Use of Software to Enhance Classroom Teaching

Conference Paper · August 2020

CITATION
1

READS
423

1 author:

Pinaki Chakraborty
Netaji Subhas Institute of Technology
2 PUBLICATIONS **7** CITATIONS

SEE PROFILE

Use of Software to Enhance Classroom Teaching*

Pinaki Chakraborty
Division of Computer Engineering, Netaji Subhas University of Technology, New Delhi 10078
E-mail: pinaki_chakraborty_163@yahoo.com

Introduction

Teachers use different types of instructional tools to enhance classroom teaching. In the last six decades, teachers have been using software tools to teach their courses. There are several advantages of implementing instructional tools as software. Software tools are more versatile than traditional instructional tools, can be customized as per the requirements of individual students, can be shared among a large number of students and require little maintenance. Consequently, educational software is now used to foster formal and informal education at various levels.

Nowadays, children get exposed to software at around two years of age when they start playing with smartphone apps (Yadav and Chakraborty, 2017). Students use different types of software on computers and smartphones as they grow up. Educational software is now available for kindergarteners (Jain *et al*., 2018a), schoolchildren (Yadav and Chakraborty, 2018a,b) and university students (Jain *et al*., 2018b).

India and other countries in South Asia have seen commendable advance in digital technologies in the last two decades. Today, software is used by a large proportion of people in these countries for various purposes. Additionally, the governments and the population of these countries strongly appreciate the role of digital technologies in everyday life. Educational software is now being developed and used in these countries in a non-trivial scale.

In this paper, we discuss the use of educational software in an Indian university. We discuss the use of software tools to teach five courses of different types. We have used software tools to teach courses on theory of automata, computer programming, operating systems, compiler construction and English pronunciation. Undergraduate and graduate computer science students attended these courses.

The rest of the paper discusses the evolution of educational software, the differences between educational software for computers and smartphones, and our experience of using software tools for teaching these courses.

Evolution of Educational Software

The first computers were developed in the mid-1940s. Those early computers were owned and operated by the military organizations of a few countries. Within a decade, however, several government agencies and big companies started using computers. By the early-1960s, computers have arrived in several universities. Educationists were quick to understand that computers can be used to enhance the delivery of information to students and started developing purpose-built educational software. Although the early computers could be used to enhance teaching, they had several limitations like being large, expensive and not easy to use.

In the early-1980s, personal computers became widely available. These personal computers were less expensive than their predecessors and small enough to be placed on a desk. One reason that contributed in the decrease in both price and size was the use of microprocessors as the central processing units in these computers. The personal computers were typically sold with a graphical user interface which allowed people to use them with ease. The personal computers were mass-produced and were procured in large numbers by universities. Professors took help of these computers to enrich their courses and developed necessary educational software.

* Cite as: Chakraborty, P. 2019. Use of software to enhance classroom teaching. *Proceedings of the International Conference on Higher Education in South Asia: Challenges and Possibilities*, pp. 295-304.

In the early-1990s, the World Wide Web was invented. The World Wide Web allowed easy, inexpensive and fast dissemination of educational software around the world. Professors had to earlier struggle to procure software for their courses and to share the software they developed. The World Wide Web solved both the problems. Professors now started sharing the software they developed through their webpages and other online forums.

Laptop computers became popular in the mid-1990s. Although they were technologically similar to personal computers, laptop computers were portable and students could carry them to the classroom. The same software worked on personal computers and laptop computers. Laptop computers ensured that students could use educational software in the classroom and not just in the computer laboratory.

Smartphones became available in the early-2000s. They were mobile phones supporting many features of personal computers. Users could install and use software according to their needs on smartphones. Professors soon started implementing educational software as smartphone apps. Smartphones were smaller and easier to carry than laptop computers. Smartphones could allow students to use educational software anywhere and anytime.

