitions of key terms of the main technologies used in this study. It defines cloud computing, providing an overview of its advantages and disadvantages and describes the different service models.. Further, it defines and describes the main software engineering technologies and workflow and outlines their benefits.

2.1. Cloud Computing

2.1.1. What is the cloud?

There is a non-standard definition of cloud computing [2]. Nonetheless the National Institute of Standards and Technology defines it as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]. Cloud computing has been the focus of much research because it offers flexible dynamic IT infrastructures, computing environments and efficient software services [3].

2.1.2. What are the advantages of the cloud?

The cloud has five main advantageous characteristics [4]:

- **On-demand self-service:** users can service themselves without needing a supporting team.
- **Broad network access:** users can access their assets by means of the Internet through different devices.
- **Resource pooling:** many users can access the same data at the same time.
- **Rapid Elasticity:** the flexibility of controlling the usability of the cloud. In other words, consumers can increase and decrease the capability of accessing and using the resources in the cloud.
- **Measured Service:** Users can measure the usability of the cloud and be charged per use. Therefore, consumers can monitor and control usage.

2.1.3. What are the disadvantages of cloud computing?

There are some drawbacks of using cloud computing. However, the following are the major disadvantages which could affect this dissertation.

- **Security**: Each company has security standards which are hard to meet by cloud providers [5]. Each cloud provider has certain features which may not satisfy all specifications for all customers.
- **Data Location and Privacy:** physical location of the servers is very important because different countries have different laws and some companies want to store their data in the same way as their privacy management laws in their countries [6].
- **Privilege User Access:** Sensitive data can be managed by non-employees. It means that cloud servers are not under the control of organizational managers [6].
- **Internet Dependency, performance and Latency:** availability is a crucial aspect for consumers and it could be affected by Internet providers [6].

2.1.4. Cloud Computing Principal Service Models

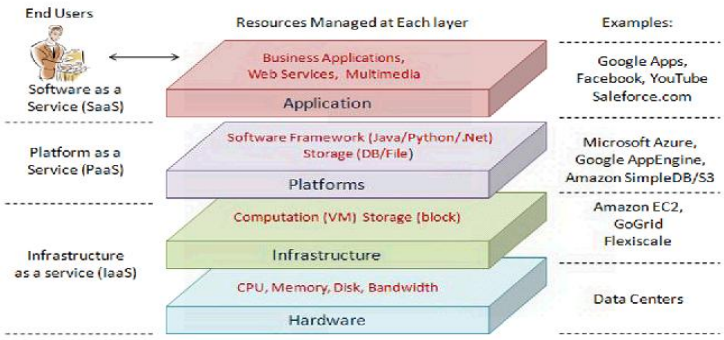

Fig.2. cloud computing architecture [7]

Figure 2 shows the three models and the following are their descriptions:

Software as Service (SaaS): Users of the cloud are allowed to install, operate and run applications in the cloud without managing the infrastructure and the platform of the cloud. The difference between the SaaS cloud's applications and normal PCs' applications is that the cloud applications offer elasticity and can be accessed by more than one user, while PC applications are only accessed by its users [9]. Moreover, the consumers can access the cloud applications from different devices such as mobile phones. Examples of the SaaS service are SalesForce.com [8], YouTube and Facebook. The customers of this service pay monthly for using the cloud.

Platform as Service (PaaS): This allows users to implement and deploy applications by providing a programming environment, for instance Java Runtime Environment (JRE). However, users cannot control or manage the infrastructure of the cloud such as the operating system of the cloud. A good example of this model is Google App Engine. It offers a good environment for running Java, JavaScript and Python applications [10]. Usually, the customers of this service pay per use.

Infrastructure as Service (IaaS): The customers of this model are able to build the whole cloud services starting from the infrastructure ending with applications. They can control storage, Internet, processing and the Operating System images, however they do not own the hardware used to support the cloud [11]. An example of this cloud is Amazon EC2 and it is on a pay-per-use basis.

2.2. Software Engineering

The term "software engineering" (SE) encompasses the activities to analyse, design, construct and test software [25]. A software engineering environment (SEE) supports the tasks of all software engineering processes [48]. Such an environment has many tools to support these tasks.

2.2.1. Software Engineering Tools

The purpose of software engineering tools is to "assist software engineering managers and practitioners in every activity associated with the software process" [18]. Developers can use these tools to analyse, design, implement and test their programs. They can assist developers to easily achieve high quality programs. The quality of software is the key factor in companies that depend upon programs for their work [32]. The following are some of these tools and a brief definition of each of them.

Static Analysis Tools: Static analysis tools can find any indication of bugs in programs before execution [18]. Examples of these tools are Findbugs [19] and PMD [20]. They do not only discover errors but they can report functional and designer errors at a deep level [21].

Testing Tools: Testing tools are used to improve software quality by practising and coordinating testing cases [22]. In other words, they allow users to input test data and verify the output whether it is expected or not. An example of this tool is Agitar [23] and CodeTest [24].

Dynamic Analysis Tools: Dynamic analysis tools deal with running programs to assure that program behaviour is running exactly as expected [25]. These tools interact with executing software, whilst normal testing tools test software in fake environments. There are two types of dynamic tools [25]:

- Intensive tools: they provide extra instruction that is written with the software to ensure that it runs acceptably.
- Non-intrusive tools: these use extra hardware processes that execute in parallel with the process that runs the main software.

Analysis and Design Tools: They allow users to build models of the system that represent data, function and behaviour and characterization of the data, interface design and component-level [25].

Coverage Tools: Coverage tools show the percentage of the functions that were not used [38]. For example, if a user uses only 40% of the functions, the tools will display 60%. This is beneficial in testing because it shows which functions have not been tested, allowing the user to go back and test them.

2.2.2. Software Engineering Methods

There are many software engineering models. However, the following explains two of them because they are used in this dissertation. The Waterfall model has been improved to the iterative model and this improvement will be described in this section.

Fig. 3. Waterfall model

Due to the Waterfall model being able to improve the work-task between the project teams, this model was able to reduce the effort of the developer and increase the performance of producing programs [39]. However, after using this model, it has been identified by means of the imperfect results that the model is inadequate [39] because it cannot go back to previous stages if there are any problems in any of the phases, thus, this issue may be resolved by iterating this model until accomplishing high quality programs [40] see Figure 3.

Fig. 4. Iterative model

Working in the iterative model's phases (see Figure. 4) has some features that could be considered. Each team in each phase has certain tools to help it in their task [25]. For example, the verification team has static analysis tools to verify the program after producing it from the implementation team. Every phase is separated and the outputs of the previous phase are the input for the next phase, for instance, the outputs of the implementation phase is the inputs of the verification phase. The verification phase decides whether or not the programs are ready for release. If the program is not ready the team will return the program with a report to the requirements team.

2.3. Workflow

Hollingsworth indicates that workflow is ‘‘concerned with the automation of procedures where documents, information or tasks are passed between participants” [26]. It concludes connected steps and each step is finished before the next step begins. Potential benefits of workflow models are that [27]:

- It is able to organize and manage distributed systems.
- It provides an approach for interaction between organizations.
- It increases outputs and decreases costs by using resources in certain domains.
- It includes a variety of administrative domains to acquire throughput.
- It has the ability to integrate between multiple teams and applications.

3. Evaluation Scenario and Criteria:

This section is focused in definition of basis scenarios for evaluating the deployment of software engineering tools in the cloud. It begins by setting the assumed cloud usage cost before going on to explain the basis for cost estimation using COCOMO. The three scenarios are then introduced.

Before proceeding further with the investigation of the three scenarios about three companies, it is essential to define the assumed cost of renting the cloud after deploying software engineering tools to help in its evaluation. After deploying these tools in the cloud, the cloud model would be the same as any SaaS model. Thus it is worthwhile to obtain the average cost of applications in the SaaS cloud to ascertain the suspected renting price. Consequently, evaluation will be based on this price. The costs of three providers are:

1- Zoho: It costs $ 40 monthly for 5 GB.
2- Opsource: It costs $49.64 monthly for 5 GB.
3- Rockspace: It costs $ 33.75 monthly for 5 GB.

Therefore, the average of all these prices is (40 + 49.64 + 33.75)/3= 41.13. This is nearly $ 41 per month for 5 GB.

COCOMO standard, which is a Constructive Cost Model, will calculate the estimated price for creating software [33]. This model divides the software projects into three categories which are Organic, Semi-detach and Embedded. This study creates a scenario where the small company deals with Organic projects, the medium company deals with Semi-detach projects and the big company deals with Embedded projects. COCOMO tools will be used to calculate the three formulas [34]. The COCOMO is based on three formulas [34]:

Effort Applied (E) = $a_b(KLOC)^{b}_{b}$ **[man-months]**

Development Time (D) = $c_b(\text{Effort Applied})^{d}_{b}$ **[months]**

People required (P) = Effort Applied / Development Time **[count]**

Where the following table provides a_b, b_b, c_b and d_b in Table1:

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

Hollingsworth indicates that workflow is ‘‘concerned with the automation of procedures where documents, information or tasks are passed between participants” [26]. It concludes connected steps and each step is finished before the next step begins. Potential benefits of workflow models are that [27]:

- It is able to organize and manage distributed systems.
- It provides an approach for interaction between organizations.
- It increases outputs and decreases costs by using resources in certain domains.
- It includes a variety of administrative domains to acquire throughput.
- It has the ability to integrate between multiple teams and applications.

3. Evaluation Scenario and Criteria:

This section is focused in definition of basis scenarios for evaluating the deployment of software engineering tools in the cloud. It begins by setting the assumed cloud usage cost before going on to explain the basis for cost estimation using COCOMO. The three scenarios are then introduced.

Before proceeding further with the investigation of the three scenarios about three companies, it is essential to define the assumed cost of renting the cloud after deploying software engineering tools to help in its evaluation. After deploying these tools in the cloud, the cloud model would be the same as any SaaS model. Thus it is worthwhile to obtain the average cost of applications in the SaaS cloud to ascertain the suspected renting price. Consequently, evaluation will be based on this price. The costs of three providers are:

1- Zoho: It costs $ 40 monthly for 5 GB.
2- Opsource: It costs $49.64 monthly for 5 GB.
3- Rockspace: It costs $ 33.75 monthly for 5 GB.

Therefore, the average of all these prices is (40 + 49.64 + 33.75)/3= 41.13. This is nearly $ 41 per month for 5 GB.

COCOMO standard, which is a Constructive Cost Model, will calculate the estimated price for creating software [33]. This model divides the software projects into three categories which are Organic, Semi-detach and Embedded. This study creates a scenario where the small company deals with Organic projects, the medium company deals with Semi-detach projects and the big company deals with Embedded projects. COCOMO tools will be used to calculate the three formulas [34]. The COCOMO is based on three formulas [34]:

Effort Applied (E) = $a_b(KLOC)^{b}_{b}$ **[man-months]**

Development Time (D) = $c_b(\text{Effort Applied})^{d}_{b}$ **[months]**

People required (P) = Effort Applied / Development Time **[count]**

Where the following table provides a_b, b_b, c_b and d_b in Table1:

Table1. Given variables for COCOMO formulas

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

According to Jacek, Thomas and Herald, the percentage of finding defects in a program in a real environment will be approximately 78% if refactorings are not applied in the development environment [45]. Refactoring improves the code by reducing complexity and making the programs readable to help developers to maintain their programs. However the percentage would be approximately 31% if refactorings are applied [45]. This research assumes that the small company does not apply a refactorings technique, while the medium and big companies do.

All these companies depend upon the iterative model. SEE in all companies in this study do not have any software engineering tools. In the verification phase, they verify the programs manually by writing acceptance test documents and test the programs without any tools. Once the functionalities of the programs satisfy the acceptance test, they will release the programs; otherwise they will return them to the requirement phase.

It is assumed that all three companies employ the Scrum concept. Scrum is a software development process that divides developers into small teams, and each team can do all the work that is related to any program [48]. The team can handle the program from the first phase of the Iterative model to the last phase. A team's roles are:

Product owner: who is responsible for customer requirements.

Development team: Its tasks are analysis, design, development, testing and verification.

Scrum Master: he is the team leader.

In the small company the task is not big. Thus, it can be done by one Scrum team and then be released without integration. The task in the medium and big companies

is divided into small parts and each Scrum team is assigned a different part. After finishing the assigned task, one of the Scrum teams integrates the parts together (see Figure 5).

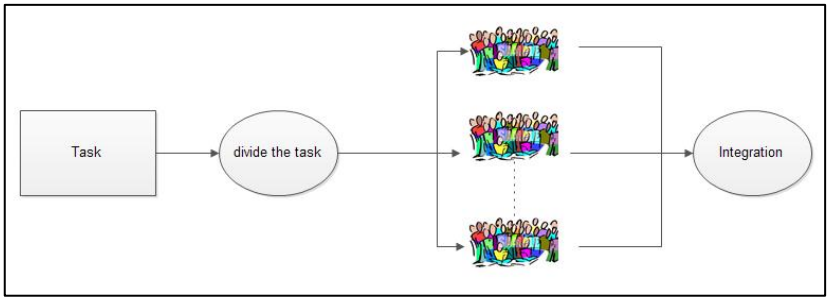

Fig. 5. Dividing task and integration

3.1. First Scenario

The small company has between 1 to 20 employees (2 Scrum teams). This team is highly familiar with software language. The team works at home and the average salary for them is $1000. The average of the delivered source code is one KDSI for one month, which is one thousand, delivered source instructions for one month. The COCOMO tool is used to calculate the efforts and development cost. Figure 6 shows the inputs in the COCOMO tool and Figure 7 shows the outputs.

Inputs

Development	
Delivered Source Instructions (thousands) (KDSI)	1
Development Mode	Organic
Average Cost Rate ($/PM)	1000
Maintenance	
KDSI added (annual)	0
KDSI modified (annual)	0
Average Cost Rate ($/PM)	0

Recalc

Fig. 6. The inputs of the COCOMO tool for first scenario.

Results

Effort	2	person-months (PM)
Schedule	3	months
Development Cost	2000	
Productivity	500	instructions per person-month
Average Staffing	0.7	full-time-equivalent software personnel
Annual Maintenance Effort	0	person-months
Annual Maintenance Cost	0	

Fig. 7. The outputs of COCOMO tool for the first scenario.

Consequently, the estimated payment for implementing one KDSI is $2000, and this payment is only for two developers. However, the study assumes that after the customer returns the code fragment, because of bugs, the company needs the effort of one developer for one month. Therefore, the cost of fixing the bugs is $1000. The percentage of finding bugs by customers in produced programs is 78% (see Figure 3). Assuming that the company sells 12 KDSI annually, thus, the annual operating expense is $24000, which is if the customers do not return the software because of defects.

Evaluation: To make a decision on whether or not software engineering tools are useful, it is beneficial to calculate probability factors to provide evidence that could help to make a decision [35]. The probability would be calculated by running 12 random cases which represent 12 months and based on Figure 8. Running the random cases will be done by a Java program.

```
int run = 12; // one year
int a = 0,b = 0;
Random randomGenerator = new Random();
For (int i=0;i<run;++i)
{

double randomNum= randomGenerator.nextDouble();
if (randomNum>0.78){
a++; // No bugs

}else b++; // Find bugs by customer
}

double percentageOfNoBugs= (a/(double)run)*100;
double percentageOfFindBugsByCustomer= (b/(double)run)*100;
```

```
double RewardA= percentageOfNoBugs *(+41);
double penaltyB=percentageOfFindBugsByCustomer*(-1000);
System.out.println(" the company will lose  = $ " +penaltyB + " ,if it does not rent the cloud service " );
System.out.println(" the company will earn = $ " +RewardA + " ,if it does not rent the cloud service " );
double utility = RewardA + penaltyB;
System.out.println("the final result is $"+ utility);
```

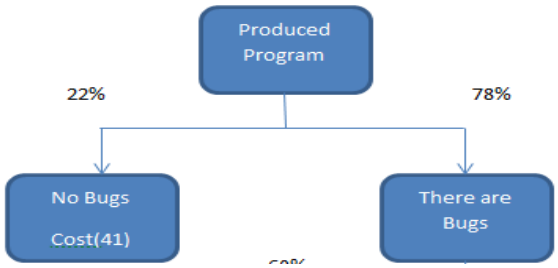

Fig. 8. Probability tree

Result:

```
  the company will lose  = $ -75000.0,if it does
not rent the cloud service
   the company will earn = $ 1025.0,if it does
not rent the cloud service
  the final result is $-73975.0
```

Fig. 9. The results of the probability program

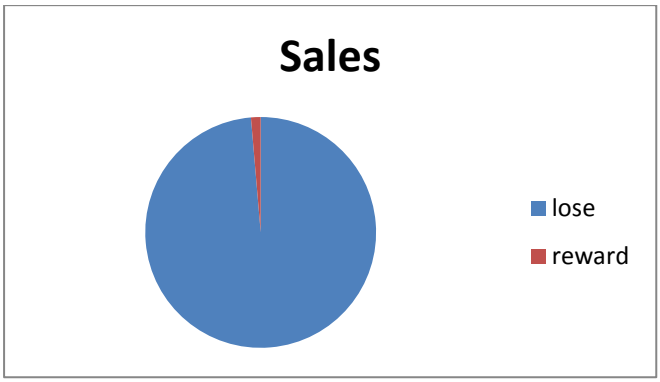

Fig. 10. Annual profit in the case of not renting the tools in the cloud for the small company

Figures 9 and 10 show the outputs of running 12 random cases. The utility is $-73975.0 which is the amount that could be lost if the company does not rely upon software engineering tools to verify and test their code fragment in the cloud. By adding the annual operating expense to this amount, the final operating expense is $97975, an increase of 75%. Therefore, there is a significant negative of not renting software engineering tools in the cloud for the small company in terms of the economic aspect.

