

2008 Senior Thesis Project Reports

Iliano Cervesato* **Majd Sakr*** **Mark Stehlik†**
Bernardine Dias*‡ **Lynn Carter***

May 2008, revised June 2010
CMU-CS-QTR-101

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Qatar campus. †Department of Computer Science. ‡Robotics Institute.

The editors of this report include the members of the Senior Thesis Committee on the Qatar campus, and the advisors of the students whose theses are included herein.

Abstract

This technical report retrospectively collects the final reports of the undergraduate Computer Science majors from the Qatar Campus of Carnegie Mellon University who elected to complete a senior research thesis in the academic year 2009–10 as part of their degree. These projects have spanned the students' entire senior year, during which they have worked closely with their faculty advisors to plan and carry out their projects. This work counts as 18 units of academic credit each semester. In addition to doing the research, the students presented a brief midterm progress report each semester, presented a public poster session in December, presented an oral summary in the year-end campus-wide Meeting of the Minds and submitted a written thesis in May.

Keywords: Assistive Computing Technology, Technology for the Developing World, Literacy, Software Engineering, Code Verification.

Contents

Noura Mohammed El-Moughny

Assistive Computing Technology for Learning to Write Braille 1

Advisor: M. Bernardine Dias

Amer Hasan Obeidah

Design – Code Verification: When Design Deviates from Code 35

Advisor: Lynn Robert Carter

Assistive Computing Technology for Learning to Write Braille

*Undergraduate Senior Thesis
April 24, 2008*

جامعة كارنيغي ميلون في قطر
Carnegie Mellon
QATAR CAMPUS



By

Noura Mohammed El-Moughny
Computer Science Undergraduate, Senior
Carnegie Mellon University

Advisor

M. Bernardine Dias, Ph. D.
Research Scientist, The Robotics Institute
Founder and Director, TechBridgeWorld (www.techbridgeworld.org)
Carnegie Mellon University

ABSTRACT

If they are to play a meaningful role in modern society, people who are visually impaired need to obtain information in an effective and timely manner. Accessing information requires the ability to read and write fluently. The Braille language provides a mechanism for the visually impaired to be fully literate participants in modern-day society. However, learning to write Braille is a non-trivial process that often involves long hours of tedious work. Learning to write Braille is even more difficult for young blind children due to many factors such as the required physical and mental exertion, and the delayed feedback on what was written. To address these needs, the TechBridgeWorld program at Carnegie Mellon University (www.techbridgeworld.org) developed an Adaptive Braille writing Tutor (ABT) that uses audio feedback to provide guided practice for young children learning to write Braille. Through extensive interactions with the Al-Noor Institute, a school for blind children in Qatar, we extend the capabilities and potential impact of this Braille tutor.

This honors thesis in Computer Science enhances the ABT in three important dimensions: Relevance to the Arab world, methodology of software design, and motivational factor for the students. Our interactions with Al Noor revealed a need for expanding the vocabulary of the ABT to include Arabic Braille, and also a strong need for increasing the enthusiasm and learning-efficiency for blind children learning to write Braille. To address these needs we make three important enhancements to the ABT. First, we enable the tutor to provide guided practice for the Arabic alphabet characters in Braille and facilitate the interface between the Braille tutor and the screen reading software used at the Al Noor Institute. Next, we improve on the ad-hoc design of the ABT software components by combining research methodologies in Assistive Technology, Intelligent Tutoring Systems, and Artificial Intelligence to propose a principled re-design of the ABT software. Finally, we study the literature on Educational Game Design and create an educational ABT-computer game to increase the motivation of children learning to write Braille. The outcome of this project is an improved Adaptive Braille Writing tutor that enhances the state of art in educational technology for the visually impaired.

TABLE OF CONTENTS

1	INTRODUCTION.....	1
2	ASSISTIVE TECHNOLOGY AT THE AL-NOOR INSTITUE.....	3
2.1	SCREEN READER SOFTWARE.....	3
2.2	BRAILLENOTE.....	3
2.3	KEYBOARDS.....	4
3	TECHBRIDGEWORLD'S ADAPTIVE BRAILLE WRITING TUTOR.....	6
4	GUIDED PRACTICE FOR ARABIC BRAILLE LETTERS.....	8
5	INTELLIGENT TUTORING SYSTM.....	11
5.1	DOMAIN KNOWLEDGE.....	12
5.2	PEDAGOGICAL MODULE.....	14
5.3	EXPERT MODEL.....	17
5.4	COMMUNICATION MODEL.....	18
5.5	STUDENT MODEL.....	18
6	EDUCATIONAL COMPUTER GAMES.....	22
6.1	GAME INSPIRATION.....	22
6.2	I/O STRUCTURE.....	23
6.3	GAME STRUCTURE AND RULES.....	23
6.4	PROGRAM STRUCTURE.....	25
7	CONCLUSIONS AND FUTURE WORK.....	26
	ACKNOWLEDGEMENTS.....	27
	REFERENCES.....	27
	APPENDIX: Arabic Alphabet Mapping to Braille.....	29

1 INTRODUCTION

Braille¹ is a widely-used language that allows visually impaired people to read and write. Each Braille character is formed using six dots placed in a cell of two columns and three rows. A subset of these six dots is embossed to represent each character. Thus, a dot may be raised at any of the six positions to form sixty-four (2^6) unique combinations. The positions of the six dots are universally numbered from one to six. Figure 1 below (presented by Kalra et al [1]) shows schematics of a Braille cell, and pictures of a slate and stylus used to write Braille and of a written Braille sample.



Figure 1[1]: (a) Schematic of a Braille cell (b) The Braille letter “T”: The black circles represent embossed dots, and the light grey circles represent the positions of the dots that are not embossed (c) A traditional slate and stylus (d) Written Braille

The Braille language has enabled visually-impaired people to actively participate in modern-day society. Despite its significance, and the accessibility it brings, learning to write Braille still has a number of barriers. More than 90% of the world's 161 million visually-impaired people live in developing communities [2]. The literacy rate of this population is estimated to be below 3% [3]. Unfortunately, poorer areas tend to have both a disproportionately high number of blind people [2] and fewer resources for educating them [1]. Therefore, the need to improve literacy for the blind in affordable ways is paramount.

The traditional method of writing Braille itself creates formidable challenges to literacy. In developed countries, Braille is usually embossed with a six-key typewriter known as a Braille which is shown in Figure 2 (a). These devices are fast and easy to use but usually cost over US\$600 each [5]. In developing countries, such devices are prohibitively expensive and Braille is almost always written with a slate and stylus as shown in Figure 1 (c). Using these tools, Braille is written from right to left so that the page can be read from left to right when it is removed from the slate and turned over. Learning to write Braille in this manner can be difficult. First, children must learn mirror images of all letters, which doubles the alphabet and creates a disparity between the written and read forms of each letter. Second, feedback is delayed until the paper is removed and then flipped over and read. For young children, this delay can make Braille conceptually challenging since the act of writing has no discernable, immediate effect. It also takes longer for both the student and the teacher to identify and correct mistakes, and this slows learning. Some of the newest slates, such as the example shown in Figure 2(b), allow the writer to read the Braille through the back of the slate as they write. However, the cost of these slates is significantly higher compared to the regular slate and stylus, and hence they are not affordable or easily available in developing communities[7]. Even with the newest slates or the Braille

¹ See <http://www.learnthat.com/define/view.asp?id=7394> for more information on Braille

feedback to students is delayed compared to sighted students since the feedback requires tactile contact between the student's fingers and the written Braille. Finally, even the thick paper used to write Braille may be expensive or in limited supply [9]. In sum, these challenges contribute to the problem of illiteracy among the blind in both developing and developed communities. Thus, assistive Braille-writing technology relevant and accessible to developing communities could have significant impact on millions of lives. To address these needs, the TechBridgeWorld program at Carnegie Mellon University (www.techbridgeworld.org) developed an Adaptive Braille writing Tutor (ABT) that uses audio feedback to provide guided practice for young children learning to write Braille (www.techbridgeworld.org/brailletutor/).



Figure 2: (a) [4] Brailier (b) [6] New slate and stylus allows users to read as they write

This senior honors thesis in Computer Science aims to serve the needs of visually impaired children learning to write Braille. We are specifically interested in serving the needs of visually impaired children in developing communities and in Qatar. To this end, we established a strong partnership with the Al-Noor Institute, a school for blind children in Qatar, and surveyed the literature to understand the needs and challenges of the visually impaired in developing communities. Based on our findings, we extend the capabilities and potential impact of the ABT in three important directions. First, we enable the tutor to provide guided practice for the Arabic alphabet characters in Braille and facilitate the interface between the ABT and the screen reader software used at the Al Noor Institute. Our second improvement addresses the ad-hoc design of the ABT software components by combining research methodologies in Assistive Technology, Intelligent Tutoring Systems (ITS), and Artificial Intelligence to propose a principled re-design of the ABT software. Finally, we study the literature on Educational Game Design and create an educational ABT-computer game to increase the motivation of children learning to write Braille. The outcome of this project is an improved Adaptive Braille Writing tutor that enhances the state of art in educational technology for the visually impaired and reduces the number of barriers to Braille literacy in developing communities and in Qatar.

We organize the remainder of this report as follows. In section 2 we describe the current technology tools used at the Al-Noor Institute for the Blind in Qatar. In section 3 we provide a brief introduction to Carnegie Mellon University's Adaptive Braille Writing Tutor and focus on the second version of the tutor that is most relevant to our work. Section 4 follows with implementation details for enabling the tutor to provide guided practice for the Arabic alphabet in Braille. Section 5 details the re-designed components of the Adaptive Braille Writing Tutor software according to the ITS methodology. Section 6 discusses the creation of an educational computer-tutor game for increasing enthusiasm for learning to write Braille. Finally, we summarize our work in section 7 and conclude the report in section 8 with proposed directions for future research.

2 ASSISTIVE TECHNOLOGY AT THE AL-NOOR INSTITUTE

Before we describe our work with the Adaptive Braille Writing Tutor, we introduce the Al Noor Institute, our primary source of information for the pedagogy and challenges of teaching Braille. The Al-Noor Institute for the Blind is a school for visually-impaired children in Qatar. Inaugurated in 1998 under the support of Her Highness Sheikha Mozah Bint Nasser Al-Missned, the institute offers many programs and facilities that serve the needs of the visually impaired in Qatar. The school works to extend educational services to the blind to help them overcome their disabilities and prepare them to undertake more productive roles in society. We have learned much about their teaching methods, needs, and challenges through several site visits. During these visits we observed the abundance of technology available for blind individuals at the institute. The available assistive technology varies in its use and caters to the different needs of the visually-impaired. The information we report is primarily based on conversations with Mr. *Yasser Al-Shrafai* and Mr. *Suleiman M. Abu Azab* who are both teachers at the Al-Noor Institute.

2.1 SCREEN READER SOFTWARE

The Al Noor Institute equips its computers with three types of screen reader software that assist blind individuals to use computer applications. These software programs are “IBSAR,”² “JAWS”³, and “HAL”⁴. These programs help visually impaired users to navigate between different applications on the computer. They audibly pronounce, in Arabic, the command level, the content of the computer’s screen, and the different applications browsed by the user. Therefore, they mainly depend on the hearing ability of the user. Figure 3 shows a blind individual using Microsoft Word with the assistance of the IBSAR software.

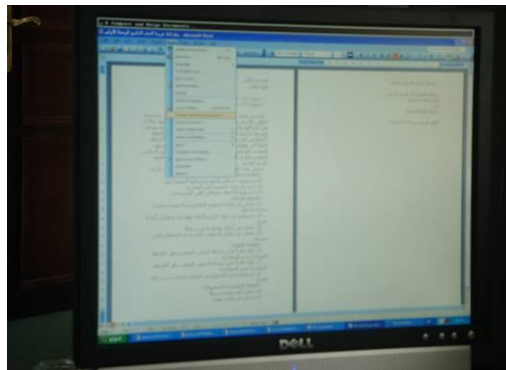


Figure 3: IBSAR software audibly pronounces different commands in Microsoft Word

2.2 BRAILLENOTE

The BrailleNote assistive technology allows blind individuals to read the text written on the computer screen in Braille format. It translates the text from different applications browsed by

² See <http://www.sakhr.com/products/Ibsar/Default.aspx?sec=Product&item=Ibsar> for more information about IBSAR software

³ See [http://en.wikipedia.org/wiki/JAWS_\(screen_reader\)](http://en.wikipedia.org/wiki/JAWS_(screen_reader)) for more information about JAW software

⁴ See <http://www.synapseadaptive.com/dolphin/hal.htm> for more information about HAL

the user to written Braille. This tool is useful for visually-impaired people who are familiar with reading Braille and prefer reading to using screen reader software. Figure 4 shows a picture of BrailleNote being used by a visually impaired user.



Figure 4: A teacher at Al-Noor Institute using BrailleNote

2.3 KEYBOARDS

Students at the Al-Noor Institute are taught to use a standard keyboard from grade one since it is the main tool that they can use to navigate between different applications on the computer. Blind individuals can't use the mouse as sighted people do. Therefore, students are taught to use certain keys that help them to move easily between browsing different applications on a computer (such as the arrows, shift, caps lock, and alt keys). These keys are essential for students to work easily and choose between different commands shown on the computer screen.

Another way of using the keyboard is to make use of only six keys. These keys represent the Braille cell. Specifically, the keys F, D, and S on the keyboard represent the dots 1, 2, and 3 respectively and the keys J, K, and L represent the dots 4, 5, and 6 respectively. This technique is practical for users who are not familiar with the configuration of keys on the keyboard. They can use those six keys to type different letters in Braille and those letters are translated to printed text on the computer. Then the printed text can be translated back to Braille through the BrailleNote. Figure 5 shows one of the teachers at the Al-Noor Institute using the BrailleNote and keyboard with the IBSAR software while working on the computer.



Figure 5: A BrailleNote, the keyboard, and the IBSAR software being used on a computer

Along with the assistive technology mentioned above, it is worth mentioning the enhanced technology used in printing books in Braille for the students. In addition to teaching, Al-Noor is the only place in Qatar that prints school books in Braille for students from the age of 6 until the final year in college. The printing department is equipped with enhanced technology that assists in meeting these educational needs of the visually impaired. The Al-Noor Institute also has an integration program that aims to integrate blind youth aged 12 years and older into the public schools in Qatar.

3 TECHBRIDGEWORLD'S ADAPTIVE BRAILLE WRITING TUTOR

The Adaptive Braille Writing Tutor (www.techbridgeworld.org/brailletutor/) is a tool designed by Carnegie Mellon University's TechBridgeWorld program to assist blind children to learn the art of writing Braille. It consists of an electronic slate and stylus known as the E-slate which monitors the student's writing and transmits data in real time to the computer. The tutor's software runs on an external PC and translates the data from the E-slate to provide immediate audio feedback to the user [1]. The original version of this tutor was field tested in the Mathru School for blind children in Bangalore, India (see Figure 6).



