Enhanced Teaching Model (ETM) for Teaching Programming Languages

Fawaz Alajmi
De Montfort University

Ahmad AA Alkhatib
Alzaytoonah University of Jordan

ABSTRACT
Expectations from academics and the industry, to have students and employees who are independent and capable of quickly writing code to resolve work-related issues, are growing high. However, teaching and learning programming is certainly not easy and very challenging. Literature shows that a lot of work has been done to improve this. Nonetheless, it is evident that little effect of this work has had impact on the actual practice of teaching and learning of Software Development programming skills. This gap has been addressed in this paper to enhance the teaching and learning process of programming to students. Furthermore, teaching programming literature research has been classified into 3 categories; teaching approach, teaching model and teaching tool. As a result, this paper proposes the following objectives to tackle this problem:

- Identify what research has found out about how to teach and learn programming and other aspects of Software Development
- Investigate how and why this research has not been applied to teaching Software Development

How more use could be made of it to improve teaching?

Finally, an Enhanced Teaching Model (ETM) has been proposed, which combines several teaching approaches and models from literature. In addition, this model uses teaching tools to provide goal-focused exercises, assess students' performance and obtain feedback from the learning community. Last but not least, a discussion about the future work required in order to assess the model and thus improve it.

Keywords
Enhanced Teaching Model (ETM), Intelligent Teaching System (ITS), Teaching Approach, Teaching Model, Teaching Tool, Software Development.

1. INTRODUCTION
Computers are ubiquitous and our needs to implement things are pretty much dependent on them. Therefore, the demand for calibres to develop software and write efficient code is souring high, i.e. T. Hüsing et al. (2013). Unfortunately, the rate of ICT University dropouts has been increasing over the last decade, i.e. Kori, Külli, et al (2015).

Programming languages are used to solve problems. However, problem solving is beyond the syntax of programming languages (Linn & Dalbey (1985) and Perkins, Schwartz & Simmons (1988)). We believe that the following two questions remain unanswered and will form the space of the research subject:

- Investigate how and why this research has not been applied to teaching Software Development?
- How more could be made of it to improve teaching?

Researchers have been successful in identifying what has been causing the dropouts. Teaching and learning programming has always been challenging for all parties involved, especially to novices (Blayney (2009); Ramalingam, LaBelle and Wiedenbeck (2004); Robins, Rountree and Rountree (2003)). Interestingly, Sleeman (1986) described programming as the new Latin of the school syllabus. One does not have to look far to prove this crystal-clear fact. In reality, most students and teachers, who take part in this skill-building cycle, can simply approve this view. But one might wonder why this is the case where there are as complex subjects, if not more than programming, such as physics. Literature shows that there are issues with the teaching of programming languages to students, which can be briefly highlighted as below:

- Lack of interest and appreciation in programming languages by students
- Individual learning of programming is not practical and tedious
- Use of unsuitable languages to teach programming concepts
- Lack of visual programming
- Too much theory and emphasis on the language syntax itself rather than the programming concepts
- Lack of emphasis on the teaching of programming solving techniques
- Lack of feedback from students and teachers
- Tools are only based on tutorials and quizzes
- On the other hand, researchers provide solutions for those issues, which can be summarised as follows:
- Orient students with the aspects of programming
- Adopt LET US DO IT ALL TOGETHER approach, Ngo-Ye and Park (2014).
- peer programming, peer tutoring and problem solving strategies
- The language should be selected based on pedagogical suitability and not popularity in the industry
- Only calibre teachers should teach programming
- Programming courses should be flexible to all students to learn in different ways
- visual is essential to learning programming

A lot of research has been done on: Why programming is challenging and How programming skills can be taught effectively in Software Development. However, it is evident that little effect of this work has had impact on the actual practice of teaching and learning Software Development programming skills; Students dropouts is still increasing since 2006, according to Kori, Külli and et al (2015). (Ford and Venema, 2010; Thomas, Ratcliffe, Woodbury, et al., 2002; Bornat, Dehnadi and Simon, 2008) have proved that most graduates do not seem to be able to write good code and do not have good understanding of programming concepts. Most of the research is usually done locally and a sample of students and results get published with no official authority to adopt it. Literature is full of examples on this.

2. SUMMARY OF RELATED WORK

Since 1970s, researchers have been providing different approaches to teaching programming. This might seem reasonable since technology evolution introduces complex concepts in programming languages. However, many of these approaches are based on theory or teacher's expertise. In addition, some even developed teaching models and used them locally to provide statistical results that prove their models. However, there is no evidence that these models are used anywhere else. Developers introduced ITS tools as an alternative approach to one-to-one teaching to reduce cost. However, these tools do not seem to kick off and there has been no sign of replacement to traditional teaching. Based on all above, we classified the research, on teaching programming, into 3 categories:

- Teaching Approach
  - Teaching approach is made of one or a set of methods. There methods collectively will help reach to successful teaching results. For example, Adopt LET US DO IT ALL TOGETHER approach, by Ngo-Ye and Park (2014), is made of several methods, i.e. pear programming, interactive exercises in classrooms and hands-on-teaching. An approach is defined by a teacher or authority and they decide on the methods that this approach is made of.
- Teaching Model
  - Teaching model is made of one or multiple approaches. In addition, it contains a process on how they should be implemented, a process on how feedback should be obtained and a process on how to integrate this feedback into the model for improvement. In other words, Teaching Approach is part of the Teaching Model For example, let's build on the 'Adopt LET US DO IT ALL TOGETHER' approach. If this method is used, there will also be some processes that explain how the method's approaches are implemented, feedback obtained from teachers and students and finally how feedback is integrated into the model
- Teaching Tool
  - It is basically a tool that is used to teach programming to students. This tool could sometimes be used to gather feedback from students. For example, online teaching tools such as SCRATCH.
  - A teaching tool on its own is not efficient as it usually only contains exercises, guidelines and sometimes animations to learners. However, if used as part of a model, then it becomes much more helpful to the teaching of programming languages.