3.2. Second Scenario

The medium company has between 20 and 50 employees (2 to 5 Scrum teams). This company can develop big and small code fragments. The average salary for them is $1500. The company produces approximately 6 KDSI monthly. Assuming that, the team members are professionals and they can implement high quality programs by applying the refactorings mechanism in the software. Therefore, the percentage of finding defects in the programs by customers is 31%. The COCOMO tool is used to calculate the effort and development cost. Figure 11 shows the inputs in the COCOMO tool and Figure 12 shows the outputs.

Inputs

Development	
Delivered Source Instructions (thousands) (KDSI)	6
Development Mode	Semidetached
Average Cost Rate ($/PM)	1500
Maintenance	
KDSI added (annual)	0
KDSI modified (annual)	0
Average Cost Rate ($/PM)	10000

Recalc

Fig. 11. The inputs of the COCOMO tool for the second scenario.

Results

Effort	22	person-months (PM)
Schedule	7	months
Development Cost	33000	
Productivity	273	instructions per person-month
Average Staffing	3.1	full-time-equivalent software personnel
Annual Maintenance Effort	0	person-months
Annual Maintenance Cost	0	

Fig. 12. The outputs of the COCOMO tool for the second scenario.

Consequently, the estimated payment for implementing 6 KDSI is $33000, and this payment is only for 22 developers. However, this study assumes that after the customers return the code fragment, because of bugs, the company needs the effort of half of this number which is 11 developers. Therefore, the cost of fixing the bugs is $16500. Assuming that the company sells 72 KDSI annually, thus, the annual operating expense is $396000, which is if the customers do not return the software because of defects.

Evaluation: The probability is calculated by running 12 random cases which represent 12 months. Figure 13 shows the probability tree which will be run 12 times. The random number will be generated to discover the probability. If the number is more than 79 that means that the customers find defects, on the other hand, if it is less than 79 that means that the customers do not find defects. The Java program will be implemented to calculate the probability.

```
int run = 12; // one year
int a = 0,b = 0;
Random randomGenerator = new Random();
For (int i=0;i<run;++i)
{

double randomNum= randomGenerator.nextDouble();
if (randomNum>0.31){
a++; // No bugs

}else b++; // Find bugs by customer
}

double percentageOfNoBugs= (a/(double)run)*100;
double percentageOfFindBugsByCustomer= (b/(double)run)*100;

double RewardA= percentageOfNoBugs *(+41);
double penaltyB=percentageOfFindBugsByCustomer*(-16500);
System.out.println(" the company will lose  = $ " +penaltyB + ", if it does not rent
the cloud service " );
System.out.println(" the company will earn = $ " +RewardA + ", if it does not rent
the cloud service " );
double utility = RewardA + penaltyB;
System.out.println("the final result is $"+ utility);
```

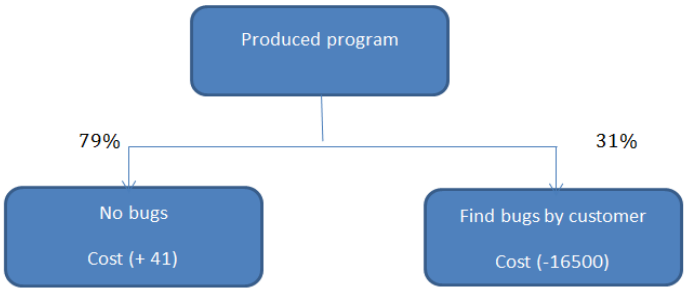

Fig. 13. Probability tree

Result:

```
  the company will lose  = $ -412500.0,if it
does not rent the cloud service
  the company will earn = $ 3075.0,if it does
not rent the cloud service
  the final result is $-409425.0
```

Fig. 14. The program output

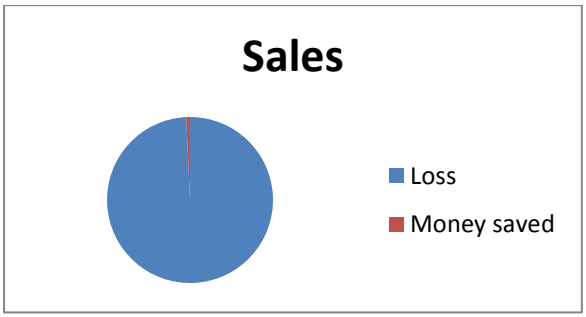

Fig. 15. Annual profit in case of not renting the tools in the cloud for the medium company

Figures 14 and 15 show the outputs of running 12 random cases. The utility is $-409425.0 which is the amount that could be lost if the company does not rely upon software engineering tools to verify and test their code fragment in the cloud. By adding the annual operating expense to this amount, the final operating expense is $805425, an increase of 49%. Therefore, there is a significant negative of not renting software engineering tools in the cloud for the small company in terms of the economic aspect.

3.3. Third Scenario

The medium company has more than 50 employees (more than 5 Scrum teams). This company specializes in the implementation of big code fragments. The average salary for them is $2000. The company produces approximately 12 KDSI monthly. Assuming that, the team members are professionals and they can implement high quality programs by applying the refactorings mechanism in the software. Therefore, the percentage of finding defects in the programs by customers is 31%.The COCOMO tool is used to calculate the effort and development cost. Figure 16 shows the inputs in the COCOMO tool and Figure 17 shows the outputs.

Inputs

Development	
Delivered Source Instructions (thousands) (KDSI)	12
Development Mode	Organic
Average Cost Rate ($/PM)	2000
Maintenance	
KDSI added (annual)	0
KDSI modified (annual)	0
Average Cost Rate ($/PM)	10000

Recalc

Fig 16. The inputs of the COCOMO tool for the third scenario

Results

Effort	33	person-months (PM)
Schedule	9	months
Development Cost	66000	
Productivity	364	instructions per person-month
Average Staffing	3.7	full-time-equivalent software personnel
Annual Maintenance Effort	0	person-months
Annual Maintenance Cost	0	

Fig. 17. The outputs of the COCOMO tool for the third scenario

Consequently, the estimated payment for implementing 12 KDSI is $66000, and this payment is for the effort of 33 developers (see Figure 13). However, this study assumes that after the customers return the code fragment, because of bugs, the company needs the effort of half of this number which is 16.5 developers. Therefore, the cost of fixing the bugs is $ 33000. Assuming that the company sells 144 KDSI annually, thus, the annual operating expense is $9504000, that is, if the customers do not return the software because of defects.

Evaluation: The probability will be calculated by running 12 random cases which represent 12 months. Figure 9 shows the probability tree which will be run 12 times. The random number will be generated to discover the probability. If the number is more than 79 that means customers find defects, on the other hand, if it is less than 79 that means that the customers do not find defects. The Java program will be implemented to calculate the probability.

```
int run = 12; // one year
int a = 0,b = 0;
Random randomGenerator = new Random();
For (int i=0;i<run;++i)
{

double randomNum= randomGenerator.nextDouble();
if (randomNum>0.31){
a++; // No bugs

}else b++; // Find bugs by customer
}

double percentageOfNoBugs= (a/(double)run)*100;
double percentageOfFindBugsByCustomer= (b/(double)run)*100;
```

```
double RewardA= percentageOfNoBugs *(+41);
double penaltyB=percentageOfFindBugsByCustomer*(-33000);
System.out.println(" the company will lose  = $ " +penaltyB + " ,if it does not rent the cloud sevice " );
System.out.println(" the company will earn = $ " +RewardA + " ,if it does not rent the cloud sevice " );
double utility = RewardA + penaltyB;
System.out.println("the final result is $"+ utility);
```

Result:

```
  the company will lose  = $ -825000.0, if it
does not rent the cloud service
  the company will earn = $ 3075.0, if it does
not rent the cloud service
  the final result is $-821925.0
```

Fig. 18. The program output

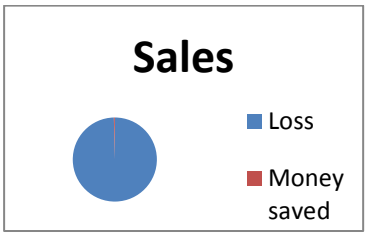

Fig. 19. Annual profit in case of not renting the tools in the cloud for the medium company

Figures 18 and 19 show the outputs of running 12 random cases. The utility is $821925.0 which is the amount that could be lost if the company does not rely upon software engineering tools to verify and test their code fragments in the cloud. By adding the annual operating expense to this amount, the final operating expense is $10325925,an increase of 92%. Therefore, there is a significant negative of not renting software engineering tools in the cloud for the small company in terms of the economic aspect.

3.4. Conclusion

The study in this chapter has illustrated the loss that might happen in small, medium and big companies if they do not use software engineering tools in the cloud. However, the above is based upon the assumption that these companies do not have any static analysis tools that could analyse and verify their programs. The COCOMO standard has been used to calculate the estimated cost of creating programs. The probabilities approach has been adopted to find the possible profit that may be generated or lost. These probabilities are justified by prior work in this field.

Table 2. Final result for all scenarios

Company	Utility	Operating expense	Percentage of increase
Small	$-73975.0	$ 97975	75%
Medium	$-409425.0	$ 805425	49%
Big	$-821925.0	$10325925	92%

Table 2 compares the three companies. It is apparent from this table that there is a significant effect in the big company because it produces a large size of code fragments without using any software engineering tools, whilst the medium company experiences the smallest effect because it applies the refactorings technique. Nevertheless, all of the companies have relatively high percentages with regard to the increase in operating expenses. Consequently, the results of the study in this section provide strong evidence for the claim that a company should treat their programs by using software engineering tools in the cloud.

However, this evaluation is only for one part of SEE which is to verify and test the programs before release. Therefore, it could be beneficial if the cloud were used in all SEE parts such as the design process.

4. Requirements

The purpose of this section is to explain the requirements that should be met to use software engineering tools in the cloud. These requirements also consider the SEE where Scrum teams work. The requirements are classified into four aspects according to the cloud and the SEE's components which are cloud requirements, design requirements, software engineering tools requirements and the requirements on customers. Although there are many requirements that could be specified, the following are some of those which rely on what we find in published papers and cloud documents

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

3.4. Conclusion

The study in this chapter has illustrated the loss that might happen in small, medium and big companies if they do not use software engineering tools in the cloud. However, the above is based upon the assumption that these companies do not have any static analysis tools that could analyse and verify their programs. The COCOMO standard has been used to calculate the estimated cost of creating programs. The probabilities approach has been adopted to find the possible profit that may be generated or lost. These probabilities are justified by prior work in this field.

Table 2. Final result for all scenarios

Company	Utility	Operating expense	Percentage of increase
Small	$-73975.0	$ 97975	75%
Medium	$-409425.0	$ 805425	49%
Big	$-821925.0	$10325925	92%

Table 2 compares the three companies. It is apparent from this table that there is a significant effect in the big company because it produces a large size of code fragments without using any software engineering tools, whilst the medium company experiences the smallest effect because it applies the refactorings technique. Nevertheless, all of the companies have relatively high percentages with regard to the increase in operating expenses. Consequently, the results of the study in this section provide strong evidence for the claim that a company should treat their programs by using software engineering tools in the cloud.

However, this evaluation is only for one part of SEE which is to verify and test the programs before release. Therefore, it could be beneficial if the cloud were used in all SEE parts such as the design process.

4. Requirements

The purpose of this section is to explain the requirements that should be met to use software engineering tools in the cloud. These requirements also consider the SEE where Scrum teams work. The requirements are classified into four aspects according to the cloud and the SEE's components which are cloud requirements, design requirements, software engineering tools requirements and the requirements on customers. Although there are many requirements that could be specified, the following are some of those which rely on what we find in published papers and cloud documents

4.1. Cloud Requirements

The following are the requirements that should be offered to reap the benefits of using software engineering as a service in the cloud and to avoid any issues:

Pricing: It is expected that a company which wants to use the cloud has the ability to calculate the cost and make payments for using the cloud by means of invoicing and e-invoicing; it can still make payment by a variety of traditional methods such as PayPal [46].

Availability: The availability of the cloud needs to be very high. The reason for this is that some tools take a long time to perform their tasks [44].

Security, Privacy & Trust concerns: the assets of the users need to be held in a secure place [49]. Customers want their data to be fully protected and providers to apply proper security measures. Cloud security involves computer security, network security and information security [49]. **Privacy** is an important part in all the challenges that may be faced in cloud computing. Users want to ensure that their identities, information, policy components during integration and transaction histories are protected [50]. **Trust** is non-quantitative and difficult to measure [50]. However, it is defined as "the extent to which one party is willing to participate in a given action with a given partner, considering the risks and incentives involved" [51]. The cloud needs to be trustworthy to attract customers to use it.

Service Level Agreement (SLA) should be clear and satisfactory. Users should be aware of the terms and conditions governing the use of the cloud.

4.2. Design Requirements

Tight coupling between software engineering tools and user: This is required because these tools need to be much more interactive than the e-Science application, which runs an automatic workflow [43] [52].

Tools connection: The tools are connected together to allow users to move from one to the other efficiently..

Assessment on cloud viability: Before using the cloud in SEE, the program should be assessed by Scrum teams. Therefore, the design of the SEE should consider this assessment. The assessment should be based upon certain criteria to provide the Scrum teams with the ability to make a decision on whether or not they use the tools in the cloud.

4.3. Software engineering tools requirements

Usability: It is very important that end-users interact easily with the tools. The tools should allow users to specify the pattern of verifying and testing the program.

The tools are for a single program language: If selection of software engineering tools is for more than one tool, the tools should be for a single language.

4.4. Requirements on customers

Skills: Users should realize how they use the cloud resources [47]. If users do not know how they can use the cloud, they will face difficulty in reaping its benefits.

5. Approaches

We could argue that if the approach succeeds in one phase in the Iterative model, it should work with other phases. Since the key issue of losing money in the companies in Section 3 is in the verification phase because the programs are released before being adequately tested. Therefore, the selections concentrate on static analysis and testing tools. These selections consider the requirements in Section 4.

In order to achieve the main aim, we have two approaches. The first approach devised for this study is 'indirect interaction' and the second approach is 'direct interaction'.

5.1. Indirect Interaction Approach

The indirect interaction approach refers to users using cloud to run the tools which are hosted in another cloud (see Figure 20). The users use the tools through the first cloud and run the tools in it. It is necessary to select two cloud computing providers and tools in order to prove this concept.

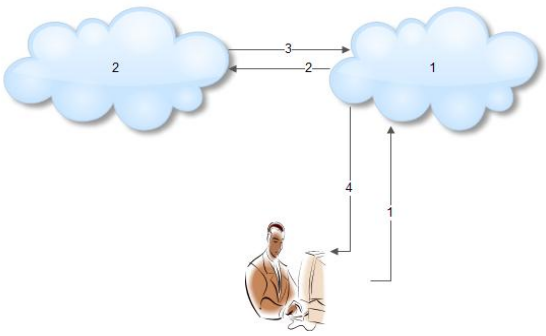

Fig. 20. Overview of Indirection Design

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

4.4. Requirements on customers

Skills: Users should realize how they use the cloud resources [47]. If users do not know how they can use the cloud, they will face difficulty in reaping its benefits.

5. Approaches

We could argue that if the approach succeeds in one phase in the Iterative model, it should work with other phases. Since the key issue of losing money in the companies in Section 3 is in the verification phase because the programs are released before being adequately tested. Therefore, the selections concentrate on static analysis and testing tools. These selections consider the requirements in Section 4.

In order to achieve the main aim, we have two approaches. The first approach devised for this study is 'indirect interaction' and the second approach is 'direct interaction'.

5.1. Indirect Interaction Approach

The indirect interaction approach refers to users using cloud to run the tools which are hosted in another cloud (see Figure 20). The users use the tools through the first cloud and run the tools in it. It is necessary to select two cloud computing providers and tools in order to prove this concept.

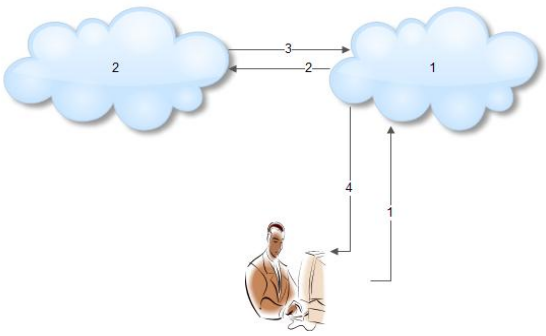

Fig. 20. Overview of Indirection Design

5.1.1. Selection cloud computing

The selections are for two clouds. The first cloud is for running the tools in it. The second cloud is for deployment of these tools.