Nowadays, most students own laptop computers and smartphones. Professors recommend suitable educational software to enrich their courses. Students use the recommended software on their laptop computers and smartphones. Many professors often develop their own software to teach their courses. Students typically use their laptop computers and smartphones to connect to the Internet. They use their devices to collaborate with their peers and professors. Educational software that supports such collaboration is also available.

These developments have led many universities to move towards a Bring Your Own Device paradigm. Following Bring Your Own Device paradigm frees up universities from procuring and maintaining computing facilities for all their students. The universities can focus on providing information services in other forms instead. It also allows students to use their customized devices for class-work and then take home the work done during the class.

Educational Software for Computers and Smartphones

Although smartphones have many similarities with personal computers and laptop computers, there are some differences in the software used on smartphones and computers. The technological differences of computers and smartphones should be taken into consideration while developing educational software for them.

Computers have faster processors and can be used for intense calculations. Computers have more secondary memory which can be used to store large volumes of data. Textual and numeric data are entered in a computer using a physical keyboard and the process is fast and accurate. Computers also have an on-screen pointer that is controlled by a mouse or a touch pad. The on-screen pointer can be used to efficiently manipulate objects displayed on the screen.

Smartphones have a small screen and can display limited information at a time. The screen typically displays only one window at a time. The touch screen of smartphones acts as both an input device and an output device. The users can manipulate the icons displayed on the screen by touching them with their fingers directly. This type of interaction is particularly helpful for non-expert users. However, the finger lacks the precision of an on-screen pointer and hides a part of the screen. Textual data is entered using an on-screen keyboard and numeric data is entered using a smaller on-screen numeric key pad. The process is slow, error prone and tiring. Nevertheless, smartphones have their own advantages. Smartphones can literally be used anywhere and anytime. Smartphones require less time to become active from an inactive state. Smartphones can work for longer hours without charging. Smartphones come with a wide range of sensors which may also be accessed by educational software. Moreover, smartphones have multimedia support and Internet connectivity similar to computers.

Educational software can work on computers as well as smartphones. However, educational software should be developed for a device which better suits the requirements of the course.

Educational software should be used on computers for courses requiring intense calculations, large volumes of data, displaying large amount of information and precise graphical editing. Alternatively, educational software should be used on smartphones for courses requiring quick and simple calculations and working in unconventional study environments (Kanika *et al*., 2019a,b).

There can be a model of educational software that combines computers and smartphones. A simple version of the software may work on smartphones which students may use during lectures and fieldwork. This version may support limited calculation and information visualization functionalities. Alternatively, a comprehensive version of the software should work on computers supporting intense calculation and other advanced functionalities. Students may use this version during lab sessions and self-learning sessions. It should be however easy to synchronize the two versions of the software.

Irrespective of whether educational software is developed for computers or smartphones, it should be widely disseminated. The software tools may be distributed through the World Wide Web either for free or for a suitable price. The source code of the software tools may also be shared with the users. This may lead to further research and development using the software tools. Professors who use educational software in their classrooms should collect empirical data on the efficacy of the tools in teaching and the results may be published in the research literature.

Software for Teaching Theoretical Concepts

Many courses at undergraduate and graduate levels contain substantial portions devoted to theoretical concepts. Educational software may be used to teach and learn these courses. A course on theory of automata is typically included in an undergraduate curriculum in computer science. The course discusses various abstract models of computations known as automata. The course is typically considered difficult to teach by professors and difficult to study by students. Consequently, specialized software tools are being used to teach the course on theory of automata since the mid-1960s (Coffin *et al*., 1963). Professors typically use automata simulators to explain the concepts to students and students use the same to practice designing automata (Chakraborty *et al*., 2012). Several types of automata simulators have been developed so far (Chakraborty *et al*., 2011a; Chakraborty, 2012).

The first type of automata simulators to be developed was the programming language based simulators. In these tools, an automaton is described as a program written in a suitably designed programming language and then its behavior is simulated (Coffin *et al*., 1963; Chakraborty, 2007; Chakraborty *et al*., 2011b, 2013). Programming language based simulators can be used to simulate automata of any size and complexity. However, students need to learn the programming language associated with the tool before using it.