2.1 Teaching Approach

With regards to teaching methods, Pears, Seidman, Malmi, Mannila, Adams, Bennedsen, Devlin and Paterson (2007) report that individual learning of programming is no longer efficient. Also, Sarpong, Arthur and Amoako (2013) seem to agree with this since they report that peer programming, peer tutoring and problem solving strategies are key to improve students' knowledge and interest. In addition, Ngo-Ye and Park (2014) report that hands-on approach has proved to be the best.

With respect to programming languages, Mannila & Raadt (2006) provide good evidence that Python and Eiffel are ideal languages to use when teaching programming to students. On the contrary, BRUSILOVSKY, CALABRESE, HVORECKY, KOUCHNIRENKO and MILLER (1997) believe that mini languages should be designed and used for teaching programming as scope of the languages will be more focused.

When it comes to human minds, MILNE and ROWE (2002) confirm that students learn better and more efficiently when they visualise objects. They believe that visual programming approach and tools are critical to teaching programming.

But, these approaches are not applied on a wide scale. In addition, some of these approaches could be combined to get an enhanced teaching approach for CS programming teaching. Table 1 is a summary of teaching approach.

2.2 Teaching Model

Researchers have spent time and energy designing what they claim different teaching models. Personally speaking, they all present differently and use interestingly varied terminology. However, the core is the same.

Horváth and Javorský (2013), Brito and Sá-Soares (2013), Vihavainen, Paksula and Luukkainen (2011) and Caspersen and Bennedsen (2007) all seem to present models that revolve around the following steps:

- Provide relevant teaching material
- Hands on teaching
- Group interaction
- Obtain feedback constantly
- Scaffold the feedback into the model

All these teaching models are only implemented in few universities or schools. There do not seem to be any serious impact of these models on a large scale. Table 2 presents the teaching models summary.

2.3 Teaching Tools

Unfortunately, there is not much literature on CS programming tools contrary to what tools one might find online. Apparently, there are tools that may be used for teaching multiple languages as Quinson and Oster (2014) claim. While, Soares (2014) describe a tool for teaching languages that can be used for novice and experienced students. However, these tools do not seem to be very popular or adopted on wide scale. Table 3 is the summary of the founded tools:

Table 1: Teaching Approach

Paper	Approach	Advantages
Ngo-Ye and Park (2014)	- Orient students with the aspects of programming - Adopt LET US DO IT ALL TOGETHER approach	Students show better appreciation and interest in learning programming
Sarpong, Arthur and Amoako (2013)	- peer programming, peer tutoring and problem solving strategies	Improve students' interest and knowledge which therefore will reflect positively on their performance
Pears, Seidman, Malmi, Mannila, Adams, Bennedsen, Devlin and Paterson (2007)	- Conduct a large-scale research to provide a better long-term understanding of how to teach programming to learners	Proves that individual learning of programming is no longer efficient
Mannila & Raadt (2006)	- Identified criteria for identifying and analysing ultimate languages for teaching programming	Python and Eiffel are more suitable for teaching programming as good care was taken during their design
Jenkins (2002)	- Programming must not be taught before University 2nd year -The language should be selected based on pedagogical suitability and not popularity in the industry - Only calibre teachers should teach programming - Programming courses should be flexible to all students to learn in different ways - No continuous assessment to ease pressure on students - Adequate support should always be available to students	Identify what institutions should do to refine their teaching approach to programming
MILNE and ROWE (2002)	- Visualization is essential to learning programming	It helps students understand what happens in memory when program executes
Warren (2001)	Students should be started with Scripting languages such as Java Script. Never with system programming languages such as Java or C++	Scripting languages are simpler and more flexible
Gal-Ezer and Zeldes (2000)	- Integration between the theoretical and practical aspects of programming - Theory should always be available and extensive; however teachers do not have to follow it to the letter - Competent teachers should be able to work out what level of depth should they go for when teaching students	Strongly helps students' understand how to design software solutions for algorithmic problems
BRUSILOVSKY, CALABRESE, HVORECKY, KOUCHNIRENKO and MILLER (1997)	- Design mini languages to use over general purpose languages for teaching programming	It will help control the behaviour and content of the software and ensure that students are learning exactly what they need
SHNEIDERMAN (1976)	Spiral approach of teaching involves: - The syntactic knowledge is constructed by frequent rehearsal and repletion which helps anchoring knowledge - The semantic knowledge is built by meaningful teaching in small partitions	Psychological principles are very critical and go in parallel with successful teaching programming for novice students