First cloud: It was necessary to select the IaaS cloud to have the ability to build the infrastructure that was required. It is necessary that the cloud provider offer a proper image. The proper image refers to offering applications and an environment to run the tool. Most of the IaaS providers have infrastructures so that users can build the environment that they need. Therefore, Amazon provider (AWS) which is one of IaaS cloud was chosen in this study. The image of the cloud would be created after choosing the tools to select the necessary application.

Second cloud: Development environment is one of the most important aspects of selection cloud computing because it assists in running the deployed application in the cloud. This service is only in PaaS and IaaS clouds, while it is not in an SaaS cloud. Therefore, the SaaS cloud is eliminated from selection cloud options. However, if a user wants to use IaaS an infrastructure needs to be built which costs money and takes a long time [29]. It is not necessary to construct the cloud infrastructure to deploy tools whilst it is ready in PaaS cloud with an outstanding environment for deployment. Consequently, PaaS is the most suitable cloud for this project.

Table 3 shows the comparison between six PaaS cloud providers. This comparison considers selected criteria which are:

1- Support: Supporting users by a cloud provider and handling any issues free.
2- Security feature: protection for the users' assets by offering security features such as Firewall, back up storage and secure permissions.
3- Operating System: the OS which is used in providers' servers such as UNIX and Windows.
4- Cost. The minimum price.
5- Program language supported: the development environment that is in the cloud.

Table 3: Comparison between PaaS cloud providers

Cloud Provider	Support	Security Features	Operating System	Cost	Program Language
App Engine [10]	Yes	Backup storage	Linux and Windows Server 2008	Free	Python, Java and Go [12]
Boomi [13]	Yes	N/A	Cent, Windows server 2003 and Windows server 2008	$550/month* for 2 connections	APL and Java
AT&T Synaptic [14]	Yes	Backup Storage, Data protection	Linux and Windows Server 2008	Pay-as-you-go service.	No
BIMIXS [15]	Yes	Firewall, Backup storage	Windows Server 2008	Free	No
Citrix [16]	Yes	Data protection	Windows server 2003 and Windows server 2008	Estimated price	No
JumbBox [17]	Yes	Bootable Mode and Backup storage	Cent OS and Linux.	$49/month.	PHP, Python, SQL and Visual Basic.

It is clear that AppEngine, Boomi and AT&T Synaptic support some program languages while the others do not. This factor is very important because this project aims to perform verification and testing for program languages and the tools may be created by one of these languages. However, there are no significant differences between the services of the three providers and this project is not going to use a very large number of resources, hence selection is for a free provider which is AppEngine.

AppEngine's server is a virtual server [10]; this means that it has the quality of scalability which continues to work well even if it has a very large size of software engineering tools. Users do not need to book the space that they need before using the server because it is very difficult to decide how many bytes they need to verify and test the programs. As a result, AppEngine cloud will be helpful if it is used to verify the programs it runs.

5.1.2. Selection Software engineering tools

As this project aims to demonstrate and evaluate deployment Software Engineering tools in the cloud, it is beneficial to choose tools that are open source to manipulate and test them without any constraints. AppEngine is targeted to perform web applications [30], thus these tools need to be web-based applications. Java Script language is one of the most important web languages which can help to create web-based applications [31]. JavaScript is a client-side server; hence it does not run in the server. Therefore, this feature could be used to run the tools, which are implemented in JavaScript, in the first cloud in Figure 20 after deploying them in the second cloud.

There are several open sources of software engineering tools that could improve the quality of the JavaScript application. Selection of these tools will be for two static analysis tools and one coverage tool which can measure code coverage for JavaScript software. Since users can verify their code fragments by two tools, thus they obtain the opportunity to ensure that their programs run correctly. After that they can test their programs manually in certain cases and measure the code to find which functions have not tested. The following are the tools which were selected for this study (Appendix B show all screenshot for all selected tools).

JSHint: JSHint can detect potential problems in JavaScript code before executing the program, and it has many options that allow users to select the pattern that they want [36]. For example, users can ignore testing the functions inside the loops if they do not choose the option which does that.

JSLint: JSLint can be used to verify any JavaScript code [37]. It has other options which are different from the options of JSHints. Therefore, there is a significant possibility that JavaScript code could be verified by a variety of options.

JSCoverage: When users test their JavaScript code, the JSCoverage tool will display the functions that are not tested [38]. Due to the fact that there are no testing tools appopriate for all potential software [32], users can test their code fragments manually in certain cases and then discover the functions that have not been covered, by the JSCoverage tool.

5.1.3. Deployment

Before the deployment of the selected tools in AppEngine, it is necessary to register an application ID and the version of these tools. AppEngine has an administrator console to do the registration. Once this is completed, the cloud is ready to accept the application and then the application will be prepared for deployment.

To deploy software engineering tools in the AppEngine cloud, the Yaml file for each tool needs to be implemented (see Figure 20). This file specifies the paths of static files such as Cascading Style Sheets (CSS). It also has information about the tools such as ID and the version of the tool. After implementation, AppEngine Launcher needs to be used to deploy the tools in the cloud (see Figure 21).

```
application: statictest1234
version: 14
runtime: python
api_version: 1

handlers:
- url: /
  static_files: code/jscoverage.html
  upload: code/jscoverage.html

- url: /
  static_dir: code
```

Fig. 21. YAML for JSCoverage tool.

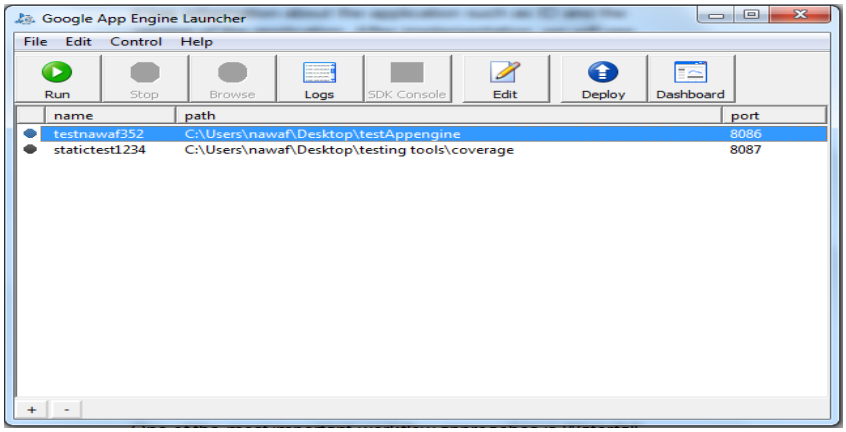

Fig. 22. AppEngine Launcher

In order to upload the tools, it is necessary to put YAML with the tool in the same folder, and then use the Launcher to send the folder to the cloud. Alternatively, it could upload the folder by Eclipse after downloading the AppEngine plugin [10], but AppEngine Launcher might be better than Eclipse because it offers very efficient features which can assist users to manage and control the tools before and after deployment.

5.1.5. Image

Amazon Machine Image (AMI) is a simple interface which allows users to obtain and configure capacity with minimal friction. It can offer certain applications that would be useful to run the tools. The application, that is required to run the tools in this study, is any browser. Although there are many kinds of images, DescktopAnyWhere image is free and it offers Firefox browser that can be used to open the selected tools after deployment in the AppEngine [54] (see Figure 23).

Fig. 23. AMI Screen shot

5.1.6. Design

The overview design in Figure 20 displays the interaction between users and the tools. Firstly, users access the AMI in AWS and then open the Firefox browser. Secondly, users use Firefox to open the tools from AppEngine by typing the address of the tools. Finally, users run the tools. The tools will run in AWS not in AppEngine since the tools are implemented in JavaScript.

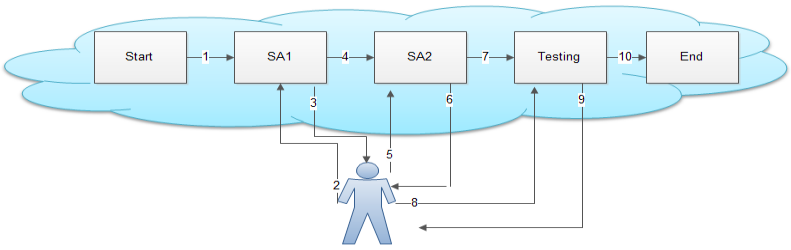

Fig. 24. Software engineering tools workflow & interaction

The interaction design in Figure 24 shows the workflow of software engineering tools in the cloud, while SA1, SA2 and Test are JSHint, JSLint and JSCoverage respectively. It also shows the interaction between users and the tools in the cloud. The scenario of this interaction is:

1- Starting by first static analysis, which is the JSHint tool.

2- Inputting JavaScript code fragment and verifying this code.
3- Printing reports.
4- Moving to the next static analysis tool which is JSLint.
5- Verifying the code by the JSLint tool.
6- Printing reports.
7- Moving to the next tool which is the JSCoverage tool to test the code.
8- Test the code and ensure that testing covered all functions.
9- Test the functions that were not covered.
10- Finish

The pattern of interaction: This will provide users with the opportunity to evaluate the output from each tool and then decide whether or not to proceed to another tool. It allows users to verify their programs, and then fix any errors before moving to the next tool. The JSCoverage tool is at the end of the workflow to validate the programs manually and then identify which functions were not tested in order for them to be tested. Therefore, the programs will be checked twice automatically before being tested manually by the JSCoverage tool. Consequently, the possibility of acquiring a program without defects is very high.

The reports of software engineering tools: Another benefit of the design is that in order to make a comparison between the tools' reports, users can print out reports about their programs from each tool and save them as a hard copy (see Figure 25). Each tool has a different report which may inform users about different defects.

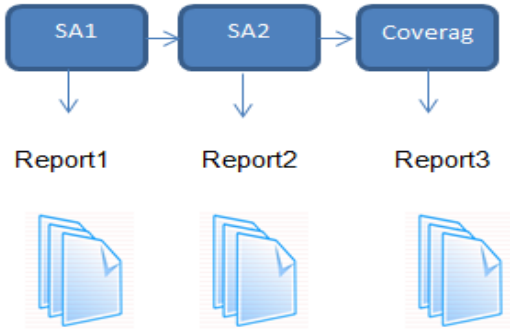

Fig. 25. Software engineering tools' reports

5.2. Direct Interaction Approach

In this type of approach the interaction is between users and the tools in the cloud directly. In order to prove this approach, we need to select a cloud computing provider and tools.

Cloud Selection: The IaaS cloud must be selected to have the ability to build the infrastructure that is needed. The cloud provider needs to offer an appropriate image. The appropriate image should have the application and the environment to run the tool. In fact, most of the IaaS providers have infrastructures that users can use to build the environment that they need. Therefore, the Amazon provider (AWS), which is one of the IaaS clouds, was chosen in this study. This cloud is explained in detail in Sections 5.1.1 and 5.1.5.

Tool Selection: There are many static analysis and testing tools. Nevertheless, the tools which are selected in Section 5.1.2 are used in this approach. These tools can be opened by any browsers; hence the Firefox browser from the AWS image could be used to open the tool in the cloud as was explained in the last section.

Deployment of the tool in the cloud in this approach is easy and it can be done by creating Dropbox folder in the virtual machine desktop and another folder for the same Dropbox account in PC, and then once the source code of the tools is saved in the Dropbox folder in PC it will synchronise with the folder in the cloud. After that the tools can be opened by Firefox browser.

5.3. Integration Design

Based on the scenarios in Section 3, it is necessary to integrate the software engineering tools in the cloud with SEE. The architecture is suitable for most companies. This integration performs with both approaches.

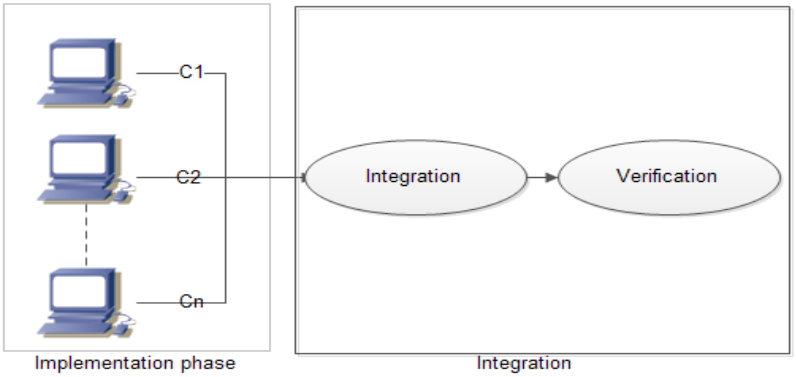

Fig. 26. Integration between implementation and verification phases

In the verification phase, the pieces of code, which are sent from Scrum teams, are integrated [41] (see Figure 26). After that, the assessment of the whole program after integration is done to know whether or not it needs to use the cloud (see Figure 27). The assessment will consider the size of the program. Some programs may not need to use software engineering tools in the cloud, while the others need to use the cloud resources in terms of the size. The assessment also considers the infrastructure and the tools and whether the infrastructure is available in the company or not.

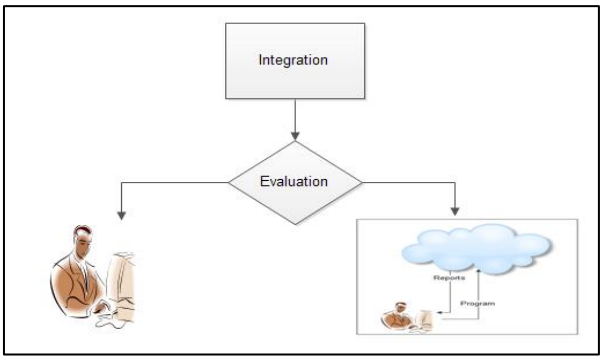

Fig. 27. Evaluation of the program to decide which method is suitable

Fig. 28. Using software engineering tools in the cloud

After deciding that the program needs to use software engineering tools in the cloud, the tools will be used to get reports about the program (see Figure 28). If the program has serious defects, the tester will send the reports and the programs back to the requirement phase.

6. Evaluation

The evaluation considers the approaches that are used and it is based on three aspects that are the scenarios in Section 3 and the requirements in Section 4.

6.1. Selected Clouds Evaluation

Pricing: All clouds which are used are free clouds, so the pricing requirement cannot be evaluated. However, AppEngine and AWS can send bills to their customers to explain their consumption. These clouds have limitations in using AWS which are 613 MB of memory, 32-bit and 64-bit platform support and 750 hours. If users use more than these limitations, they will pay based on the calculator in the AWS system [55]. Therefore, this calculator is helpful because users can calculate the price they will pay before using the clouds.

Availability: AppEngine and AWS are built in various global regions. Therefore, the availability is very high.

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

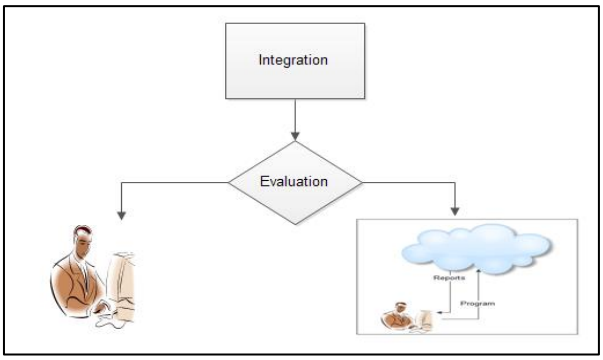

Fig. 27. Evaluation of the program to decide which method is suitable

Fig. 28. Using software engineering tools in the cloud

After deciding that the program needs to use software engineering tools in the cloud, the tools will be used to get reports about the program (see Figure 28). If the program has serious defects, the tester will send the reports and the programs back to the requirement phase.

6. Evaluation

The evaluation considers the approaches that are used and it is based on three aspects that are the scenarios in Section 3 and the requirements in Section 4.

6.1. Selected Clouds Evaluation

Pricing: All clouds which are used are free clouds, so the pricing requirement cannot be evaluated. However, AppEngine and AWS can send bills to their customers to explain their consumption. These clouds have limitations in using AWS which are 613 MB of memory, 32-bit and 64-bit platform support and 750 hours. If users use more than these limitations, they will pay based on the calculator in the AWS system [55]. Therefore, this calculator is helpful because users can calculate the price they will pay before using the clouds.

Availability: AppEngine and AWS are built in various global regions. Therefore, the availability is very high.

Security, Privacy & Trust concerns: The software engineering tools are modified in the cloud by the authorized user. A username and password are required to identify authenticated users. This authentication is used to access log files and the applications' console. However, the tools in the AppEngine are public and they can be used by all people and users can run them on their computers, but if they want to perform them in AWS because of scalability they must have an AWS account. Additionally, this dissertation does not use AppEngine's database or backup. Whilst AWS has many security features, the following are taken for the infrastructure of this study [56]:

1- **Physical Security.** AWS hardware is housed in Amazon-controlled data centres around the world and the data centres are secured with a variety of physical security.
2- **Secure Service.** There is complete Firewall, and AWS uses Secure Shell (SSH) to secure data communication between users and virtual server.
3- **Data Privacy.** The users can encrypt their personal or business data in AWS and they can produce backup for it.

