Automatic Source Code Generation with Intelligent Wizard Technique: Smart Home Software Composer Case Study

Samer Alhaddadin
ad0884@iu.edu.jo
Dept. of Software Engineering, Faculty of Information Technology, Isra University
Amman, Jordan

Ayad Tareq Imam
alzobaydi_ayad@iu.edu.jo
Dept. of Computer Science, Faculty of Information Technology, Isra University
Amman, Jordan

Mohammad S. Saraireh
m_srayreh@mutah.edu.jo
Computer Engineering Dept., Faculty of Engineering, Mutah University
Karak, Jordan

Abstract

The current Computer-Aided Software Engineering (CASE) tools are of notable help to developers to compose programs. With the increase in the complexity of programs' composition, there is a need for more adaptive and flexible supporting software tools. This paper proposes the definition of the Intelligent Wizard Technique (IWT) as a strategy to compose a program by collecting answers to the wizard's questions from different resources - in addition to the user. As a case study for IWT, a new Automatic Code Generator (ACG) that generates a Python language source code for the smart home application was developed, and its resulting code had been tested on a real home, which showed the code's soundness. The assessment of the IWT was achieved by using the objective measure of performance and the subjective measure of usability. IWT can be classified as an Intelligent Computer-Aided Software Engineering (I-CASE) tool.

Keywords: Automatic Source Code Generation; Wizard; I-CASE; Smart Home; Raspberry Pi; Python

1 Introduction

Code building is a technique that is used to quickly update and develop software using Automatic Code Generation (ACG) software. ACG software is an automated process intended for normal coding duties of a certain software design. ACG has exciting potential for developing programs in a faster way because it helps save time and effort, improve program quality, become more accurate, and help developers get rid of tedious routine assignments. Code generation technology is widely used and has facilitated the development of lots of several types of code generators. For example, the Java decompiler (JAD) translates byte code to Java source code [1]. We cannot ignore the significant role of the current ACG technologies, although they mainly depend on a human to get the required information to accomplish their duty of composing a source code. This is expected because the programmer's job requires innovation and creativity- it is considered a creative (non-routine) job. ACG directs software engineers to perform non-routine tasks rather than replacing them entirely. The passive code generator is a code generator type that creates a code that would be revised by the programmer [1]. ACG technique is used to develop many applications like a wizard.

The Wizard of Oz (WoZ) technology requires applicants to interact with a system that appears to be self-operated but in actuality, it is run by a person [2].

Examples of WoZ systems that are currently in use include a type of Woz system, which was created to investigate the suitability of using natural languages in Information Retrieval (IR) systems. Such systems are seen in the telephone information services like reservation services, travel information, and phone directories, which all have shown fruitful [3] [4]. Tape recordings of the questions and replies are made for subsequent transcription and analysis. Interrogation of databases or advisory systems [5] [6] [7], as well as conversations with Expert Systems (ES) [8] [2], are examples of other case studies. Most of these examples attempted to gather vocabulary corpora to fine-tune and enhance the robustness of natural language (whether spoken or written) recognition tools. Dahlbäck [2] describes a framework that allows the coupling of the observation of a graphical direct with natural language control. This framework was used to implement Turvy [9].

Despite their restricted reach, Woz’s experiments have already produced a fascinating body of research regarding wizards and assessment experts. Wizards have taught us a lot. The fact that wizards' duties are cognitively costly, despite their seeming simplicity, is an intriguing consequence of the Woz findings. The equipment's realism necessitates that the wizard's activities be constant in substance, manner, and tempo [3].

A particular order from the subject must elicit the same response from the wizard in identical situations.

The wizard's reaction time must meet the subject's expectations: if the wizard reacts too slowly, the subject may avoid utilizing simulated functions or feel the system is overburdened.

In conclusion, wizards cannot afford to improvise. Wizards must be taught well-defined duties and aided by strong tools to attain acceptable consistent behavior. To this aim, certain Woz

systems provide limited but helpful methods like a set of prepared responses or menus with pre-stored sections of answers [10].

Recent studies recommend a two-wizard setup to reduce cognitive stress [3], with one wizard dedicated to I/O operations, and the other for achieving task-level processing. The task wizard takes the requests, which had been translated by the I/O wizard, to produce the answers. Consistency is more probable with this collaborative work sharing. If the wizards are well taught, it has no discernible effect on reaction time. Another experiment [11] that used a two-wizard setup was successful.

This paper aims to propose an Intelligent Wizard Technique (IWT) that utilizes Artificial Intelligence (AI) techniques and methods to compose a source code in a high-level language. As a case study, we used our proposed IWT to create a Python source code for a smart home application.

This paper is organized as follows: the second section is the related works that report the previous ideas, proposals, and suggested approaches for developing wizards. The third section is about our proposed Intelligent Wizard Technique (IWT). The fourth section reports the case study which is the Smart Home Compose. The results section is presented next, and last, the conclusions and future work are given in the last section.

2 Related Works

In this section, we will show the approaches to Automated Source Code Generation (ASCG) using AI approaches and techniques. We present and highlight some of the works that are relevant to the problem defined in this paper.

The design pattern strategy that was introduced by Eric Gama [12], is based on the research work of Floyd [13] and the research work of Knuth [14].

Several software development tools appeared in the 1970s and 1980s: UI developers, wizards, the fourth Generation Languages (4GLs) application generators, state tools, assemblers, and compilers. These software tools are also considered ACG techniques and have been used for creating source code since long ago [15].