Table 2: Teaching Model

Paper	Model	Advantages
Horváth and Javorský (2013)	- Get first hand-feedback from students on the current Java course being taught using interviews - Modify the course structure immediately and roll it over in the syllabus straight away so that students can benefit	- Teachers' interpretation simplification - Give more time for problem analysis presented in class

	from it - This loop of review, modification and roll-out was done over three-year period on University students	- Continuously review curriculum - Creation of electronic learning materials - Giving more emphasis on the home work - Quick puzzles at the beginning of each class - Continuously encourage students to work together
Brito and Sá-Soares (2013)	- Students should always read the reference book first - Students should participate in the class - Students should try everything - Students should use the lab classes to self-assessment - Assessment - Tutorial guidance - Use surveys to gather feedback about the process from students	- Students achieve better results - Students are more aware of the course content and structure - Students drop-outs decrease significantly
Vihavainen, Paksula and Luukkainen (2011)	Extreme Apprenticeship model involves: - Learn by doing - Continuous feedback - Continuous practice - Avoid tons of preaching during the lectures - Relevant topics and exercises during lectures - Start exercises as early as the first lecture - Help in the labs should always be available in the labs by competent instructors - Small goal exercises - Exercises are mandatory - High amount of exercises and repetition - Exercises should have clean guidelines -Encourage students to look for information	- Feedback on what is being taught and scaffolding it straight into the teaching method is vital to the learning and teaching cycle of programming languages - Extreme Apprenticeship can be used as a model for teaching at Universities since it has been experimented and proved successful - Increase students' interest and motivation in learning programming.
Caspersen and Bennedsen (2007)	Based on psychological theories, i.e. cognitive load theory, cognitive apprenticeship and worked examples, the course model is organised into six phases: - Getting starter - Learning the basics - Conceptual framework and coding recipes - Programming method - Subject specific assignment - Practice	- The use of psychological theories is important to ensure good quality of learning and delivery.
Leutenegger and Edgington (2007)	- Teaching game programming course - Use surveys to gather data about student's knowledge and understanding of programming concepts, interest and level of satisfaction	- Using games to teach introductory programming to students increases their motivation and interest - It helps them learn the concepts of programming more effectively since they can visualise the mistakes they make in the code manifested in the resultant graphics
Keefe, Sheard and Dick (2006)	Extreme Programming (XP) practices to teaching OO programming to Students	Improves students' programming skills, however, their problem solving technique remains the same
Pillay (2003)	- Present and explain programming concepts - Provide different type of programming problems to students to test different programming aspects - Assist students to develop solutions - Automatically assessing programs written by students	Describes a generic architecture of how ITS can be used to develop viable intelligent teaching tools to programming students

	- Assessing students to debug programs for semantic errors	
Kanemune, Nakatani, Mitarai, Fukui and Kuno (2002)	Dolittle model which involves: - Simple syntax programming language - Incremental programming, where one line of code can do the job. Students do not have to write functions or classes which make code complicated -Text-based programming, which provide freedom and flexibility in writing code OO, where students can manipulate objects using instructions - Open expandability, where objects can be controlled remotely over networks - Use graphics to illustrate the impact of code changes on objects	Students enjoy their programming experience, achieve the tasks they are supposed to implement and find the overall concepts easy to follow and comprehend
Deek and McHugh (1998)	Most Teaching model(s) have the following issues: - Absence of problem solving software frameworks - Too much emphasis on language syntax - Inadequate user interfaces - Incomplete systems - Complex examples	Investment in developing intelligent teaching tools have not paid off since it cannot provide good problem solving comprehension techniques

Table 3: Teaching Tool

Paper	Tool	Advantages
Quinson and Oster (2014)	Programmer's Learning Machine (PLM): - It provides flexibility in designing lessons and grouping them - It provides flexibility in designing exercises and grouping them - It has a total available exercises are 12,000 - It integrates both the teaching and programming elements - 3 different languages can be used in this tool, i.e. Java, Python and Scala - Freely available project online	- Tool can be used for teaching multiple programming languages - It saves students time and energy in learning about different educational tools - Tool itself can be run on multiple environments, i.e. windows and linux
Soares (2014)	- Inventor App. is a teaching programming tool that has been developed by Google and currently hosted by MIT - It can be used to teach students how to develop Android applications	Tool is simple and flexible and can be used by both novice and non-novice students
Meerbaum-Salant, Armoni, & Ben-Ari (2013)	- Scratch is a visual programming environment that is used to teach CS programming concepts - Scratch holds a community of 1.5 members who use this environment for learning programming - Tool is made of scripts and those scripts are created by drag-drop of blocks that represent programming components such as expressions, conditions, etc - Tool eliminates syntax errors and gives immediate visual feedback. In other words, students do not have to write code to understand programming	- Students achieved good level of understanding of CS programming concepts - Some topics were still difficult to students, such as repeated execution and variables, however, there could be encountered with careful and more detailed teaching.

3. METHOD

Generally, there are few research methods available for use. A researcher can mix more than one method to achieve the research objectives. Decision, on which one to use, could be challenging. However, McNamara (2007) presents a very interesting comparison between all research methods available. His conclusion on the effectiveness of the research methods is illustrated in Figure 1.