Service Level Agreement: The cloud issues, explained in Section 2, have to be discussed with cloud providers when users want to agree a contract with them. Nevertheless, the small company does not have specialist employees to read SLA and discuss the terms with the cloud provider. While the medium and big companies have the opportunity to read the terms of using the clouds.

6.2. Design Evaluation

Tight coupling between software engineering tools and user: Interaction between the tools and users is in all workflow processes. The tools' outputs can be printed individually. Therefore this requirement is met. However, users have to copy and paste the programs in all the tools. Although it is very efficient that the tools can read the programs from the cloud server instead of posting them to all the tools, users may need to edit their programs during the verification task.

Tools connection: This requirement is met because users can move to another tool easily. However, users cannot post their programs from tool to tool, and they have to copy and paste their programs whenever they want verification.

Assessment on cloud viability: This requirement is dealt with in Section 5.3 where a set of criteria need to be considered and assessed by the user when making a decision on whether or not cloud is a viable option for their specific program requirements.

6.3. Software Engineering Tools Evaluation

Usability: The tools which are selected are more efficient to use and they take less time to accomplish a particular task because they have a friendly interface which can be used easily.

The tools are for one program language: All tools which are selected are for JavaScript language.

However, the selection tools are free and there are better than these tools but they are not free. The good example for these tools is Klocwork[1]. This tool can verify C/C++, Java and C# and it can detects a comprehensive range of reliability, security, and maintainability issues in code fragments.

6.4. Requirements on Customer Evaluation

Skills: The members of the Scrum team are developers. Therefore, it can be assumed that they have the ability to deal with clouds' resources and tools.

6.5. Comparison between Approaches

There are no significant differences between indirect and direct interaction approaches because both of them use AWS cloud and the same software engineering tools. In the indirect interaction approach, users open the tools from AppEngine through AWS and the tools are implemented in JavaScript which is a client-side server. Hence the tools will be run in AWS as AWSs' services. Nevertheless, in the indirect interaction approach, it is possible to run the tools in users' machines or use AWS cloud for performing. Whilst the tools in the direct approach are hosted in AWS cloud and users must have a contract with Amazon cloud to use them. Therefore, the second approach is stricter than the first approach and users may have to pay if they use more than 613 MB or increase the speed of the CPU to perform verification or if the program performance takes a long time. AWS is also free for one year; however after this users will have to pay per use.

Both of the two approaches use two static analyses and one JSCoverage tool to test programs manually. Therefore, static analysis tools can find defects and the JSCoverage tools can expose deep design and functional errors. Consequently, the potentiality of having programs without defects is very high and this is evident when this study's approaches with the companies are evaluated.

[1] http://www.klocwork.com/

6.6. Evaluation of Approaches in the Companies

Based on the percentages in Section 3 the percentage of finding defects if refactoring is not applied in the development environment is 78%. 40% of all defects could be detected by using static analysis tools [57]. Consequently, the percentage of finding defects after using static analysis is approximately 46%. This percentage is used in small companies because it was assumed that this company does not apply a refactoring mechanism.

The percentage of detecting errors is 31% if refactoring is applied. Consequently, the possibility of finding defects is around 18% after using static analysis. This percentage is used in medium and big companies because it was assumed they apply a refactoring mechanism.

Small Companies: Instead of running a Java program to calculate the probability, the results in Section 3 can be used. The amount that could be lost if the small company does not use static analysis tools is $-73975.0 and 40% of this amount will be saved, which is $29590, if the company uses static analysis tools (See Figure 29).

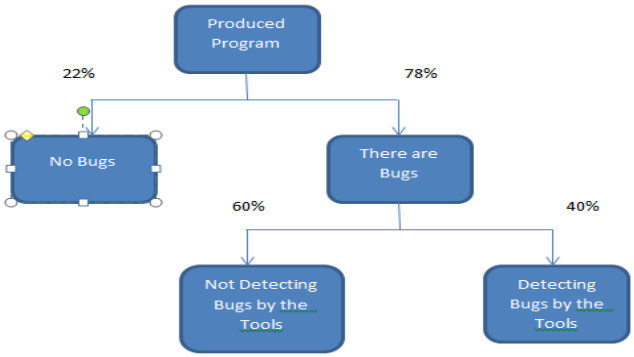

Fig. 29. The probability of finding bugs after using static analysis tools in the small company

Medium Company: The amount that could be lost if the small company does not use static analysis tools is $-409425.0 and 40% of this amount will be saved, which is $163770, if the company uses static analysis tools (See Figure 30).

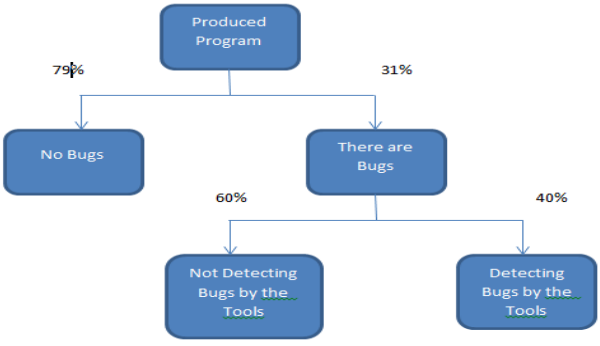

Fig. 30. The probability of finding bugs after using static analysis tools in the medium and big company

Big Company: The amount that could be lost if the big company does not use static analysis tools is $-821925.0 and 40% of this amount will be saved, which is $328770, if the company uses static analysis tools (See Figure 30).

All companies could save money if they apply static analysis tools. However, as explained before these tools sometimes need a very large space of servers and these servers need maintenance and upgrading frequently. Therefore, companies could save more money if they used the tools in cloud computing.

6.7. Evaluation Against the Objectives

1. Review: The review of the state-of-the-art technology in cloud computing, software engineering tools and some technologies that are used in this dissertation are discussed in Section 2.
2. Selection: Software engineering tools and clouds are selected and explained in Section 5 which is the approach section.
3. Define evaluation scenarios and criteria: The scenario for three companies is defined in Section 3 and the criteria were to decide whether or not using the tools in the cloud is useful.
4. Deployment: Three tools are deployed in the cloud in two approaches which are explained in Section 5.
5. Workflow: The tools are linked together to allow the users to move from one to the other easily.
6. Evaluation: The investigation of the scenarios is carried out in Sections 3 and 6.6.

7. Conclusion & Further Work

In this paper, the deployment of software engineering tools in the cloud have been demonstrated and evaluated. The state-of-the-art technology in the cloud has been reviewed, and a comparison made between cloud providers to select one which is appropriate for this dissertation. Evaluation scenarios were defined for three companies as a motivation for the research. Two approaches were explained with regard to using software engineering tools in the cloud and both of these were demonstrated. The first approach is indirect interaction and the second approach is direct interaction. Three tools were selected which are JSHint, JSLint and JSCoverage tools, and deployed in AppEngine in the indirect interaction approach and they are deployed in AWS for the second approach. After that, the tools were linked together as workflow for both approaches. At the end of the dissertation, the whole project was evaluated.

If more time were available, the following may be achieved:

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

Fig. 30. The probability of finding bugs after using static analysis tools in the medium and big company

Big Company: The amount that could be lost if the big company does not use static analysis tools is $-821925.0 and 40% of this amount will be saved, which is $328770, if the company uses static analysis tools (See Figure 30).

All companies could save money if they apply static analysis tools. However, as explained before these tools sometimes need a very large space of servers and these servers need maintenance and upgrading frequently. Therefore, companies could save more money if they used the tools in cloud computing.

6.7. Evaluation Against the Objectives

1. Review: The review of the state-of-the-art technology in cloud computing, software engineering tools and some technologies that are used in this dissertation are discussed in Section 2.
2. Selection: Software engineering tools and clouds are selected and explained in Section 5 which is the approach section.
3. Define evaluation scenarios and criteria: The scenario for three companies is defined in Section 3 and the criteria were to decide whether or not using the tools in the cloud is useful.
4. Deployment: Three tools are deployed in the cloud in two approaches which are explained in Section 5.
5. Workflow: The tools are linked together to allow the users to move from one to the other easily.
6. Evaluation: The investigation of the scenarios is carried out in Sections 3 and 6.6.

7. Conclusion & Further Work

In this paper, the deployment of software engineering tools in the cloud have been demonstrated and evaluated. The state-of-the-art technology in the cloud has been reviewed, and a comparison made between cloud providers to select one which is appropriate for this dissertation. Evaluation scenarios were defined for three companies as a motivation for the research. Two approaches were explained with regard to using software engineering tools in the cloud and both of these were demonstrated. The first approach is indirect interaction and the second approach is direct interaction. Three tools were selected which are JSHint, JSLint and JSCoverage tools, and deployed in AppEngine in the indirect interaction approach and they are deployed in AWS for the second approach. After that, the tools were linked together as workflow for both approaches. At the end of the dissertation, the whole project was evaluated.

If more time were available, the following may be achieved:

- Users could be allowed to upload their programs to the cloud database as text file which can be read by the tools in the cloud and save the outputs in the database as well. This approach may be much more efficient than sending the programs manually to each individual tool. Especially, when the users want to verify large code fragments.
- An access control method could be implemented to prevent unauthorized users to access the tools in AppEngine directly.
- A program could be implemented to make a comparison between the verification tools to display the differences between them.

Acknowledgments

This research project would not have been possible without the support of many people. The author wishes to express his gratitude to his supervisor Prof. John Fitzgerald who was abundantly helpful and offered invaluable assistance, support and guidance. The author would also like to convey thanks to Newcastle University for providing laboratory facilities and libraries. The author wishes to avail himself of this opportunity to express a sense of gratitude and love to his mother for her manual support, strength and help and for everything. Last but not least the author wishes to express gratitude to his wife for her understanding and endless love through the duration of his studies.

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

8. References

1. Mell, P., Grance, T.: (Published: Jan 2011, Accessed: 21st July 2012).The NIST Definition of Cloud Computing. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
2. Grossman, R.L.; "The Case for Cloud Computing," IT Professional , vol.11, no.2, pp.23-27, March-April 2009
3. Wang, L., G., Laszewski, V.: "Cloud Computing: a Perspective Study." New Generation Computing 28(2): 137-146. (2010).
4. Burton S., Kaliski, Jr. and Pauley, W.: Toward risk assessment as a service in cloud environments. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10). USENIX Association, Berkeley, CA, USA, 13-13. (2010)
5. Hofmann, P.; Woods, D.; , "Cloud Computing: The Limits of Public Clouds for Business Applications," Internet Computing, IEEE , vol.14, no.6, pp.90-93, Nov.-Dec. 2010
6. Kandukuri, B., Paturi, V., Rakshit, A., "Cloud Security Issues," Services Computing, 2009. SCC '09. IEEE International Conference on , vol., no., pp.517-520, 21-25 Sept. 2009
7. Zhang, Q., Cheng, L.,"Cloud computing: state-of-the-art and research challenges." Journal of Internet Services and Applications 1(1): 7-18. (2010).
8. SalesForce.com. (Published: 2011, Accessed: 8th April 2012). SalesForce. Available: http://www.salesforce.com/uk/
9. Chou, T, 2011, Introduction to CloudComputing. 2nd ed. United State: Active Book Press.
10. Appengine.google.com. (Published: 2012, Accessed: 8th April 2012). AppEngine. Available: https://appengine.google.com/
11. Amazon Web Services. (Published: 2012, Accessed: 8th May 2012). Amazon Elastic Compute Cloud (Amazon EC2). Available: http://aws.amazon.com/ec2/
12. Appengine.google.com. (Published: 2012, Accessed: 8th April 2012). AppEngine. Available: http://code.google.com/status/appengine
13. boomi.com/. (Published: 2012, Accessed: 8th April 2012). boomi. Available: http://www.boomi.com/
14. AT&T. (Published: 2012, Accessed: 8th April 2012). AT&T. Available: https://www.synaptic.att.com/
15. bimixs. (Published: 2012, Accessed: 8th April 2012). bimixs. Available: https://www.bimixs.com/metro/apps/MainFrame.htm
16. citrix. (Published: 2012, Accessed: 8th April 2012). citrix. Available: http://www.citrix.com/lang/English/lp/lp_2313912.asp
17. jumpbox. (Published: 2012, Accessed: 8th April 2012). jumpbox. Available: http://www.jumpbox.com/
18. Wichmann, B. A., A. A. Canning, D. L. Clutterbuck, L. A. Winsbarrow, N. J. Ward, and D. W. R. Marsh Industrial Perspective on Static Analysis. Software Engineering Journal Mar. 1995: 69-75. Available: http://www.ida.liu.se/~TDDC90/papers/industrial95.pdf
19. findbugs. (Published: 2012, Accessed: 8th April 2012). findbugs. Available: http://findbugs.sourceforge.net/

20. pmd. (Published: 2012, Accessed: 20th April 2012). pmd. Available: http://pmd.sourceforge.net/pmd-5.0.0/
21. Nagappan, N and Ball, T., Static analysis tools as early indicators of pre-release defect density. In Proceedings of the 27th international conference on Software engineering (ICSE '05). ACM, New York, NY, USA, 580-586. 2005.
22. Yang,Q. , Jenny Li, J., and Weiss, D: A survey of coverage based testing tools. In Proceedings of the 2006 international workshop on Automation of software test (AST '06). ACM, New York, NY, USA, 99-103. 2006.
23. agitar.com. (Published: 2012, Accessed: 25th April 2012). agitar. Available: http://www.agitar.com/
24. lynuxworks.com. (Published: 2012, Accessed: 27th April 2012). lynuxworks. Available:http://www.lynuxworks.com/partners/show_product.php?ID=268
25. Roger S.. Software Engineering. Mc Graw Hill International Edition. 2001.
26. Hollingsworth, D.: Workflow Management Coalition The Workflow Reference Model. 1.1. 1995.
27. D.P. Spooner, J., Cao, S. A. Jarvis, L. He, and G. R. Nudd. Performance-aware Workflow Management for Grid Computing. The Computer Journal, Oxford University Press, London, UK, 2004.
28. Huo M., Verner J., Zhu L., Ali Babar M.: Software Quality and Agile Methods, compsac,vol. 1, pp.520-525, 28th Annual International Computer Software and Applications Conference, COMPSAC'04, (2004).
29. Khajeh-Hosseini, A., Greenwood, D., Sommerville, I. , "Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS," Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on , vol., no., pp.450-457, 5-10 July 2010
30. Armbrust, M., Fox, A., Griffith, R., Anthony D. Joseph, Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. : A view of cloud computing. Commun. ACM 53, 50-58, April 2010.
31. Crockford,D.,: Javascript: The Good Parts. O'Reilly Media, Inc.,2008.
32. Yang, Q.,Jenny J. and Weiss, D. : A survey of coverage based testing tools. In Proceedings of the 2006 international workshop on Automation of software test (AST '06). ACM, New York, NY, USA, 99-103. 2006.
33. COCOMO, (no date), Accessed 15th Aug , Available : http://www.softstarsystems.com/overview.htm
34. COCOMO, (no date), Accessed 15th Aug , Available : http://cost.jsc.nasa.gov/cocomo.html
35. Stroock, D. w., Probability Theory. 2nd ed. Dubai: Cambridge University, 2010.
36. JSHint, (no date), Accessed 15th Aug , Available : https://github.com/jshint/jshint/
37. JSLint, (no date), Accessed 15th Aug , Available : https://github.com/douglascrockford/JSLint
38. JSCoverage, (2010), Accessed 15th Aug , Available : http://siliconforks.com/jscoverage/manual.html
39. Iqbal, M.; Rizwan, M.; , "Application of 80/20 rule in software engineering Waterfall Model," Information and Communication Technologies, 2009. ICICT '09. International Conference on , vol., no., pp.223-228, 15-16 Aug. 2009
40. Advances in Computer Science and Information Technology. Computer Science and Engineering Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2012.