The symbols that resulted from the ACG methods and tools, still have a great need and human involvement. This is clearly shown by the latest ACG systems utilized by various companies or organizations (such as NASA) that need to define very accurate models for the (under development) system before creating the code for the system [1] [14]. To reduce the need for human involvement, AI techniques are used to automate code generation, such as the case-based reasoning (CBR) approach, Natural Language Processing (NLP), Genetic Algorithms (GA), and Artificial Neural Networks (ANN) [16].

The CBR approach to generating a source code can be seen in the work of Danilchenko [17] that proposed the Automatic Coder using Artificial Intelligence (ACAI) software to generate a Java-like source code from predefined cases and text queries. The CBR approach of this work

was a combination of routine design, state-based logic, and template programming that develops programs that deal directly with database operations. Another CBR approach is seen in the work proposed by Imam et al [1] as an expert code generator, which uses rule-based and frames knowledge representation techniques (ECG-RF) to generate a source code. The point of this work was the user-directed inference system that populates predefined frames of static structures software with code snippets, which were retrieved from a knowledge base. Another point of this work is the use of a wizard in terms of questions given to the users to help with code editing based on the answers to these questions. While this approach is easy to understand and use, creating rule-based systems becomes more difficult as rules become more complex, and the paper does not provide guidance on how to do this.

Talking about the wizard technique in the code generators can be seen also in the work of [18], which proposed the idea of Learning from a Wizard (LFW) to answer the question of the wizard to create a code to control a robot.

The GA approach can be seen in the work of Becker et al [19], which proposed Machine Learning (ML) software for automating the generation of complete software programs with small guidance from a human. This system, which is called AI Programmer, implements the GA, along with a firmly restricted programming language, which reduces the required search space by its ML. Another work that utilizes GA to generate source code is the one proposed by Molderez et al [20]. In this work, GA is used along with fitness functions for automating the generalization and enhancing a group of code templates that would be searched or its source code would be transformed.

The ANN approach for generating (or assisting in the generating) of a source code can be seen by Murali et al [21], which is called BAYOU. This is a system based on a neural generator to generate API-heavy Java source code. Also, the Glass box is another neural-based code generator that was proposed by Christakopoulou et al [22]. In this work, the requirements and specifications information presented by the auditor's source code of a specific program was used to validate the generated source code. The verification output was used to direct the creation of the program that meets the specifications. This approach needs the writing of a program first, which is considered unsatisfactory since it requires experience in the development and programming process. Although it could be easier to compose a program to validate an answer than to write a program to produce a correct answer, the problem of requiring a program to be written firstly makes it more difficult to be recommended because not all people or users are experienced in programming or development.

The NLP approach of AI had been used for generating source code from pseudocode, which is a natural language form by the proposed work of Imam et al [23]. The NLP approach was used also in the work of Aaqib et al [24]

2.1Conclusion of Related Works

In conclusion of the related works, ASCG tools and approaches have been developed and used to speed up and facilitate the coding process of software development. The techniques and

approaches have been leveled up by engaging AI techniques. Yet, each work implements one technique and requires human involvement. To minimize the human involvement in the coding (and somehow the solution finding), we contribute the wizard approach, for generating source code, by providing it with multiple techniques to get the required information. These techniques are either usual techniques like getting the answer from the user, or AI-based techniques like inferring an answer, searching the Internet, and learning from the environment using observation and sensors.

Based on previous related works we have gone through, we can classify the approaches used to develop ASCG into three main types, namely, the template filling, the syntax generators, and the lookup table. Fig. 1 illustrates the three approaches and the techniques used in each one. Of course, each approach has its pros and cons, which can be specified by a careful comparative study that identifies where and how to use each approach and technique.

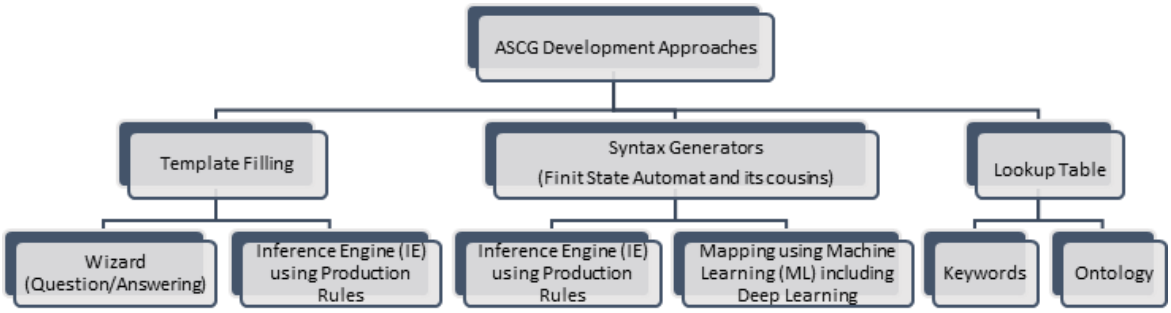


Fig. 1. ASCG Development Approaches

In this paper, we modified the production rule of the syntax generator approach to developing an ASCG as a set of production rules, by using thematic roles with the antecedents of production rules (to reveal the semantics of a sentence's words and thus control the process of generating source code from the natural language textual description) and the use of GA to select the production rules of best performance. This is why we called this modified approach of ASCG a Complex Automated Approach.

