Advances In Multidisciplinary & Scientific Research
A Multidisciplinary & Interdisciplinary Journal
Vol. 3 No. 3, September 2017

A Learning Tool for Demonstrating Automatic Java Code Generation from UML Class Diagrams

Mukhtar, M.I., Galadanci, B.S. & Muaz, S.A.
Department of Software Engineering
Faculty of Computer Science & Information Technology
Bayero University
Kano, Nigeria
mimukhtar.se@buk.edu.ng, bashirgaladanci@yahoo.com, samaaz.cs@buk.edu.ng

ABSTRACT

The significance of automatic code generation from UML diagrams has increased over the traditional software development process due to benefits such as being less error prone, leading to more rapid delivery of software, cost reduction, less time consumption, maintainability and accuracy. However, majority of students and software engineers are still neither aware of these benefits nor fully acquainted with its principles and methods. This paper presents a learning tool to demonstrate automatic java code generation from UML class diagrams. An overview of the tool is presented giving its architectural design based on its three components; the learning module, the trial module and the quiz module. With the help of diagrams, the functionalities of the modules are explicitly described. The results of an evaluation survey conducted by administering a usability questionnaire to a group of lecturers and students are explained and discussed. The evaluation results show that the tool will likely be accepted by a majority of people because of its ease of use and potential usefulness. The learning tool can be used by universities and individuals to demonstrate to students the fundamentals of automatic code generation from UML class diagrams.

Keywords: Automatic Code Generation; Model Driven Engineering; UML Diagrams; Class Diagrams.

Aims Research Journal Reference Format:
Mukhtar, M.I., Galadanci, B.S. & Muaz, S.A.(2017): A Learning Tool for Demonstrating Automatic Java Code Generation from UML Class Diagrams. Advances in Multidisciplinary & Scientific Research Journal. Vol. 3. No.3, Pp11-20

1. INTRODUCTION

Over the years, the demand for more complex and sophisticated software has increased and with this increased complexity, a lot of programming skills is now needed. Nowadays, the development of software is being achieved through system modelling (Kent, 2002). System modelling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system (Summerville, 2009). System modelling involves using Unified Modeling Language (UML) graphical notations. In fact these UML notations have become generally accepted as the de-facto standard modeling notation for the analysis and design of object- oriented software systems (and I. Jacobson, 1999). As system modeling gets more and more popular (Rumbaugh, Blaha, Premerlani, Eddy, & Lorenson, 1991), the automatic generation of the program code on the basis of high-level models are becoming an important issue(Pinter & Majz, 2003).In fact, the concept of Automatic Code Generation that would eliminate the need for programming by generating source code from design models is gaining a lot of attention in Software Engineering because it has many benefits including being less error prone than writing code manually, maintainability and accuracy(Niaz, 2005).

The Object Management Group (OMG) has realized these benefits of automatic code generation and has introduced Model-driven Architecture (MDA) that uses a subset of UML diagrams such as class diagrams, sequence diagrams and state diagrams (Summerville, 2009). In view of the increasing significance of automatic code generation and the difficulty in programming, there is the need for students and software engineers to be fully familiar with the principles and methods of automatic code generation. This paper discusses a learning tool which is mainly meant for students in order to teach automatic java code generation from UML class diagrams.

The paper is organized as follows: Section 2 presents some related works while Section 3 presents a detailed description of the proposed learning tool. Section 4 discusses the experimental results of the tool and Section 5 gives some concluding remarks and recommendations for further research.

2. RELATED WORK

One of the major tasks in MDA is to eradicate the task of programming by automatically generating source code from design models. Majority of the work on automatic code generation is currently being done on Unified Modeling Language (UML) diagrams that have become the de-facto standard for use in object-oriented software development (Usman & Nadeem, 2009). A number of studies as described in [(Knapp & Merz, 2002); (Wehrmeister, Freitas, Pereira, & Rammig, 2008); (Fertalj & Brcic, 2008); (Lindlar & Zimmermann, 2008);(Derezińska & Pilitowski, 2009); (A Jakimi & Koutbi, 2009);(Parada, Siegert, & Brisolara, 2011);(Usman & Nadeem, 2009); (Abdeslam Jakimi & Elkoutbi, 2009); (Yi, Niu, Ancha, & Lakshmipathy, 2009); (Bennett, Cooper, & Dai, 2010) and (Vadakkumcheril, Mythily, & Valarmathi, 2013)] have developed approaches and tools to automatically generate source code from UML design models such as class diagrams, sequence diagrams and activity diagrams. A few such as [(Mcdonald-maier, Akehurst, & Howells, 2007); (Kim, Lee, Phan, & Sokolsky, 2013) and (Hussein & Salah, 2013)] focus more on the development of broad frameworks. From these studies, class diagrams are the models that have been the most used in automatic code generation as they describe the static structure of a system in terms of attributes and methods and are therefore easy to automate (Abdeslam Jakimi & Elkoutbi, 2009). Most of the work done so far has been on automatic java code generation from UML design models.