Figure 1: Effectiveness of Research methods

Interestingly, this indicates that Meta-Analysis research method is highly recommended due to its proven effectiveness. In fact, this makes a lot of sense as synthesis of the results of others' relevant studies will help identify interference and gaps in the research subject area. But what is Meta-Analysis ?

It a methodology that is based on synthesising results from multiple studies to identify the impact it might have across the actual study. This will be illustrated in this work by:

- Synthesising the results of multiple teaching approaches, models and tools that have proven to be efficient in their own right- this shown in the tables 1,2 and 3 in 'Related work' part.
- Then, identifying similarity and interference between them and how that would impact the subject of research

In order to ensure quality and good research coverage, significant attention has been paid to the selection criteria of papers. Based on academic experience, the following list has been compiled:

- Only academic papers must be used.
- Mostly journal papers will be used. However, to avoid bias, some conference papers will be used too.
- The research must be relevant to the field.
- Papers should provide supporting evidence. Although, some papers have been chosen which have good arguments that are based on relevant experience of researchers
- Papers should have been cited by other researchers.

Based on the above list, there has been a limit on the number of papers available for this research subject. Unfortunately, the Teaching Tools papers have been the least available papers in literature.

A list of critique criteria has been compiled, which has been used to critique the papers. This list is a combination of several online resources, such as Wood (2003) and personal experience. The list is shown in Table 4.

Table 4: Research Critique Criteria

Critique steps	Answer(s)
Purpose of Author/Article	
Why does the researcher intend to do this research?	
What methods/techniques/approach has been used?	
Why were these methods used?	
How was each method performed?	
What data were obtained from those methods?	
Constraints encountered by Author?	
How author mitigated the constraints?	
Evidence of data from each method?	
What tool/environment has been used by the Author? And Why?	
How were data interpreted?	
What results were obtained from each method?	
Has the article met its purpose?	
Article's coverage?	
Article's usefulness?	
Article's bias?	

4. ANALYSIS

4.1 Discussion

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

Overall, the research done on the teaching models of programming languages has provided better quality literature than teaching approaches and tools. Tables 5, 6 and 7, in APPENDICES, clearly illustrate that. We believe that such a conclusion is fair since a teaching model involves approach(s) with method(s) that are applied strategically on a sample over some time. Then, obtaining results and feedback to assess the model and thus improve it. This is an ideal way, which has standards that are certainly higher than the other two categories.

- *Teaching Approach*

Most interestingly, it is evident from literature that most researchers, teachers and students are not aware of the major difference between the teaching of programming and the teaching of a programming language. This point is critical since the teaching of programming could be done using a simple language or script where the focus would be on the principles of programming and problem solving techniques but never on the actual syntax of the language. However, the teaching of a programming language would focus on the syntax and features of the language itself. Table 1, illustrates a detailed analysis of all the teaching approach literature.

What seems to be clear is that all approaches are incomplete and there is a lack of collective vision and coverage to encounter all teaching approaches issues. As a result, an enhanced approach is required to combine those several methods and languages together.

- *Teaching Model*

Interestingly, literature provides pretty much similar models with few methods variations. However, those methods are not complete or as not thorough at those provided in the teaching approach literature. Therefore, an enhanced model is required. Table 2, illustrates all the teaching approach literatures analysis.

- *Enhanced teaching tool*

Unfortunately there is no evidence of tool variety in literature. We believes that tools are usually developed for commercial reasons and this is why literature does not focus much on this. It is clearly seen that each model tries to automate the teaching of programming. This is simply done by providing exercises and assessments on each feature of the programming language. However, these tools can probably help teach the syntax and features of a language, but never able to develop the learner's problem solving skills. Table 3, illustrates a detailed analysis of all the teaching tool literature.

We believe that time and effort should be spent in developing tools that assess the performance of students and obtain feedback to enhance the teaching approach within the teaching model. In other words, an enhanced teaching tool is required too.

It is certain that further improvements to any research in this field should always involve discussion of constraints and how to mitigate them and evidence of data and their interpretation.

Finally, it is crystal clear that software development industry has not witnessed great deal of this research field and not benefited from it. We believe this is caused by the following factors:

- **No authority available to enforce particular practices and standards on how teaching models and tools should be used**
- **No official standards and guidance are available**
- **All successful results from tools and models only represent a small sample of people and is not representative of the learning community**
- **Most of the literature results does not provide real statistics to support their claims**
- **A lot of the tools and models are based on experience or on psychological learning theories. However, an integration between both is never seen anywhere**

So, what more can be done?

4.2 Proposed solution

In literature, there has been no collective solution towards a model that combines multiple proven approaches with a tool that provides exercises, content and feedback. Figure 2 clearly proposes an enhanced teaching model, which innovates than all other models by:

- Using a combination of proven teaching approaches to ensure perfection and completeness.
- Using a scaffolding approach which loops feedback, from students and staff, back into the approach and teaching content, then rolling it directly to the learning community.
- Using teaching tool to provide goal-focused exercises, assess students' performance and obtain feedback from the learning community. This tool would be integrated as part of the teaching model.

However, if the focus of the teaching approach is to teach a specific programming language, then a challenge rises to develop a tool that provides exercises, assessment and feedback for specific required programming language.