The semantic derivation of a word or a sentence, which is a critical step in natural language-based works, is a standalone subject that has sorts of techniques. Generally, sorts of linguistic attributes to reveal the semantics of a word were used in a set of works other than ASCG. For example, Knuth [28] used the attribute grammar technique to control the work of a set of production rules. Following the appearance of general NLP tools such as sentence tokenizers, detectors, POS taggers, and treebank parsers, the work in the revealing semantics of a word becomes easier. For example, the Stanford Parser was used as an NLP tool to generate a domain of discourse including sort conditions and labeled sentences to treat the input natural language use cases; there was, therefore, no need to develop a dedicated NLP tool for this work. Another example is the work of Yue et al. [29], which used the Stanford Parser to analyze input requirement sentences and classify their words into articles, nouns, pronouns, verbs, adverbs,

adjectives, and other linguistic classes. These annotated words were then used to recognize the elements of a sequence diagram. The work presented by Gulia et al. [30] encompassed syntax and semantic processes that applied a Part Of Speech (POS) to an input sentence of software requirements in natural language, to obtain more accurate recognition of the elements of the sequence diagram. A hybrid approach of semantic importance and fuzzy graph connectivity measures [31] [32]. Ayad et al [8] used verb classification, thematic roles, and semantic role labeling (SRL) to analyze the pseudocode and convert it to C# code.

3 The Proposed Intelligent Wizard Technique (IWT)

The software wizard or setup assistant is a user interface that presents the user with a series of dialogue boxes, and the user fills in the data and transfers it to the other box depending on the inputs that the user has entered and leads the user through a series of well-defined steps. Tasks that are complex, erratic, or unfamiliar to the wizard may be easier to do [1]. The will-be-adopted methodology aims to define and implement an Intelligent Wizard Technique (IWT). Fig 2 illustrates the flow of the suggested IWT solution, which contains the following parts:

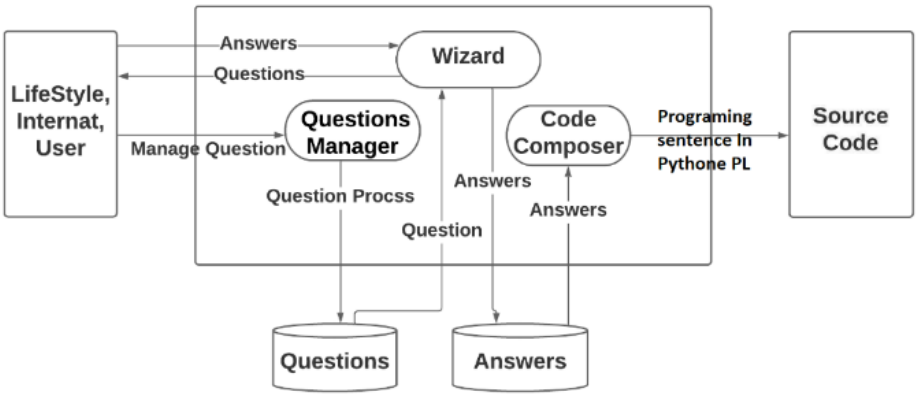


Fig. 2. The Architecture of the Proposed IWT

3.1 Lifestyle, Internet, and User

These are the resources from which our proposed IWT gets the information that is required to compose the code of a targeted application. The lifestyle is a recorder that records the way or style or behaviors of the beneficiary of the targeted (resulted) software application, which could be another application or a human. The lifestyle could be developed as a table to be part of the database of the IWT. The lifestyle is updated frequently by monitoring the user's activities while handling (or solving) a certain problem. The Internet is another source that IWT can get information from. No doubt that the Internet is a vast storage of information that anyone can get benefit from, but the main problem is how to develop a wise search for a specific datum from the Internet. Finally, the user is the human that runs the IWT, who may give direct information to the IWT to help compose the code of the targeted software. A sample example of the human source of information is what we experience in some preparing programs like Windows.

3.2 Wizard

It is the process that reading the questions from the question database and finding their answers either from a lifestyle table, the Internet, or the user. After conducting the answers, Wizard puts them in the answers database. The Pseudocode of the wizard is:

```
Wizard ( )
{
   Read a Question from the 'Questions' database
   Look for the answer in the Lifestyle Table
   If the answer is not found in the Lifestyle Table
   Then Look for the answer on the Internet
   If the answer is not founded on the Internet
        Then get the answer from the User
   Save the answer in the 'Answer' Database
}
```

3.3 Questions Manager

It is the process that manages the question database via adding, removing, updating, or listing the contents of the questions database. The questions manager process mainly the contents of the questions database regularly revising the contest by the admin of IWT to keep sure the suitability of the questions database to the targeted software application aimed to be composed. The Pseudocode for the questions manager is:

```
Questions Manager ( )
{
  Read Option
  If Option = 'Edit' Then Call the EditQuestion method
  If Option = 'Delete' Then Call the DeleteQuestion method
  If Option = ''Append' Then Call AppendQuestion
  If Option = 'List' Then the Call ListQuestions method
}
```

3.4 Code Composer

It is the process that is responsible for creating/ editing/ composing a textual form of the targeted program in a certain programming language code based on the answers, which were saved in the answers database. The code composer process uses a standard template and fills it out with a programming statement that would utilize the answers in the answers database to compose the code. The Pseudocode for the code composer is:

```
Code Composer ( )
{
     Open Answer database for reading
```

Open the code template for writing
While not the EOF Answer Database
Read an answer from the Answer Database
Apply text processing to the answer to compose a programming statement
Write the programming statements in the code template
End While
}

3.5 Questions Database

It is a database that contains a set of questions, which help supply the composer with the information it needs for composing a code. These questions are used by the wizard component of IWT, which is given to the lifestyle, the Internet, or the user. Each question is saved textually in a file. The questions vary from one application type to another and should be set by the user of the IWT before running it to compose the code of a targeted application. The Question file is controlled by the 'Question Manager' process, which is described earlier.

3.6 Answers Database

It is a database that contains the answers to the set of questions, which were given by the wizard component to the lifestyle, the Internet, or the user. These answers are used by the composer component of IWT to compose the code of the targeted software. Each answer is saved textually in the Answer file. The Answer data file is controlled by the 'Answer Manager' process, which is described earlier.