41. Essence of Waterfall: (no date), Accessed 15th Aug , Available : http://ezinearticles.com/?Essence-Of-Waterfall-Model&id=352291
42. Amazon, (no date), Accessed 15th Aug , Available : http://aws.amazon.com/ec2/
43. Goecks, J., Nekrutenko, A., Taylor, J. and Team, T. G.: Galaxy: a Comprehensive Approach for Supporting Accessible, Reproducible and Transparent Computational Research in the Life Sciences: Genome Biology, vol. 11, p. R86, (2010).
44. Bessey, A.: A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. Commun. ACM, pp. 66-75, February 2010.
45. Ratzinger, J., Sigmund, T. and Harald C. Gall. On the relation of refactorings and software defect prediction. In Proceedings of the 2008 international working conference on Mining software repositories (MSR '08). ACM, New York, NY, USA, 35-38. 2008.
46. Riungu, L., Taipale, O., Smolander, K., "Research Issues for Software Testing in the Cloud," Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on , vol., no., pp.557-564, Nov. 30 2010-Dec. 3 2010.
47. Riungu, L., Taipale, O., Smolander, K. , "Software Testing as an Online Service: Observations from Practice," Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on , vol., no., pp.418-423, 6-10 April 2010.
48. Sharon, D.; Anderson, T.; "A complete software engineering environment," Software, IEEE , vol.14, no.2, pp.123-125, Mar/Apr 1997.
49. Kaur, P. and Kaushal, S., Security Concerns in Cloud Computing High Performance Architecture and Grid Computing. A. Mantri, S. Nandi, G. Kumar and S. Kumar, Springer Berlin Heidelberg. 169: 103-112. (2011).
50. Takabi, H.; Joshi, J., Ahn, G.,"Security and Privacy Challenges in Cloud Computing Environments," Security & Privacy, IEEE , vol.8, no.6, pp.24-31, Nov.-Dec. 2010.
51. Ruohomaa, S. and Kutvonen, L., Trust Management Survey Trust Management. P. Herrmann, V. Issarny and S. Shiu, Springer Berlin / Heidelberg. 3477: 77-92. (2005).
52. E-Science centre, Publish: (2012), Accessed 15th Aug , Available : http://www.esciencecentral.co.uk/
53. Ramgovind, S., Eloff, M., Smith, E., The management of security in Cloud computing. Information Security for South Africa (ISSA). 2010 vol., no., pp.1-7, 2-4 Aug. (2010).
54. Desktopanywhere, (2012), Accessed 15th Aug , Available : https://www.desktopanywhere.com/
55. Amazon, (2012), Accessed 15th Aug , Available : http://calculator.s3.amazonaws.com/calc5.html
56. Amazon, (2012), Accessed 15th Aug , Available : http://aws.amazon.com/security/
57. Piyush, J., Ramarkrishna, R., Sathyanad, B., Challenge in deploying static analysis tools, (no date), Accessed 15th Aug , Available : http://www.crosstalkonline.org/storage/issue-archives/2011/201107/201107-Jain.pdf.

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

Deployment of Software Engineering Tools as Cloud Services

Nawaf Almutairi
School of Computing Science, Newcastle University
n.m.m.al-mutairi@newcastle.ac.uk

Abstract. While much research has been carried out on the use of cloud computing, there has been little investigation of the cloud in software engineering. This study aims to demonstrate and evaluate software engineering tools in the cloud. In order to achieve this aim, technology in cloud computing and software engineering are reviewed and a workflow is created to facilitate the use of tools in the cloud for indirect and direct interaction approaches. Three differently sized case study scenarios are created to evaluate the use of the cloud in terms of cost, size and user efficiency. In addition, requirements of the service and users are explained and analysed. Consideration for the workflow design and interaction between users and the tools in the cloud are perceived as important aspects. This research also concerns the integration between the software engineering environment and the tools in the cloud and in particular how Scrum teams work with this integration. Evaluation of the results indicates that deployment of the cloud in the case study companies can be cost, time and user efficient.

Declaration: I declare that this dissertation represents my own work except where otherwise stated.

1. Introduction

Cloud computing has the potential to revolutionize computer technology, and because the cloud has several advantages, it has been used for many purposes in business and management [53]. Firstly, the cloud has qualities of elasticity and scalability, which refer to the ability to increase or decrease the number of resources automatically based on needs [4]. Moreover, it offers the ability to pay per unit of usage instead of buying, running and maintaining large servers, networks and storage [42]. In addition, developers can validate their programs through a number of standard mechanisms such as PDAs [4]. These benefits may attract developers to migrate their software engineering tools to cloud.

Providing an environment that is similar to a real environment needs a significant space for testing and verification to deal with the whole program. The results of verifications of large programs require servers with a large amount of space in order for these results to be displayed. This is clearly evident when trying to verify the small PROMELA code fragment (179 lines) (see Appendix A) in the Spin tool with a memory of 128 MB in the PC, it barely verifies, yielding the suggested action in Figure 1:

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

CSC8299 Deployment of Software Engineering Tools as Cloud Services

Individual Project

8/24/2012

Computer Security & Resilience
Student: Nawaf Almutairi
Student Number: 099098133
Prof. J. S Fitzgerald

Deployment of Software Engineering Tools as Cloud Services

Nawaf Almutairi
School of Computing Science, Newcastle University
n.m.m.al-mutairi@newcastle.ac.uk

Abstract. While much research has been carried out on the use of cloud computing, there has been little investigation of the cloud in software engineering. This study aims to demonstrate and evaluate software engineering tools in the cloud. In order to achieve this aim, technology in cloud computing and software engineering are reviewed and a workflow is created to facilitate the use of tools in the cloud for indirect and direct interaction approaches. Three differently sized case study scenarios are created to evaluate the use of the cloud in terms of cost, size and user efficiency. In addition, requirements of the service and users are explained and analysed. Consideration for the workflow design and interaction between users and the tools in the cloud are perceived as important aspects. This research also concerns the integration between the software engineering environment and the tools in the cloud and in particular how Scrum teams work with this integration. Evaluation of the results indicates that deployment of the cloud in the case study companies can be cost, time and user efficient.

Declaration: I declare that this dissertation represents my own work except where otherwise stated.

1. Introduction

Cloud computing has the potential to revolutionize computer technology, and because the cloud has several advantages, it has been used for many purposes in business and management [53]. Firstly, the cloud has qualities of elasticity and scalability, which refer to the ability to increase or decrease the number of resources automatically based on needs [4]. Moreover, it offers the ability to pay per unit of usage instead of buying, running and maintaining large servers, networks and storage [42]. In addition, developers can validate their programs through a number of standard mechanisms such as PDAs [4]. These benefits may attract developers to migrate their software engineering tools to cloud.

Providing an environment that is similar to a real environment needs a significant space for testing and verification to deal with the whole program. The results of verifications of large programs require servers with a large amount of space in order for these results to be displayed. This is clearly evident when trying to verify the small PROMELA code fragment (179 lines) (see Appendix A) in the Spin tool with a memory of 128 MB in the PC, it barely verifies, yielding the suggested action in Figure 1:

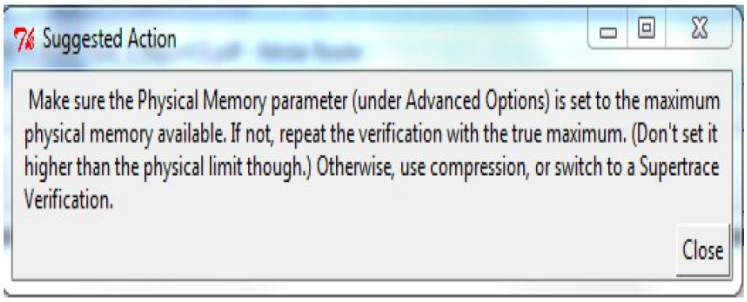

Fig. 1. Suggested action from Spin tool

The cloud can offer any infrastructure such as operating system [42]. Therefore, users can select the same infrastructure that would be found in the real environment. For example, some projects work in UNIX and the tester can select the same operating system in the cloud to test the projects.

The provision of big servers costs companies a significant amount of money, whereas cloud users only pay for what they use. These big servers also need maintenance and updates frequently. Nevertheless, the advantages of the cloud can handle all these difficulties.

The process of testing can be carried out by a large group of people [45]. Therefore, the software can be tested by many testers at the same time. This is very useful because it provides an outstanding opportunity to cover all the functions of the software. Developers can upload their programs in the cloud and ask testers from offshore locations to undertake testing and provide feedback on the programs.

Unfortunately, software engineering technology has not been widely deployed in cloud. Hence, the aim of this project is to demonstrate the deployment of this technology in the cloud. However, developers might use more than one tool in order to achieve high-quality results. Using a variety of tools requires them to be organized and connected together by workflow in order for them to be efficient and usable.

1.1. Objectives

1. **Review**: review the state-of-the-art technology in cloud computing and software engineering tools. The success criteria for this objective would be by providing a review of the stat-of-the-art technology in cloud computing and software engineering tools.
2. **Selection**: select cloud computing and software engineering tools for the project. The success criteria for this objective would be by selecting cloud provider and software engineering tools
3. **Define evaluation scenario and criteria:** assume scenario to discuss whether or not the deployment is helpful in this scenario. The success criteria for this objective would be by providing scenarios to discuss whether or not the deployment is helpful in these scenarios.

4. **Deployment**: demonstrate and evaluate the deployment of the software engineering tools in the cloud. The success criteria for this objective would be by providing demonstration and evaluation of the deployment of the software engineering tools in the cloud.
5. **Workflow**: create processes that could help the developers to validate and test their programs and move from tool to tool. The success criteria for this objective would be by creating processes that could help the developers to validate and test their programs and move from tool to tool.
6. **Evaluation**: investigate the scenario and evaluate the outputs and help decision-makers decide whether or not the deployment could be helpful. The success criteria for this objective would be by providing evaluation of the scenarios and make decision on whether or not the deployment is useful.

1.2. Methodology

This project will focus on the demonstration and evaluation of the deployment of software engineering tools in the cloud. The study provides three scenarios for three companies in different areas of business and with different development environments. These scenarios will be analysed before they used as the basis for evaluating the cloud deployment. A comparison will be made between different service models of cloud providers to select the appropriate cloud to suit the deployment. One of the important aspects which must be understood in the selection of the cloud is the development environment that allows the tools to be run. Next, the software engineering tools will be chosen for deployment.. Subsequently, the cloud will be used to deploy the selected tools and the process will then be created to allow the users to move from tool to tool. Finally, the project will be evaluated based on the scenarios which are assumed before and then investigating these scenarios to decide whether or not the deployment is helpful.

1.3. Structure of Dissertation

Section 2 provides the background to the study and reviews the technologies used. The scenarios and evaluation criteria motivating the work are described in detail in Section 3. The requirements needed to achieve the main project goal of effective cloud deployment of software engineering tools are discussed in Section 4 and the main technical approaches that have been implemented are described in Section 5. The evaluation (Section 6) is followed by conclusions and suggestions for further work (Section 7).

2. Background

This section provides definitions of key terms of the main technologies used in this study. It defines cloud computing, providing an overview of its advantages and disadvantages and describes the different service models.. Further, it defines and describes the main software engineering technologies and workflow and outlines their benefits.

2.1. Cloud Computing

2.1.1. What is the cloud?

There is a non-standard definition of cloud computing [2]. Nonetheless the National Institute of Standards and Technology defines it as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]. Cloud computing has been the focus of much research because it offers flexible dynamic IT infrastructures, computing environments and efficient software services [3].

2.1.2. What are the advantages of the cloud?

The cloud has five main advantageous characteristics [4]:

- **On-demand self-service:** users can service themselves without needing a supporting team.
- **Broad network access:** users can access their assets by means of the Internet through different devices.
- **Resource pooling:** many users can access the same data at the same time.
- **Rapid Elasticity:** the flexibility of controlling the usability of the cloud. In other words, consumers can increase and decrease the capability of accessing and using the resources in the cloud.
- **Measured Service:** Users can measure the usability of the cloud and be charged per use. Therefore, consumers can monitor and control usage.

2.1.3. What are the disadvantages of cloud computing?

There are some drawbacks of using cloud computing. However, the following are the major disadvantages which could affect this dissertation.

- **Security**: Each company has security standards which are hard to meet by cloud providers [5]. Each cloud provider has certain features which may not satisfy all specifications for all customers.
- **Data Location and Privacy:** physical location of the servers is very important because different countries have different laws and some companies want to store their data in the same way as their privacy management laws in their countries [6].
- **Privilege User Access:** Sensitive data can be managed by non-employees. It means that cloud servers are not under the control of organizational managers [6].
- **Internet Dependency, performance and Latency:** availability is a crucial aspect for consumers and it could be affected by Internet providers [6].

2.1.4. Cloud Computing Principal Service Models

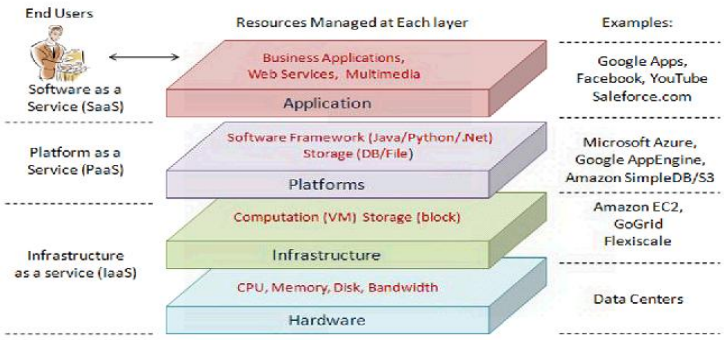

Fig.2. cloud computing architecture [7]

Figure 2 shows the three models and the following are their descriptions:

Software as Service (SaaS): Users of the cloud are allowed to install, operate and run applications in the cloud without managing the infrastructure and the platform of the cloud. The difference between the SaaS cloud's applications and normal PCs' applications is that the cloud applications offer elasticity and can be accessed by more than one user, while PC applications are only accessed by its users [9]. Moreover, the consumers can access the cloud applications from different devices such as mobile phones. Examples of the SaaS service are SalesForce.com [8], YouTube and Facebook. The customers of this service pay monthly for using the cloud.

Platform as Service (PaaS): This allows users to implement and deploy applications by providing a programming environment, for instance Java Runtime Environment (JRE). However, users cannot control or manage the infrastructure of the cloud such as the operating system of the cloud. A good example of this model is Google App Engine. It offers a good environment for running Java, JavaScript and Python applications [10]. Usually, the customers of this service pay per use.

Infrastructure as Service (IaaS): The customers of this model are able to build the whole cloud services starting from the infrastructure ending with applications. They can control storage, Internet, processing and the Operating System images, however they do not own the hardware used to support the cloud [11]. An example of this cloud is Amazon EC2 and it is on a pay-per-use basis.

2.2. Software Engineering

The term "software engineering" (SE) encompasses the activities to analyse, design, construct and test software [25]. A software engineering environment (SEE) supports the tasks of all software engineering processes [48]. Such an environment has many tools to support these tasks.

2.2.1. Software Engineering Tools

The purpose of software engineering tools is to "assist software engineering managers and practitioners in every activity associated with the software process" [18]. Developers can use these tools to analyse, design, implement and test their programs. They can assist developers to easily achieve high quality programs. The quality of software is the key factor in companies that depend upon programs for their work [32]. The following are some of these tools and a brief definition of each of them.

Static Analysis Tools: Static analysis tools can find any indication of bugs in programs before execution [18]. Examples of these tools are Findbugs [19] and PMD [20]. They do not only discover errors but they can report functional and designer errors at a deep level [21].

Testing Tools: Testing tools are used to improve software quality by practising and coordinating testing cases [22]. In other words, they allow users to input test data and verify the output whether it is expected or not. An example of this tool is Agitar [23] and CodeTest [24].

Dynamic Analysis Tools: Dynamic analysis tools deal with running programs to assure that program behaviour is running exactly as expected [25]. These tools interact with executing software, whilst normal testing tools test software in fake environments. There are two types of dynamic tools [25]:

- Intensive tools: they provide extra instruction that is written with the software to ensure that it runs acceptably.
- Non-intrusive tools: these use extra hardware processes that execute in parallel with the process that runs the main software.

Analysis and Design Tools: They allow users to build models of the system that represent data, function and behaviour and characterization of the data, interface design and component-level [25].

Coverage Tools: Coverage tools show the percentage of the functions that were not used [38]. For example, if a user uses only 40% of the functions, the tools will display 60%. This is beneficial in testing because it shows which functions have not been tested, allowing the user to go back and test them.

2.2.2. Software Engineering Methods

There are many software engineering models. However, the following explains two of them because they are used in this dissertation. The Waterfall model has been improved to the iterative model and this improvement will be described in this section.

Fig. 3. Waterfall model

Due to the Waterfall model being able to improve the work-task between the project teams, this model was able to reduce the effort of the developer and increase the performance of producing programs [39]. However, after using this model, it has been identified by means of the imperfect results that the model is inadequate [39] because it cannot go back to previous stages if there are any problems in any of the phases, thus, this issue may be resolved by iterating this model until accomplishing high quality programs [40] see Figure 3.

Fig. 4. Iterative model

Working in the iterative model's phases (see Figure. 4) has some features that could be considered. Each team in each phase has certain tools to help it in their task [25]. For example, the verification team has static analysis tools to verify the program after producing it from the implementation team. Every phase is separated and the outputs of the previous phase are the input for the next phase, for instance, the outputs of the implementation phase is the inputs of the verification phase. The verification phase decides whether or not the programs are ready for release. If the program is not ready the team will return the program with a report to the requirements team.

2.3. Workflow

Hollingsworth indicates that workflow is ‘‘concerned with the automation of procedures where documents, information or tasks are passed between participants” [26]. It concludes connected steps and each step is finished before the next step begins. Potential benefits of workflow models are that [27]:

- It is able to organize and manage distributed systems.
- It provides an approach for interaction between organizations.
- It increases outputs and decreases costs by using resources in certain domains.
- It includes a variety of administrative domains to acquire throughput.
- It has the ability to integrate between multiple teams and applications.

3. Evaluation Scenario and Criteria:

This section is focused in definition of basis scenarios for evaluating the deployment of software engineering tools in the cloud. It begins by setting the assumed cloud usage cost before going on to explain the basis for cost estimation using COCOMO. The three scenarios are then introduced.