The proposed Enhanced Teaching Model (ETM) involves the following:

1. Approach identification

Based on the teaching objectives, the teaching authority will decide on one of the following approaches:

a) Teaching of programming

The sole focus of this approach is to teach programming concepts, i.e. OOP, etc. As a result, the teaching authority can either use traditional scripting languages, such as JavaScript, or specifically designed mini languages for teaching purposes.

b) Teaching of Specific language

The sole focus of this approach is to teach a specific programming language, i.e. Java, Python, etc.

2. Orient students with aspects of programming

This ensures students appreciation and interest in learning programming. This involves explaining:

a) Why programming is important and how it impacts our lives

b) The tools, IDEs and frameworks used

c) The overall cycle of software development

d) Applicability of course content in practice and industry

3. Apply teaching methods

Several teaching methods have been adopted from literature. The combination of these methods is the key of this enhanced model.

a) Peer Programming and Peer Teaching

Sarpong, Arthur and Amoako (2013) have proved that this method improves students' interest and knowledge, which therefore will reflect positively on their performance.

b) Problem Solving Techniques

There are many problem solving techniques available to use. This paper is not focused on naming a specific one. However, this method should focus on teaching the analysis and debugging techniques of requirements and issues.

c) Calibre Teachers

Having calibre teachers to teach programming is vital in delivering the teaching material to the students. In addition, it increases the students' interest in learning programming. **Jenkins (2002)** proved this in his model.

d) No Continuous Assessment

Students dread assessments. It adds more pressure on them and distracts them from the main goal of the course. **Jenkins (2002) a**rgues this case as he believes it eases the teaching process. The writers believe that reducing assessments and using the tool focused exercises to obtain indirect feedback on students' performance in the way forward.

e) Visualisation

MILNE and ROWE (2002) discuss how visual programming helps students in understanding what happens in memory when a program executes. In addition, visual programming makes it much more interesting and easier to grasp complex programming concepts.

f) Frequent Rehearsal

Practice makes perfection is a way of thinking about this. Also, **SHNEIDERMAN (1976)** proved how rehearsal anchors knowledge in students' minds during the teaching of programming.

g) Teaching in small partitions

This will keep the students interested and focused. In addition, this technique is significant when teaching complex concepts. **Vihavainen, Paksula and Luukkainen (2011)** and **SHNEIDERMAN (1976)** have all agreed on the importance of this method.

h) Intensive theory with focus on practice

The writers believe that theory is important and should be available extensively for students on the actual programming tool as a learning material. However, the main focus should be on practise during the lectures and labs as they are the ultimate way for learning programming.

i) Start Exercises as early as first lecture

The writers believe the traditional way of teaching programming is not appropriate anymore. Going on for weak teaching students theoretical concepts of programming is not right at all. In fact, the writers advocate for practical exercises to be used as early as the first lecture. In literature, many agree with this too. **Vihavainen, Paksula and Luukkainen (2011)** experimented this on a sample of students and the research results proved the importance of this method.

j) Exercises should be small goal

This method goes hand in hand with the 'Teaching in small partitions' method. Small exercises usually would focus on particular concept, and will eliminate any complexities that could divert the students' from the main exercise objective. **SHNEIDERMAN (1976)** is an advocate of this approach too and his research proves how positive this method on the teaching process.

k) Applying game programming

Game programming is the ultimate way for teaching programming. It is a method that keeps students not only interested, but also excited and joyful about programming. In literature, **Leutenegger and Edgington (2007)** provided evidence from this research that proves this too.Obtain and assess feedback

This involves using the actual teaching tool to obtain feedback directly from the students on the actual teaching material, exercises and delivery approach. Then, analyse the students' answers to the exercises as well as the feedback. In addition, feedback could be obtained from staff directly on how the students have performed.

4. Integrate feedback into teaching methods and tool

Finally, make decisive changes, which are required to modify the teaching methods or tool exercises in order to improve the quality of teaching and delivery to students. This should be a continuous and prompt process. The feedback, obtained from staff and students is essential to this ETM.

4.3 ETM in Practice

In practice, a teaching authority, such as a school, college, university and even a teacher, can use this model. ETM is flexible and adaptable. In other words, it should cater for all different teaching environments. A teaching authority should use the model in Figure (2) to decide on the approach, language and content of delivery. The tool can be designed locally or adopted from a successful ETM implementation. The key is to ensure that methods, feedback and scaffolding of feedback are all implemented.

4.4 What is Next

This paper has focused on identifying a model that promises to enhance the teaching of programming. Assessing this proposed model, i.e. ETM, will be the next step. A proposal is being drafted and will be submitted to the Public Authority for Applied Education, in Kuwait, to implement this model for two semesters, i.e. one academic year. It will be based on teaching the Java programming, which is a module taught for 1st year Computer Science students.

The writers are planning to keep detailed logs of the entire implementation process and will publish the final model and results at the end of the implementation stage.