3.7 Source Code File

This is the text file of a source code that is generated automatically by our proposed IWT. It comes as a template that is defined according to the style of the Integrated Development Environment (IDE) that is used to edit, compile, and run programming source code. This template should be defined before running IWT and be filled by the composer part of the IWT.

4 Case Study: Smart Home Software Composer

As a case study for our proposed IWT, we developed a Smart Home Software Composer. A smart home application aims to provide people with a comfortable and comfortable life that contains all the means of comfort and protection [25]. A smart home consists of a group of sensors and controllers that are equipped with different objects in the home and are contacted with each other by using modern tools and technologies such as Ethernet wires. A smart home consists of electronic devices and photovoltaic energy systems connected and there is a responsible and controlling system for every part of it [26], and their information can be controlled and transmitted from/to outside the house by using smart home gates. The center-role part of a smart home is the controller like Raspberry Pi, which needs to be programmed using a programming code.

Raspberry Pi is a small, palm-sized computer with an ARMV8 microprocessor and 4GB ram. The Raspberry Pi was developed in the UK by the Raspberry Pi Foundation. They first brought

it to market in 2012 and was a huge hit. Raspberry Pi meets the needs of many things and people. For beginners and hobbyists, it was the ideal device and the best option due to its low price and at the same time powerful enough that can be easily used anywhere or to run small applications. There are some safety advantages of the Raspberry Pi, like its cheap price. Good for all groups, it can be easily installed around the house to run whatever application. Raspberry Pi is often related to monitoring or voice interaction and controlling home matters such as lighting, water consumption management, and door control. In addition, they can be part of complex projects. For example, use them to control all parts of the house. Also, the advantage when making a small project, you will get results as soon as possible [27]. Raspberry Pi can be programmed using certain programming languages like Python programming language.

Python programming language works as the most common programming language. Thanks to its highly interactive nature and mature ecosystem of scientific libraries, it is the best choice for software algorithm development and data analysis [28] [29]. However, as a code language, it is used not only in the field of computers, and it works in the fields of industry as well and in many programs [30]. Python is a very easy programming language for learning and reading. The origin of word the Python is taken from the English comics group Monty Python [31].

Not to forget, C# programming language had been used to implement the Smart Home Composer, which aims to generate a Python source code for smart home control.

4.1 Lifestyle, Internet, and User

In this case study, we utilized the Lifestyle, the Internet, and the user to be the resources from which the Smart Home Software Composer can get the information that is required to compose the code of a smart home software. As shown in Fig. 3, the lifestyle was developed as a table, and it was displayed in the GUI of the Smart Home Software Composer. Its data is got from the daily uses of the home's resident, while he/she uses the objects controlled by Raspberry Pi. Each controlled object has its record, which holds the information of the last use by the home's resident, and it will be used for future composing of the Smart Home Software. The metadata of this file encompasses the names of the object to be controlled (Name), the status of the object (status), the timing engaged with status (s-time), and finally the pin configuration in which the object was connected (again).

Name	status	s_time	gpio
AC	On	09:46:49	24
AC	Off	09:46:57	24
TV	On	10:20:25	19
TV	On	10:28:05	19
Door	On	10:29:02	25
AC	On	10:29:03	24

Fig. 3. Lifestyle Table

The second source of the data was the Internet, which was used to get the daily sunrise and sunset times of the location of the home. Such information is of big importance to automatically compose the code for controlling the lights (especially the outdoor lights) depending on such information. Of course, other unusual conditions such as being cloudy were considered and had been got from the weather website.

The last resource was the user, who gives the information directly either by selecting a predetermined alternative or typing the data directly. Worth to mention here, that Smart Home Software Composer embedded the voice recognition facility in the resulting code to give the user the ability to ask the Smart Home Software controller about some information like time, and date, or even invoke a setting process of an object under controlling.

4.2 The Wizard Part of the Smart Home Software Composer

This is the part that is responsible for collecting the answers to the questions that are used to compose the required source code. Fig 4 illustrates the Graphical User Interface (GUI) of the Smart Home Software Composer.