Figure 6: Field-test of the original version of the Braille tutor at the Mathru School in India

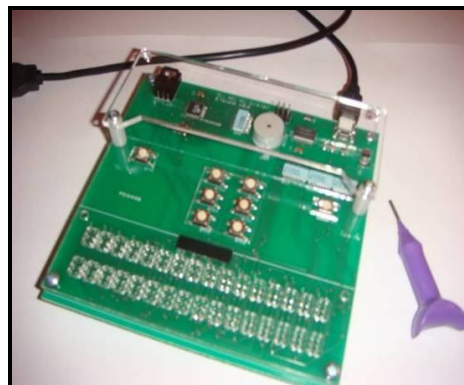


Figure 7: Adaptive Braille Tutor developed by the TechBridgeWorld program

Based on feedback from this early field-test, a second version of the tutor (shown in Figure 7) was designed and implemented by several Carnegie Mellon University students and faculty. Our work is implemented on this second version of the tutor which provides guided practice for learning to write Braille using a slate and stylus, or six buttons that represent the six dots of the Braille cell. The E-slate monitors the student's writing and transmits the data in real time to a computer linked via a USB cable. The transmitted data is then interpreted to provide immediate audio feedback to the user via text-to-speech synthesis or the teacher's recorded voice [1].

Further details of the hardware and software implementations for both versions of the tutor are discussed in prior publications by Kalra et al. ([1] and [12]).

The second version of the Adaptive Braille Writing Tutor consists of two rows of 16 Braille cells and six buttons placed over the top of the two rows (shown in Figure 8) to work as an input area on the E-slate. The stylus is a standard Braille stylus that connects to the E-slate by its metal tip. Moreover, students can press on one of the two additional control buttons placed on the E-slate to indicate the completion of a character or a word.

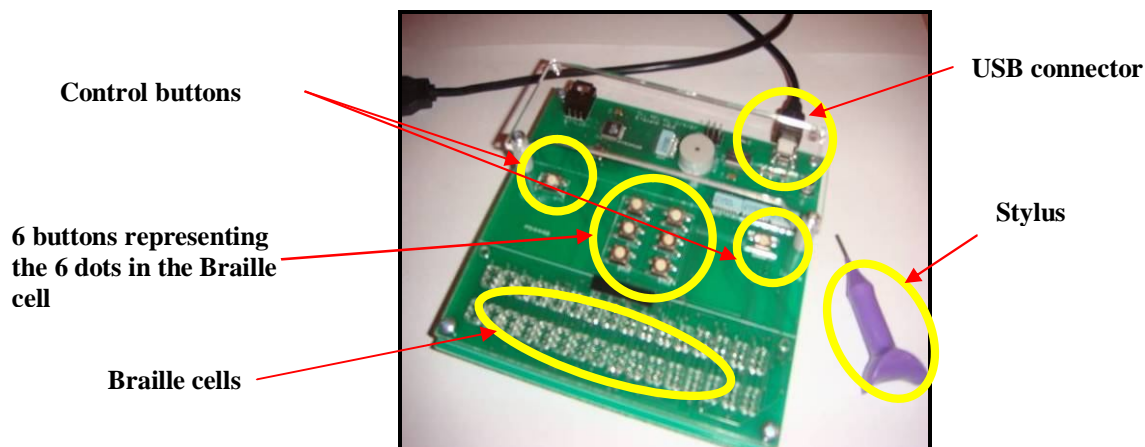


Figure 8: Main Features of the Adaptive Braille Tutor

The ABT software is implemented in C++ and both the software and hardware for the two versions of the tutor will soon be available under Open Source licenses via the TechBridgeWorld website (www.techbridgeworld.org/brailletutor/downloads.html). Tom Stepleton, a Ph.D. student in the Robotics Institute at Carnegie Mellon University, developed the software for the second version of the ABT and created a detailed installation guide for using the tutor and some initial instructions for developing software for the tutor [8]. Compiling the Braille Tutor library on a Windows machine (the most commonly available Operating System in the schools for the blind we have worked with) requires the installation and compilation of several software packages. (Note that the second version of the ABT software is cross-platform). The instructions for compiling the ABT software are adapted to use tools that are affordable for programmers in developing communities. The first key component of the instructions involves downloading the GNU C++ compiler that is part of MinGW. The GNU C++ compiler is a free compiler for Microsoft Windows that can be used to compile the Braille Writing Tutor software. The next requirement is to download and compile the Boost library which is a multiplatform compilation of C++ libraries used to enable threading in the ABT. Finally, the Braille Tutor interface library must be downloaded and compiled (using the GNU C++ compiler). Once these three major components are complete, the tutor hardware is connected to the computer via a USB connector, enabling the use of the tutor as well as software development and testing.

Three areas for improvement in the second version of the tutor are its limitation to supporting guided practice for English Braille only, its ad-hoc design of the intelligent tutoring system, and its limited mechanisms for generating enthusiasm for learning to write Braille. We address these three limitations in our work and describe the details of our enhancements of the Adaptive Braille Writing Tutor in the following sections of this document.

4 GUIDED PRACTICE FOR ARABIC BRAILLE LETTERS

Our interactions with the Al Noor Institute in Qatar revealed the need for expanding the tutor capabilities to include guided practice and instructions on writing Arabic Braille. Adding the functionality of tutoring Arabic Braille letters to the Adaptive Braille Writing Tutor makes the tutor relevant to visually impaired students learning to write Arabic Braille, and is especially relevant in Qatar. Further interactions with Al Noor taught us that it is also useful to save the cumulative output of the student's writing in a format accessible by their screen reader software. This allows each student to keep track of his/her progress during a tutoring session. Therefore, our first task for improving the ABT was to enable the tutor to provide guided practice for the Arabic alphabet characters in Braille and facilitate the interface between the ABT and the screen reader software used at the Al Noor Institute. This initial task also provided us with the opportunity to familiarize ourselves with the ABT software, hardware, and operations.

The underlying structure of the tutor software developed by Tom Stepleton was designed to allow for easy extension for character maps other than English Braille where the number of characters was less than 64 (2^6). Having understood the basic underlying structure of the tutor software, we were able to extend the tutor capabilities to include tutoring Arabic Braille. Close coordination with several teachers at the Al-Noor Institute was of great value during this implementation process, especially in helping us to learn the mapping of the Arabic alphabet to their corresponding Braille characters.

A flow control diagram illustrating the operation of the tutor when recognizing letters and providing audio feedback to the student is shown in Figure 9. We implemented the mapping of dots to Arabic letters by creating a file consisting of a series of pairings between Braille dot patterns and Arabic characters. This mapping is recorded in a file with each pairing occupying its own line (shown in the Appendix). We use UTF-8 Unicode encoding⁵ to capture this mapping. This character mapping is then uploaded after initializing the Braille Tutor (Figure 9(a)) and the IOEventParser is informed to wait for user input. The IOEventParser component of the Braille Tutor interface library can then parse and interpret Braille cell dot patterns into letters based on the specific mapping provided in the uploaded file. In other words, once the student enters the letter using the stylus or buttons on the tutor, the IOEventParser uses the UTF-8 file to search for a match between the combination of "dots" entered in the Braille cell and an Arabic letter. If a match is found, the tutor prints the appropriate letter to a file and plays the appropriate sound file as shown in Figure 9(b). The tutor recognizes that a character is complete by satisfying either of the three conditions which are: the tutor waits for 5 seconds with no events, the student presses either of the control buttons, or if using the Braille cells, moving to a different cell. Once the user is done with the tutor, he/she can quit the program as shown in Figure 9(c) by either disconnecting the tutor, using the computer keyboard key combination of Ctrl+x+c or by filling in only the 6th Braille dot (which maps to the character 'ع').

For the audio component of the tutor, we recorded individual sound files for each Arabic alphabet character as it was spoken, and saved these sound files with appropriate labeling. We also integrated an audio player with the source library of the Braille tutor code to play the recorded sound files. When a student uses the tutor, the software extracts the character that maps

⁵ UTF-8 is a widely used variable-length character encoding for Unicode supported by most modern text editors.

to the combination of dots and plays the corresponding sound file based on matching the character to the name of the sound file.

The tutor also prints this character to a text file that is saved on the computer. We enabled the student to keep track of his/her progress during a tutoring session by storing the output of the written Braille characters into a text file with an intuitive naming convention. These files were then readable by the “IBSAR” software [9] used at Al Noor. The chosen naming convention for the text file was BrailleTutor. There is an initial verbal message at the beginning to tell the student what the name of the output text file is. When the student exits, there is a reminder that asks the student to rename the output text file if they want to access it later, otherwise it will overwrite the previous created file.

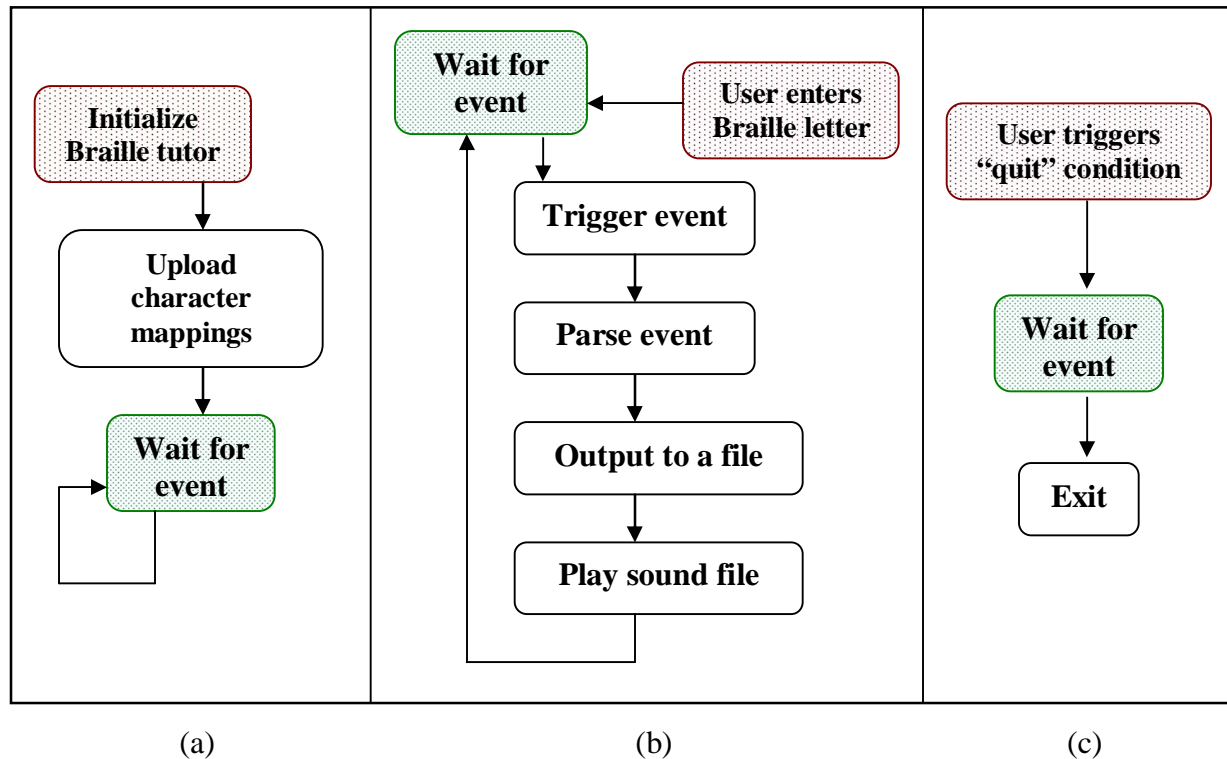


Figure 9: Control flow diagram for the Arabic Braille letter tutoring implementation (a) Braille tutor initialization results in waiting for users to enter letters (b) User enters a letter that interrupts the waiting process, interprets the entered letter, and outputs the result via audio and to a text file (c) User triggers the Braille tutor to quit

The pseudo code of this implementation is as follows:

```

//Before initializing the Braille tutor
Charset mychar(""); // create an object of type charset that takes any string

// load the information of the character mappings into the created object
Mychar.read(path of the file that has all the character mappings UTF-8);

//Create a Voice object
Voice myvoice("path to the sound files");
  
```

```
Wait for an event
If an event is triggered,
    // compare the valid letters outputted into the file with themselves. If this is true, then say
    // the sound file that is named according to the outputted letter
    If (entered Braille letter == "letter")
        Myvoice.say("the entered letter .wav);

    //The tutor will quit if user enters the letter ء
    If (the entered Braille letter == "ء")
        Exit;
```

The resulting implementation was successfully able to monitor a user's Arabic Braille alphabet writing, and provide immediate audio feedback. We tested this implementation with the help of Mr. Yasser Al-Shafai, a visually impaired teacher at Al-Noor.

5 INTELLIGENT TUTORING SYSTEM

The second improvement we undertook addresses the ad-hoc design of the ABT software components. To overcome this weakness we reviewed the literature on research methodologies in Assistive Technology, Intelligent Tutoring Systems (ITS), and Artificial Intelligence to propose a principled re-design of the ABT software. Intelligent Tutoring Systems (ITS) are interactive learning environments based on instruction and guided practice assisted by computers. They offer the ability to present educational materials through a communication interface to the computer and respond intelligently and instructively to the student's performance. The intelligent response of the tutor is adapted to the particular student through a process described in three phases: (i) getting the information from the student, (ii) processing this information to initialize and update a model of the student, and (iii) using the student model to provide the customized feedback [10]. Accomplishing these three phases requires collaborative work between the five core components of an ITS: The Domain Knowledge, the Pedagogical Module, the Expert Model, the Communication Model, and the Student Model. Figure 10 illustrates the interactions between these components of an ITS.

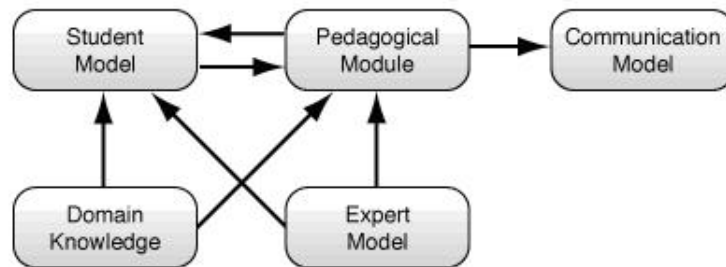


Figure 10 [11]: The principle components of an ITS and the interaction between them

Each of these components has a specific role in an ITS. The Domain Knowledge contains the information being taught by the ITS. The Pedagogical Module captures the teaching process and makes decisions about how to guide the student. The Expert Model captures the skills of an expert of the knowledge stored in the Domain Knowledge. The Communication Model dictates the interaction with the learner and addresses the question of how to present the material to the student in the most effective manner. Finally, the Student Model gathers and stores information specific to the current student, tracks the progress of the student based on the model, and provides useful data to the Pedagogical Module. Each of these five components needs to be customized to fit the needs of the particular ITS being implemented. Based on our survey of the relevant literature, and based on our observations and interactions with the Al Noor Institute, we were able to map the five components of an ITS to the corresponding customizations for the Adaptive Braille Writing Tutor.

In the next few sections we provide detailed recommendations for how each of the five components should be implemented for the ABT. We also discuss the how the components exchange information and work together to provide adaptive feedback to a young student learning to write Braille.