The majority of students studying Software Engineering have little or no knowledge about automatic code generation and the variety of tools that have been developed since the discovery of MDA. To make things worse, even though there are quite a number of learning tools to support teaching and learning of UML diagrams and their semantics, such as those described in (Baghaei, Mitrovic, & Irwin, 2007)' (Ali, Shukur, & Idris, 2007), and (Soler, Boada, Prados, & Poch, 2010), and a number of tools to support teaching and learning of programming concepts, such as those described in [(Cooper, Pausch, & Dann, 2003);(Sanders & Dorn, 2003); (Al-imamy, 2006) and (Parson & Haden, 2006)] learning tools for automatic code generation from UML diagrams are very scarce. It is therefore evident that there is the need for learning tools to demonstrate automatic code generation from UML diagrams.

3. OVERVIEW OF THE LEARNING TOOL

The proposed tool is used for converting class diagrams to an equivalent line-by-line Java program which can be executed in the tool environment. In this research, the Prototyping Software Development Model was used to analyze, design, implement and test the tool. An analysis was first carried out to, with the help of use case diagrams, identify the functional requirements and non-functional requirements of the learning tool including ease of use and quick response time.The design of the learning tool was then made using sequence diagrams, class diagrams and structured charts. With respect to sequence diagrams, one diagram was drawn for each use case to further explain the execution flow of the activity. The architecture of the tool is depicted in Figure 1. The learning tool has 3 modules which the user can select from. Each module is described in the subsections below:

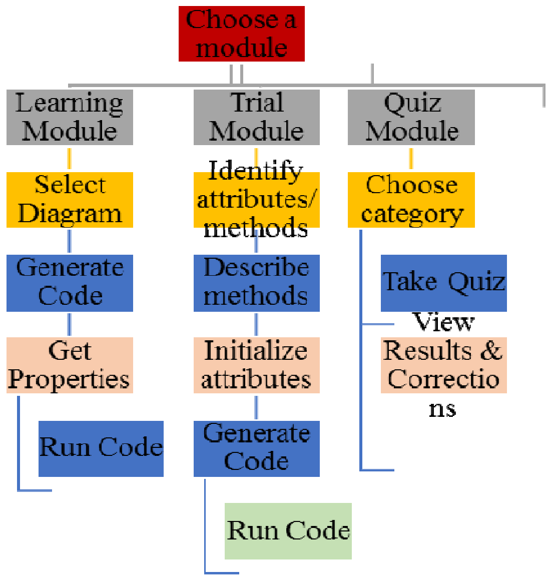
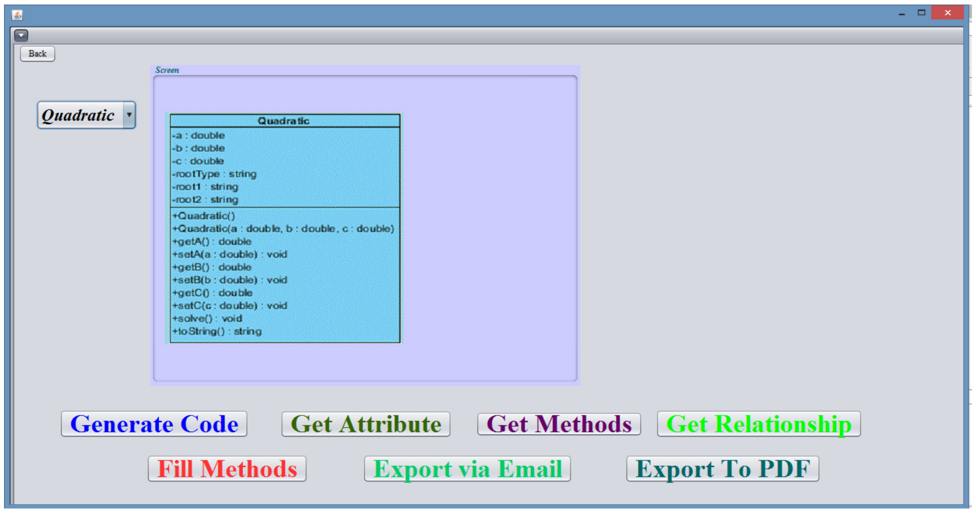