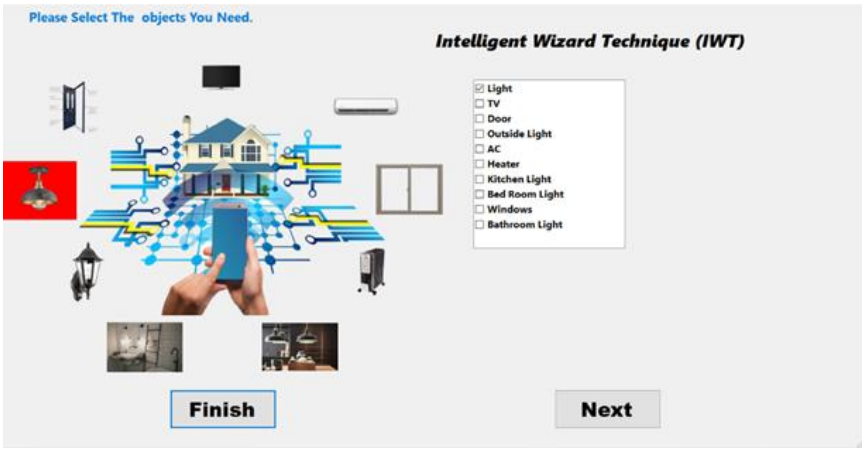


Fig. 4. The Graphical User Interface (GUI) of the Smart Home Software Composer

The wizard part had the inference ability and the flexibility property that made it able to collect the data with minimum need to the user. The inference ability was achieved by utilizing the lifestyle table and the Internet to get the answers. The flexibility property was achieved via the existence of three alternatives as a source of answers. Fig. 5 illustrates the method that functions as a wizard part of the Smart Home Software Composer.

```
//wizerd To Select Objects
var input = checkedListBox1.CheckedItems.Count;
//var input = Int32.Parse(Alaa.Text);
for (int i = 1; i <= input; i++)

{
    // string promptValue = Prompt.ShowDialog(xxx + " " + arr[i + count], "Button " + i);
    Button btn = new Button();
    //   btn.Text = promptValue.ToString();
    btn.Name = count.ToString();
    btn.Size = new Size(150, 100);
    btn.Font = new Font(btn.Font.FontFamily, 16);
    btn.Location = new Point(50, 110 * y);
    // ------------------------------------------------------

    //  MySqlConnection connection = new MySqlConnection("server=86.108.18.247;port=13606;Database=gpioDB;User ID=root;Password=123");
    MySqlConnection connection = new MySqlConnection("server=94.249.2.85;port=39379;Database=gpioDB;User ID=root;Password=123");


    MySqlCommand commandDatabase = new MySqlCommand("INSERT INTO `Answers` (`id`, `gpio_no`, `status`, `Name`, `used`) VALUES ('1','" + arr[i
    count--;
    commandDatabase.CommandTimeout = 60;
```

Fig. 5. The Wizard Method of the Smart Home Software Composer

4.3 Questions Manager Part of the Smart Home Software Composer

It is the process that is responsible for managing the contents of the 'Question' database file. It offers services for adding, removing, modifying, and deleting the questions in the 'Questions' file. The importance of this process is shown by the required need to set up the composer to compose a source code for a certain application, which is different from other types of applications, and thus, it needs a different set of questions. Fig. 6 represents a screenshot of the code of this method.

```
//Questions Manager
MySqlConnection con2 = new MySqlConnection("server=94.249.2.85;port=39379;Database=gpioDB;User ID=root;Password=123");
//MySqlConnection conc = new MySqlConnection("server=86.108.18.247;port=13606;Database=gpioDB;User ID=root;Password=123");

MySqlCommand cmd2 = new MySqlCommand("select question from Questions where id=2", con2);

con2.Open();

MySqlDataReader rdr = cmd2.ExecuteReader();

rdr.Read();

//var reader = cmd.ExecuteReader();
var xxx = rdr.GetString(0);
// MessageBox.Show(xxx.ToString());
rdr.Close();
```

Fig. 6. Question Manager Method

4.4 Code Composer Part of the Smart Home Software Composer

This is the heart part of the Smart Home Software Composer. The code composer process functions at creating or generating a Python source code for controlling a home (smart home application). Code composer fills (by writing) a predefined template, which is a textual file that meets the structuring of a Python source code. In addition to the header and other complementary statements, the main type of statement is the 'IF – THEN' statement, which requires the condition part and the action part. Both of these two parts were got from the 'Answers' database file, which had been filed already by the wizard part of the Smart Home Software Composer. Fig. 7 is a screenshot of this method.

```
//Generating Python Files
using var client = new SshClient("94.249.2.85",34459, "pi", "123");
client.Connect();
// client.RunCommand("mkdir "+DateTime.Now.ToString());
var cmd = "import os\nos.system(\"echo " + "'1'" + " > /sys/class/gpio/gpio" + arr[i + count] + "/value\")";
var cmdx = "import os\nos.system(\"echo " + arr[i + count] + " > /sys/class/gpio/unexport\")\n" +
     "os.system(\"echo " + arr[i + count] + " > /sys/class/gpio/export\")\n" +
     "os.system(\"sudo chmod 777 /sys/class/gpio/gpio" + arr[i + count] + "/value\")\n" +
     "os.system(\"echo 'out' > /sys/class/gpio/gpio" + arr[i + count] + "/direction\")";
//  MessageBox.Show(cmdx);
client.RunCommand("echo '" + cmdx + "' > /home/pi/samer/gpio" + arr[i + count] + ".py");
cmd = "import os\nos.system(\"echo " + "'1'" + " > /sys/class/gpio/gpio" + arr[i + count] + "/value\")";
//  MessageBox.Show(cmd);
client.RunCommand("echo '" + cmd + "' > /home/pi/samer/gpio" + arr[i + count] + "off.py");
cmd = "import os\nos.system(\"echo " + "'0'" + " > /sys/class/gpio/gpio" + arr[i + count] + "/value\")";
//  MessageBox.Show(cmd);
client.RunCommand("echo '" + cmd + "' > /home/pi/samer/gpio" + arr[i + count] + "on.py");

client.Disconnect();
```

Fig. 7. A Screenshot of Code Composer Part of the Smart Home Software Composer

4.5 Questions Database Part of the Smart Home Software Composer

These are the questions aimed to collect the data related to the Smart Home controller. These questions had been set by us and saved in the 'Questions' database file. Examples of these questions are:

- 'How many buttons do you want to connect your system?'

- ‘What time do you like to turn on the TV’
- ‘How many degrees do you like the temperature of the room to be?’

Note that some questions are chained together since they form the conditions or actions of a single programmed object. Fig. 8 is a screenshot of the ‘Question’ database file, which is developed as a table that consists of 25 questions. Each question was saved as a record that encompasses three fields namely: id (the question number), question (the question’s text), and note (if there is something to be considered about this question).