5.1 DOMAIN KNOWLEDGE

The first ITS component we discuss is designed to capture the domain knowledge in the tutor's area of expertise. For the purpose of the Braille writing tutor, this component will contain information about Braille character mappings from the alphabet of a given language to the correct combination of dots in the Braille cell. It also includes basic knowledge on the numbering of the six dots.

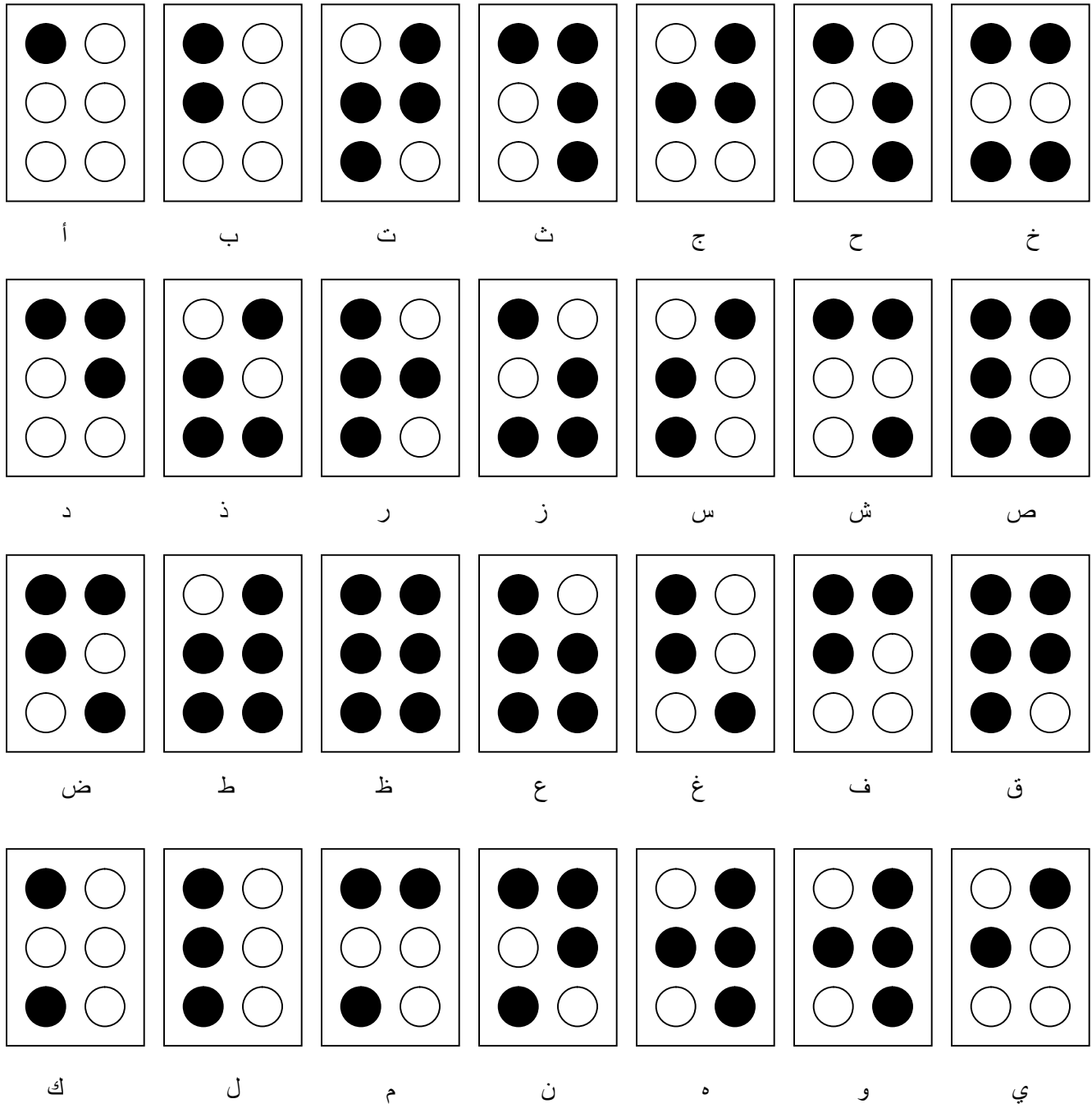


Figure 11: Arabic alphabet written in Braille

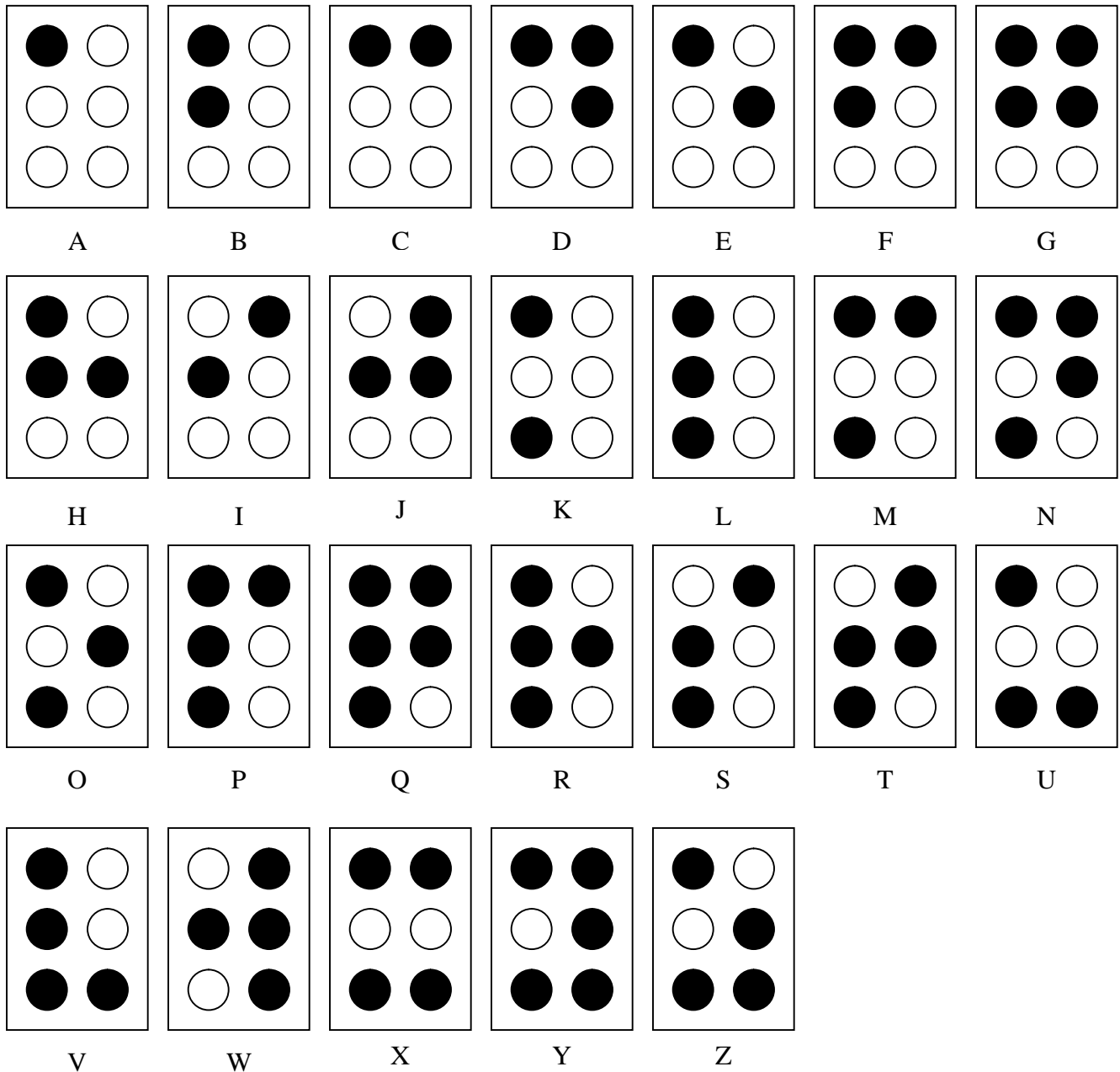


Figure 12: English alphabet written in Braille

Note that there are some differences in the British and American Braille systems, especially when representing math symbols, but for the basic alphabets they both share the same mappings. In this work, we only include alphabets since we are targeting young kids just starting to learn the how to write Braille. More advanced concepts such as words, spelling, punctuation, grammar and contractions can be added at a later stage for more advanced students. The complexity of dealing with Braille contractions will require significant extensions to the ABT software.

5.2 PEDAGOGICAL MODULE

The Pedagogical Module in the Intelligent Tutoring Systems is derived from the teacher's role in the classroom. The purpose of this ITS component is to capture the teaching process and make decisions about how to guide students. This guidance is provided in the form of one-on-one interaction to maintain the individualized instructions that the Intelligent Tutoring Systems provides. Intelligent Tutoring Systems use two forms of the individualized instructions, meta-strategies and instructional strategies [13]. Meta-strategies refer to the overall picture of the teaching process while instructional strategies refer to the methods used to teach a particular concept. In a classroom setting at Al-Noor Institute, the teachers tend to adopt the following five strategies when teaching [14]:

Students' Trust

Based on their qualifications, the teachers give reasoning and explanations for the information delivered to students in the classroom, and this gives the students the confidence in their teacher's accuracy and wisdom. The teaching process progresses smoothly once the teacher earns the students' trust. Therefore, this stage is critical for successful teaching or tutoring.

Curriculum

Typically, teachers deliver information to students based on a particular curriculum, and in this case, the curriculum for Grade One is teaching both the English and the Arabic Braille alphabets and then moving to instructions on writing simple words.

Classroom Activities

Teachers engage with students in the classroom through a mixture of lectures, discussion, hints, exercises, and other forms of activities.

Continuous Assessment

Teachers often stop after a number of classroom activities to evaluate their educational outcomes of and the effectiveness of using the activities in the teaching process. For example, short quizzes can give some indication of whether or not the students understood the material in that module. This continuous assessment process enhances the educational outcomes by continuously enriching the teacher's knowledge about the students' strengths and weaknesses, and thus allowing the teacher to adapt instructional techniques to respond to the students' needs.

Final Assessment

Final assessment is mostly important in measuring the overall level of understanding of the entire curriculum for a given class or grade level. This measurement is often a significant factor in the decision of whether or not to move the student to a more advanced level.

The following illustration shows the five aforementioned strategies' categorized as meta-strategies and instructional strategies in the Intelligent Tutoring System.

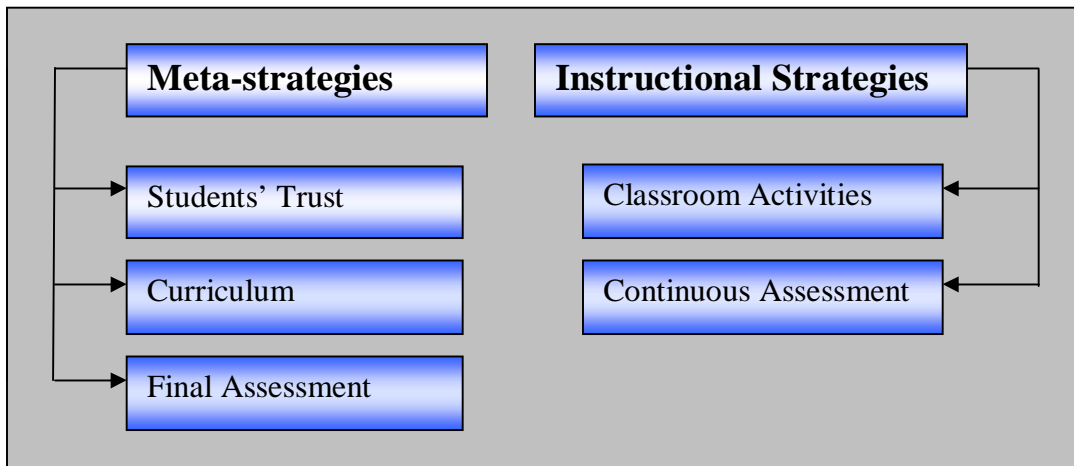


Figure 13: Categorization of the five strategies of teaching employed by the Al Noor Institute

The meta-strategies include strategies that impact an entire grade level and include earning the students' trust, designing and covering all of the material in the curriculum, and assessing the each student's ability to continue to the next grade level through a final assessment process. The instructional strategies include the two strategies used for teaching components of the overall curriculum; these are classroom activities and continuous assessment. If we map these strategies back to the pedagogical module, the meta strategies govern the overall strategies used to teach the domain knowledge, and the instructional strategies are used to update instruction techniques (exercises and interventions) based on the feedback fro the student model. Thus the ITS is able to cover the necessary information while providing the most appropriate interventions for a particular student. Next, we propose a detailed implementation strategy for a Pedagogical Module relevant to the Braille writing tutor.

Meta-strategies:

Each one of the three meta-strategies (earning student trust, designing a curriculum to cover all the relevant information, and assessing the skill-level of the student when deciding on promotions) are next explained in detail.

Earning the Students' Trust:

Student trust in the tutor depends on two primary factors: robustness of the tutoring mechanism, and effectiveness of the tutor's ability to teach. For the first aspect, we can employ a variety of strategies such as recording the teacher's (or a trusted person's) voice for the audio feedback or using any other "happy sounds" to eliminate the student's fears of using a new device. We can improve the trust in the tutor's ability to teach by implementing a Braille Tutor System that employs sound pedagogical techniques, reliably produces correct answers, adaptively provides useful feedback and interventions, and by getting a respected person (such as a teacher) to endorse the tutor and introduce it to the student.

Information Delivered:

Writing Braille in Arabic is based on a one-to-one mapping of characters to Braille patterns. Thus, the tutor should deliver three types of information:

- Different character mappings in Braille for both English and Arabic alphabets. (The set of both alphabets in Braille is shown in Figure 11 and Figure 12).
- The structure and the numbering of the six dots in the Braille cell (shown in Figure 1(a)).
- Instructions on how to use the Braille tutor. (These instructions should be either provided by a teacher when introducing the student to the tutor, or be a part of the tutor's initialization process or "welcome" to the student).

Final Assessment:

Final assessment measures the overall understanding of two main aspects of writing Braille:

- Knowledge of the structure and the numbering of the six dots in the Braille cell.
 - An example of an exam that could be used to evaluate the student's knowledge of the six dot positions could be a sequence of exercises that ask the student to press on a sequence of dots using the six buttons on the middle of the E-slate (shown in Figure 8). For example, press on dot 1, 2, 4, 6, 3, 5, 3, 2, 6.
- Knowledge and speed of writing the different mappings in the Braille alphabets.
 - An example of an exam to test the student's knowledge and speed of writing Braille alphabets could be an assignment that asks the student to write a sequence of characters (English and/or Arabic) and measure the speed and accuracy of the student's solution.

Finally, we need to determine criteria for deciding whether the student can advance to the next level, should stay in the same level, or should be demoted to a lower level. For example, if the student passed an exam on the structure and numbering of the Braille dots with a score of 85% or above he/she can proceed to the higher level of learning characters. If the score is between 85% and 50% the student should remain at the same level. If the score is lower than 50% the student could be diagnosed as needing more fundamental help.