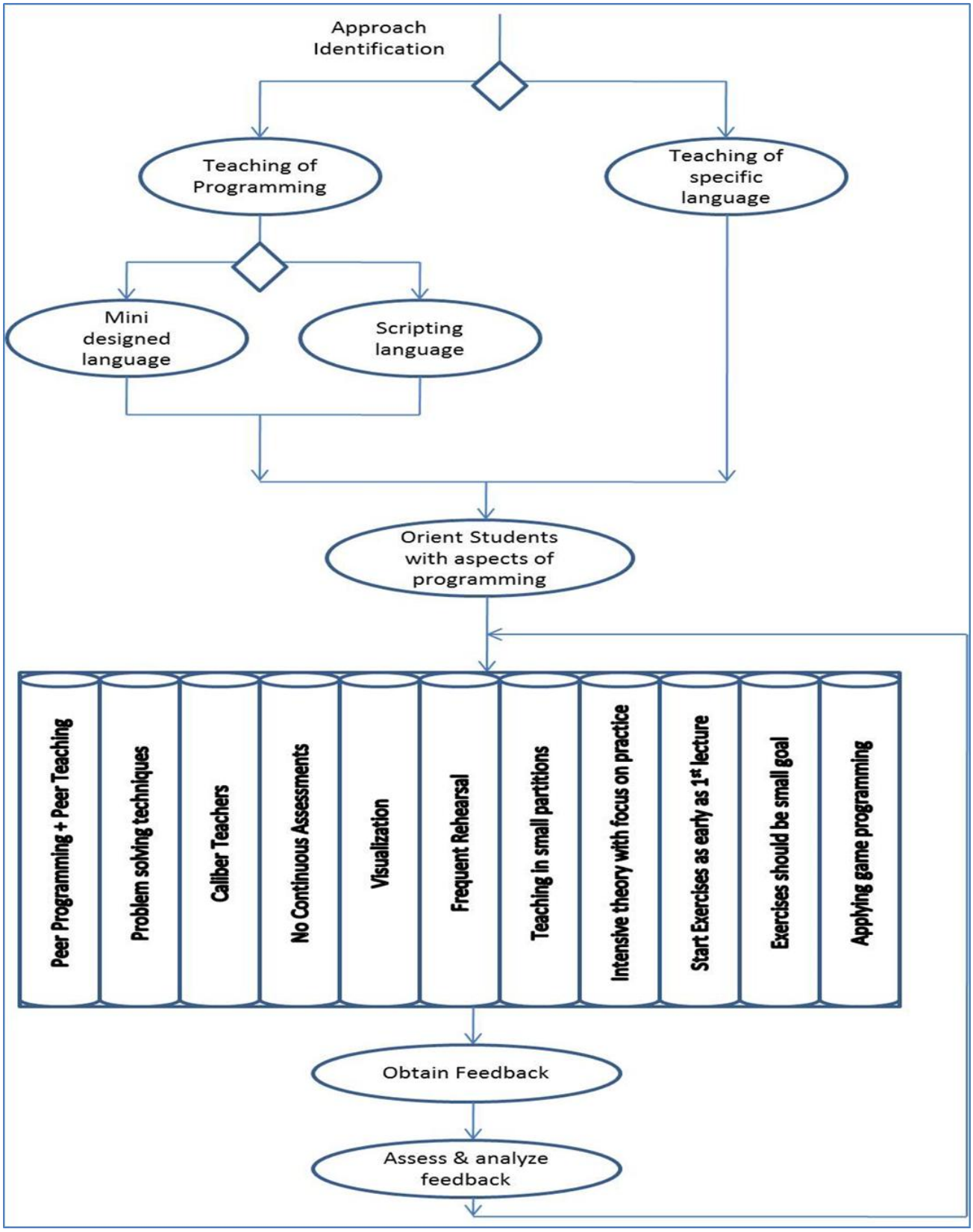

Figure 2: Proposed enhanced teaching programming model

The writers believe that the dynamic approach of scaffolding feedback into the teaching model is key toward ensuring flexibility and adaptability in future development of the ETM.

5. CONCLUSION

Teaching and learning programming is certainly challenging for both students and teachers. Literature shows that a lot of work has been done to improve this; however, it is evident that little effect of this work has had impact on the actual practice of teaching and learning of Software Development programming skills. Therefore, this gap has been addressed in this project to enhance the teaching and learning process of programming to students.

Meta-analysis research methodology has been used due to its effectiveness, i.e. McNamara (2007). In addition, to the identified specific criteria for literature paper selection and research critique. These criteria have enabled us to choose relevant literature and properly critique it. This reflected positively on the quality of research results obtained in this paper. We classified the teaching programming literature research into 3 categories: Teaching approach, Teaching model and Teaching tool

Moreover, it is lucid that the software development industry has not seen the fruition of this literature and not benefited from it. We believes this is caused by the following factors:

•No authority available to enforce particular practices and standards on how teaching models and tools should be used

•No official standards and guidance are available

•All successful results from tools and models only represent a small sample of people and is not representative of the learning community

•Most of the literature results does not provide statistics to support their claims and as a result appear unconvincing

•A lot of the tools and models are based on experience or on psychological learning theories. However, an integration between both is ever seen anywhere

As a result, the proposed Enhanced Teaching Model (ETM) combines several teaching approaches and models from literature. Then, proposes using teaching tools to provide goal-focused exercises, assess students' performance and obtain feedback from the learning community. However, if the focus of the teaching approach is to teach a specific programming language, then a challenge rises to develop a tool that provides exercises, assessment and feedback for specific required programming language.

Finally, a proposal is being drafted to the Public Authority for Applied Education, in Kuwait, to allow the implementation and assessment of this model.

6. ACKNOWLEDGMENTS

Our thanks to the experts who have contributed towards development of the template.