Figure 1: Learning Tool Architecture

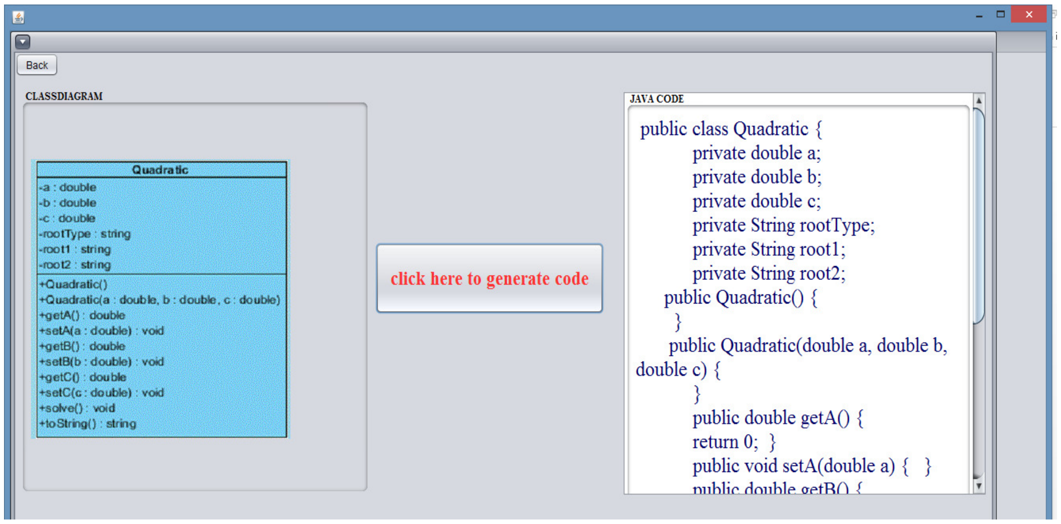
3.1 The Learning Module

The "Learning Module" teaches the concept of automatic java code generation by providing the user with some predefined class diagrams. The user selects from a list of class diagrams. Once selected, the user can click on a button to generate a line-by-line equivalent Java code. Because long lines of code can be difficult to comprehend, buttons that extract the properties of the selected class diagram such as the attributes, methods and relationships are provided for the user. Under attributes a user can view private as well as public attributes; under methods, a user can view ordinary methods in the class as well as special Java methods such as constructors, setters and getters by using appropriate buttons provided; and under relationships, the user can view inheritance and composition relationships if any exist in the selected class diagram by using the appropriate buttons. The user can proceed to run the selected class diagram by clicking the fill method button which fills in the body of the methods using some predefined methods provided in the learning tool. The user may also export the generated code either through an email from the tool environment or create a PDF file for later use. Figure 3, 4 and 5 shows some frames of the learning module.



```
Quadratic
-a : double
-b : double
-c : double
-rootType : string
-root1 : string
-root2 : string
+Quadratic()
+Quadratic(a : double, b : double, c : double)
+getA() : double
+setA(a : double) : void
+getB() : double
+setB(b : double) : void
+getC() : double
+setC(c : double) : void
+solve() : void
+toString() : string
```

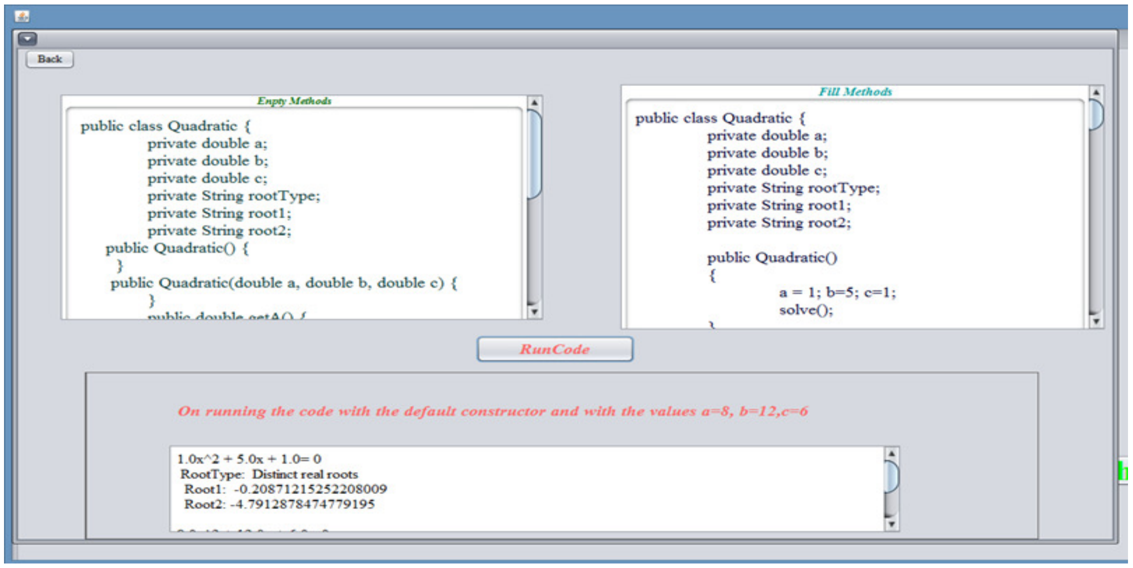
Figure 2: Learning Module Selection Frame



```
Quadratic
-a : double
-b : double
-c : double
-rootType : string
-root1 : string
-root2 : string
+Quadratic()
+Quadratic(a : double, b : double, c : double)
+getA() : double
+setA(a : double) : void
+getB() : double
+setB(b : double) : void
+getC() : double
+setC(c : double) : void
+solve() : void
+toString() : string
```

```
public class Quadratic {
        private double a;
        private double b;
        private double c;
        private String rootType;
        private String root1;
        private String root2;
    public Quadratic() {
     }
     public Quadratic(double a, double b,
double c) {
        }
        public double getA() {
        return 0;  }
        public void setA(double a) {   }
        public double getB() {
```