Before proceeding further with the investigation of the three scenarios about three companies, it is essential to define the assumed cost of renting the cloud after deploying software engineering tools to help in its evaluation. After deploying these tools in the cloud, the cloud model would be the same as any SaaS model. Thus it is worthwhile to obtain the average cost of applications in the SaaS cloud to ascertain the suspected renting price. Consequently, evaluation will be based on this price. The costs of three providers are:

1- Zoho: It costs $ 40 monthly for 5 GB.
2- Opsource: It costs $49.64 monthly for 5 GB.
3- Rockspace: It costs $ 33.75 monthly for 5 GB.

Therefore, the average of all these prices is (40 + 49.64 + 33.75)/3= 41.13. This is nearly $ 41 per month for 5 GB.

COCOMO standard, which is a Constructive Cost Model, will calculate the estimated price for creating software [33]. This model divides the software projects into three categories which are Organic, Semi-detach and Embedded. This study creates a scenario where the small company deals with Organic projects, the medium company deals with Semi-detach projects and the big company deals with Embedded projects. COCOMO tools will be used to calculate the three formulas [34]. The COCOMO is based on three formulas [34]:

Effort Applied (E) = $a_b(KLOC)^{b}_{b}$ **[man-months]**

Development Time (D) = $c_b(\text{Effort Applied})^{d}_{b}$ **[months]**

People required (P) = Effort Applied / Development Time **[count]**

Where the following table provides a_b, b_b, c_b and d_b in Table1:

Table1. Given variables for COCOMO formulas

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

According to Jacek, Thomas and Herald, the percentage of finding defects in a program in a real environment will be approximately 78% if refactorings are not applied in the development environment [45]. Refactoring improves the code by reducing complexity and making the programs readable to help developers to maintain their programs. However the percentage would be approximately 31% if refactorings are applied [45]. This research assumes that the small company does not apply a refactorings technique, while the medium and big companies do.

All these companies depend upon the iterative model. SEE in all companies in this study do not have any software engineering tools. In the verification phase, they verify the programs manually by writing acceptance test documents and test the programs without any tools. Once the functionalities of the programs satisfy the acceptance test, they will release the programs; otherwise they will return them to the requirement phase.

It is assumed that all three companies employ the Scrum concept. Scrum is a software development process that divides developers into small teams, and each team can do all the work that is related to any program [48]. The team can handle the program from the first phase of the Iterative model to the last phase. A team's roles are:

Product owner: who is responsible for customer requirements.

Development team: Its tasks are analysis, design, development, testing and verification.

Scrum Master: he is the team leader.

In the small company the task is not big. Thus, it can be done by one Scrum team and then be released without integration. The task in the medium and big companies

is divided into small parts and each Scrum team is assigned a different part. After finishing the assigned task, one of the Scrum teams integrates the parts together (see Figure 5).

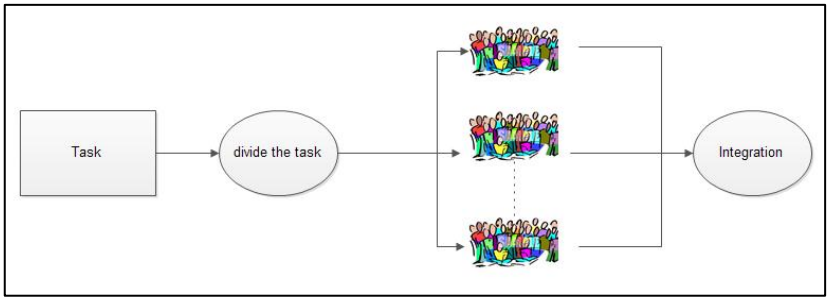

Fig. 5. Dividing task and integration

3.1. First Scenario

The small company has between 1 to 20 employees (2 Scrum teams). This team is highly familiar with software language. The team works at home and the average salary for them is $1000. The average of the delivered source code is one KDSI for one month, which is one thousand, delivered source instructions for one month. The COCOMO tool is used to calculate the efforts and development cost. Figure 6 shows the inputs in the COCOMO tool and Figure 7 shows the outputs.

Inputs

Development	
Delivered Source Instructions (thousands) (KDSI)	1
Development Mode	Organic
Average Cost Rate ($/PM)	1000
Maintenance	
KDSI added (annual)	0
KDSI modified (annual)	0
Average Cost Rate ($/PM)	0

Recalc

Fig. 6. The inputs of the COCOMO tool for first scenario.

Results

Effort	2	person-months (PM)
Schedule	3	months
Development Cost	2000	
Productivity	500	instructions per person-month
Average Staffing	0.7	full-time-equivalent software personnel
Annual Maintenance Effort	0	person-months
Annual Maintenance Cost	0	

Fig. 7. The outputs of COCOMO tool for the first scenario.

Consequently, the estimated payment for implementing one KDSI is $2000, and this payment is only for two developers. However, the study assumes that after the customer returns the code fragment, because of bugs, the company needs the effort of one developer for one month. Therefore, the cost of fixing the bugs is $1000. The percentage of finding bugs by customers in produced programs is 78% (see Figure 3). Assuming that the company sells 12 KDSI annually, thus, the annual operating expense is $24000, which is if the customers do not return the software because of defects.

Evaluation: To make a decision on whether or not software engineering tools are useful, it is beneficial to calculate probability factors to provide evidence that could help to make a decision [35]. The probability would be calculated by running 12 random cases which represent 12 months and based on Figure 8. Running the random cases will be done by a Java program.

```
int run = 12; // one year
int a = 0,b = 0;
Random randomGenerator = new Random();
For (int i=0;i<run;++i)
{

double randomNum= randomGenerator.nextDouble();
if (randomNum>0.78){
a++; // No bugs

}else b++; // Find bugs by customer
}

double percentageOfNoBugs= (a/(double)run)*100;
double percentageOfFindBugsByCustomer= (b/(double)run)*100;
```

```
double RewardA= percentageOfNoBugs *(+41);
double penaltyB=percentageOfFindBugsByCustomer*(-1000);
System.out.println(" the company will lose  = $ " +penaltyB + " ,if it does not rent the cloud service " );
System.out.println(" the company will earn = $ " +RewardA + " ,if it does not rent the cloud service " );
double utility = RewardA + penaltyB;
System.out.println("the final result is $"+ utility);
```

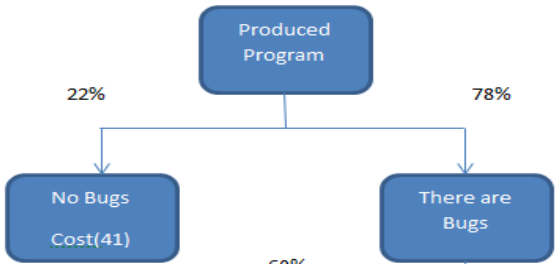

Fig. 8. Probability tree

Result:

```
  the company will lose  = $ -75000.0,if it does
not rent the cloud service
   the company will earn = $ 1025.0,if it does
not rent the cloud service
  the final result is $-73975.0
```

Fig. 9. The results of the probability program

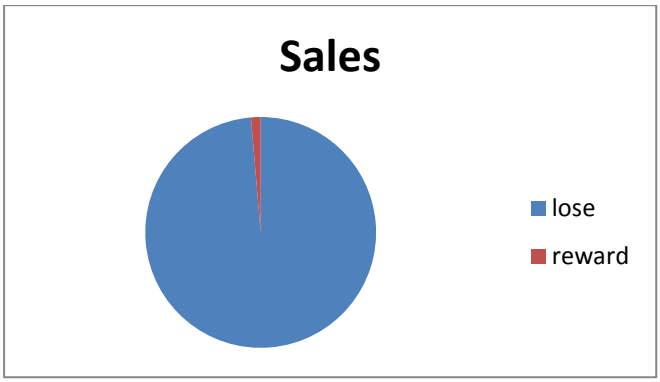

Fig. 10. Annual profit in the case of not renting the tools in the cloud for the small company

Figures 9 and 10 show the outputs of running 12 random cases. The utility is $-73975.0 which is the amount that could be lost if the company does not rely upon software engineering tools to verify and test their code fragment in the cloud. By adding the annual operating expense to this amount, the final operating expense is $97975, an increase of 75%. Therefore, there is a significant negative of not renting software engineering tools in the cloud for the small company in terms of the economic aspect.

3.2. Second Scenario

The medium company has between 20 and 50 employees (2 to 5 Scrum teams). This company can develop big and small code fragments. The average salary for them is $1500. The company produces approximately 6 KDSI monthly. Assuming that, the team members are professionals and they can implement high quality programs by applying the refactorings mechanism in the software. Therefore, the percentage of finding defects in the programs by customers is 31%. The COCOMO tool is used to calculate the effort and development cost. Figure 11 shows the inputs in the COCOMO tool and Figure 12 shows the outputs.

Inputs

Development	
Delivered Source Instructions (thousands) (KDSI)	6
Development Mode	Semidetached
Average Cost Rate ($/PM)	1500
Maintenance	
KDSI added (annual)	0
KDSI modified (annual)	0
Average Cost Rate ($/PM)	10000

Recalc

Fig. 11. The inputs of the COCOMO tool for the second scenario.

Results

Effort	22	person-months (PM)
Schedule	7	months
Development Cost	33000	
Productivity	273	instructions per person-month
Average Staffing	3.1	full-time-equivalent software personnel
Annual Maintenance Effort	0	person-months
Annual Maintenance Cost	0	

Fig. 12. The outputs of the COCOMO tool for the second scenario.

Consequently, the estimated payment for implementing 6 KDSI is $33000, and this payment is only for 22 developers. However, this study assumes that after the customers return the code fragment, because of bugs, the company needs the effort of half of this number which is 11 developers. Therefore, the cost of fixing the bugs is $16500. Assuming that the company sells 72 KDSI annually, thus, the annual operating expense is $396000, which is if the customers do not return the software because of defects.

Evaluation: The probability is calculated by running 12 random cases which represent 12 months. Figure 13 shows the probability tree which will be run 12 times. The random number will be generated to discover the probability. If the number is more than 79 that means that the customers find defects, on the other hand, if it is less than 79 that means that the customers do not find defects. The Java program will be implemented to calculate the probability.

```
int run = 12; // one year
int a = 0,b = 0;
Random randomGenerator = new Random();
For (int i=0;i<run;++i)
{

double randomNum= randomGenerator.nextDouble();
if (randomNum>0.31){
a++; // No bugs

}else b++; // Find bugs by customer
}

double percentageOfNoBugs= (a/(double)run)*100;
double percentageOfFindBugsByCustomer= (b/(double)run)*100;

double RewardA= percentageOfNoBugs *(+41);
double penaltyB=percentageOfFindBugsByCustomer*(-16500);
System.out.println(" the company will lose  = $ " +penaltyB + ", if it does not rent
the cloud service " );
System.out.println(" the company will earn = $ " +RewardA + ", if it does not rent
the cloud service " );
double utility = RewardA + penaltyB;
System.out.println("the final result is $"+ utility);
```

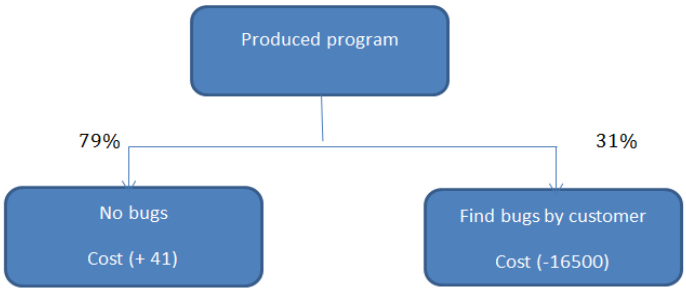

Fig. 13. Probability tree

Result:

```
  the company will lose  = $ -412500.0,if it
does not rent the cloud service
  the company will earn = $ 3075.0,if it does
not rent the cloud service
  the final result is $-409425.0
```

Fig. 14. The program output

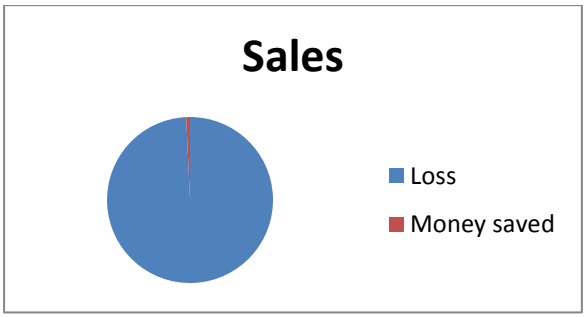

Fig. 15. Annual profit in case of not renting the tools in the cloud for the medium company

Figures 14 and 15 show the outputs of running 12 random cases. The utility is $-409425.0 which is the amount that could be lost if the company does not rely upon software engineering tools to verify and test their code fragment in the cloud. By adding the annual operating expense to this amount, the final operating expense is $805425, an increase of 49%. Therefore, there is a significant negative of not renting software engineering tools in the cloud for the small company in terms of the economic aspect.

3.3. Third Scenario

The medium company has more than 50 employees (more than 5 Scrum teams). This company specializes in the implementation of big code fragments. The average salary for them is $2000. The company produces approximately 12 KDSI monthly. Assuming that, the team members are professionals and they can implement high quality programs by applying the refactorings mechanism in the software. Therefore, the percentage of finding defects in the programs by customers is 31%.The COCOMO tool is used to calculate the effort and development cost. Figure 16 shows the inputs in the COCOMO tool and Figure 17 shows the outputs.

Inputs

Development	
Delivered Source Instructions (thousands) (KDSI)	12
Development Mode	Organic
Average Cost Rate ($/PM)	2000
Maintenance	
KDSI added (annual)	0
KDSI modified (annual)	0
Average Cost Rate ($/PM)	10000

Recalc

Fig 16. The inputs of the COCOMO tool for the third scenario

Results

Effort	33	person-months (PM)
Schedule	9	months
Development Cost	66000	
Productivity	364	instructions per person-month
Average Staffing	3.7	full-time-equivalent software personnel
Annual Maintenance Effort	0	person-months
Annual Maintenance Cost	0	

Fig. 17. The outputs of the COCOMO tool for the third scenario

Consequently, the estimated payment for implementing 12 KDSI is $66000, and this payment is for the effort of 33 developers (see Figure 13). However, this study assumes that after the customers return the code fragment, because of bugs, the company needs the effort of half of this number which is 16.5 developers. Therefore, the cost of fixing the bugs is $ 33000. Assuming that the company sells 144 KDSI annually, thus, the annual operating expense is $9504000, that is, if the customers do not return the software because of defects.

Evaluation: The probability will be calculated by running 12 random cases which represent 12 months. Figure 9 shows the probability tree which will be run 12 times. The random number will be generated to discover the probability. If the number is more than 79 that means customers find defects, on the other hand, if it is less than 79 that means that the customers do not find defects. The Java program will be implemented to calculate the probability.

```
int run = 12; // one year
int a = 0,b = 0;
Random randomGenerator = new Random();
For (int i=0;i<run;++i)
{

double randomNum= randomGenerator.nextDouble();
if (randomNum>0.31){
a++; // No bugs

}else b++; // Find bugs by customer
}

double percentageOfNoBugs= (a/(double)run)*100;
double percentageOfFindBugsByCustomer= (b/(double)run)*100;
```

```
double RewardA= percentageOfNoBugs *(+41);
double penaltyB=percentageOfFindBugsByCustomer*(-33000);
System.out.println(" the company will lose  = $ " +penaltyB + " ,if it does not rent the cloud sevice " );
System.out.println(" the company will earn = $ " +RewardA + " ,if it does not rent the cloud sevice " );
double utility = RewardA + penaltyB;
System.out.println("the final result is $"+ utility);
```

Result:

```
  the company will lose  = $ -825000.0, if it
does not rent the cloud service
  the company will earn = $ 3075.0, if it does
not rent the cloud service
  the final result is $-821925.0
```

Fig. 18. The program output

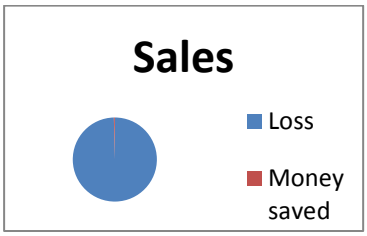

Fig. 19. Annual profit in case of not renting the tools in the cloud for the medium company

Figures 18 and 19 show the outputs of running 12 random cases. The utility is $821925.0 which is the amount that could be lost if the company does not rely upon software engineering tools to verify and test their code fragments in the cloud. By adding the annual operating expense to this amount, the final operating expense is $10325925,an increase of 92%. Therefore, there is a significant negative of not renting software engineering tools in the cloud for the small company in terms of the economic aspect.

3.4. Conclusion

The study in this chapter has illustrated the loss that might happen in small, medium and big companies if they do not use software engineering tools in the cloud. However, the above is based upon the assumption that these companies do not have any static analysis tools that could analyse and verify their programs. The COCOMO standard has been used to calculate the estimated cost of creating programs. The probabilities approach has been adopted to find the possible profit that may be generated or lost. These probabilities are justified by prior work in this field.