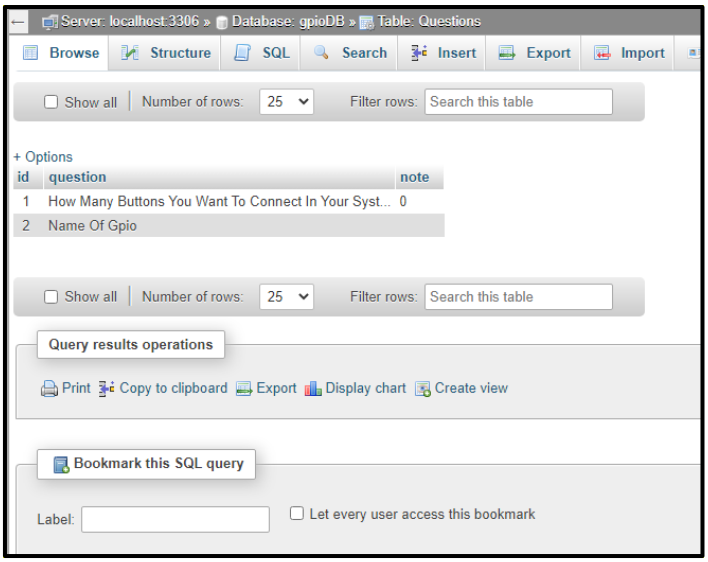


Fig. 8. A Screenshot of the ‘Question’ Database File

4.6 Answers Database Part of the Smart Home Software Composer

This database file holds the answers that would be collected by the wizard part of the Smart Home Software Composer. As shown in Fig. 9, the ’Answer’ database file had been designed as a table that consists of five fields, which are:

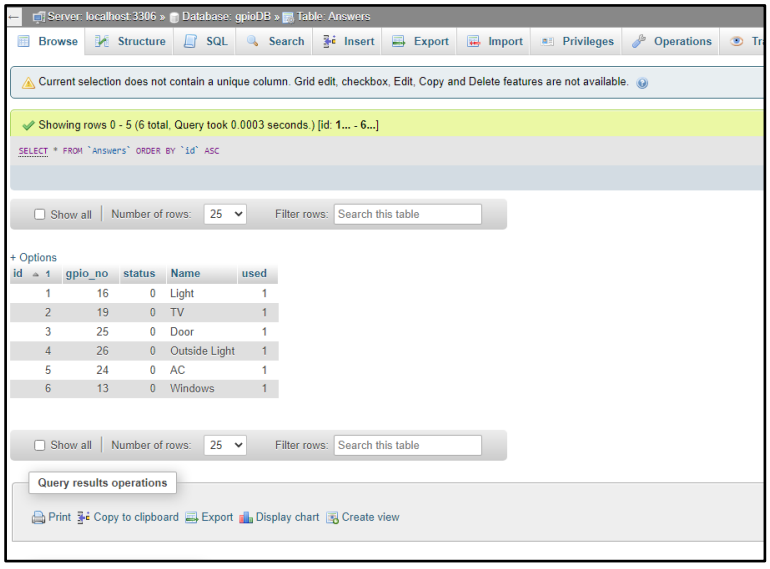


Fig. 9. A Screenshot of the ‘Answers’ Database File

- id: the sequence of the answer
- gpio_no: the Raspberry Pi PIN, where a certain object had been controlled.
- status: numerical representation of on/off
- Name: the device name
- used: numerical representation of the used/unused pin

Finally, this database file got its contents from the wizard part and used the composer part to assist in composing the Raspberry Pi Python source code.

4.7 The Resulted Source Code File

This is the text file of the Raspberry Pi Python source code that had been generated automatically by the composer part of our proposed IWT. It came as a template that is defined according to the style of the Raspberry Pi Python source code. Fig. 10 shows a sample of the Python code.

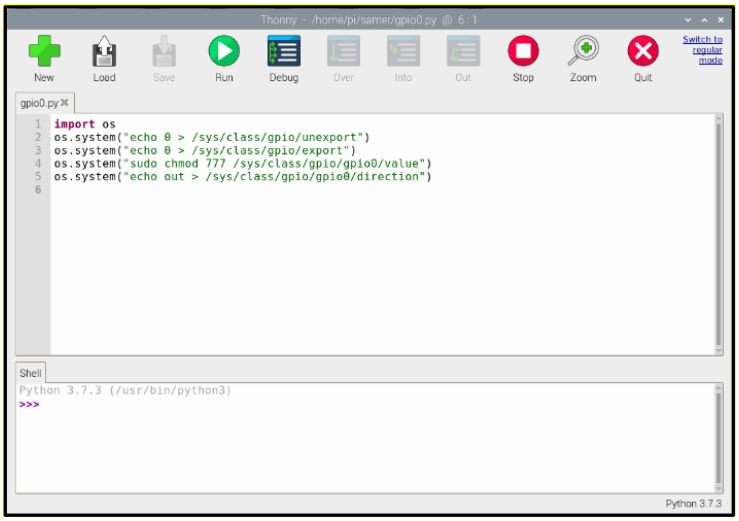


Fig. 10. Sample of the Python Code

4.8 The Results and Evaluation of the Smart Home Software Composer

Fig 11 illustrates the GUI of the Smart Home Software Controller. This interface was the fixed part of the template that had been filled by the composer. The result Smart Home Software controller is very useful in monitoring and controlling the smart home environment.

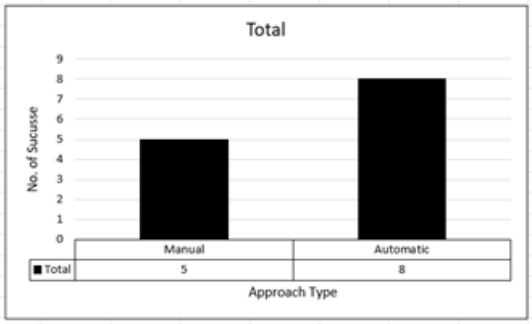
Fig. 11. The GUI of the Smart Home Software Controller

The evaluation of the Smart Home Software Composer has been done by using the objective measure of performance, and the subjective measure of Usability. These two measures were performed depending on the feedback we got after using Smart Home Software Composer from 10 programmers, who had been asked to compose the Smart Home Software Controller manually at first, and later automatically using the Smart Home Software Composer.