Figure 3: Learning Module Generate Code Frame



```
public class Quadratic {
          private double a;
          private double b;
          private double c;
          private String rootType;
          private String root1;
          private String root2;
    public Quadratic() {
     }
     public Quadratic(double a, double b, double c) {
          }
```

```
public class Quadratic {
          private double a;
          private double b;
          private double c;
          private String rootType;
          private String root1;
          private String root2;

          public Quadratic()
          {
                    a = 1; b=5; c=1;
                    solve();
```

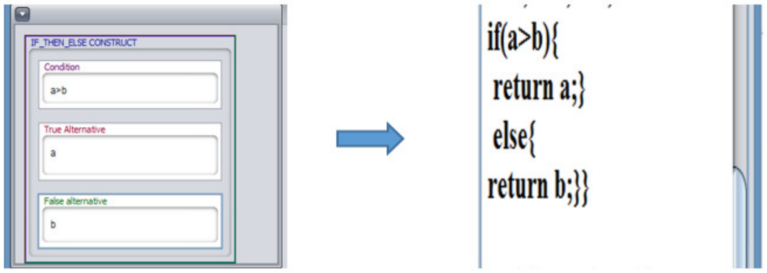
On running the code with the default constructor and with the values a=8, b=12,c=6

```
1.0x^2 + 5.0x + 1.0= 0
 RootType:  Distinct real roots
  Root1:  -0.20871215252208009
  Root2: -4.7912878474779195
```

Figure 3: Learning Module Run Code Frame

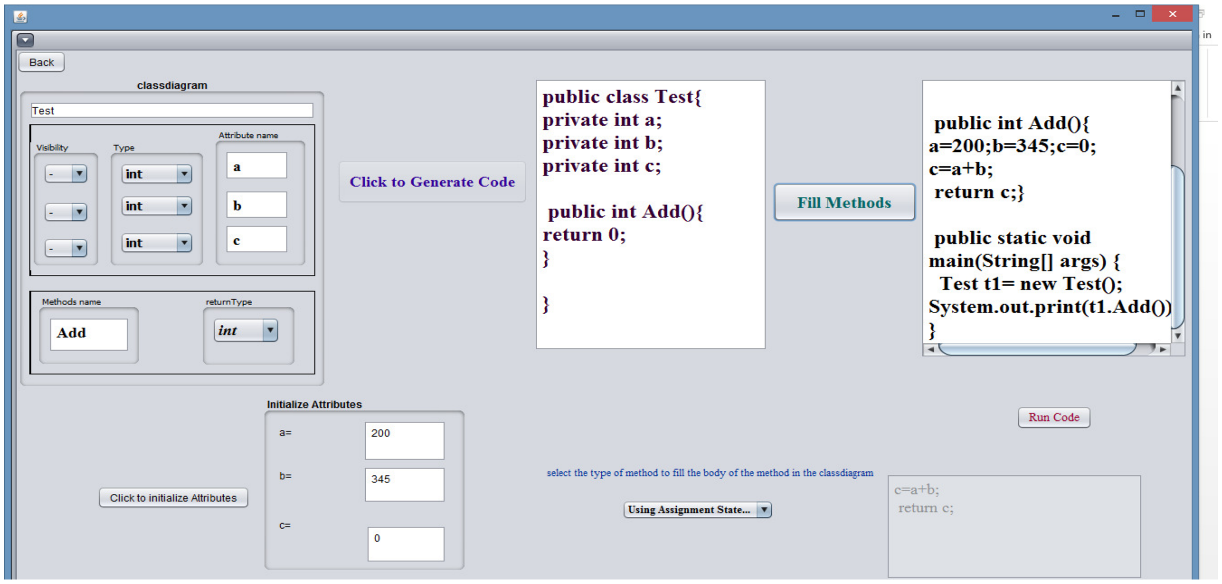
3.2. The Trial Module

The "Trial Module" allows a user to create a class diagram and convert it to an equivalent Java code. The user first enters the attributes (which can be of int, double or String data types) and method (that can return int, double, String or Void) into the class diagram construct provided and press on the Generate Code button. This button will automatically generate an equivalent Java code based on the user's entries. As the tool is aimed at executing the generated code in the tool environment, unlike many case tools that only provide the user with the code, the process does not stop here. The user proceeds to initialize the attributes using an attribute construct provided and this is followed by describing the body of the method. Three (3) constructs are provided for the user to describe his/her method; the assignment, IF and while construct. It is worth nothing that all the user needs to do is to fill in the spaces provided in the construct. The tool translates them automatically to a Java construct as shown in Figure 4. The user then clicks on the fill method button which combines the whole description of the class diagram attributes and methods and appends the necessary headers needed to run a Java class. Finally the user clicks on the Run Code button which invokes the command prompt and executes the user program. The output of the program is displayed to the user. However in cases where the user makes some mistakes such as in declaring data types, the errors are displayed to the user. An example of a sum program and its output is given in Figure 5 and 6.



```
if(a>b){
 return a;}
 else{
return b;}}
```