Instructional Strategies:

We now examine the two instructional strategies (classroom activities and continuous assessment) in detail.

Classroom Activities:

Classroom activities are the activities employed by the teacher to instruct the students on how to write Braille. In Arabic, Braille writing is taught in a number of stages [14]:

- Improving the touch sense
 - This involves exercising the touch sense to recognize very detailed objects. This is important and crucial in writing and reading Braille because Braille letters could differ from one to another by one dot. Hence, the touch sense is a primary use in writing Braille. This stage usually takes place during kindergarten through the first few weeks of grade one.
- Knowing numbers
 - Teaching numbers is a crucial stage that comes before starting to teach letters because each letter in Braille is defined by a number of dots and each dot in the

cell of 6 dots has a specific number assigned to it. Therefore, students should be able to understand numbers and associate the numbers they learn to the number of dots required for each letter.

- Learning the easy letters
 - The easy letters in Braille are defined to be the letters that have one, two, or three dots embossed in the left column; that is letters that only use the left column. For example, أ, ب, ج. (Arabic alphabets in Braille shown in Figure 11)
- Learning the less complex letters
 - These letters are defined to have one dot in the left column and another in the right column, e.g. ة.
- Learning the tri-letters
 - These are the letters with three embossed dots distributed in the two columns (for example س).
- The next stage is the letters with four embossed dots.
- The last stage is the fully raised cell except one dot.

Guided instructions that lead the students through the above stages, enables students to write the basic Arabic alphabet in Braille. Similar stages can be designed for teaching the basic English alphabet in Braille. These stages can be translated to instruction modules and exercises that lead the student through the learning process for the six dots and the letters using the ABT.

Continuous Assessment:

The Braille tutor should provide continuous assessment by giving the students short quizzes that cover different topics as they learn them. These short quizzes can take many forms [14]. For example, the tutor could ask the student to emboss particular dots or to write specific letters. Alternately, the tutor could give the student a word constructed from many letters, and then ask the student to recognize the letters that are familiar. As the students do these exercises, the Braille tutor needs to evaluate the student performance and detect if the student is demonstrating particular mistakes (mirroring of characters, misunderstanding of dot numbering, etc.) and provide appropriate remedial actions such as hints, new exercises, additional instructions, or encouragement in the form of verbal commentary.

5.3 EXPERT MODEL

The role of the Expert Model component of the ITS is to represent the knowledge and skills of an expert in the domain. It provides expert solutions to all the exercises and quizzes assigned to the student thus providing a benchmark for comparison and evaluation of the student's solutions.

The expert model is commonly implemented using one or both of two approaches [11]:

1. The first approach captures subject matter and expertise in rules. These rules are combined and applied to solve different problems.
2. The second approach provides information based on different scenario definitions. In this approach the expert specifies how the student should respond to a specific scenario.

The first approaches would meet our target of implementing the Expert Model. Specifically, writing Braille is constructed by a set of rules. These rules map out each character to a unique subset combination of the six dots. Therefore, the expert model will capture expertise in writing

Arabic alphabet, and the tutor will use the expert model solution to different exercises for comparison with the student's solutions to pinpoint places where the student has difficulties.

5.4 COMMUNICATION MODEL

The communication model determines the interaction between the student and the tutor and addresses the question of how to present material to the student and receive feedback from the student in the most effective manner. A key difference between most Intelligent Tutoring Systems and the Adaptive Braille Writing Tutor is the use of graphical interfaces and visual cues. Visual feedback is inaccessible to blind students. (Note however that visual output to the screen can be useful to sighted teachers and to sighted debuggers). Hence, the ABT must rely primarily on audio feedback to the student. The major drawback of audio feedback is that it prevents the use of the tutor by the deaf-blind. Another viable interaction method for the ABT is via a tactile interface. However, this capability is not supported by the tutor hardware and is often much more expensive because it requires moving mechanical components that must operate robustly.

Deciding what type of audio is best will depend upon the student's age, culture, and level of progress. One of the options we have for very young children is recording their teacher's voice to provide feedback to the student. A second option that older children might find more exciting or "cool" is using a synthetic voice. It is also important to make sure that the audio feedback is relevant to the student's culture; that is, issues such as language, dialect, and accent need to be taken into consideration. Finally, it is possible to include "fun sounds" or different audio cues that might engage the student (such as animal noises or songs) to encourage the student when he/she provides a correct answer or makes progress.

Currently, the Communication Model is implemented through recording audio wave files for each letter in the alphabet. This is tractable because the alphabets for both English and Arabic are relatively small with 26 letters in the English alphabet and 28 letters in the Arabic alphabet. While audio is the main mode of feedback from the tutor to the student, the communication model must also include an interface for the student to talk to the tutor. For the ABT the student communicates with the tutor via a tactile interface; through the buttons and stylus.

5.5 STUDENT MODEL

The student model is crucial to the ITS because it allows the tutor to provide adaptive customized feedback to each student. This ITS component gathers and stores information specific to each student, identifies the current level of knowledge of the student, tracks the progress of each student based on the model, and provides useful data to the Pedagogical Module. Some Student Models are built to recognize student plans [15], and some are built to evaluate student performance or problem solving skills [14] depending on the scenario the student is in and what aspect of the student we are trying to model. The design of a Student Model depends on the answer to one question: What aspects of the student should we model in a specific Intelligent Tutoring System? [10]. The answer to this question for the Adaptive Braille Writing Tutor is that we want to:

- Identify the student's level of competence in writing Braille and initialize the start state of the Student Model accordingly because this will determine the first level of exercises and interventions provided to the student)
- Monitor the evolving capability of the student to write Arabic Braille letters and trigger appropriate transitions to new states in the Student Model. This is important to evaluate progress (or regression) in the student's performance.
- Generate sufficient data to accurately determine the skill-level of the student so that the Pedagogical Module can determine the necessary interventions and instructions.

There are several algorithms that are commonly used to implement the Student Model. In our literature review we encountered many of the common approaches such as Bayes Nets, Overlay Models, Case-Based Reasoning, and the Stereotype Model. Bayesian Networks can be used for long-term knowledge assessment, plan recognition, and prediction of student's actions during problem solving [15]. The Overlay Model is defined around the assumption that the student's knowledge is a subset of the expert's knowledge [11]. Case-Based Reasoning is based on the notion that student's problem-solving capabilities can be evaluated by looking at how other students (in previous cases) solved problems which are similar to the current situation [10]. Finally, stereotype-based reasoning builds inferences based on an initial impression of the student and default assumptions of the stereotype assigned to the student state [16]. Because of the small number of levels we are dealing with in the ABT applied to teaching alphabet characters, and because of the small amount of feedback available to the tutor from the student, we chose the stereotype approach for the Student Model. The stereotype approach allows the system to make many inferences based on a small set of observations by mapping the observations to stereotypes, and making many inferences from this classification.

Before proceeding with the implementation strategy for the proposed stereotyped Student Model for the Braille writing tutor, we present some an introduction to the stereotype approach. Suppose we have a stereotype M which is part of the student model in a system. M consists of a set of components $\{c_j\}$, where each component represents some aspect of the user. A set of trigger conditions $\{tM_i\}$ and retractions, composed of individual components or some function that combines several components, activate and de-activate M . When M is activated, many inferences can be derived where each inference is governed by a threshold probability.

In summary:

A stereotype M is activated when any of the trigger conditions, tM_i , become true:

$$\text{If } \exists i, tM_i = \text{true} \rightarrow \text{active}(M)$$

M is deactivated when any of the retraction conditions, rM_j , becomes true:

$$\text{If } \exists j, rM_j = \text{true} \rightarrow \text{not active}(M)$$

In addition, we have to keep in mind that some of the triggers may be "essential" to a stereotype being active:

$$\exists e, (tM_e \in \{tM_e\}) \text{ and } (\text{not } tM_e \in \{rM_e\})$$

While a stereotype M is activated, we can derive many inferences $\{sM_k\}$:

$$\text{Active}(M) \rightarrow \{sM_k\}$$

And finally, a probability, P_M , thresholds the inclusion of inferences in the stereotype.:

$$\forall i, sM_i \in \{sM_i\}, p(sM_i) > P_M$$

For example, if M is a stereotype about a student who lacks the fundamental knowledge about the structure and numbering of the six dots on the Braille cell, and the probability threshold for M is 90%, then the inclusion of inferences (such as “the student is a beginner” and “the student doesn’t know the letter mappings yet” and “the student doesn’t know the location of dot5”) are all governed by the probability of 90%. In other words, on average, we expect 90% of the people who don’t know the fundamentals of the Braille dots to be governed by the inferences (or default assumptions) we make. These threshold probabilities are derived using expert knowledge, observations, or surveys. The power of the stereotyped model is that we can make many inferences based on a few observations, and use these inferences to guide the Pedagogical Module in its choice of interventions.

In our design for the ABT we are using a Hand-Crafted Stereotype model based largely on the expert knowledge of the teachers at Al Noor. Hand-Crafted Stereotype models make assumptions about the stereotype groups as well as the triggers based on the designer’s intuitions or access to expert knowledge. We propose a combination of observing the students interaction with the tutor and diagnostic tasks to trigger the stereotype groups.

Next, we describe an implementation strategy for the proposed stereotyped Student Model framework for the Braille writing tutor. We recommend four stereotypes defined as follows:

M₁: The student is considered an expert in writing the alphabet in Braille

M₂: The student knows the mapping of the alphabet to Braille but makes careless mistakes

M₃: The student does not know the mapping of the alphabet to Braille

M₄: The student is a true beginner and lacks the fundamental knowledge of structure and numbering of the six dots in the Braille cell

Each stereotype will have its own inferences, components, triggers, retractions and threshold probabilities that can be crafted based on the expert teachers at Al Noor Institute and student’s interaction with the tutor.

We propose two important components that will be of value to these stereotypes. The first is a distance metric value, which measures the “distance” between the student’s answer and the expert solution. For example, let’s assume the assignment is to write the letter “A” which corresponds to the first dot (in the upper left corner of the Braille cell). If the student entered the mirror image dot (i.e dot 4), then the corresponding distance between the two answers is low. On the other hand, if the student selects dot 6, then the distance between the solutions is high. Additional penalties could apply for selecting the wrong number of dots or selecting the wrong row. For example, if a student presses 4 buttons for a letter that only requires 1 dot, it is a clear indication that the student does not know that letter.

The second category of components we propose is based more directly on observed performance of the student. These observations will be chosen based on the knowledge of the level of difficulty of particular skills or the level of commonality of specific errors. For example, if a student does well on a set of tri-letters (letters with 3 dots distributed over both columns) we can assume they understand the fundamentals. In contrast, one of the hardest concepts to get across to students and retain in their memory is the structure and numbering of the Braille dots. Perhaps a common mistake is to pick the mirror image of the numbering. This would be an important component to monitor.

Components for each stereotype can be exclusive or overlap with other stereotypes. These components, individually or in combination, can form triggers that activate stereotypes or form retractions that deactivate stereotypes. Then, we start to build inferences based on how well the student is progressing and therefore which stereotype is activated. The proposed stereotyped model is illustrated in Figure 14.

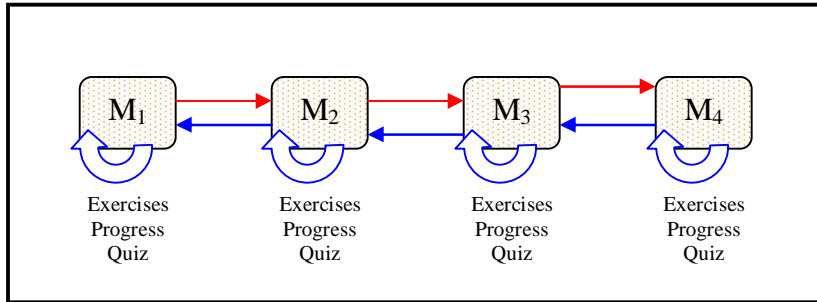


Figure 14: Control flow diagram of the working definition of the student stereotype model

The default assumption of our stereotype model for the Adaptive Braille Writing Tutor is that the student does not know the six dots well. This assumption is based on information from the teachers at Al Noor.

6 EDUCATIONAL COMPUTER GAMES

The final component of our reported work addresses the need for increasing the enthusiasm of children when learning to write Braille. Educational computer games have been gaining rapid acclaim as motivational educational tools for children and youth [17]. However, to the best of our knowledge, most of the current educational computer games don't cater to the needs of visually impaired children. Hence, our third task in this project was to develop an educational computer game that motivates visually impaired children to learn to write Braille.

The Thorne-EMI puzzles are good examples in the computer educational field [18]. Also, HANGMAN⁶, HAMMURABI⁷ and LUNAR LANDER⁸ are some early classic entries of computer games [19]. Educational computer games can assist people, expand concepts, and teach learners certain subjects as they play. They simulate many elements of traditional media such as characters, music, plot and sound. Computer games are seen as interactive narratives, remediated cinema and procedurals stories inspired from the fields of theater, drama and film studies [19]. They are a host of player challenges on decision-making and problem solving strategies [20]. Formally, one can think of games as a systematic study of the relationship between rules, choice and outcomes in competitive situations [21].

Despite the large number of existing educational computer games, we could not discover any such games that are catered to motivating the visually impaired to learn Braille. However, there are some games that have been developed for and by visually impaired people⁹. Some design constraints for educational games for the visually impaired are very different from the design constraints for sighted players. Most notably, the constraint of visual components being disallowed in the game is paramount for blind players. Since the vast majority of popular computer games are highly dependent on computer graphics and animation, and since most computer games rely on some form of visual interface, it is a considerable challenge to build an educational computer game for visually impaired children.

In this section we present a detailed description of an educational computer game designed to motivate visually impaired students to enjoy learning to write Braille. The game is specifically targeted towards motivating students to learn the Arabic Braille alphabet. The following section discusses the environment in which the game will be played by giving concrete conditions and events in which the abstracted goal of learning the Arabic Braille alphabet is embedded in an educational computer-tutor game.

6.1 GAME INSPIRATION

This section describes the motivating concept for the computer-tutor Braille alphabet game we designed. The main inspiration for our game design comes from the game of Dominos [22] where the domino tile (shown in Figure 15) has a rectangular shape with a line dividing its face into two square ends; each is marked with a number of black spots or blank. The spots are generally arranged as they are on six-sided dice. In spite of that, we are structuring our game on

⁶ See <http://www.hangman.learningtogether.net/> for the description of hangman game

⁷ See <http://www.atariarchives.org/basicgames/showpage.php?page=78> for the description of Hammurabi game

⁸ See [http://en.wikipedia.org/wiki/Lunar_Lander_\(computer_game\)](http://en.wikipedia.org/wiki/Lunar_Lander_(computer_game)) for more description of Lunar Lander game

⁹ See <http://www.tsbvi.edu/technology/games.htm> for some examples of accessible computer games

the light of the Domino game's rules and concept. We were inspired by the similarity between the Braille letters and the tiles of the Domino game, and decided to exploit this similarity in our game design. Braille letters are formed using six dots placed in a cell of two columns and three rows; the positions of the six dots are universally numbered from one to six and a subset of these six dots is embossed to represent each letter. The Arabic Braille alphabet consists of 28 letters; each represented by a unique subset of the six dots embossed. The construction of domino tiles is very similar to Braille letters.