Table based automata simulators receive the description of an automaton in a tabular format and then simulates its behavior (Jagielski, 1988). Table based automata simulators are easy and quicker to use than the other types of automata simulators. However, they can be used to simulate and teach small and simple automata only.

The advent of interactive personal computers in the 1980s allowed canvas based automata simulators to be developed. Such a tool provides a canvas on which students can draw the transition diagram of an automaton and then simulate its behavior (Rodger and Finley, 2006). Canvas based automata simulators are more interactive than other types of automata simulators. Consequently, canvas based automata simulators are being used widely to teach the course on theory of automata since the 1990s.

Automata simulators are considered to be one of the most successful forms educational software used to teach computer science courses (Chakraborty *et al*., 2012). They had been in continuous use for more than five decades (Chakraborty *et al*., 2011a). All automata simulators developed till the mid-2010s worked on computers. However, some professors have recently implemented automata simulators as smartphone apps and used them to teach in Slovakia (Chuda *et al*., 2015), Brazil (Pereira and Terra, 2018) and India (Singh *et al*., 2019). These automata

simulators follow the table based approach (Chuda *et al*., 2015; Singh *et al*., 2019) and the canvas based approach (Chuda *et al*., 2015; Pereira and Terra, 2018). However, it has been observed that it is more convenient to use a table based automata simulator on a mobile phone. Singh *et al*. (2019) used their tool to teach the course on theory of automata to 118 undergraduate students in an Indian university during the autumn of 2018. The tool helped to motivate the students to study the course and score better in examination.

Software for Teaching Computer Programming

Programming is at the core of computer science. Software developers write programs to solve various real-world tasks. Programs specify in detail what a computer has to do to solve the tasks. Programs are written in programming languages which are artificial languages comprising of English-like words and phrases and mathematics-like symbols. Computer science students are taught the principles and techniques of programming in the early part of their undergraduate program. A typical undergraduate curriculum in computer science contains four or more courses devoted to programming. Additionally, the concepts taught in these courses are used in other courses that are taught in the later part of the program. Professors use different types of educational tools to teach the courses on programming.

Visual programming tools are being used widely to teach programming since the advent of personal computers. A visual programming tool provides a canvas were students can draw a flowchart of the program they want to write. The tool then interprets the flowchart and produces output according to the given input. Glinert and Tanimoto (1984) developed the first visual programming tool. More sophisticated visual programming tools were later developed by the likes of Pausch *et al*. (1995) and Maloney *et al*. (2004). Visual programming tools are interactive and can be used to attract students to learn programming. These tools are suitable for teaching introductory courses on programming.

Some professors use game based approach for teaching programming. They use software tools in which students have to write programs to play visual games. The students have to design strategies to play the games and then write programs to implement those strategies (Corral *et al*., 2014). The game based approach can be used to attract and motivate students, and can be used to teach both introductory and advanced courses on programming.

Some professors recommend students to practice programming in pairs or small teams (Gomez *et al*., 2017). Students use tools that support such collaboration. Pair programming and collaborative programming help students to learn from one another and write better programs.

Students like to receive quick feedback on their exercises. Professors typically require a lot of time to evaluate the programming exercises submitted by students. Specialized software tools known as assessment systems can be employed to assess programming assignments submitted by students, grade them and provide them feedback in natural language (Ahoniemi and Reinikainen, 2006). Assessment systems can perform in-depth analysis of programs and provide students detailed feedback that helps them to improve their programming skill. Assessment systems can be use to support classroom teaching as well as MOOCs which students often attend to learn programming (Sra and Chakraborty, 2018).

Software for Teaching Algorithmic Concepts

An algorithm is a sequence of steps that has to be followed to solve a task. Computer programs are written following algorithms. Computer science students require studying various types of algorithms at undergraduate and graduate levels. Professors often use algorithm visualization tools to teach algorithms in classroom. An algorithm visualization tool illustrates the working of algorithms step by step using suitable diagrams.