7. REFERENCES

[1] Kori, Külli, et al. (2015): The Role of Programming Experience in ICT Students' Learning Motivation and Academic Achievement. International Journal of Information and Education Technology Vol. 6.

[2] T. Hüsing et al. (2013). E-leadership, e-skills for competitiveness and innovation vision, roadmap and foresight scenarios final report. European Commision E-Skills Vision.

[3] Ford, M. and Venema, S. (2010): Assessing the Success of an Introductory Programming Course. Journal of Information Technology Education 9:133-145.

[4] Thomas, L., Ratcliffe, M., Woodbury, J., Jarman, E.(2002): Learning styles and performance in the introductory programming sequence . SIGCSE '02 Proceedings of the 33rd SIGCSE technical symposium on Computer science education

[5] Bornat, R., Dehnadi, S., and Simon (2008): Mental models, consistency and programming aptitude. ACE '08: Proceedings of the tenth conference on Australasian computing education Vol. 78.

[6] Sleeman, D. (1986). The Challenges of Teaching Computer Programming. Communications of the ACM. 29 (9). p.840-841

[7] Blayney, P. J. (2009). Knowledge gap? Accounting practitioners lacking computer programming concepts as essential knowledge. In G. Siemens and C. Fulford (Ed.), Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (pp. 151-159). Chesapeake, VA: AACE.

[8] Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., Nunamaker, J. F., Sipior & J. C., et al. (2010). IS 2010: Curriculum guidelines for undergraduate degree programs in information systems. Communications of the Association for Information Systems, 26, 359-428

[9] Ramalingam, V., LaBelle, D. & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. Association for Computing Machinery SIGCSE Bulletin, 36(3), 171-175

[10] Gonzalez, G. (2004). Constructivism in an introduction to programming course. Journal of Computing Sciences in Colleges, 19(4), 299-303.

[11] Robins, A., Rountree, J. & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer Science Education, 13(2), 137-172

[12] ISMAIL, M. N., NGAH, N. A. & UMAR, I. N. (2010). Instructional strategy in the teaching of computer programming: a need assessment analyses. TOJET: The Turkish Online Journal of Educational Technology, volume 9 Issue 2, pp. 569–571

[13] Butler, M. & Morgan, M. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. Proceedings ascilite Singapore, pp. 99 – 107

[14] Smith, P. A. & Webb, G. I. (2000). The Efficacy of a Low-Level Program Visualisation Tool for Teaching Programming Concepts to Novice C Programmers. Journal of Educational Computing Research, 22 (2), 27–39

[15] Boroni, C. M.et al. (1996). Dancing with Dynalab. In Proceedings of the 27th SIGCSE Technical Symposium on CS Education. Philadelphia, February. pp. 135–139

[16] Rowe, G. R. (2000). VINCE— An on-line tutorial tool for teaching introductory programming. British Journal of Educational Technology, 31(4), 359–369

[17] Fleury, A. E. (1993). Students' beliefs about Pascal programming. Journal of Educational Computing Research, 9(3), 355–371

[18] Scheftic, C. & Goldenson, D. (1986). Teaching programming methods and problem solving: the role of programming environments based on structure editors. In Proceedings of the National Educational Computing Conference, pp.231–6

[19] Brusilovsky, P. (1994). Program visualisation as a debugging tool for novices. In Proceedings of INTERCHI '93 (Adjunct proceedings) Amsterdam, 24 – 9April 1993, pp. 29–30

[20] Ranjeeth, S. & Naidoo, R. (2007). An investigation into the relationship between the level of cognitive maturity and the types of errors made by students in a computer programming course. College Teaching Methods and Styles Journal, 3, 31–40

[21] Adams, J.C. (1998). Chance-it: an OO capstone project for cs-1, SIGCSE'98, 10-14

[22] Becker, K. (2001). Teaching with games: the minesweeper and asteroids experience, J. Comput. Small Coll, 17(2), 23-33

[23] Lorenzen, T. & Heilman, W. (2002). Cs1 and cs2: write computer games in java! SIGCSE Bull., 34(4), 99-100

[24] Trono, J.A. (1994). Taxman revisited, SIGCSE Bull., 26(4), 56-58

[25] Beck, K. (1999). Embracing change with extreme programming. Computer, 32(10), 70-77

[26] Hedin, G., Bendix, L., & Magnusson, B. (2003).Introducing software engineering by means of Extreme Programming, Proceedings of the 25th International conference on Software Engineering (pp. 586-593). Portland, Oregon: IEEE Computer Society

[27] Mills, G. (2000). Action research: a guide for the teacher researcher. Upper Saddle River, New Jersey:Prentice-Hall

[28] Williamsons, K., Burstein, F., & McKemmish (2002).The two major traditions of research. In K. Williamsons (Ed.), Research methods for students, academics and professionals (2nd ed.). Wagga Wagga, New South Wales: Centre for Information Studies, Charles Sturt University

[29] Odekirk, E. (2000). "Analysing Student Programs, in SIGCSE Bulletin: Conference Proceedings of the 5th Annual SIGCSE/SIGUE Conference on Innovation and Technology in Computer Science Education", ITiCSE 2000, Vol. 32, No. 3, pg. 191-191, ACM Press