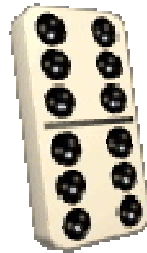


Figure 15 [23]: domino tile

The following section of the paper explains a concrete design of our game, Braille Cell Winner (BCW). Based on the frame work proposed in “*The Art of Computer Game Design*” written by Chris Crawford [18], we outline three structures critical to our game design: the I/O structure, the game structure, and the program structure.

6.2 I/O STRUCTURE

The I/O structure is the system that communicates information between the computer (and in our case, the tutor) and the player. I/O structure is composed of outputs and inputs. The computer has two forms of outputs to the players: graphics on the screen for debugging and demonstrating to sighted people, and sound [18]. So, our target is more oriented towards sighted or visually impaired people. Making the game comprehensible with only sound output was a one of the most challenging aspects of the game design. On the other hand, the player's input is accomplished through the input device of the Braille writing tutor

6.3 GAME STRUCTURE AND RULES

The game structure is the internal architecture of casual relationships that defines the obstacles the player must overcome in the course of the game [18]. A key component in this game is entering Braille letters that can match a single letter from both ends. That is, if a player is assigned the “left side” he/she must pick a letter where its corresponding dots in the 2nd column match the dots in the first column of the letter chosen by the computer. This process is illustrated below in Figure 16.

Rules

- Number of players: two players using one Braille tutor (taking turns)
- The deck: the 28 letters of the Arabic Braille alphabet
- Game environment: the computer randomly picks a letter from the deck and announces the letter verbally. The computer also announces whose turns it is (that is player 1 or

player 2) and which side (left or right) should be attempted for matching the announced letter. The assigned player then tries to find the “heaviest” letter (the letter with the most embossed dots) to match the assigned side of the announced letter (as shown in

- Figure 16). In the next round, the second player is chosen to do the same activity to match the next announced letter on a side (left or right) randomly selected by the computer. Assigning sides is done randomly each round and players are alternated sequentially. The challenge is competing against each other to match the heaviest letter within the constraints and within the allotted time.

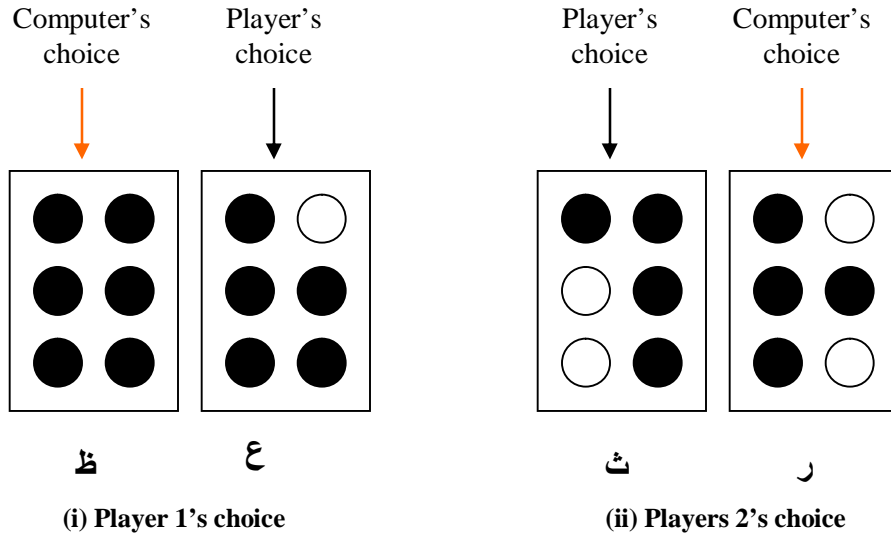


Figure 16: Two consecutive rounds (i) Player’s 1 choice: the player was able to get 5 points by match “ع” from the right side of the computer’s choice (ii) Player’s 2 choice: the player was able to get 4 points by match “ث” from the left side of the computer’s choice. Hence, the first player is declared to be the winner.

Example Scenario

Assume in the first round, the computer announces character “ظ” and assigned player one the right side randomly. Player one has to think of the heaviest character that can match the right column of character “ظ”

Figure 16(i) Players 1’s choice shows that player 1 chose ع to match the computer’s choice. In the second round, the computer announces character ر and assign the left side to the player. The player came up with ث which can match the computer’s choice from the left. As seen, player one was able to get five points whereas player two was able to get only four points. Thus, player one is declared to be the winner.

Scoring

- Each round a player is assigned a side by the computer. There is a fixed time assigned as well for entering the matched character. If the player didn’t enter anything within the time limit, or he entered an invalid character then he will get zero.

- If he was able to enter a character within the time limit, and the character was valid, then the program counts the number of dots embossed for the entered character and announces that number
- There are 28 rounds, after each round the program announces the sum of the total dots gained by each player from the rounds played. The winner is declared at the end based on the higher number of dots
- If both players had the same number of dots at the end, then there is a tie

6.4 PROGRAM STRUCTURE

- Use control button to activate game
- Keep track of whose turn it is (initialize to player A), which side is next (initialize to left), what the score is for each player (initialize to 0 for each player), and what set of numbers from 0 to 27 have been used (initialize to empty set)
- End conditions: all numbers have been used or “end of game” is triggered by a player using one of the control buttons
- While end conditions aren’t true
 - Computer picks a random number between 0 and 27 discarding any numbers already used
 - Update used numbers with newly picked number
 - Choose letter from alphabet corresponding to number (i.e. first letter is number 0 and last letter is number 27)
 - Announce letter, player, and side
 - Wait a fixed time for player to response
 - End wait if player responds or time elapses
 - Announce end condition (i.e. time elapsed or player entered dots 1,5,and 6) via audio feedback
 - Check response
 - If the player didn’t respond in time – do nothing
 - If the player responded in time
 - Check if the response was a valid letter
 - If the response was not valid announce it
 - If the response was valid, announce it and compute the score
 - Update the score for that player
 - Switch player
 - With a random draw switch sides
 - Announce the score for each player
- If end condition is true
 - Announce final scores for both players
 - Announce winner or tie
 - exit

7 CONCLUSIONS AND FUTURE WORK

Braille literacy is required for blind people to play a meaningful role in modern society. Our research on teaching Arabic Braille, and our interactions with the Al-Noor Institute for the Blind in Qatar have helped us to identify some of the challenges faced by visually-impaired people during the learning process for writing Braille. These challenges motivated us to improve the Adaptive Braille writing Tutor (ABT) that provides guided practice using audio feedback for young children to learn to write Braille. The tutor was developed by the TechBridgeWorld program at Carnegie Mellon University (www.techbridgeworld.org). We enhanced on the (ABT) to cover three main dimensions: Relevance to the Arab World, methodology of software design, and motivational factor for the student. For the first dimension, we enabled the (ABT) to provide guided practice for the Arabic alphabet characters in Braille and facilitate the interface between the Braille tutor and the screen reading software used at the Al Noor Institute. Next, we improved the ad-hoc design of the ABT software components by combining research methodologies in Assistive Technology, Intelligent Tutoring Systems, and Artificial Intelligence to propose a principled re-design of the ABT software. Finally, we studied the literature on Educational Game Design and created an educational ABT-computer game to increase the motivation of children learning to write Braille. The outcome of this project is an improved Adaptive Braille Writing tutor that enhances the state of art in educational technology for the visually impaired.

The experience of working on this honor Senior Thesis has been tremendously rewarding in many ways. We hope to motivate other students to share in this experience. An important aspect of future work will be to implement the proposed design of the Intelligent Tutoring Systems components for the ABT. The tutor should also be extended to provide instruction on numbers, mathematical operators, words, punctuation, sentences, etc. in Braille. Another important enhancement to increase the impact of the tutor will be adding the capability to teach new languages in Braille. Several other extensions such as new games, networked games, additional voices and sound effects, a set of curricula and exercises, and user studies will all greatly benefit the tutor's impact. Finally, systematic longitudinal studies in different parts of the world are also important to evaluate and document the long-term impact of the tutor on global Braille literacy.

ACKNOWLEDGEMENTS

We are grateful to the Qatar National Research Fund (QNRF) Undergraduate Research Experience Program (UREP) for sponsoring this project. The author also wishes to thank the TechBridgeWorld team for their help in providing the Adaptive Braille Tutor, and especially Tom Stepleton for his assistance with navigating the ABT software. We are indebted to the administrators, teachers, staff, and students at the Al-Noor Institute, and especially to Mr. Yasser Al-Shafai for the significant time and effort he devoted to providing us feedback. Finally, we could not have completed this project without the support and encouragement of the Carnegie Mellon University Qatar (CMUQ) faculty, staff and students.

REFERENCES

- [1] Nidhi Kalra, Tom Lauwers, and M. Bernardine Dias, “A Braille Writing Tutor to Combat Illiteracy in Developing Communities,” accepted paper, Artificial Intelligence in Information Communication Technology for Development workshop at IJCAI 2007.
- [2] A. Nemeth, “Braille: The agony and the ecstasy,” Braille Monitor, pp. 324–328, July 1998.
- [3] World Health Organization, “Fact sheet 282: Magnitude and causes of visual impairment,” World Health Organization, November 2004.
- [4] Conventional Support Tools for the Blind and Visually-impaired Students, University Marburg of Marburg:
http://www.uni-marburg.de/studium-en/specialneeds/visuallyhandicapped/brailleur/image_preview
- [5] Perkins, “Perkins brailleur,” Catalog of Products; Howe Press of the Perkins School for the Blind, 2006.
- [6] Image of slate and stylus, March 8, 2008:
http://www.rehabmart.com/resizeimage_send.asp?path=/imagesfrommma/202600L.jpg&width=150&height=150
- [7] <http://www.brailleslates.org/>
- [8] <http://braille-tutor.livejournal.com/4395.html>
- [9] <http://www.sakhr.com/products/Ibsar/Default.aspx?sec=Product&item=Ibsar>
- [10] <http://www.braillepaper.com/braillepaper.php>
- [11] Application of AI in Education. Joseph Beck, Mia Stern and Erik Haugsiaa 7 September, 2007
<http://www.acm.org/crossroads/xrds3-1/aied.html>
- [12] Nidhi Kalra, Tom Lauwers, D. Dewey, Tom Stepleton, and M. B. Dias, Iterative Design of A Braille Writing Tutor to Combat Illiteracy, IEEE/ACM International Conference on Information and Communication Technologies and Development (ICTD), 2007.
- [13] Burns Tom & Sinfield Sandra. Teaching, Learning & Study Skills. SAGE Publications Inc, 2455 Teller Road, Thousand Oaks, California 2004.
- [14] Interview with Professor Yasser Mohammed Al-Shafai, Arabic Teacher at Al-Noor Institute
- [15] Conati Critina, Gertner Abigail, VanLehn Kurt, Druzdel J. Marek. “On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks”. University of Pittsburgh, PA, U.S.A.
- [16] Kay Judy. Stereotypes, Student Models and Scrutability. Basser Dept of Computer Science. University of Sydney
- [17] http://www.futurelab.org.uk/resources/publications_reports_articles/literature_reviews/Literature_Review130

- [18] <http://www.vancouver.wsu.edu/fac/peabody/game-book/Chapter3.html#Educational>
- [19] <http://www.gamestudies.org/0101/eskelinen/>
- [20] http://gamestudies.org/0701/articles/elnasr_niedenthal_knez_almeida_zupko
- [21] http://gamestudies.org/0601/articles/heide_smith
- [22] <http://www.domino-games.com/>
- [23] http://www.sfsl.org/animations/domino_tile_21_wobble_md_clr.gif

APPENDIX: Arabic Alphabet Mapping to Braille

This appendix shows the UTF-8 Unicode encoding of a series of pairings between Braille dot patterns and Arabic characters with each pairing occupying its own line. Note that the mapping of each letter is reversed in this file because the tutor accommodates learning mirror images of letters when using the slate and stylus, but the students at Al-Noor do not learn the mirror images of the Arabic Braille characters.

!UTF-8 BEGIN My Arabic character set

4	أ
45	ب
1256	ت
1234	ث
125	ج
234	ح
1436	خ
124	د
1356	ذ
2456	ر
2346	ز
156	س
134	ش
13456	ص
1345	ض
12356	ط
123456	ظ
23456	ع
345	ف
145	ق
12456	ك
46	ك
456	ل
146	م
1246	ن
245	ه
1235	و
15	ي
6	ء

!UTF-8 END

Design – Code Verification

When Design Deviates From Code



Student: Amer Hasan Obeidah

aobeidah@qatar.cmu.edu

Advisor: Lynn Robert Carter

LRCarter@cmu.edu

Carnegie Mellon University – Qatar Campus

School of Computer Science

2008

Submitted to School of Computer Science in fulfillment of the requirements for
Undergraduate Research Program

ACKNOWLEDGEMENT

I am indebted to Professor Lynn Robert Carter for his dedication and constant mentoring throughout the course of this research paper. He has been one of the prime motivators to the success of this research paper and he will always remain one for any future research projects I pursue. I would also like to thank all the people who have helped me in making this research paper a reality that I will never forget.

ABSTRACT

In the business of software today, developers have a problem keeping design and code in synchronization. This thesis covers my research into this topic and describes an implementation of a tool that speeds up the resynchronization process of design and code. The resynchronization process requires a comparison process between code and design, but such comparison is too complex with too many unsolved problems to be fully covered in just one thesis. As a result, this thesis focuses on UML Class and Sequence diagrams, since they are the most widely and commonly used among the different design notations. We also restrict the study to just the Java programming language. This thesis highlights the key aspects of the (“in-sync”) problem, proposes the obvious solutions, explains the issues that keep them from working, and then introduces solutions that do work addressing important pieces of the problem. Additionally, this thesis proposes an extensible framework to support people struggling with the “in-sync” problem and populates that framework with an initial set of analysis modules. The thesis then concludes with an analysis of the work and explains how others can extend it to cover more pieces of the problem.

TABEL OF CONTENTS

INTRODCUTION.....	1
1 THE IN-SYNC PROBLEM	4
2 MOTIVATION.....	4
3 PREVIOUS WORK.....	5
4 THE OBVIOUS SOLUTIONS.....	5
5 SOLVING THE IN-SYNC PROBLEM MANUALLY	6
6 OUR APPROACH.....	10
6.1 Differences Between Design and Code Notations.....	14
6.1.1 UML Relationships.....	14
6.1.2 UML Multiplicity.....	15
6.2 Methods Overriding and Dynamic Binding.....	15
6.3 Methods Overloading and Sequence Diagrams.....	16
6.4 Control Flow Structures.....	16
7 JDCV FRAMEWORK: JAVA DESIGN CODE VERIFICATION.....	17
8 CONCLUSIONS.....	23
9 FUTURE WORK.....	23
REFERENCES	25

Introduction:

In software engineering, the system development life cycle is a process of understanding how the use of technologies and information systems can support people in different fields to solve the problems they face in their work domain. In 1996, the Standish Group conducted a survey about the Information Systems that were being built to address problems in different business domains. Unfortunately, the statistical results of the survey show that 42 percent of those projects were canceled before completion. Also, the results show that the projects that were completed are delivered significantly late with fewer features than originally planned [1].