An operating system is a piece of software that manages all resources of a computer system and transforms the computer hardware into a usable computing device. Operating systems are found in all computers and smartphones. Operating systems need to solve different types of tasks

and use a wide range of algorithms. A course on the structure and internal working on operating systems is typically included in undergraduate computer science curriculums. Over the years, several techniques have been used to teach the course (Chakraborty, 2008; Chakraborty and Gupta, 2008; Chakraborty and Saxena, 2009).

Several professors have developed tools to visualize the algorithms used in operating systems and used them to teach their courses on operating systems (Khuri and Hsu, 1999; Fischbach, 2013; Garmpis, 2013). These tools worked on computers and most of them could visualize only a few common algorithms. Nevertheless, encouraging feedback was received from students.

The algorithms used in operating systems are comparatively simple and use numeric data. This means that a tool to visualize these algorithms may be implemented as a smartphone app. Chakraborty *et al*. (2019) developed a smartphone app to visualize the algorithms used in operating systems. The tool is more comprehensive than the tools developed earlier and can visualize almost all important algorithms used in operating systems. The tool covers algorithms for managing the processor and the memory devices, *viz*. RAM and hard disk. The tool can visualize a total of twenty-four algorithms. Since the tool has been implemented as a smartphone app, students can use them during lectures. The tool optimally uses the limited space of the screen to display its output. The output of the tool is a combination of textual and graphical information. The different algorithms are visualized using appropriate diagrams like Gantt charts and memory maps.

The tool was used to teach undergraduate students in an Indian university in the spring of 2016 through 2018. A total of 243 students attended the course. An analysis of the examination results showed that using the tool helped students to score 6% more marks.

Software for Teaching Complex Algorithmic Concepts

Some software products use algorithms that are overtly complex and process non-numeric data. For example, compilers are software products that are used to process programs written by software engineers and compilers use extremely complex algorithms to check the syntax of the programs. Courses on compilers are taught to computer science students at both undergraduate and graduate levels, and professors use various techniques to teach these courses (Chakraborty *et al*., 2011c, 2014).

The algorithms used in compilers to verify the syntax of programs written by software engineers are colloquially called parsing algorithms. Many professors use visualization tools to teach the parsing algorithms to students. The first such tool was developed by Blythe *et al*. (1994). The tools to visualize parsing algorithms take large amount of textual data as input, require performing lot of processing, and display voluminous output in forms of text and graphics. These issues make it difficult to implement tools to visualize parsing algorithms as smartphone apps and all tools to visualize parsing algorithms work on computers.

Jain *et al*. (2017) and Sangal *et al*. (2018) developed a tool to visualize six parsing algorithms of different types. The level of difficulty of these algorithms can be estimated from the fact that professors devout twenty to thirty percent duration of the course discussing these algorithms. In the tool developed by Sangal *et al*. (2018), students can either type in the inputs or upload the same from a file. After processing, the tool produces output that contains text, tables and diagrams. The volume of the output is often too large to be displayed together on the screen of a computer. Therefore, the tool saves the output in a file which students can later refer to. The tool has been used to teach courses on compilers at both undergraduate and graduate levels in an Indian university in the autumn of 2015 through 2019 and has received positive feedback from the students who attended those courses.

Software for Interactive Learning

English is perhaps the most successful language in the history of human communication. Today, English is spoken by more than one billion people and many of them use English either as a second language or a foreign language. Non-native speakers often face problem in pronouncing

English words and sentences correctly. Computer Aided Pronunciation Training (CAPT) software may be used to improve the pronunciation of students at various levels (Agrawal and Chakraborty, 2019).

A CAPT software tool records the speech of a student, analyzes it, detects the mispronunciations in it and suggests a way for the student to avoid those mistakes. The first CAPT software tool for English was developed by Kalikow and Swets (1972) and more advanced tools have been developed since. CAPT tools try to achieve two orthogonal objectives, *viz*. detecting mispronunciations accurately and providing suggestions to students according to their profiles. Four types of CAPT software tools have been developed so far for teaching English pronunciation, *viz*. visual simulation based tools, game based tools, comparative phonetics based tools and artificial neural network based tools.