Table 2. Final result for all scenarios

Company	Utility	Operating expense	Percentage of increase
Small	$-73975.0	$ 97975	75%
Medium	$-409425.0	$ 805425	49%
Big	$-821925.0	$10325925	92%

Table 2 compares the three companies. It is apparent from this table that there is a significant effect in the big company because it produces a large size of code fragments without using any software engineering tools, whilst the medium company experiences the smallest effect because it applies the refactorings technique. Nevertheless, all of the companies have relatively high percentages with regard to the increase in operating expenses. Consequently, the results of the study in this section provide strong evidence for the claim that a company should treat their programs by using software engineering tools in the cloud.

However, this evaluation is only for one part of SEE which is to verify and test the programs before release. Therefore, it could be beneficial if the cloud were used in all SEE parts such as the design process.

4. Requirements

The purpose of this section is to explain the requirements that should be met to use software engineering tools in the cloud. These requirements also consider the SEE where Scrum teams work. The requirements are classified into four aspects according to the cloud and the SEE's components which are cloud requirements, design requirements, software engineering tools requirements and the requirements on customers. Although there are many requirements that could be specified, the following are some of those which rely on what we find in published papers and cloud documents

4.1. Cloud Requirements

The following are the requirements that should be offered to reap the benefits of using software engineering as a service in the cloud and to avoid any issues:

Pricing: It is expected that a company which wants to use the cloud has the ability to calculate the cost and make payments for using the cloud by means of invoicing and e-invoicing; it can still make payment by a variety of traditional methods such as PayPal [46].

Availability: The availability of the cloud needs to be very high. The reason for this is that some tools take a long time to perform their tasks [44].

Security, Privacy & Trust concerns: the assets of the users need to be held in a secure place [49]. Customers want their data to be fully protected and providers to apply proper security measures. Cloud security involves computer security, network security and information security [49]. **Privacy** is an important part in all the challenges that may be faced in cloud computing. Users want to ensure that their identities, information, policy components during integration and transaction histories are protected [50]. **Trust** is non-quantitative and difficult to measure [50]. However, it is defined as "the extent to which one party is willing to participate in a given action with a given partner, considering the risks and incentives involved" [51]. The cloud needs to be trustworthy to attract customers to use it.

Service Level Agreement (SLA) should be clear and satisfactory. Users should be aware of the terms and conditions governing the use of the cloud.

4.2. Design Requirements

Tight coupling between software engineering tools and user: This is required because these tools need to be much more interactive than the e-Science application, which runs an automatic workflow [43] [52].

Tools connection: The tools are connected together to allow users to move from one to the other efficiently..

Assessment on cloud viability: Before using the cloud in SEE, the program should be assessed by Scrum teams. Therefore, the design of the SEE should consider this assessment. The assessment should be based upon certain criteria to provide the Scrum teams with the ability to make a decision on whether or not they use the tools in the cloud.

4.3. Software engineering tools requirements

Usability: It is very important that end-users interact easily with the tools. The tools should allow users to specify the pattern of verifying and testing the program.

The tools are for a single program language: If selection of software engineering tools is for more than one tool, the tools should be for a single language.

4.4. Requirements on customers

Skills: Users should realize how they use the cloud resources [47]. If users do not know how they can use the cloud, they will face difficulty in reaping its benefits.

5. Approaches

We could argue that if the approach succeeds in one phase in the Iterative model, it should work with other phases. Since the key issue of losing money in the companies in Section 3 is in the verification phase because the programs are released before being adequately tested. Therefore, the selections concentrate on static analysis and testing tools. These selections consider the requirements in Section 4.

In order to achieve the main aim, we have two approaches. The first approach devised for this study is 'indirect interaction' and the second approach is 'direct interaction'.

5.1. Indirect Interaction Approach

The indirect interaction approach refers to users using cloud to run the tools which are hosted in another cloud (see Figure 20). The users use the tools through the first cloud and run the tools in it. It is necessary to select two cloud computing providers and tools in order to prove this concept.

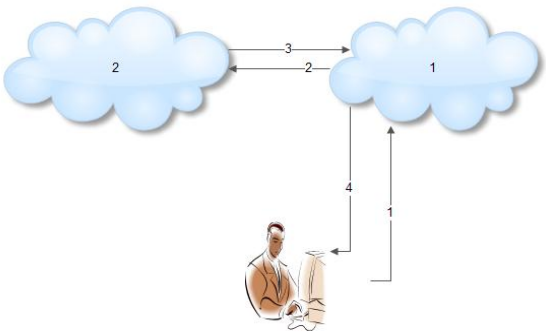

Fig. 20. Overview of Indirection Design

5.1.1. Selection cloud computing

The selections are for two clouds. The first cloud is for running the tools in it. The second cloud is for deployment of these tools.

First cloud: It was necessary to select the IaaS cloud to have the ability to build the infrastructure that was required. It is necessary that the cloud provider offer a proper image. The proper image refers to offering applications and an environment to run the tool. Most of the IaaS providers have infrastructures so that users can build the environment that they need. Therefore, Amazon provider (AWS) which is one of IaaS cloud was chosen in this study. The image of the cloud would be created after choosing the tools to select the necessary application.

Second cloud: Development environment is one of the most important aspects of selection cloud computing because it assists in running the deployed application in the cloud. This service is only in PaaS and IaaS clouds, while it is not in an SaaS cloud. Therefore, the SaaS cloud is eliminated from selection cloud options. However, if a user wants to use IaaS an infrastructure needs to be built which costs money and takes a long time [29]. It is not necessary to construct the cloud infrastructure to deploy tools whilst it is ready in PaaS cloud with an outstanding environment for deployment. Consequently, PaaS is the most suitable cloud for this project.

Table 3 shows the comparison between six PaaS cloud providers. This comparison considers selected criteria which are:

1- Support: Supporting users by a cloud provider and handling any issues free.
2- Security feature: protection for the users' assets by offering security features such as Firewall, back up storage and secure permissions.
3- Operating System: the OS which is used in providers' servers such as UNIX and Windows.
4- Cost. The minimum price.
5- Program language supported: the development environment that is in the cloud.

Table 3: Comparison between PaaS cloud providers

Cloud Provider	Support	Security Features	Operating System	Cost	Program Language
App Engine [10]	Yes	Backup storage	Linux and Windows Server 2008	Free	Python, Java and Go [12]
Boomi [13]	Yes	N/A	Cent, Windows server 2003 and Windows server 2008	$550/month* for 2 connections	APL and Java
AT&T Synaptic [14]	Yes	Backup Storage, Data protection	Linux and Windows Server 2008	Pay-as-you-go service.	No
BIMIXS [15]	Yes	Firewall, Backup storage	Windows Server 2008	Free	No
Citrix [16]	Yes	Data protection	Windows server 2003 and Windows server 2008	Estimated price	No
JumbBox [17]	Yes	Bootable Mode and Backup storage	Cent OS and Linux.	$49/month.	PHP, Python, SQL and Visual Basic.

It is clear that AppEngine, Boomi and AT&T Synaptic support some program languages while the others do not. This factor is very important because this project aims to perform verification and testing for program languages and the tools may be created by one of these languages. However, there are no significant differences between the services of the three providers and this project is not going to use a very large number of resources, hence selection is for a free provider which is AppEngine.

AppEngine's server is a virtual server [10]; this means that it has the quality of scalability which continues to work well even if it has a very large size of software engineering tools. Users do not need to book the space that they need before using the server because it is very difficult to decide how many bytes they need to verify and test the programs. As a result, AppEngine cloud will be helpful if it is used to verify the programs it runs.

5.1.2. Selection Software engineering tools

As this project aims to demonstrate and evaluate deployment Software Engineering tools in the cloud, it is beneficial to choose tools that are open source to manipulate and test them without any constraints. AppEngine is targeted to perform web applications [30], thus these tools need to be web-based applications. Java Script language is one of the most important web languages which can help to create web-based applications [31]. JavaScript is a client-side server; hence it does not run in the server. Therefore, this feature could be used to run the tools, which are implemented in JavaScript, in the first cloud in Figure 20 after deploying them in the second cloud.

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

8. References

1. Mell, P., Grance, T.: (Published: Jan 2011, Accessed: 21st July 2012).The NIST Definition of Cloud Computing. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
2. Grossman, R.L.; "The Case for Cloud Computing," IT Professional , vol.11, no.2, pp.23-27, March-April 2009
3. Wang, L., G., Laszewski, V.: "Cloud Computing: a Perspective Study." New Generation Computing 28(2): 137-146. (2010).
4. Burton S., Kaliski, Jr. and Pauley, W.: Toward risk assessment as a service in cloud environments. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10). USENIX Association, Berkeley, CA, USA, 13-13. (2010)
5. Hofmann, P.; Woods, D.; , "Cloud Computing: The Limits of Public Clouds for Business Applications," Internet Computing, IEEE , vol.14, no.6, pp.90-93, Nov.-Dec. 2010
6. Kandukuri, B., Paturi, V., Rakshit, A., "Cloud Security Issues," Services Computing, 2009. SCC '09. IEEE International Conference on , vol., no., pp.517-520, 21-25 Sept. 2009
7. Zhang, Q., Cheng, L.,"Cloud computing: state-of-the-art and research challenges." Journal of Internet Services and Applications 1(1): 7-18. (2010).
8. SalesForce.com. (Published: 2011, Accessed: 8th April 2012). SalesForce. Available: http://www.salesforce.com/uk/
9. Chou, T, 2011, Introduction to CloudComputing. 2nd ed. United State: Active Book Press.
10. Appengine.google.com. (Published: 2012, Accessed: 8th April 2012). AppEngine. Available: https://appengine.google.com/
11. Amazon Web Services. (Published: 2012, Accessed: 8th May 2012). Amazon Elastic Compute Cloud (Amazon EC2). Available: http://aws.amazon.com/ec2/
12. Appengine.google.com. (Published: 2012, Accessed: 8th April 2012). AppEngine. Available: http://code.google.com/status/appengine
13. boomi.com/. (Published: 2012, Accessed: 8th April 2012). boomi. Available: http://www.boomi.com/
14. AT&T. (Published: 2012, Accessed: 8th April 2012). AT&T. Available: https://www.synaptic.att.com/
15. bimixs. (Published: 2012, Accessed: 8th April 2012). bimixs. Available: https://www.bimixs.com/metro/apps/MainFrame.htm
16. citrix. (Published: 2012, Accessed: 8th April 2012). citrix. Available: http://www.citrix.com/lang/English/lp/lp_2313912.asp
17. jumpbox. (Published: 2012, Accessed: 8th April 2012). jumpbox. Available: http://www.jumpbox.com/
18. Wichmann, B. A., A. A. Canning, D. L. Clutterbuck, L. A. Winsbarrow, N. J. Ward, and D. W. R. Marsh Industrial Perspective on Static Analysis. Software Engineering Journal Mar. 1995: 69-75. Available: http://www.ida.liu.se/~TDDC90/papers/industrial95.pdf
19. findbugs. (Published: 2012, Accessed: 8th April 2012). findbugs. Available: http://findbugs.sourceforge.net/

20. pmd. (Published: 2012, Accessed: 20th April 2012). pmd. Available: http://pmd.sourceforge.net/pmd-5.0.0/
21. Nagappan, N and Ball, T., Static analysis tools as early indicators of pre-release defect density. In Proceedings of the 27th international conference on Software engineering (ICSE '05). ACM, New York, NY, USA, 580-586. 2005.
22. Yang,Q. , Jenny Li, J., and Weiss, D: A survey of coverage based testing tools. In Proceedings of the 2006 international workshop on Automation of software test (AST '06). ACM, New York, NY, USA, 99-103. 2006.
23. agitar.com. (Published: 2012, Accessed: 25th April 2012). agitar. Available: http://www.agitar.com/
24. lynuxworks.com. (Published: 2012, Accessed: 27th April 2012). lynuxworks. Available:http://www.lynuxworks.com/partners/show_product.php?ID=268
25. Roger S.. Software Engineering. Mc Graw Hill International Edition. 2001.
26. Hollingsworth, D.: Workflow Management Coalition The Workflow Reference Model. 1.1. 1995.
27. D.P. Spooner, J., Cao, S. A. Jarvis, L. He, and G. R. Nudd. Performance-aware Workflow Management for Grid Computing. The Computer Journal, Oxford University Press, London, UK, 2004.
28. Huo M., Verner J., Zhu L., Ali Babar M.: Software Quality and Agile Methods, compsac,vol. 1, pp.520-525, 28th Annual International Computer Software and Applications Conference, COMPSAC'04, (2004).
29. Khajeh-Hosseini, A., Greenwood, D., Sommerville, I. , "Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS," Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on , vol., no., pp.450-457, 5-10 July 2010
30. Armbrust, M., Fox, A., Griffith, R., Anthony D. Joseph, Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. : A view of cloud computing. Commun. ACM 53, 50-58, April 2010.
31. Crockford,D.,: Javascript: The Good Parts. O'Reilly Media, Inc.,2008.
32. Yang, Q.,Jenny J. and Weiss, D. : A survey of coverage based testing tools. In Proceedings of the 2006 international workshop on Automation of software test (AST '06). ACM, New York, NY, USA, 99-103. 2006.
33. COCOMO, (no date), Accessed 15th Aug , Available : http://www.softstarsystems.com/overview.htm
34. COCOMO, (no date), Accessed 15th Aug , Available : http://cost.jsc.nasa.gov/cocomo.html
35. Stroock, D. w., Probability Theory. 2nd ed. Dubai: Cambridge University, 2010.
36. JSHint, (no date), Accessed 15th Aug , Available : https://github.com/jshint/jshint/
37. JSLint, (no date), Accessed 15th Aug , Available : https://github.com/douglascrockford/JSLint
38. JSCoverage, (2010), Accessed 15th Aug , Available : http://siliconforks.com/jscoverage/manual.html
39. Iqbal, M.; Rizwan, M.; , "Application of 80/20 rule in software engineering Waterfall Model," Information and Communication Technologies, 2009. ICICT '09. International Conference on , vol., no., pp.223-228, 15-16 Aug. 2009
40. Advances in Computer Science and Information Technology. Computer Science and Engineering Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2012.

41. Essence of Waterfall: (no date), Accessed 15th Aug , Available : http://ezinearticles.com/?Essence-Of-Waterfall-Model&id=352291
42. Amazon, (no date), Accessed 15th Aug , Available : http://aws.amazon.com/ec2/
43. Goecks, J., Nekrutenko, A., Taylor, J. and Team, T. G.: Galaxy: a Comprehensive Approach for Supporting Accessible, Reproducible and Transparent Computational Research in the Life Sciences: Genome Biology, vol. 11, p. R86, (2010).
44. Bessey, A.: A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. Commun. ACM, pp. 66-75, February 2010.
45. Ratzinger, J., Sigmund, T. and Harald C. Gall. On the relation of refactorings and software defect prediction. In Proceedings of the 2008 international working conference on Mining software repositories (MSR '08). ACM, New York, NY, USA, 35-38. 2008.
46. Riungu, L., Taipale, O., Smolander, K., "Research Issues for Software Testing in the Cloud," Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on , vol., no., pp.557-564, Nov. 30 2010-Dec. 3 2010.
47. Riungu, L., Taipale, O., Smolander, K. , "Software Testing as an Online Service: Observations from Practice," Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on , vol., no., pp.418-423, 6-10 April 2010.
48. Sharon, D.; Anderson, T.; "A complete software engineering environment," Software, IEEE , vol.14, no.2, pp.123-125, Mar/Apr 1997.
49. Kaur, P. and Kaushal, S., Security Concerns in Cloud Computing High Performance Architecture and Grid Computing. A. Mantri, S. Nandi, G. Kumar and S. Kumar, Springer Berlin Heidelberg. 169: 103-112. (2011).
50. Takabi, H.; Joshi, J., Ahn, G.,"Security and Privacy Challenges in Cloud Computing Environments," Security & Privacy, IEEE , vol.8, no.6, pp.24-31, Nov.-Dec. 2010.
51. Ruohomaa, S. and Kutvonen, L., Trust Management Survey Trust Management. P. Herrmann, V. Issarny and S. Shiu, Springer Berlin / Heidelberg. 3477: 77-92. (2005).
52. E-Science centre, Publish: (2012), Accessed 15th Aug , Available : http://www.esciencecentral.co.uk/
53. Ramgovind, S., Eloff, M., Smith, E., The management of security in Cloud computing. Information Security for South Africa (ISSA). 2010 vol., no., pp.1-7, 2-4 Aug. (2010).
54. Desktopanywhere, (2012), Accessed 15th Aug , Available : https://www.desktopanywhere.com/
55. Amazon, (2012), Accessed 15th Aug , Available : http://calculator.s3.amazonaws.com/calc5.html
56. Amazon, (2012), Accessed 15th Aug , Available : http://aws.amazon.com/security/
57. Piyush, J., Ramarkrishna, R., Sathyanad, B., Challenge in deploying static analysis tools, (no date), Accessed 15th Aug , Available : http://www.crosstalkonline.org/storage/issue-archives/2011/201107/201107-Jain.pdf.