Many people would think that the main reason behind software failure is due to poor design. Therefore, software development companies spend large amounts of time and resources planning and designing software products before reaching the implementation phase of development. Design is very important since design documents can be created more quickly than the code and when they are effective, they allow professionals to communicate far more quickly and correctly than is possible with code.

In the design phase, software engineers decide how the system will work. Specifically, they describe how the system will operate and they address key aspects of the system in terms of its underlying architecture such as the hardware, software etc.

In order to produce this detailed description of the modeled system, many software organizations produce different design notations or modeling languages in order to facilitate the process of software development. Between 1989 and 1994, the number of the modeling languages increased from less than 10 to more than 50 [2]. In 1994, Grady Booch, Jim Rumbaugh, and Ivar Jacobson from the Object Management Group (OMG) merged their own notations to create a Unified Modeling Language called UML. From an analysis of books [3] and [4] about software design, we have concluded that UML appears to be the most popular notation to encode the software design.

The Unified Modeling Language (UML) supports twelve different types of diagrams that can be used to portray the designed system from structural, behavioral, and managerial points of view [5]. However, from our analysis of the above books, we have learned that Class and Sequence Diagrams are the most commonly used. More specifically, software developers start off their design by creating Class Diagrams. These Diagrams show the major components of the system and their interrelationships [6]. Also, they show some details about class members such as attributes (Fields) and operations (Methods) upon them. As a second step, the software developers explain how the system's objects (classes) will interact with each other. Also, they will be interested in knowing the flow and timing of such interactions. For these reasons, the software developers will build Sequence Diagrams since they can learn more about the interactions between objects in sequential order.

When the software developers are done with the design, they enter the next phase of the software development process, the implementation phase. In this phase, software developers concentrate more on code and they spend a lot of time testing the artifacts and fixing the encountered defects. Specifically, when developers work under pressure, they think that finishing the code is the most important task because they need to deliver the product. Consequently, there is a tendency to become stuck in a testing/fixing loop (debugging loop) where they find the defects in code, fix them, and start the process all over again as shown in the figure (1).

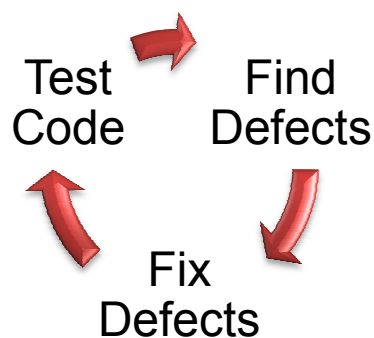


Figure (1): Testing/Fixing Loop: (Debugging Loop)

As a result of this loop, software developers tend to postpone updating the design documents. Therefore, the code artifacts start to deviate from the original design introducing a problem which we call the “in-sync” problem. Moreover, any changes in the requirements during the implementation phase puts the developers under great pressure which will eventually lead them to the debugging loop rather quickly, thus to the “in-sync” problem.

In the first section of this paper, we give a general definition of the “in-sync” problem. In the second section, we explain our motivation toward solving the “in-sync” problem. The third section gives a brief overview of previous and related work done in this area. Then, we talk about the obvious solutions to the “in-sync” problem and what prevent them from work in the fourth section. The fifth section gives an example on comparing Class and Sequence diagrams to their code representation using the manual approach. In the sixth section, we discuss our approach towards automating the manual approach by building an extensible framework called JDCV. The seventh section describes the implementation of JDCV, how it applies the automated approach, and how people can extend it to solve more parts of the “in-sync” problem. Finally, the paper describes possible future work in the eighth section and concludes the work in the ninth section.

1- THE IN-SYNC PROBLEM

In actuality, the “in-sync” problem cannot be defined by one specific definition. This is due to the wide nature of the problem composed of numerous other sub-problems that will be discussed later in this study. A very general definition of the “in-sync” problem would be the incompatibility between design and code or vice versa. More specifically, we can look at the “in-sync” problem as having design documents that do not reflect the actual code implementation, or equivalently, having a code implementation that deviates from the original design and does not reflect the intent of the design.

2- MOTIVATION

As a result of holding a design that is out of sync with its corresponding code, software developers face hardships adding new functionality and updating the system. More precisely, when software developers want to change or update certain aspects of the modeled system, they tend to go back to design documents since an effective design allows faster communication. However, with the existence of the “in-sync” problem, design documents become ineffective if not useless.

Developers need to amend the design so as to return it to a state of synchronization with the code. Nonetheless, if developers were to do the resynchronization manually, they will deviate from the actual time plan and suffer as a consequence of the late delivery of the software product. There is therefore a crucial need for effective tools that are able to automatically capture the differences between code and design documents. Such tools will ease the resynchronization process between code and design as well as save a great deal of time and business resources determining where the code has deviated from design. Thus, studying the “in-sync” problem carefully and capturing its details support us in building useful tools that can help software developers get the code back into synchronization with design. Moreover, there is an increased interest in the “in-sync” problem as the design process of software development is considered to be the starting point of the

development life-cycle. Therefore, we have realized the importance of having such tools that ease the process of software development and make it more productive.

3- PREVIOUS WORK

A great deal of work has been done on comparison of programs. Nevertheless, we have been unable to find work that specifically addresses the “in-sync” problem of a program and its design documentation. On the one hand, some CASE¹ tools [7] can take design and produce some (if not all) of the code. Also, some CASE tools [8] can take the source code and produce some of (if not all) the design documents. On the other hand, none of the CASE tools that we have been able to find will compare the design documentation against the source code and point out where there may be in-sync problems.

Furthermore, many papers addressed topics which are related to the “in-sync” problem but are not equivalent. One of such topics is the Process of Reverse Engineering of Class Diagrams. In this process, the inspected source code is converted back into a Class Diagram [9] and manually checked against design documentation. Another related topic is the concept of design differencing. Here, the design documents are diffed against each other [10] and different reports are produced showing the captured differences. However, these topics do not directly address the comparison between a program’s source code and its design documents which is the subject of this thesis.

4- THE OBVIOUS SOLUTIONS

Referring to the general definition, many people may have the wrong perception of the “in-sync” problem solution. They may consider it as an easy thing to do due to the existence of two unfeasible obvious solutions to this problem. The first solution is to convert design to code and then compare the two code artifacts. The second one follows the same conversion principle but in the opposite direction where code is converted into design and then a comparison of the two designs is performed to see if they are equivalent. Unfortunately, neither of these solutions can be made to

¹ CASE: Computer Aided Software Engineering

work. The equivalency of two programs has been shown to be reducible to a famous noncomputable problem, “The Halting Problem.”²[11]

5- SOLVING THE IN-SYNC PROBLEM MANUALLY

It is really important to understand how a software developer would solve the “in-sync” problem manually. The reason for this is to understand the approach of this working but inefficient solution. Building such understanding will help us structure our automatic approach in a better way.

In order to better understand the manual approach for the comparison process, we will walk through a simple example that portrays the fundamental ideas behind the manual approach. Specifically, we will be discussing how to manually compare UML Class and Sequence diagrams to their code representation.

Example: Bank Account Application

Figure (5.1) shows a class Diagram that represents a very simple Bank Account Application program.

² The Halting problem is a decision problem which belongs to the computability theory. The statement of the Halting problem is as follows: “Given a description of a program and a finite input, decide whether the program finishes running or will run forever, given that input.”

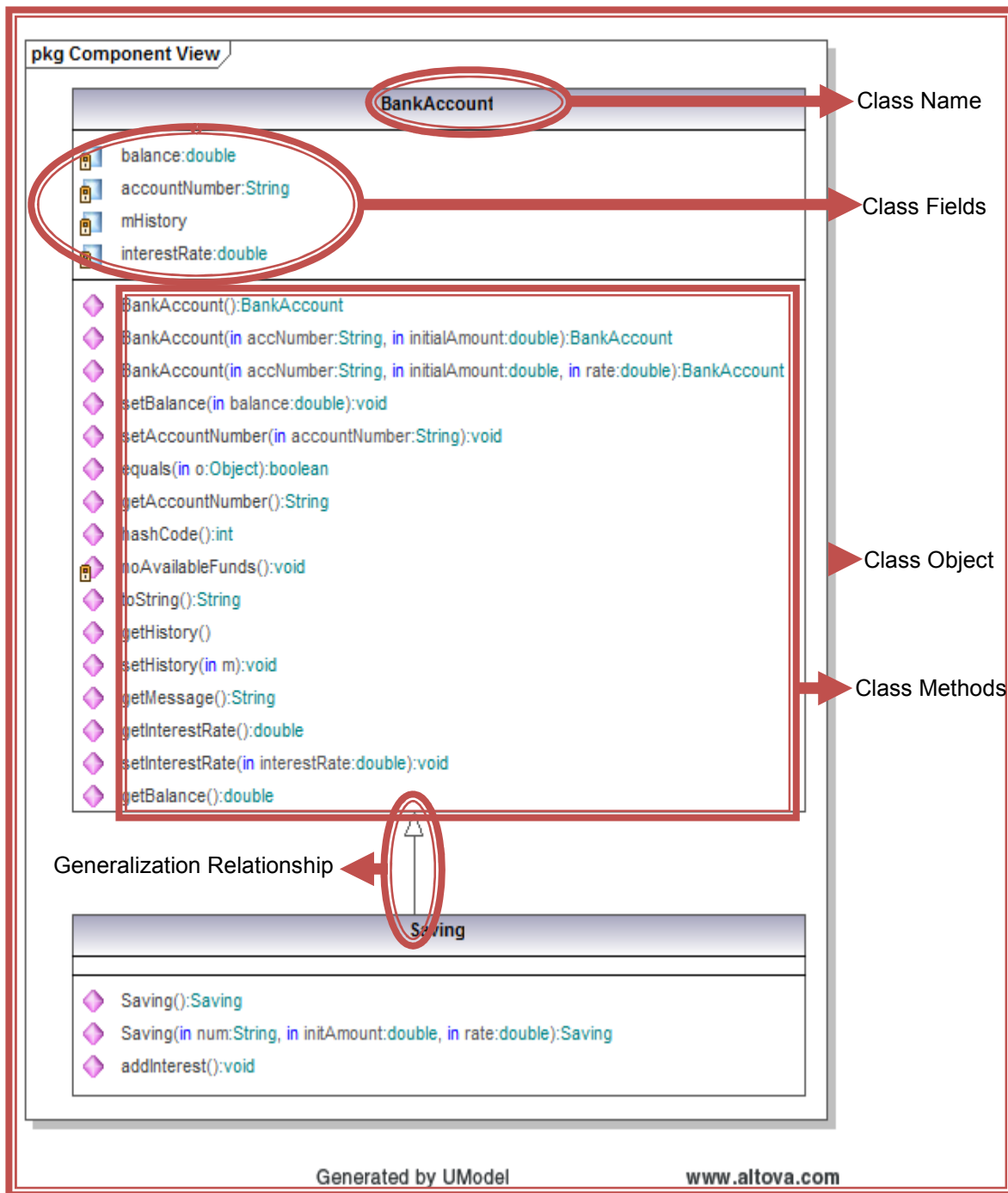


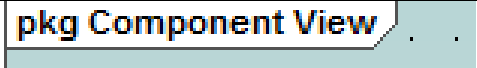
Figure (5.1): Bank Account Application – Class Diagram

The code representation of this Class diagram will be a set of two Java Source files named (“BankAccount.java” and “Saving.java”). The implementation of these files is like any regular Java class implementation. Here, we did not include the full java code for these files. However, we will show the appropriate pieces of code as we go through the comparison process.

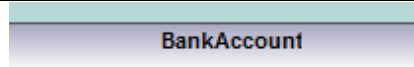
Generally, the approach to the “in-sync” problem is to compare elements’ properties (name, visibility, type, etc) in design to elements’ properties in the code. If the properties match then we have design code compatibility otherwise, the software developer will record the differences in his/her notes.

For Class diagrams, the comparison process proceeds as follows:

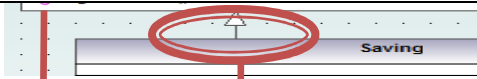
1. Comparing Packages’ Names:

Code	Design
<code>package ComponentView;</code>	


2. Comparing Classes’ Names :

Code	Design
<code>public class BankAccount</code>	

3. Comparing the extended classes’ names (if any) and the names of the implemented interfaces (if any).

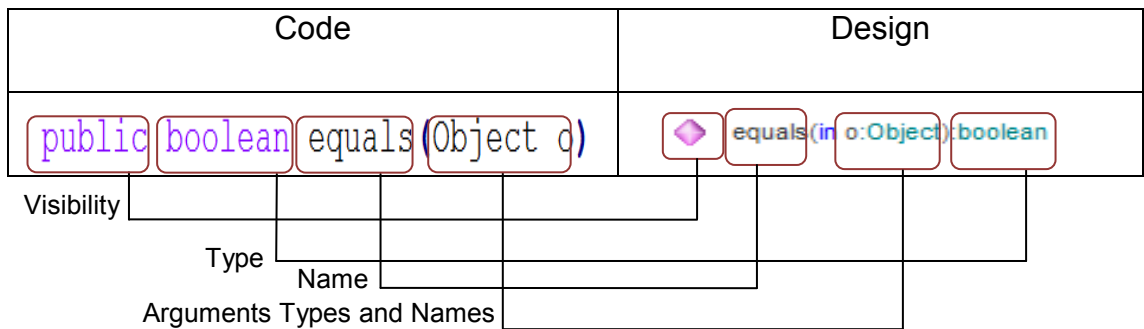
Code	Design
<code>class Saving extends BankAccount{</code>	

4. Comparing Fields :

Code	Design
<code>private double interestRate;</code>	

Visibility			
Type			
Name			

5. Comparing Methods :



Once the software developer finishes with the Class diagrams, he/she starts to manually compare the Sequence diagrams. Figure (5.2) shows one of the Sequence diagrams for our simple application.

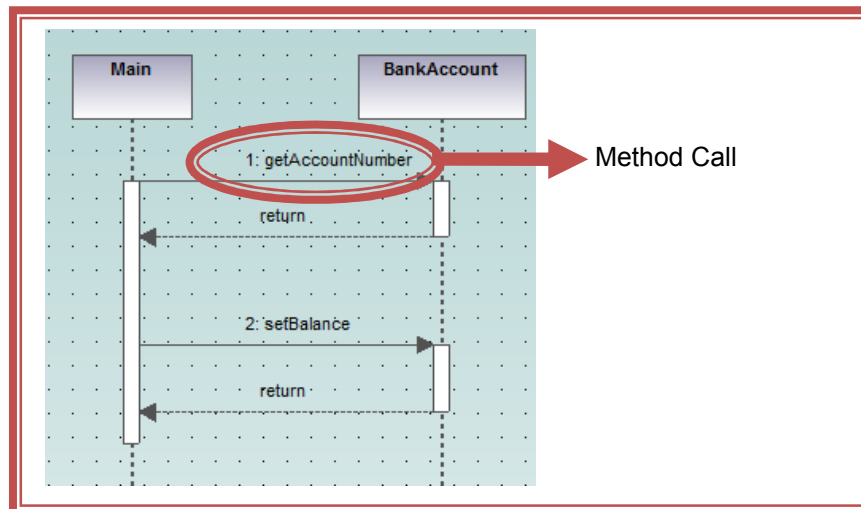


Figure (5.2): Bank Account Application – Sequence Diagram

For Sequence diagrams, the comparison process is based on matching messages that appear in the Sequence diagrams to method calls that appear in code. However, such comparison is too complex as we will see when we discuss the automated approach and the problems we faced during the implementation. Figure (5.3) shows an example of what we mean by complexity.