Visual simulation based tools record the speech of a student, analyzes it and provides feedback using videos and images (Kalikow and Swets, 1972). Such tools are suitable for young and non-expert learners. Alternatively, CAPT software tools may be implemented as games (Satria *et al*., 2017). They can be used to teach pronunciation in both formal and informal settings. Such tools help students to learn to speak according to the context. Most university students can speak their native language fluently but mispronounce while speaking in English. Comparative phonetics based tools compare the phonemes in English and the native language of the students, and advise students accordingly (Qian *et al*., 2010). Artificial neural network is a model of computation that is often used to process data of imprecise nature. CAPT software tools may use artificial neural networks to detect mispronunciations in students' speech with high accuracy (Li *et al*., 2017; Agarwal *et al*., 2019). It has been observed that CAPT software can detect up to 86% mispronunciation (Wang *et al*., 2008) and help students to lessen their pronunciation mistakes by 23% (Jing and Yong, 2014).

We recommend the merger of the four aforesaid techniques. CAPT software tools may be developed to work on both computers and smartphones. They should be interactive and support necessary visualizations. They should take into account the native language of the students and help them to speak English according to the context. They should internally use artificial neural networks to achieve high accuracy in mispronunciation detection.

CAPT software tools are typically interactive in nature. Their success in helping students to speak in English fluently shows that educational software can support interactive learning. Educational software may therefore be developed for teaching concepts and skills that traditionally require a lot of interaction. Human teachers may be supplemented, but not certainly replaced, by such interactive educational software tools. Such tools may also be customizable to adapt to the requirements of individual students.

Conclusion

Educational software is being successfully used for decades around the world to enrich classroom teaching. Technological advances have now made it easier to develop and disseminate educational software. We recommend that appropriate educational software be used in classrooms in universities in South Asia for the following three reasons. First, educational software can motivate students to study courses that are otherwise considered to be difficult. Second, use of educational software in classroom helps students to learn better and score higher in examination. Third, educational software helps professors to teach their courses and evaluate students.

Acknowledgement

I would like to thank my doctoral students, *viz*. Chesta Agarwal, Kanika and Savita Yadav for providing me inputs to write this paper.

References

Agarwal, C. and Chakraborty, P. 2019. A review of tools and techniques for computer aided pronunciation training (CAPT) in English. *Education and Information Technologies*, in press.

Agarwal, C., Chakraborty, P., Barai, S. and Goyal, V. 2019. Quantitative analysis of feature extraction techniques for isolated word recognition. *Proceedings of the Third International Conference on Advances in Computing and Data Sciences*, pp. 618-627.

Ahoniemi, T. and Reinikainen, T. 2006. ALOHA - A grading tool for semi-automatic assessment of mass programming courses. *Proceedings of the Sixth Baltic Sea Conference on Computing Education Research*, pp. 139-140.

Blythe, S. A., James, M. C. and Rodger, S. H. 1994. LLparse and LRparse: Visual and interactive tools for parsing. *ACM SIGCSE Bulletin*, **26**(1): 208-212.

Chakraborty, P. 2007. A language for easy and efficient modeling of Turing machines. *Progress in Natural Science*, **17**(7): 867-871.

Chakraborty, P. 2008. Verbose mode of operation of a pedagogical virtual machine operating system. *Proceedings of the Third National Conference on Methods and Models in Computing*, pp. 43-53.

Chakraborty, P. 2012. *A Compiler Technology Based Approach to Simulation of Basic Forms of Automata*. Ph.D. Thesis, Jawaharlal Nehru University.

Chakraborty, P. and Gupta, R. G. 2008. The design of a pedagogical operating system. *Proceedings of the Second National Conference on Computing for Nation Development*, pp. 517-527.