To evaluate the performance (the objective measure), we counted the number of people who failed to compose the Smart Home Software Controller manually (from the first try) compared to the automatic composing of it using the Smart Home Software Composer. Table I shows the number of succussed tries of the manual approach vs the automatic approach. It should be noted that the "failure" may be one of three alternatives. The first one is the failure to complete coding the Smart Home Software Controller within a given time. The second one is failing due to failing to find the correct answer to the questions of the wizard part. The third type of failure is that caused by giving the wrong answer to the questions of the wizard part.

TABLE I. THE NUMBER OF SUCCUSSED TRIES OF THE MANUAL APPROACH VS THE AUTOMATIC APPROACH

#	Manual	Automatic
1	0	1
2	1	1
3	1	0
4	0	1
5	0	1
6	1	1
7	1	0
8	0	1
9	0	1
10	1	1
Total	5	8

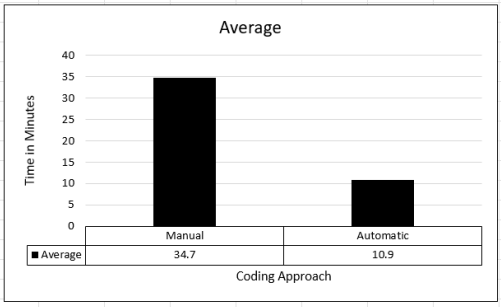


From Table I, it can be seen that fewer failures were recorded with the Smart Home Software Composer than with the manual one.

Not only the number of failings but the speed of accomplishing soundness coding had been measured also. Table II shows the average time (in minutes) spent by each programmer using the manual approach, and the automatic approach by using the Smart Home Software Composer.

TABLE II. THE AVERAGE TIME OF THE MANUAL APPROACH VS THE AUTOMATIC APPROACH

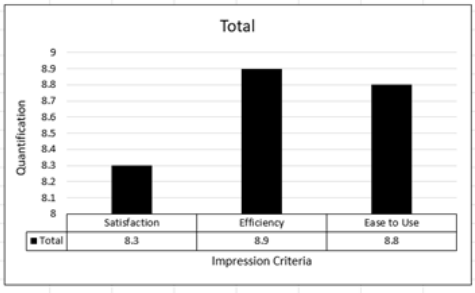
#	Manual	Automatic
1	30	10
2	33	8
3	28	12
4	35	13
5	30	9
6	45	11
7	31	10
8	34	12
9	39	11
10	42	13
Average	34.7	10.9



Here it is obvious that the average time for composing the Python source code was longer in the manual than in the use of the Smart Home Software Composer.
To evaluate Usability (the subjective measure), each programmer of the 10 participating individuals was asked to give her/his impression of using the Smart Home Software Composer in terms of three factors namely satisfaction, efficiency, and ease to use in a range from 1 to 10. The results are shown in Table III. Here we can see that the Smart Home Software Composer was – in general - favorable in the responses.

TABLE III. THE USABILITY MEASURES OF THE SMART HOME SOFTWARE COMPOSER

#	Satisfaction	Efficiency	Ease to Use
1	7	9	9
2	8	10	9
3	8	9	8
4	10	10	9
5	8	8	9
6	9	8	9
7	7	8	8
8	8	9	9
9	9	10	9
10	9	8	9
Total	8.3	8.9	8.8



6 Conclusions and Future Works

In this paper, the definition of IWT is given and has been applied to develop a case study of a code generator that composes a software controller for smart home applications. The wizard of the case study can collect answers from a variety of sources by using inference techniques, searching the Internet, and learning from the environment using observation and sensors.

Several important findings are revealed during the work. The first one is the suggestion of classification for the ASCG development approaches, which helps direct the new works based on the features of each ASCG class. The second finding is the definition of IWT, as a principal, which levels up the capabilities of the wizard technique and represents a framework for this technique. IWT contributes to I-CASE tools [32]. IWT is of high flexibility that makes any user able to generate source code in any programming language regardless of the need to gain coding skills with this programming language. This is a noteworthy result as it is considered one of the key issues of Human-Computer Interaction (HCI). Furthermore, IWT has the knowledge acquisition ability as shown by the Lifestyle updating or recording in the Smart Home Software Composer case study given in this paper. This ability can be considered a new ML approach.

The results of the evaluation show that the IWT approach scores higher in both subjective usability and objective performance when compared with the traditional manual approach to coding. Worth to note here, that the data presented here is based on a small sample (ten programmers) and should be deemed as preliminary.

For future work, we recommend utilizing ANN and GA to compose the code. These two techniques need special consideration when used for editing purposes. While the case study given in this paper has a limited ability to recognize spoken commands, we look forward to the

development of an NLP unit that would be part of the IWT to support the collection of spoken answers rather than written ones. Also, we recommend the development of IWT as an Integrated Development Environment (IDE) and supporting it with library functions. Also, making IWT can engage with well-known design, coding, and testing software tools.

References

[1] A. T. Imam, T. Rousan and S. Aljawarneh, "An Expert Code Generator using Rule-Based and Frames Knowledge Representation Techniques," in *5th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan, 2014.

[2] N. Dahlbäck, A. Jönsson and L. Ahrenberg, "Wizard of Oz studies-why and how," *Knowledge-Based Systems*, vol. 6, no. 4, pp. 258-266, 1993.