Figure 4: IF Construct

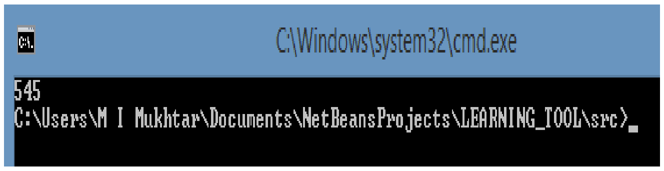


```
public class Test{
private int a;
private int b;
private int c;

 public int Add(){
return 0;
}

}
```

```
 public int Add(){
a=200;b=345;c=0;
c=a+b;
 return c;}

 public static void
main(String[] args) {
  Test t1= new Test();
System.out.print(t1.Add())
}
```

```
c=a+b;
 return c;
```

Figure 5: Frame that generates Java Code for Sum program

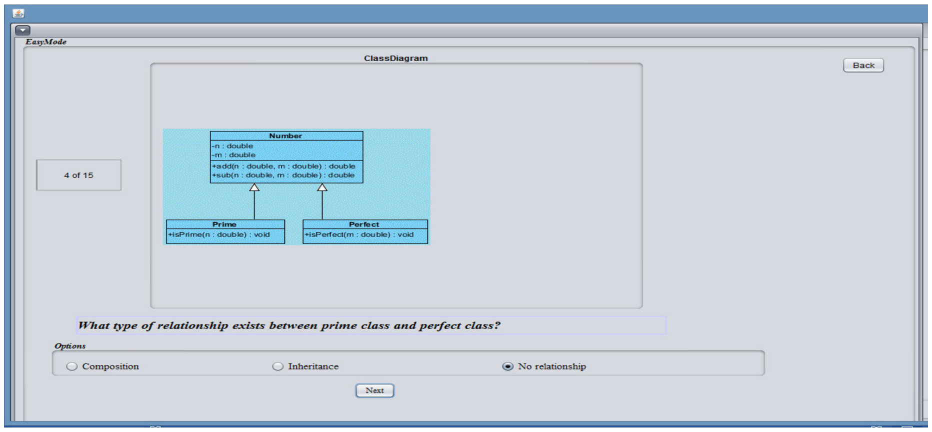


```
C:\Windows\system32\cmd.exe
545
C:\Users\M I Mukhtar\Documents\NetBeansProjects\LEARNING_TOOL\src>_
```

Figure 6: Command Prompt showing output of the Sum program

3.2. The Quiz Module

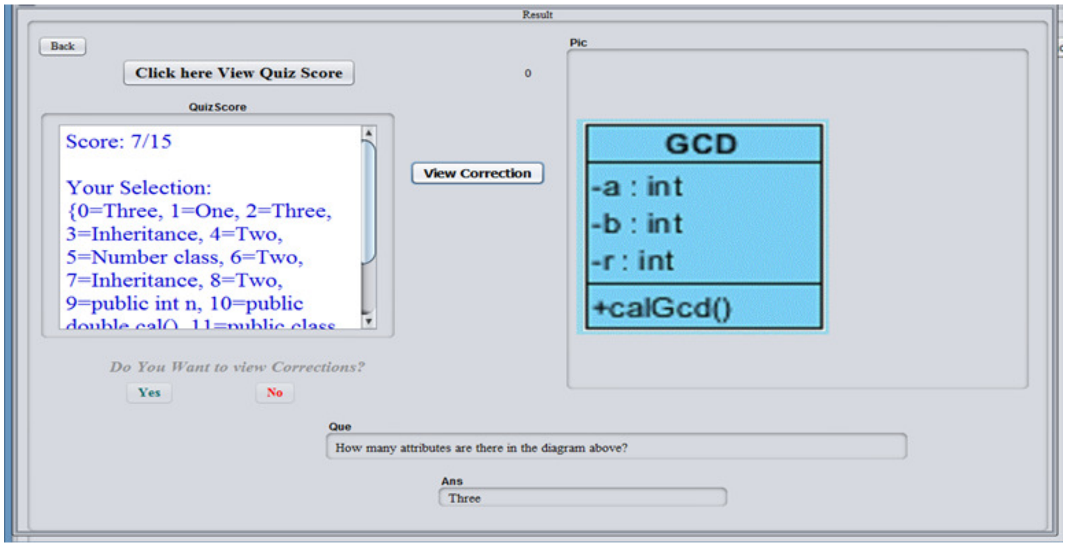
The "Quiz Module", which is shown in Figure 7 and Figure 8, enables a user to take a quiz on class diagrams. The user can select either the easy or difficult section of the quiz. In each section the user will be shown some UML diagrams and questions will be asked on them to see how much he/she has learnt from the "Learning Module". The selected section determines the complexity of the questions. Once the user starts the quiz, he/she is not allowed to go back to a previously answered question. A user is required to answer 15 questions in each quiz session. At the end of the quiz session the user can view his score as well as corrections if he/she desire.