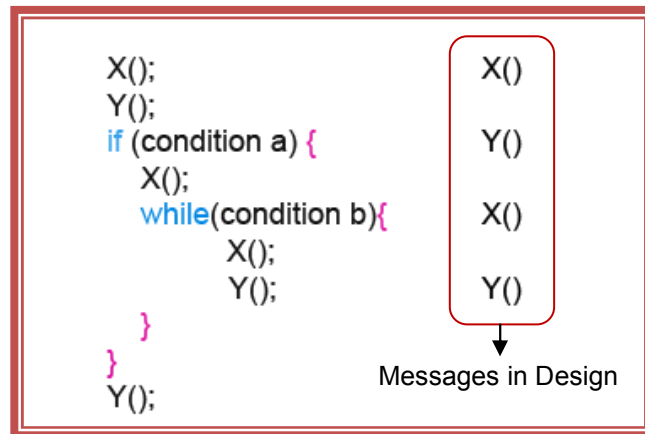


Figure (5.3): Control Flow Complexity

As we can see from Figure (5.3), we have different method calls to methods (X) and (Y). If the Sequence diagram shows the following sequence of messages: (X, Y, X, Y). Then, when the software developer compares design to code, there will be multiple scenarios that can occur. One scenario is where condition (a) is true and condition (b) is false, which means that design matches the code. The other scenarios for values (a) and (b) would result in an “in-sync” problem. Therefore, we cannot determine statically whether or not the design and code matches.

Although the manual approach is doable, it is not feasible. Basically, the manual approach is labor intensive, error prone, and requires a great deal of time in order to be conducted. Most importantly, we need to do this procedure not only for every Class and Sequence diagrams, but also for each of the components in these diagrams. Therefore, the manual approach is a long and tedious process that requires a lot of patience and attention to details. Accordingly, in the next section we will discuss our approach which automates the tedious manual one.

6- OUR APPROACH

In general, we can say that the aim of our solution is to help software developers to close, as much as possible, the gap between a program’s source code and its design. However, we will not be trying to solve every aspect of the problem since it is too complicated to be addressed in a single thesis. Instead, in our approach we will be focusing on Class and Sequence Diagrams because they are considered to be the heart of design.

As we have seen in the manual approach, software developers need to compare the elements of design and code to check for their compatibility. The reason why software developers perform the comparison in this way is due to the modular structure of both Java and UML which follows from the object-oriented paradigm. Figure (6.1) shows the modular structure of java.

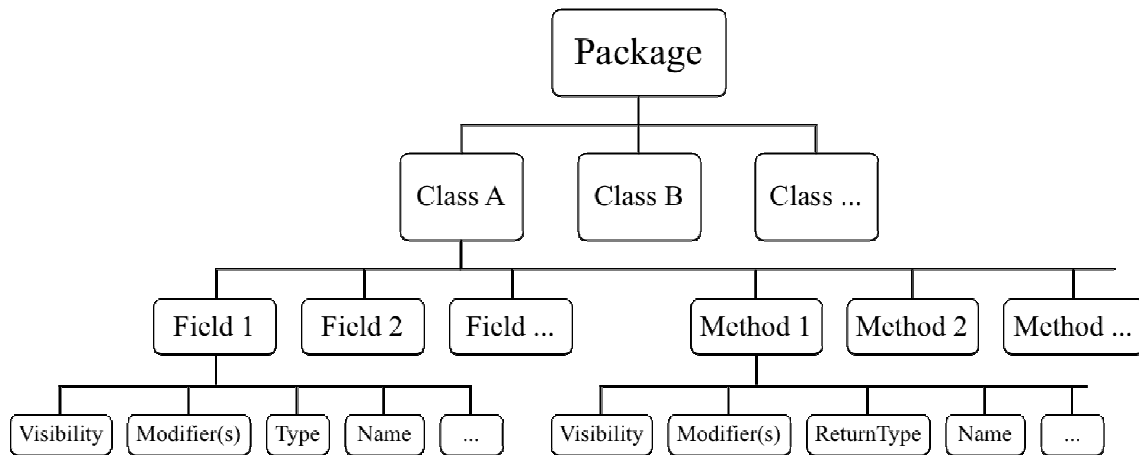


Figure (6.1): Java Modular Structure

Our approach to solve this problem is divided into a number of steps. First, we need to have the design and code in a comparable format. Comparable means having formats where we can identify the elements of design and code so that we can compare these elements. In order to do that, we have used the [12] XMI (XML Metadata Interchange) notation to encode design. XMI is standard notation where design is serialized in a structured order using XML elements and tags. The usage of XML in XMI encouraged many CASE tools manufacturers to integrate it with their products which makes it easy to convert design documents into textual representations [13]. Figure (6.2) shows an example of the XMI representation of a class method.



Figure (6.2): XMI Encoding of a design Method in a Class diagram

Once we have the XMI file which represents the design, we parse both the source code and the design (XMI file). We will parse the source code to build a data structure which stores the needed information about the code elements. For example, we can store information about classes such as class name, visibility, class members, method calls etc. Also, we will parse the design to transform the XMI representation into a data structure storing information about the design and the elements it represents. The detailed implementation of the data structures will be discussed in the next section.

The third step will compare the two data structures created in step two identifying the three main categories on which our solution is based. The first category is called “things we know ‘they are correct’”. This category reflects the fact that what the design represents is compatible with what the code represents. For instance, having a boolean variable named “stop” in the design and a boolean variable named “stop” in the code indicated design-code compatibility.

The second category is called “things we know ‘they are wrong’”. This category itself is divided into three sub categories. The first sub-category represents those elements that are found in both design and code but are incompatible. For example, a declaration of an integer variable named balance is not compatible with a declaration of a double variable named balance. i.e. elements names and types in the design must agree with those in code. The second sub-category includes elements that appear in code but not in design. For example, having some variables declared in the code without equivalents in the design. The last sub-category shows those elements which appear in design but do not appear in code.

The last main category in the third step is what we call “things we don’t know” for many reasons that we discuss next. Figure (6.3) below shows the three main categories we have just discussed.

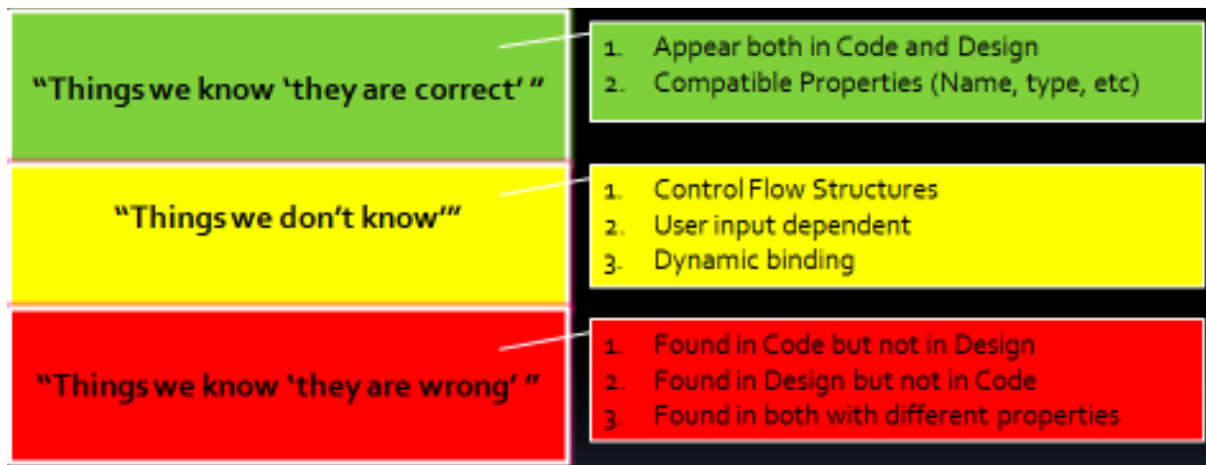


Figure (6.3): The three main categories of our approach

There are many reasons why we named the last category of our approach as “things we don’t know”. These reasons are considered to be the sub-problems of the “in-sync” problem and they are:

6.1 Differences Between Design and Code Notations

Although both of them follow the object oriented paradigm, Java and UML represent some common concepts in different ways. On the one hand, one can find some concepts that are easily represented in UML, but are not allowed or are indirectly represented in Java. For example, UML allows multiple-inheritance by allowing a class to have multiple generalization relationships with other classes. However, multiple-inheritance is not allowed directly in Java. More specifically, Java does not allow a class to extend multiple classes simultaneously. Nevertheless, Java allows a class to extend another class and implement many interfaces to achieve the goal of multiple-inheritance.

On the other hand, some of the concepts that are represented in Java have just been added to UML. For example, older versions of UML did not have the elements to represent control flow structure. One could not represent a loop structure in a Sequence diagram. Instead, the designers need to explicitly write down the behavior of the loop as many times as the loop executes. This of course made design documents larger in size and redundant.

In the newer versions of UML such as UML Version 2.1, many of those differences are addressed and the appropriate notations are integrated into the UML structure. However, there are still specific notations in UML that cannot be mapped directly to Java. The most closely related of such notations to the “in-sync” problem are UML Relationships and UML Multiplicity.

6.1.1 UML Relationships

In a class diagram, designers use different types of arrows to show different types of relationships. The most commonly used relationships are the Association and Generalization relationships. On the one hand, in an association relationship, the designer wants to show the connection between two class objects and why they should be connected. Also, he/she wants to show the rules that govern this relationship. However, there is no equivalent or specific notation in code (Java).

More precisely, if a class A is associated with class B in design, then one can represent this in code in many different ways such as having an instance of class A declared inside class B or vice versa, or having a list of instances of class A declared in class B. Such things can be checked, however, we cannot be sure that this is what the design meant with that specific association relationship.

On the other hand, in a generalization Relationship, the designer wants to show what properties a specific class inherits from other classes. This kind of relations is portrayed in code by using the “*extends*” keyword. However, the problem here is that Java does not allow multiple-inheritance while design allows a single class to be related to many other classes using the generalization relationship.

6.1.2 UML Multiplicity

Design allows designers to assign multiplicity values to relationships. Multiplicity tells the designer how many times each class can instantiate another one. Due to the different types of multiplicities design offers, some of them cannot be determined in code. For example, many-to-many multiplicities are impossible to be statically checked in code because we cannot tell how the system will behave in all possible scenarios. Therefore, we cannot know beforehand how many times the system will instantiate a certain class.

6.2 Methods Overriding and Dynamic Binding

Another difficulty that one would encounter trying to get design in-sync with the code is the problem of method overriding and dynamic binding. To better understand this problem we will explain it in an example. Suppose we have three classes (A, B, and C) and that both classes (B and C) extends class (A). Also, suppose that we define a method called (“equals”) in class (A) and that classes (B) and (C) have a different implementation of the “equals” method (i.e. they override it). Now, having a method call to “equals” can be confusing since we cannot tell from static parsing if a call to the “equals” method is associated with an instance from class (A), class (B), or class (C). Therefore, if we see in the design a method call to “equals” then we cannot be sure which one of “equals” methods the design is referring to.

6.3 Methods Overloading and Sequence Diagrams

Java allows developers to declare multiple methods with the same name, but with different arguments and/or return type. Methods overloading is considered to be a problem when we compare the design Sequence diagram to the code implementation. Specifically, Sequence diagrams represent messages or method calls by mentioning their names only. Therefore, if we have an overloaded method in a certain class, then we cannot determine which method the Sequence Diagram is referring to because in code we will have two or more different methods with the same name but they are overloaded. So, having the names only prevents us from being absolutely sure what method the Sequence Diagram is referring to.

6.4 Control Flow Structures

This is the last problem that we will address in this thesis with regards to the “in-sync” problem. In specific, this problem is related directly to the Sequence diagrams since they show the behavior of objects (flow) in the modeled system. The appearance of method calls inside a control flow structure in the code makes it difficult if not impossible for us to know whether the messages which appear in design are compatible with those calls that appear in code. For example, we cannot tell from static parsing if the condition of an if-else statement will evaluate to true or false. Another example would be having a sequence of messages that shows a call for a method “X” in the sequence diagram. Equivalently, the code can represent that in a loop structure which can be guarded by a dynamic condition that cannot be determined until run-time. These examples show the complexity added by the control flow structure as depicted in Figure (5.3) in section 5. Therefore, we classified such issues under the “things we don’t know” category.

However, in our approach, we were able to use the fact that code statements such as method calls must appear within a method body. Specifically, we have identified three main sections when analyzing the body of a method in the source code. The first section is what we call a “Prefix” includes all the statements that appear in the method’s body and before the first control structure. In the “Prefix,” method calls should appear in the sequence diagrams since we are certain that these method calls will be executed.

The second section is the “Control Structure.” This is the section where we cannot grantee the compatibility of design and code as a consequence of our uncertainty regarding the execution of the statements that appear within the control flow structure.

Lastly, we have the third section which we call the “Suffix.” The “Suffix” section includes the statements that appear after the last “Control Structure.” Like in the “Prefix” section, the calls that appear in the “Suffix” section must also appear in the Sequence diagram for validating the design and code compatibility. Figure (6.4) shows a clearer view of these three sections.

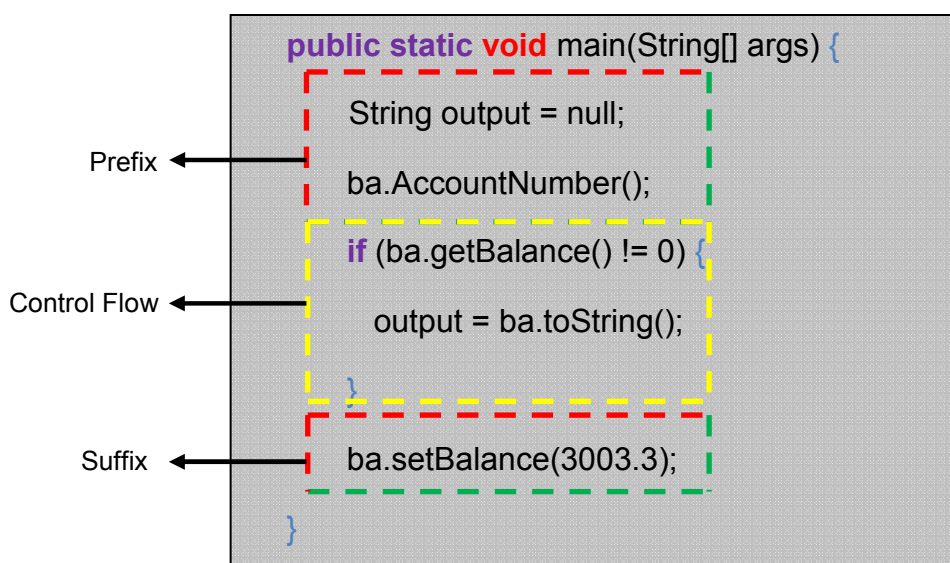


Figure (6.4): Method Body: Prefix-Control Flow-Suffix

7- JDCV FRAMEWORK: JAVA DESIGN CODE VERIFICATION

JDCV is an extensible Java framework that supports people struggling with the “in-sync” problem. We have populated the JDCV framework with an initial set of analysis modules. These modules address different parts of the “in-sync” problem regarding Class and Sequence diagrams.