[3] D. Salber and J. Coutaz, "Applying the Wizard of Oz Technique to the Study of Multimodal Systems," in *Third International Conference on Human-Computer Interaction (EWHCI '93)*, Moscow, Russia, 1993.

[4] R. Gulndon, "Grammatical and ungrammatical structures in user-adviser dialogues: evidence for sufficiency of restricted languages in natural language interfaces to advisory systems," in *The 25th annual meeting on Association for Computational Linguistics*, Stanford, California, USA , 1987.

[5] S. Whittaker and P. Stenton, "User studies and the design of Natural Language Systems," in *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, Manchester, England, 1989.

[6] A. Jönsson and N. Dahlbäck, Talking to a computer is not like talking to your best friend, Linköping, Sweden: Linköping University, 1988.

[7] D. Diaper, "The Wizard's Apprentice: A Program to Help Analyse Natural Language," in *The fifth conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*, Nottingham, U.K, 1989.

[8] Y. Polity, J.-M. Francony, R. Palermiti, P. Falzon and S. Kazma, "Recueil de dialogues Homme-machine en langue naturelle écrite," *Les cahiers du Criss*, vol. 17, 1990.

[9] D. Maulsby, S. Greenberg and R. Mander, "Prototyping an Intelligent Agent through Wizard of Oz," in *The INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, Amsterdam, The Netherlands, 1993.

[10] A. J. N Dählback, "Empirical studies of discourse representations for natural language interfaces," in *The fourth conference on European chapter of the Association for Computational Linguistics*, Manchester, England, 1989.

[11] J.-M. Francony, "Towards a methodology for wizard of oz experiments," in *Third Conference on Applied Natural Language Processing*, Trento, Italy, 1992.

[12] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns Elements of Reusable Object-Oriented Software, vol. 99, New York: ADDISON-WESLEY, 1995.

[13] R. W. Floyd, "A Descriptive Language for Symbol Manipulation," *Journal of the ACM,* vol. 8, no. 4, pp. 579--584, 1961.

[14] D. Knuth, "Semantics of Context-Free Languages," *Mathematical systems theory*, vol. 2, pp. 127--145, 1968.

[15] E. Kitzelmann, "Inductive programming: A survey of program synthesis techniques," in *Third International Workshop on Approaches and Applications of Inductive Programming,* , Edinburgh, UK, 2009.

[16] F. L. Loaiza, D. A. Wheeler and J. D. Birdwell, "A Partial Survey on AI Technologies Applicable to Automated Source Code Generation," Institute for Defense Analyses, Alexandria, USA, 2019.

[17] Y. Danilchenko and R. Fox, "Automated Code Generation using Case-Based Reasoning, Routine Design and Template-Based Programming," in *Midwest Artificial Intelligence and Cognitive Science Conference*, Cincinnati, USA, 2012.

[18] W. B. Knox, S. Spaulding and C. Breazeal, "Learning from the wizard: Programming social interaction through teleoperated demonstrations," in *The 2016 International Conference on Autonomous Agents & Multiagent Systems*, Singapore, 2016.

[19] K. Becker and J. E. Gottschlich, "AI Programmer: Autonomously Creating Software Programs using Genetic Algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Lille, France, 2021.

[20] T. Molderez and C. D. Roover, "Automated Generalization and Refinement of Code Templates with Ekeko/X," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, Osaka, Japan, 2016.

[21] V. Murali, L. Qi, S. Chaudhuri and C. Jermaine, "Neural sketch learning for conditional program generation," in *Sixth International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 2017.

[22] K. Christakopoulou and A. T. Kalai, "Glass-Box Program Synthesis: A Machine Learning Approach," in *Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, USA, 2018.

[23] A. T. Imam and A. J. Alnsour, "The Use of Natural Language Processing Approach for Converting Pseudo Code to C# Code," *Journal of Intelligent Systems,* vol. 29, no. 1, pp. 1388-1407, 2020.

[24] A. A. R. Ansari and D. R. Vora, "NLI-GSC: A Natural Language Interface for Generating Source Code," *International Journal of Advanced Computer Science and Applications,* vol. 13, no. 1, pp. 842-853, 2022.

[25] M. li, W. Gu, W. Chen, Y. He, Y. Wu and Y. Zhang, "Smart home: architecture, technologies and systems," in *8th International Congress of Information and Communication Technology*, Tianjin, 2018.

[26] E. I. Davies and V. Anireh, "Design and Implementation of Smart Home System Using Internet of Things," *Journal of Digital Innovations and Contemporary Research In Science, Engineering and Technology,* vol. 7, pp. 33--42, 17 Jan 2019.

[27] J. F. Nusairat, "Raspberry Pi," O'Reilly online learning, 2020. [Online]. Available: https://www.oreilly.com/library/view/rust-for-the/9781484258606/html/481443_1_En_8_Chapter.xhtml.

[28] P. F. Dubois, "Python: Batteries Included," *Computing in Science and Engineering,* vol. 9, no. 3, pp. 7-9, 2007.

[29] K. Milmann and M. Avaizis, "Python for Scientists and Engineers," *Computing in Science and Engineering* , vol. 13, no. 2, pp. 9-12, 2011.

[30] P. Fabian, V. Gaël, G. Alexandre, M. Vincent, T. Bertrand, G. Olivier, B. Mathieu, . P. Peter, W. Ron and D. Vincent, "Scikit-learn: Machine learning in Python," *The Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[31] B. Ballmann, Python Basics, Uster: Springer, 2020.

[32] A. T. Imam, A. J. Al-Nsour and A. Hroob, "The Definition of Intelligent Computer Aided Software Engineering (I-CASE) Tools," *Journal of Information Engineering and Applications,* vol. 5, no. 1, pp. 47-56, 2015.