EasyMode
ClassDiagram
Back
4 of 15
Number
-n : double
-m : double
+add(n : double, m : double) : double
+sub(n : double, m : double) : double
Prime
+isPrime(n : double) : void
Perfect
+isPerfect(m : double) : void
Options
Next

What type of relationship exists between prime class and perfect class?

Composition Inheritance [✓ No relationship]

Figure 7: Quiz Frame



Score: 7/15

Your Selection:
{0=Three, 1=One, 2=Three, 3=Inheritance, 4=Two, 5=Number class, 6=Two, 7=Inheritance, 8=Two, 9=public int n, 10=public double cal(), 11=public class

How many attributes are there in the diagram above?

Three

Figure 8 : Correction Frame of Quiz Module

4. EVALUATION

A questionnaire was used to evaluate the learning tool in terms of its usability. The questionnaire has two sections; Section 1 collects personal information of the respondents such as gender, educational background and position while Section 2 of the questionnaire uses the System Usability Scale (SUS) questions to evaluate the Learning Tool. The SUS comprises of 10 close-ended questions and uses the likert scale (1-Strongly disagree, 2-Disagree, 3-Not sure, 4-Agree, 5-Strongly Agree to obtain rating from participants. This questionnaire was distributed to lecturers and students of Computer Science Bayero University Kano. This is because the learning tool will be mostly used by lectures to demonstrate to students the automatic java code generation from UML class diagrams. The SUS score was computed in Microsoft Excel using the formula below:

$$((Q1-1)+(5-Q2)+(Q3-1)+(5-Q4)+(Q5-1)+(5-Q6)+(Q7-1)+(5-Q8)+(Q9-1)+(5-Q10))*2.5$$

4.1. Preliminary Results

The preliminary results are presented in this section. Figure 9 shows the average SUS score of lecturers, MSc students and undergraduate Students. The total average score of all participant is 84 which is more than the average SUS score (68). Figure 10 shows 26 participants score graph. This figure shows almost all participants may use this learning tool easily and efficiently.

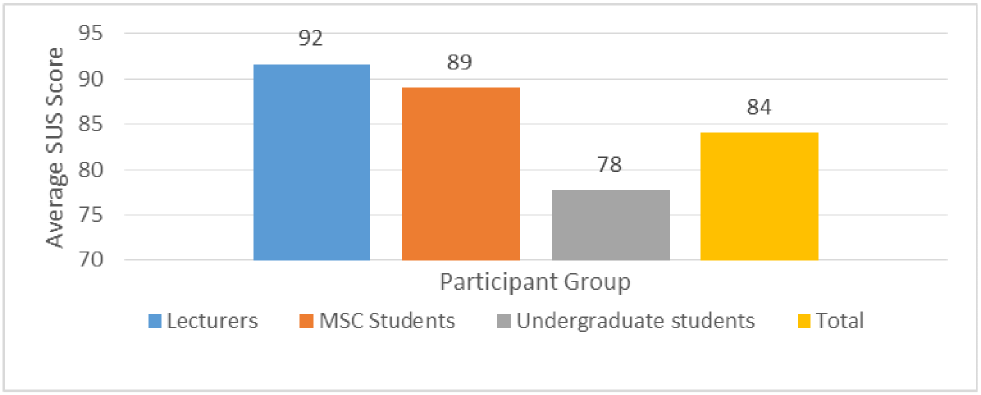


Figure 9: Average SUS Score of participants

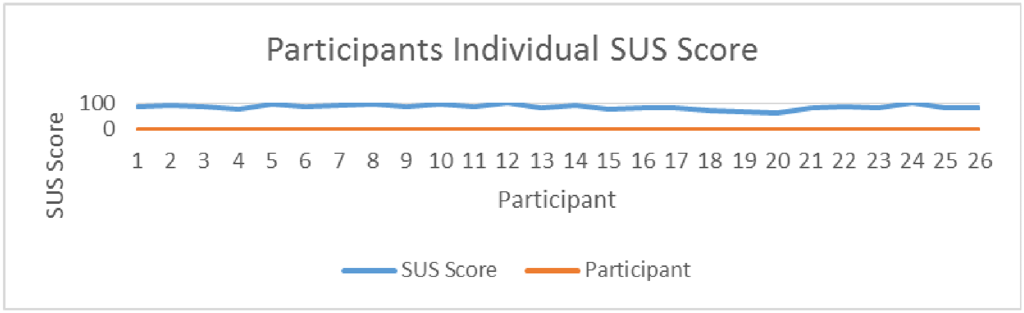


Figure 10: Participant Individual SUS Score

Table 1 shows that 87.7% of the participant's given response is 3 or 4 score (4 is the best score) this implies that the application is user friendly and effective. Only 6.9% of the participant's response is towards 0 and 1, and 5.38 % of the participant's response are neutral.