Chakraborty, P. and Saxena, P. C. 2009. Novel approaches to teach and learn courses on computer operating systems. *Computer Science and Telecommunications*, **8**(5): 174-176.

Chakraborty, P., Arora, U., Mukhija, N., Goel, V., Priyanka, Shikhar, S., Takhar, R. 2019. OSAVA: An Android app for teaching a course on operating systems. *Journal of Engineering Education Transformations*, **32**(9): 20-30.

Chakraborty, P., Saxena, P. C. and Katti, C. P. 2011b. Fifty years of automata simulation: A review. *ACM Inroads*, **2**(4): 59-70.

Chakraborty, P., Saxena, P. C. and Katti, C. P. 2012. Automata simulators: Classic tools for computer science education. *British Journal of Educational Technology*, **43**(1): E11-E13.

Chakraborty, P., Saxena, P. C. and Katti, C. P. 2013. A compiler-based toolkit to teach and learn finite automata. *Computer Applications in Engineering Education*, **21**(3): 467-474.

Chakraborty, P., Saxena, P. C., Katti, C. P., Pahwa, G. and Taneja, S. 2014. A new practicum in compiler construction. *Computer Applications in Engineering Education*, **22**(3): 429-441.

Chakraborty, P., Taneja, S., Anand, N., Jha, A., Malik, D. and Nayar, A. 2011a. An optimizing compiler for Turing machine description language. *The IUP Journal of Computer Sciences*, **5**(3): 33-39.

Chakraborty, P., Taneja, S., Saxena, P. C. and Katti, C. P. 2011c. Teaching purpose compilers – An exercise and its feedback. *ACM Inroads*, **2**(2): 47-51.

Chuda, D., Trizna, J. and Kratky, P. 2015. Android automata simulator. *Proceedings of the International Conference on e-Learning*, pp. 80-84.

Coffin, R. W., Goheen, H. E. and Stahl, W. R. 1963. Simulation of a Turing machine on a digital computer. *Proceedings of the Fall Joint Computer Conference*, pp. 35-43.

Corral, J. M. R., Balcells, A. C., Estevez, A. M., Moreno, G. J. and Ramos, M. J. F. 2014. A game-based approach to the teaching of object-oriented programming languages. *Computers & Education*, **73**: 83-92.

Fischbach, J. A. 2013. Visualization of student-implemented OS algorithms in Java. *Journal of Computing Sciences in Colleges*, **28**(3): 6-13.

Garmpis, A. 2013. Alg_OS – A web-based software tool to teach page replacement algorithms of operating systems to undergraduate students. *Computer Applications in Engineering Education*, **21**(4): 581-585.

Glinert, E. P. and Tanimoto, S. L. 1984. Pict: An interactive graphical programming environment. *IEEE Computer*, **17**(11): 7-25.

Gomez, O. S., Aguileta, A. A., Aguilar, R. A., Ucan, J. P., Rosero, R. H. and Verdin, K. C. 2017. An empirical study on the impact of an IDE tool support in the pair and solo programming. *IEEE Access*, **5**: 9175-9187.

Jagielski, R. 1988. Visual simulation of finite state machines. *ACM SIGCSE Bulletin*, **20**(4): 38-40.

Jain, A., Goyal, A. and Chakraborty, P. 2017. PPVT: A tool to visualize predictive parsing. *ACM Inroads*, **8**(1): 43-47.

Jain, D., Chakraborty, P. and Chakraverty. S. 2018b. Smartphone apps for teaching engineering courses: Experience and scope. *Journal of Educational Technology Systems*, **47**(1): 4-16.

Jain, D., Patil, A. P., Nawal, D. J. and Chakraborty, P. 2018a. ARWAK: An augmented reality wordbook smartphone app for kindergarteners. *Journal of Multi Disciplinary Engineering Technologies*, **12**(2): 59-66.

Jing, X. and Yong, L. 2014. The speech evaluation method of English phoneme mobile learning system. *Proceedings of the IEEE Workshop on Advanced Research and Technology in Industry Applications*, pp. 546-550.