In general, the way in which the tool functions is exactly as described in our proposed approach in section 6. Mainly, the tool has code and design parsing capabilities in which it parses the code artifacts and the XML representation of the

As we can see from the previous figure, JDCV consists of a graphical user interface that allows the user to interact with the system. Figure (7.2) shows the GUI of JDCV.

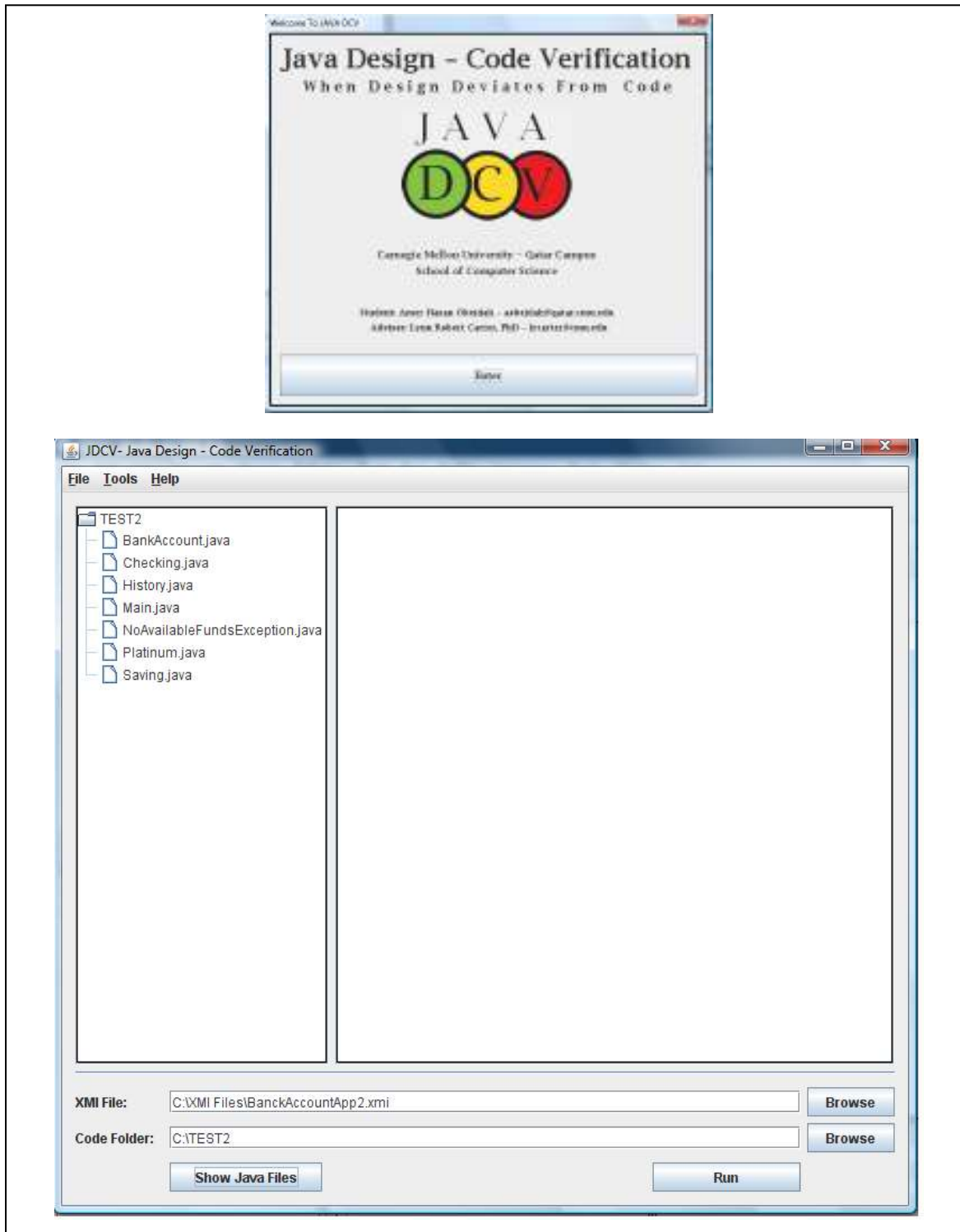


Figure (7.2): JDCV Graphical user Interface

As shown in the figure above, JDCV allows the user to specify the path of the XMI File that represents the design documents. It also requires the user to specify the path to the code folder (Package) which represents the code artifacts. At this point the user has two options: process the whole code package or process the selected Java files. The first option occurs when the user presses the “RUN” button. The second option will occur if the user presses “Show Java Files” button which will instruct JDCV to list the java files inside the specified folder for the user to select from as shown in Figure (7.2). These options are important because the code may not be fully described by the design documents.

With the appropriate option selected, when the user presses the “RUN” button, JDCV will create a “Processor” object as depicted in Figure (7.1). This object creates a “Data Structures” Object which itself constructs two internal data structures. The first one is for code and it is called “Code Data Structure”. The other one is for design and it is called “Design Data Structure”. The reason for creating this “Data Structures” object is to establish the interfaces to support the extensibility of the framework.

The way we have constructed the data structures is similar to the modular structure of java as shown in figure (6.1) in section 6. Basically, for both the design and the code, we have created a “Package Object” which contains a HashMap of the classes and another one for the interfaces. We have selected Hashmaps because they have high performance. Furthermore, for code, we cannot have two classes with the same name inside the same package. Therefore, we considered the class name as a unique key for hashing. However in the design, we have used the unique XMI ID instead of identifiers.

Each class is represented by a “Class Object” which itself contains an “Extends List”, a list of “Field Object”, a list of “Constructor Object”, and a list of “Method Object” which itself contains a list of an “Argument Object”. These objects have different attributes which represent their properties. For example, if we looked at the implementation of the “Field Object” class, we will see that there are attributes such as “Type, Name, etc”. The same issue applies to all of the other objects.

After the initialization is finished, the “Processor” object takes the input files and parses them. On the one hand, the “Processor” object sends the code files to a

Java Parser which was generated by [14]. On the other hand, it sends the XMI file to a DOM parser [15] which can be found in the Java Development Kit (JDK 6) [16]. During the parsing process [17], both parsers create and insert the obtained information into appropriate parse tree nodes.

Afterwards, the “Processor Object” invokes the “doAnalysis” method, which has access to all the available analysis modules to perform the analysis. All of these modules must extend the abstract class “Analyzer” in which the method “doAnalysis” is declared. Our initial set of analysis modules implements the straight forward analysis and matching algorithms. In other words, for each element defined in the code data structure, the matching algorithm searches for an equivalent element in the design data structure. Having said that, the JDCV framework allows other people to extend it by adding their own analysis models and matching algorithms. Specifically, others need not recompile the whole source code of JDCV. On the contrary, one of the important features of the JDCV framework is its ability to integrate external analysis modules during runtime.

The dynamic extendibility feature of JDCV is a result of using Java’s reflection package during the implementation. The way this feature works is not trivial. In specific, the “doAnalysis” method scans a predefined folder named “FUNCTIONS” where the analysis modules must be located. For every module, it studies it to see if the module extends the “Analyzer” class. If it does, JDCV will use an instance of the java class “Class” and its byte loading capability in order to load the byte code of the desired module into the Java Virtual Machine (JVM). However, there are two important key points that should be mentioned here. First, in order to load an analysis module, then we need to have the compiled version of that module and all of the other classes upon which it depends.

Moreover, adding a newer (modified) instance of a class to JDCV during a single execution requires unloading of all dependent code modules that were previously loaded.

Secondly, we did not include a dynamic unloading method to unload the added modules. Precisely, when a module is loaded to the JVM, then any other changes to that module will not take place. This is because, if we loaded version 1 of class “A”, and then we modified it to version 2. The “doAsysis” method will keep

loading the old version since it has been saved by the major Class Loader. As a result, when adding the same class again, the Class Loader by construction will refer to the cached copy of that class. Moreover, the analysis modules need to use both of the “CDS” and “DDS” classes to be able to access the code and design data structures.

Finally, in Figure (7.3) we show snapshot of the output we have obtained when running JDCV on the code and design of the Bank Account Application we have mentioned in section 5.

```

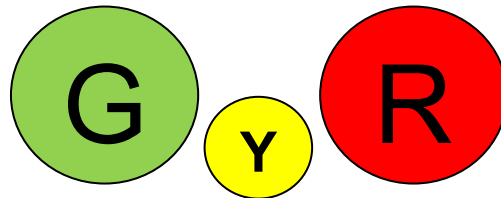
*****
----- COMPARING CONSTRUCTORS OF CODE CLASS:[Platinum] & CONSTRUCTORS OF DESIGN CLASS:[Platinum]-----
*****
- CODE Constructor Name :[Platinum] MATCHES DESIGN Constructor Name :[Platinum]
- CODE Constructor Name :[Platinum] MATCHES DESIGN Constructor Name :[Platinum]
- CODE Method Name :[getOverDraftLimit] MATCHES DESIGN Method Name :[getOverDraftLimit]
- CODE Method Name :[getAvailableFunds] MATCHES DESIGN Method Name :[getAvailableFunds]
*****
----- EXTENDS RELATION OF CODE CLASS:[Platinum] & EXTENDS RELATION OF DESIGN CLASS:[Platinum]-----
*****
CODE Class:[Platinum] Extended Class (BankAccount) -(MATCHES)- DESIGN Class [Platinum] Extended Class (BankAccount)
*****
----- FIELDS OF CODE CLASS:[Main] & FIELDS OF DESIGN CLASS:[Main]-----
*****
#####
CODE Class:[Main] -(Has NO Fields)-
#####
#####
DESIGN Class:[Main] -(Has NO Fields)-
#####
*****
----- COMPARING CONSTRUCTORS OF CODE CLASS:[Main] & CONSTRUCTORS OF DESIGN CLASS:[Main]-----
*****
- CODE Constructor Name :[Main] MATCHES DESIGN Constructor Name :[Main]
- CODE Method Name :[main] MATCHES DESIGN Method Name :[main]
*****
----- COMPARING METHOD CALLS & SEQUENCE DIAGRAMS -----
*****
public void main ( args String[] ){
    String output = null;
    BankAccount ba = new BankAccount();
    ba assignAccountNumber(0);
    if (ba.getBalance() != 0) {
        output = ba.toString();
    }
    ba setBalance(3003.3);
}
}

```

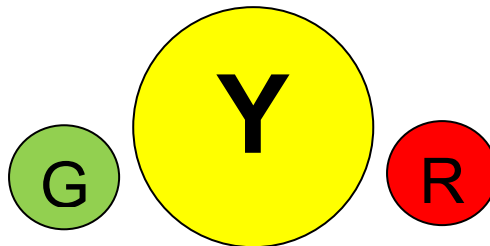
Figure (7.3): JDCV Output

8- CONCLUSIONS

When we started working on this problem, we thought that we can solve the “in-sync” problem through parsing and direct comparison. Therefore, our vision at the beginning of the project can be depicted as follows:



The green circle represents “Things We Know ‘are correct’”, the yellow circle represents “things We don’t Know”, and the red circle represents “Things We Know ‘are wrong’”. However, with the careful analysis of the “in-sync” problem, we concluded that the actual picture of the “in-sync” problem is as follows:



Based on this realization we have changed our vision for the application and learned how to produce a dynamically modifiable extensible framework.

9- FUTURE WORK

We have not tested the JDCV framework on any large projects. While we are fairly confident that the tool works, it would be valuable to study its usefulness on real world sized systems.

Moreover, we have implemented JDCV as a framework rather than a standalone tool that addresses just aspects of the “in-sync” problem. The main reason for this is because there is a huge room for further enhancements and

improvements. We intend to extend the framework by adding more analysis modules that address the UML relations in more detailed and effective way.

Additionally, we can add more sophisticated analysis modules that are based on heuristics in order to diminish the size of the YELLOW zone (“Thing we don’t know” category) in specific areas. One possible project is to integrate the SOOT Java Framework in our implementation. This in turn will equip JDCV with the capability of building Control Flow Graphs (CFGs) which will help us obtain more information about the methods calls in the code. Also, we can extend the current analysis modules to address the new UML notations for representing the control flow structure such as loops and if-statements. Specifically, such work will aid in a better support for handling Sequence diagrams. Another important issue to be addressed is when we have incompatibility among the results generated from different analysis modules examining the same code and design from different analysis tools.

REFERENCES

- [1] A.Dennis, B.H.Wixom, and D.Tegarden, *System Analysis and Design with UML Version 2.0*, 2nd ed, Hoboken, NJ: WILEY, 2005, pp. 1-2
- [2] http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/history_of_uml.htm
- [3] Paul R. Reed, Jr. *Developing Applications with JAVA and UML*, Boston MA, Pearson Education, 2001
- [4] P.Grassle, H. Baumann, and P. Baumann, *UML 2.0 in Action*, Birmingham, UK: Packt, 2005
- [5] http://www.pcmag.com/encyclopedia_term/0,2542,t=UML&i=53392,00.asp
- [6] <http://www.ibm.com/developerworks/rational/library/3101.html>; IBM Library Website
- [7] <http://office.microsoft.com/en-us/visio/default.aspx> ; Microsoft VISIO, CASE tool.
- [8] <http://www-306.ibm.com/software/rational/> ; IBM Rational Rose, CASE tool
- [9] M.Girschick, "Difference Detection and Visualization in UML Class Diagrams", TU Darmstadt, 2006
- [10] Z.Xing, and S.Eleni, "UML Diff: An Algorithm for Object-Oriented Design Differencing", University of Alberta: Edmonton AB
- [11] A. Biermann, *Great Ideas in Computer Science*, , 2nd ed, MIT Press, 1997
- [12] <http://www.omg.org/technology/xml/index.htm>; Object Management Group, XML Technology
- [13] http://www.altova.com/products/umodel/uml_tool.html; ALTOVA UModel Enterprise Edition 2008, CASE tool
- [14] <https://javacc.dev.java.net/>, JAVACC Project
- [15] http://www.w3schools.com/Dom/dom_parser.asp, DOM Parsers
- [16] <http://java.sun.com/javase/6/docs/>, Java Development Kit 6 Documentation
- [17] <https://javacc.dev.java.net/doc/docindex.html>, JAVACC Documentation