Table 1. Ten SUS items response from 0 to 4

SUS score from 0-4	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Participant response in %
0	0	3	0	0	0	0	0	3	0	3	3.46%
1	0	0	0	2	0	1	0	2	0	4	3.46%
2	0	2	1	3	1	1	2	3	0	1	5.38%
3	6	10	11	10	5	5	7	4	9	4	27.30%
4	20	11	14	11	20	19	17	14	17	14	60.40%

4.2. Statistical Analysis

The SUS score was extracted and analyzed using SPSS V20 in terms of mean, standard deviation. The result in Table 2 showed that all even numbered questions have a high mean while all odd numbers questions have a low mean showing how much participants agree with the questions.

Table 2: SUS Mean and Std. Deviation Results

	Mean	Std. Deviation
Q1. I think I would like to use this Learning Tool frequently	4.77	0.43
Q2. I found the Learning Tool to be unnecessarily complex	2.00	1.26
Q3. I thought that the Learning Tool was easy to use	4.50	0.58
Q4. I think I would need the support of a technical person to be able to use this Learning tool	1.85	0.92
Q5. I found that the various functions in this Learning Tool were well integrated	4.73	0.53
Q6. I thought there was too much inconsistency in this Learning Tool	1.38	0.75
Q7. I would imagine that most people would learn to use this Learning Tool very Quickly	4.58	0.64
Q8. I found that Learning Tool to be very cumbersome to use	2.08	1.44
Q9. I felt very confident using the Learning Tool	4.65	0.49
Q10. I needed to learn a lot of things before I could get going with this Learning Tool.	2.15	1.52

The result in Table 3 shows there was no statistically significant difference between Male and Female participants with regards to the usability of the Learning tool.

Table 3: Male and Female Differences

Gender	Mean	N	Std. Deviation
Male	85.2778	18	9.65923
Female	82.5000	8	10.52209
Total	84.4231	26	9.80581

5.. DISCUSSION

Almost all the participant score the learning tool above the average SUS score of 68 with only two (2) undergraduate Student having an SUS Score of (60) and (67.5). In general, the SUS score of Lecturers is highest, followed by MSC Students and then Undergraduate Student and this is probably because of the differences in level of experience and awareness to the MDE concepts. Results of Q1 (I think I would like to use this Learning Tool frequently) & Q9 (I felt very confident using the Learning Tool) of SUS questionnaire shows that almost all the participants are satisfied. The result of Q5 (I found that the various functions in this Learning Tool were well integrated) shows that the Learning Tool design is sound and it is well integrated. The result of Q6 (I thought there was too much inconsistency in this Learning Tool) shows that the Learning Tool is consistent; Q4(I think I would need the support of a technical person to be able to use this Learning tool) & Q10(I needed to learn a lot of things before I could get going with this Learning Tool) shows that most participants will learn to use the Learning Tool very quickly; Q2(I found the Learning Tool to be unnecessarily complex), Q3(I thought that the Learning Tool was easy to use) & Q8(I found that Learning Tool to be very cumbersome to use) shows that the Learning Tool is user friendly.

The result from Table 2 shows that Q1(I think I would like to use this Learning Tool frequently) and Q5(I found that the various functions in this Learning Tool were well integrated) have the highest mean showing the learning tool ease of use and consistency; with Q4(I think I would need the support of a technical person to be able to use this Learning tool) and Q6(I thought there was too much inconsistency in this Learning Tool) having the lowest mean signifying very few participant agree with these questions.

The result from the comparison of performance of Male and Female participants showed no statistically significant difference. Moreover, the SUS mean Score across the group is more than 80 suggesting the observed usability is high across the groups.

6.. CONCLUSION AND FUTURE WORK

With the significant benefits of automatic code generation, it is important for software engineers and students to completely understand how it works. This paper presents the development of a learning tool for demonstrating automatic code generation. The tool has 3 main modules that illustrate, in an-easy-to-understand way, the basics of automatically generating codes from UML class diagrams. The usability testing of the learning tool shows that the tool is very easy to use and beneficial. Thus it will likely be accepted by most people.

The proposed tool can be improved in a number ways. In particular, since the tool only demonstrates automatic java code generation from UML class diagrams, there is the need to improve it so as to illustrate automatic code generation from other UML diagrams. There is also the need to for demonstrating the conversion from UML diagrams to other programming languages. Also the learning tool does not allow a user to specify more than three attributes and a single method in his/her class diagram and it only allows a user to describe simple methods consisting of assignment statements, if statements and while statements. There is therefore the need to develop the tool further so as to allow a user to describe complex methods.