Kalikow, D. N. and Swets, J. A. 1972. Experiments with computer-controlled displays in second language learning. *IEEE Transactions on Audio and Electro Acoustics*, **20**(1): 23-28.

Kanika, Chakraverty, S., Chakraborty, P., Agnihotri, S., Bansal, P. and Mohapatra, S. 2019b. Linking classroom studies with dynamic environment. *Presented in the International Conference on Computing, Power and Communication Technologies*.

Kanika, Chakraverty, S., Chakraborty, P., Agnihotri, S., Mohapatra, S. and Bansal, P. 2019a. KELDEC: A recommendation system for extending classroom learning with visual environmental cues. *Proceedings of the Third International Conference on Natural Language Processing and Information Retrieval*, pp. 99-103.

Khuri, S. and Hsu, H.-C. 1999. Visualizing the CPU scheduler and page replacement algorithms. *ACM SIGCSE Bulletin*, **31**(1): 227-231.

Li, K., Qian, X. and Meng, H. 2017. Mispronunciation detection and diagnosis in L2 English speech using multi-distribution deep neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, **25**(1): 193-207.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. and Resnick, M. 2004. Scratch: A sneak preview. *Proceedings of the Second International Conference on Creating, Connecting, and Collaborating through Computing*, pp. 104-109.

Pausch, R., Burnette, T., Capeheart, A. C., Conway, M., Cosgrove, D., DeLine, R., Durbin, J., Gossweiler, R., Koga, S. and White, J. 1995. Alice: Rapid prototyping for virtual reality. *IEEE Computer Graphics and Applications*, **15**(3): 8-11.

Pereira, C. H. and Terra, R. 2018. A mobile app for teaching formal languages and automata. *Computer Applications in Engineering Education*, **26**(5): 1742-1752.

Qian, X., Soong, F. and Meng, H. 2010. Discriminative acoustic model for improving mispronunciation detection and diagnosis in computer-aided pronunciation training (CAPT). *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association*, pp. 757-760.

Rodger, S. H. and Finley, T. 2006. *JFLAP – An Interactive Formal Languages and Automata Package*. Jones and Bartlett.

Sangal, S., Kataria, S., Tyagi, T., Gupta, N., Kirtani, Y., Agrawal, S. and Chakraborty, P. 2018. PAVT: A tool to visualize and teach parsing algorithms. *Education and Information Technologies*, **23**(6): 2737-2764.

Satria, F., Aditra, H., Wibowo, M. D. A., Luthfiansyah, H., Suryani, M., Paulus, E. and Suryana, I. 2017. EFL learning media for early childhood through speech recognition application. *Proceedings of Third International Conference on Science in Information Technology*, pp. 568-572.

Singh, T., Afreen, S., Chakraborty, P., Raj, R., Yadav, S. and Jain, D. 2019. Automata simulator: A mobile app to teach theory of computation. *Computer Applications in Engineering Education*, **27**(5): 1064-1072.

Sra, P. and Chakraborty, P. 2018. Opinion of computer science instructors and students on MOOCs in an Indian university. *Journal of Educational Technology Systems*, **47**(2): 205-212.

Wang, L., Feng, X. and Helen, M. 2008. Automatic generation and pruning of phonetic mispronunciations to support computer-aided pronunciation training. *Proceedings of the Ninth Annual Conference of the International Speech Communication Association*, pp. 1729-1732.

Yadav, S. and Chakraborty, P. 2017. Children aged two to four are able to scribble and draw with a smartphone. *Acta Paediatrica*, **106**(6): 991-994.

Yadav, S. and Chakraborty, P. 2018a. Using smartphones with suitable apps can be safe and even useful if they are not misused or overused. *Acta Paediatrica*, **107**(3): 384-387.

Yadav, S. and Chakraborty, P. 2018b. Smartphone apps can entertain and educate children aged two to six but should be used with caution. *Acta Paediatrica*, **107**(10): 1834-1835.