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

Deployment of Software Engineering Tools as Cloud Services

Appendix A

Nawaf Almutairi
School of Computing Science, Newcastle University
n.m.m.al-mutairi@newcastle.ac.uk

```
# define N 2
typedef Message {
bool value;
byte count;
byte i;
}
bool ArrayA[N] ;
bool ArrayB[N] ;
bool ArrayC[N] ;
bool ArrayD[N] ;
bool ArrayZ[N];
proctype ProcA (chan chind , choutb, choutz)
{
Message M;
M.count = 0;
M.i = 0;
do
:: ( M.i < N ) ->
choutb! ArrayA[M.i], M.i , 1 -> M.i++
:: ( M.i >= N ) ->
break
od;
end1:
do
:: chind? M.value, M.i , M.count ->
if
:: ( M.count < 4) ->
if
:: ( ArrayA[M.i] == 1 || M.value == 1) ->
M.count++ -> choutb! 1 , M.i , M.count
:: ( ArrayA[M.i] == 0 && M.value == 0) ->
M.count++ -> choutb! 0 , M.i , M.count
fi;
:: ( M.count == 4) ->
choutz! M.value , M.i
fi;
od;
```

```
}
proctype ProcB (chan china , choutc, choutz)
{
Message M;
M.count = 0;
M.i = 0;
do
:: ( M.i < N ) ->
choutc! ArrayB[M.i], M.i , 1 -> M.i++
:: ( M.i >= N ) ->
break
od;
end2:
do
:: china? M.value, M.i , M.count ->
if
:: ( M.count < 4) ->
if
:: ( ArrayB[M.i] == 1 || M.value == 1) ->
M.count++ ->
choutc! 1 , M.i , M.count
:: ( ArrayB[M.i] == 0 && M.value == 0) ->
M.count++ ->
choutc! 0 , M.i , M.count
fi;
:: ( M.count == 4 ) ->
choutz! M.value , M.i
fi;
od;
}
proctype ProcC (chan chinb , choutd, choutz)
{
Message M;
M.count = 0;
M.i = 0;
do
:: ( M.i < N ) ->
choutd! ArrayC[M.i], M.i , 1 -> M.i++
:: ( M.i >= N ) ->
break
od;
end3:
do
:: chinb? M.value, M.i , M.count ->
if
:: ( M.count < 4) ->
if
```

```
:: ( ArrayC[M.i] == 1 || M.value == 1) ->
M.count++ ->
choutd! 1 , M.i , M.count
:: ( ArrayC[M.i] == 0 && M.value == 0) ->
M.count++ ->
choutd! 0 , M.i , M.count
fi;
:: ( M.count == 4 ) ->
choutz! M.value , M.i
fi;
od;
}
proctype ProcD (chan chinc , chouta, choutz)
{
Message M;
M.count = 0;
M.i = 0;
do
:: ( M.i < N ) ->
chouta! ArrayD[M.i], M.i , 1 -> M.i++
:: ( M.i >= N ) ->
break
od;
end4:
do
:: chinc? M.value, M.i , M.count ->
if
:: ( M.count < 4) ->
if
:: ( ArrayD[M.i] == 1 || M.value == 1) ->
M.count++ ->
chouta! 1 , M.i , M.count
:: ( ArrayD[M.i] == 0 && M.value == 0) ->
M.count++ ->
chouta! 0 , M.i , M.count
fi;
:: ( M.count == 4 ) ->
choutz! M.value , M.i
fi;
od;
}
proctype ProcZ (chan china, chinb, chinc, chind)
{
bool value1;
bool value2;
bool value3;
bool value4;
```

```
byte i1;
byte i2;
byte i3;
byte i4;
end5:
do
:: china? value1, i1 ->
ArrayZ[i1] = value1;
:: chinb? value2, i2 ->
ArrayZ[i2]= value2;
:: chinc? value3, i3 ->
ArrayZ[i3]= value3;
:: chind? value4, i4 ->
ArrayZ[i4]= value4;
od;
}
init {
ArrayA[0] = 0 ;
ArrayB[0] = 1 ;
ArrayC[0] = 1 ;
ArrayD[0] = 0 ;
ArrayA[1] = 1 ;
ArrayB[1] = 1 ;
ArrayC[1] = 1 ;
ArrayD[1] = 1 ;
ArrayA[2] = 0 ;
ArrayB[2] = 0 ;
ArrayC[2] = 0 ;
ArrayD[2] = 0 ;
chan AtoB = [N] of { bool , byte , byte};
chan BtoC = [N] of { bool , byte , byte};
chan CtoD = [N] of { bool , byte , byte };
chan DtoA = [N] of { bool , byte , byte};
chan AtoZ = [1] of {bool , byte };
chan BtoZ = [1] of {bool , byte };
chan CtoZ = [1] of {bool , byte };
chan DtoZ = [1] of {bool , byte };
atomic {
run ProcA (DtoA, AtoB , AtoZ);
run ProcB (AtoB, BtoC , BtoZ);
run ProcC (BtoC, CtoD , CtoZ);
run ProcD (CtoD, DtoA , DtoZ);
run ProcZ (AtoZ, BtoZ, CtoZ, DtoZ);
}
}
```

Deployment of Software Engineering Tools as Cloud Services

Appendix B

Nawaf Almutairi
School of Computing Science, Newcastle University
n.m.m.al-mutairi@newcastle.ac.uk

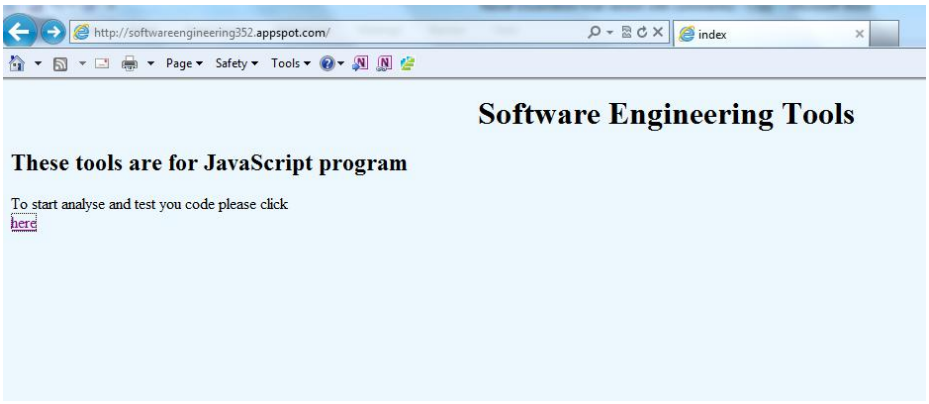

Fig. 1. Starting Page

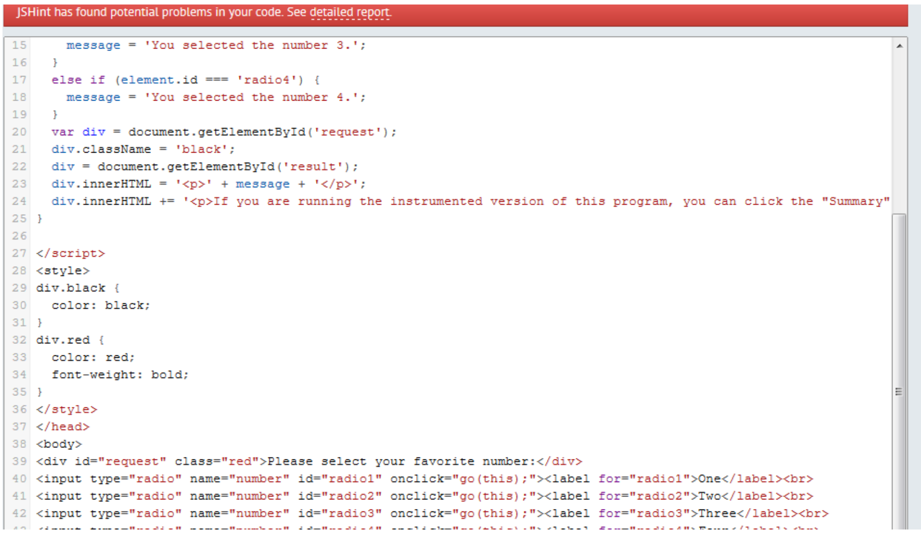

Fig. 2. JSHint Input

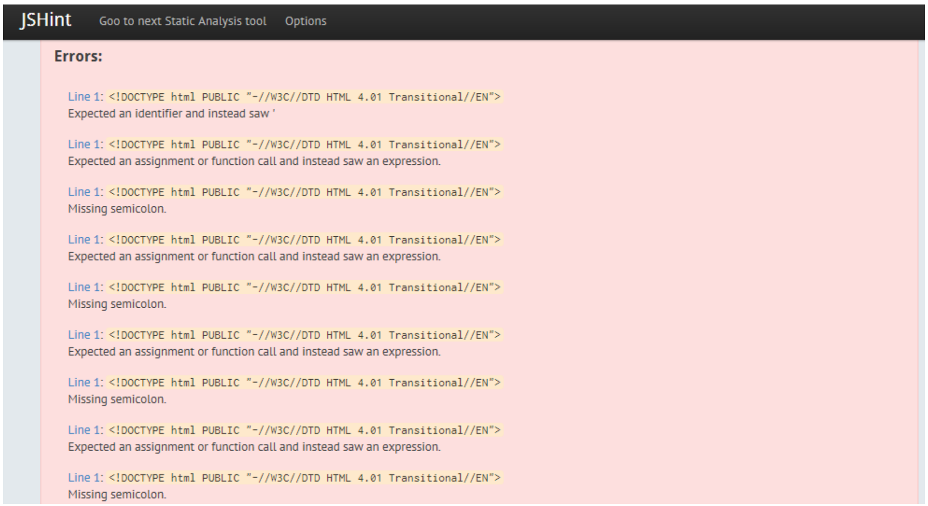

Fig. 3. JSHint outputs

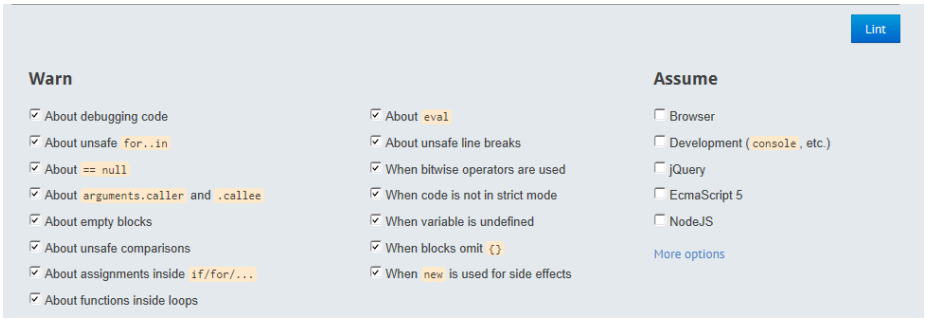

Fig. 4. JSHint Configuration

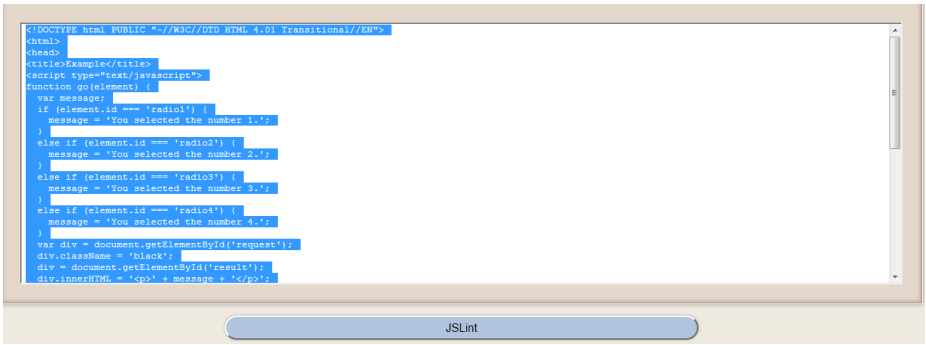

Fig. 5. JSLint Input

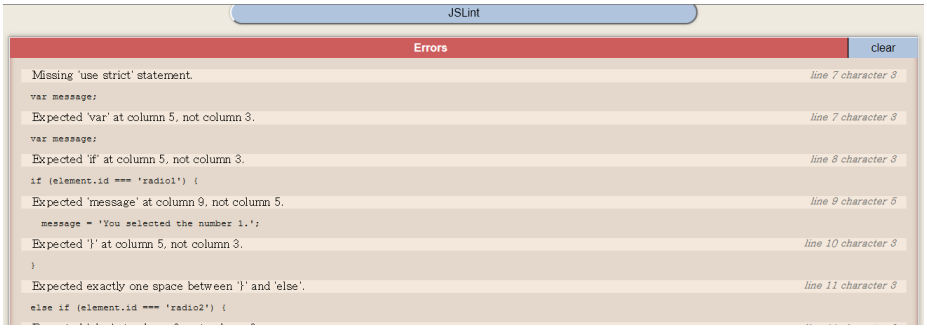

Fig. 6. JSLint Outputs

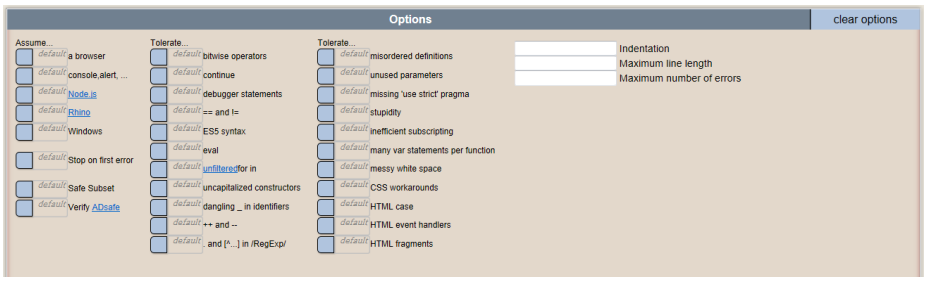

Fig. 7. JSLint Configuration

Fig. 8. The Main Screen of JSCoverage

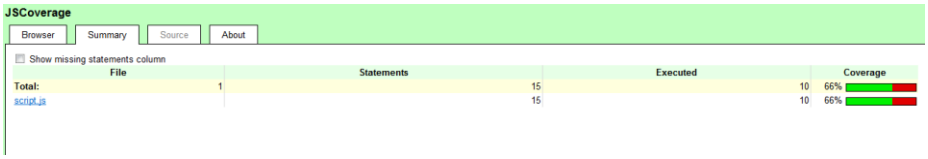

Fig. 9. The Summary of Testing in JSCoverage

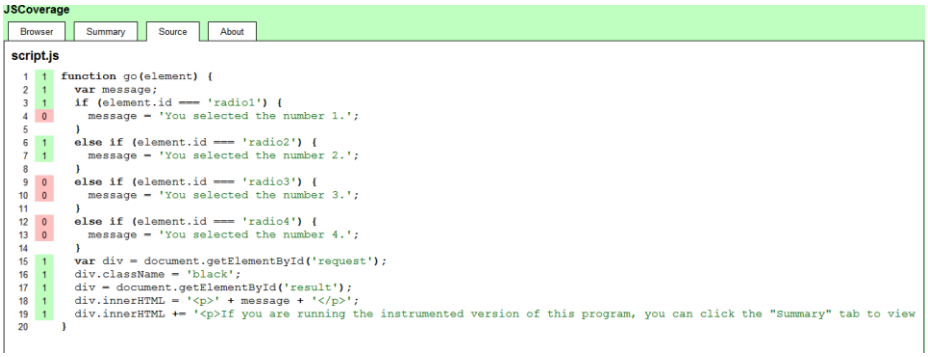

Fig. 10. The Functions which are not tested in red colour

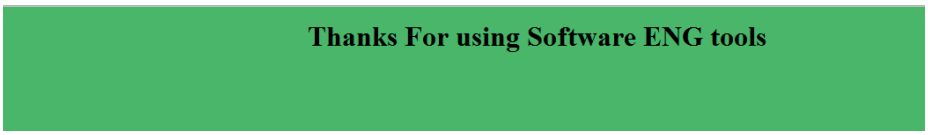

Fig. 11. End Screen

العنوان:	CSC8299 Deployment of Software Engineering Tools as Cloud Services Individual Project
المؤلف الرئيسي:	Al Mutairi, Nawaf
مؤلفين آخرين:	Fitzgerald, John(super)
التاريخ الميلادي:	2012
موقع:	نيوكاسل
الصفحات:	1 - 46
رقم MD:	607639
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	Newcastle University
الكلية:	The School Of Computing Science
الدولة:	بريطانيا
قواعد المعلومات:	Dissertations
مواضيع:	هندسة البرمجيات ، الحاسبات الإلكترونية ، الحوسبة السحابية
رابط:	https://search.mandumah.com/Record/607639

CSC8299 Deployment of Software Engineering Tools as Cloud Services

Individual Project

8/24/2012

Computer Security & Resilience
Student: Nawaf Almutairi
Student Number: 099098133
Prof. J. S Fitzgerald