REFERENCES

1. Ali, N. ., Shukur, Z., & Idris, S. (2007). Assessment System For UML Class Diagram Using Notations Extraction. *Int. J. of Comput. Sci. and Network Security*, *7*(8), pp. 181–187,.
2. Al-imamy, S. (2006). On the Development of a Programming Teaching Tool : The Effect of Teaching by Templates on the Learning Process. *Journal of Information Technology Education*, *5*, 272–283.
3. and I. Jacobson, G. B. J. R. (1999). *The Unified Modeling Language: User Guide*. Retrieved from www.google.com
4. Baghaei, N., Mitrovic, A., & Irwin, W. (2007). Supporting Collaborative learning and problem-solving in a constraint-based CSCL environment for UML class diagrams, *2*(13), pp 159–190.
5. Bennett, J., Cooper, K., & Dai, L. (2010). Aspect-oriented Model-Driven Skeleton Code Generation: A Graph based Transformation Approach. *Science of Computer Programming*, *75*(8), 689–725.
6. Cooper, S., Pausch, R., & Dann, W. (2003). Teaching objects-first in Introductory Computer Science. *ACM SIGCSE Bulletin*, *35*(1), 191–195.
7. Derezińska, A., & Pilitowski, R. (2009). Realization of UML Class and State Machine Models in the C # Code Generation and Execution Framework Related work Code generation and execution support. *Informatica(Slovenia)*, *33*(4), 431–440.
8. Fertalj, K., & Brcic, M. (2008). A Source Code Generator Based on UML Specification. *International Journal of Computer and Communications*, *2*(1), 10–19.
9. Hussein, B. M., & Salah, A. (2013). A Framework for Model-Based Code Generation from a Flowchart. *International Journal of Computing Academic Research (IJCAR)*, *2*(5), 167–181.
10. Jakimi, A., & Elkoutbi, M. (2009). Automatic Code Generation FromUML Statechart. *International Journal of Engineering and Technology*, *1*(2), 165–168.
11. Jakimi, A., & Koutbi, M. El. (2009). An Object-Oriented Approach to UML Scenarios Engineering and Code Generation. *International Journal of Computer Theory and Engineering*, *1*(1), 35–41. http://doi.org/10.7763/IJCTE.2009.V1.6
12. Kent, S. (2002). Model-driven engineering. In *Internation Conference on Integrated Formal Methods. Springer Berlin Heidelberg* (pp. 286–298).
13. Kim, B., Lee, I., Phan, L. T. ., & Sokolsky, O. (2013). Platform-Dependent Code Generation for Embedded Real-Time Software. In *International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)* (pp. 8.1–8.10).
14. Knapp, A., & Merz, S. (2002). Model Checking and Code Generation for UML State Machines and Collaborations. In *Proc. 5th Wsh. Tools for System Design and Verification* (pp. 59–64).
15. Lindlar, F. F., & Zimmermann, A. (2008). A code generation tool for embedded automotive systems based on finite state machines. In *The IEEE International Conference on Industrial Informatics* (pp. 1539–1544). http://doi.org/10.1109/INDIN.2008.4618349
16. Mcdonald-maier, K., Akehurst, D., & Howells, G. (2007). Implementing associations : UML 2 . 0 to Java 5. *Software & Systems Modelling*, *6*(1), 3–35. http://doi.org/10.1007/s10270-006-0020-1
17. Niaz, I. (2005). *Automatic Code Generation From UML Class and Statechart Diagrams*. *Graduates School of Systems and Information Engineering, University of Tsukuba, Ph.D. Thesis*.
18. Parada, A. G., Siegert, E., & Brisolara, L. B. De. (2011). Generating Java code from UML Class and Sequence Diagrams. In *Brazilian Symposium on Computing System Engineering* (pp. 99–101).
19. Parson, D., & Haden, P. (2006). Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In *Proceedings of the 8th Australasian Conference on Computing Education* (pp. 157–163).
20. Pinter, G., & Majz, I. (2003). Program Code Generation Based On UML Statechart Models. *Periodica Polytechnica*, *vol. 47*(no. 3), pp 187–204.
21. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorenson, W. (1991). *Object- Oriented Modeling and Design. New Jersey: Prentice-Hall*.
22. Sanders, D., & Dorn, B. (2003). Jeroo : A Tool For Introducing Object-Oriented Programming. *ACM SIGCSE Bulletin*, *35*(1), 201–204.
23. Soler, J., Boada, I., Prados, F., & Poch, J. (2010). A web-based e-learning tool for UML class diagrams. *Education Engineering (EDUCON) IEEE*, 973–979.

24. Summerville, I. (2009). *Software Engineering* (9th ed.). PEARSON.
25. Usman, M., & Nadeem, A. (2009). Automatic Generation of Java Code from UML Diagrams using UJECTOR. *International Journal of Software Engineering and Its Applications*, *3*(2), 21–37.
26. Vadakkumcheril, G. T., Mythily, M., & Valarmathi, M. L. (2013). A Simple Implementation of UML Sequence Diagram to Java Code Generation through XMI Representation. *International Journal of Emerging Technology and Advanced Engineering*, *3*(12), 1–5.
27. Wehrmeister, M. A., Freitas, E. P., Pereira, C. E., & Rammig, F. (2008). GenERTiCA: ATool for Code Generation and Aspect Weaving. In *Proceedings of 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)* (pp. 234–238).
28. Yi, Q., Niu, J., Ancha, A., & Lakshmipathy, J. (2009). Automatic Generation of Implementations For Object-Oriented Abstractions.