[30] Mayer, R.E. (1981). The psychology of how novices learn computer programming. ACM Computing Surveys, 3, 121–141

[31] Perkins, D.N., Hancock, C., Hobbs, R., Martin, F. & Simmons, R. (1986). Conditions of learning in novice programmers. Journal of Educational Computing research, 2, 37–56

[32] Linn, M.C. & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, access, and ability. Educational Psychologist, 20, 191–206

[33] Perkins, D.N., Schwartz, S. & Simmons, R. (1988). Instructional strategies for the problems of novice programmers. In R.E. Mayer (Ed.), Teaching and learning computer programming (pp. 153–178). Hillsdale, NJ: Lawrence Erlbaum

[34] Lippert, R.C. (1989). Expert systems: Tutors, tools, and tutees. Journal of Computer-Based Instruction, 16, 11–19

[35] Ramadhan, H. (1992). An intelligent discovery programming system. In Proceedings of ACM symposium on applied computing: Special track on visuality in computing. Kansas City, KS

[36] Rosenberg, R. (1987). A critical analysis of research on intelligent tutoring systems. Journal of Educational Technology, 27, 7–13

[37] Mannila, L. & de Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. In Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006 (pp. 32-37). ACM

[38] McIver, L., & Conway, D. (1996). Seven deadly sins of introductory programming language design. In Software Engineering: Education and Practice, 1996 Proceedings. International Conference (pp. 309-316). IEEE

[39] Soares, A. (2014). Reflections on Teaching App Inventor for Non-Beginner Programmers: Issues, Challenges and Opportunities. Information Systems Education Journal, 12(4), 56

[40] Tyler, J. (2011). App Inventor for Android: Build Your Own Apps - No Experience Required! : Wiley Publishing

[41] McNamara, C. (2007) Types of Research Methods. SERVE Center

[42] Quinson, M. & Oster, G. (2014). The Programmer's Learning Machine: A Teaching System To Learn Programming. Loria

[43] Lethbridge, T. (2014). Teaching Modeling using Umple: Principles for the Development of an Effective Tool. Software Engineering Education and Training (CSEE&T)

[44] Ngo-Ye, T. & Park, S. (2014) MOTIVATING BUSINESS MAJOR STUDENTS TO LEARN COMPUTER PROGRAMMING – A CASE STUDY. Proceedings of the Southern Association for Information Systems Conference. AIS Electronic Library

[45] Horváth, R. & Javorský, S. (2013). New Teaching Model for Java Programming Subjects. 5th World Conference on Educational Sciences. 116. p. 5188–5193. ScienceDirect

[46] Sarpong, K., Arthur, J. & Amoako, P. (2013). Causes of Failure of Students in Computer Programming Courses: The Teacher – Learner Perspective. International Journal of Computer Applications. 77 (12). p.0975–8887. IJCA Journal

[47] Brito, M. & Sá-Soares, F. (2013). Assessment frequency in introductory computer programming disciplines.Computers in Human Behaviour. 30. p.623–628. Science Direct

[48] Lethbridge, T., Mussbacher, G. & Forward, A. & Badreddin, O. (2011). Teaching UML Using Umple:Applying Model-Oriented Programming in the Classroom.Software Engineering Education and Training (CSEE&T). p.421-428. IEEE

[49] Vihavainen, A., Paksula, M. & Luukkainen, M. (2011). Extreme Apprenticeship Method in Teaching Programming for Beginners.Proceedings of the 42nd ACM technical symposium on Computer science education. p.93-98. ACM Digital Library

[50] Caspersen, M. & Bennedsen, J (2007). Instructional Design of a Programming Course −A Learning Theoretic Approach. Proceedings of the third international workshop on Computing education research. p. 111-122. ACM Digital Library

[51] Pears, A., Seidman, S. & Malmi, L. & Mannila, L. & Adams, E. & Bennedsen, J. & Devlin, M. & Paterson, J. (2007). A Survey of Literature on the Teaching of Introductory Programming. Working group reports on ITiCSE on Innovation and technology in computer science education.p.204-223. ACM Digital Library

[52] Leutenegger, S. & Edgington, J. (2007). A Games First Approach to Teaching Introductory Programming. Proceedings of the 38th SIGCSE technical symposium on Computer science education.p.115-118. ACM Digital Library

[53] Keefe, K., Sheard, J. & Dick, M. (2006). Adopting XPPractices for Teaching Object Oriented Programming. Proceedings of the 8th Australasian Conference on Computing Education. 52. p.91-100. ACM Digital Library

[54] Pillay, N. (2003). Developing Intelligent Programming Tutors for Novice Programmers. ACM SIGCSE Bulletin. 35 (2). p.78-82. ACM Digital Library

[55] Grimes, D. A. & Schulz, K. F. (2002). Bias and causal associations in observational research. The Lancet, 359(9302), 248-252.

[56] Jenkins, T. (2002). ON THE DIFFICULTY OF LEARNING TO PROGRAM. 3rd Annual LTSN-ICS Conference. University of Ulster, LTSN Centre for Information and Computer Sciences

[57] Kanemune, S., Nakatani, T., Mitarai, R., Fukui, S. & Kuno, Y. (2002). Dolittle — Experiences in Teaching Programming at K12 Schools. The Second International Conference on Creating, Connecting and Collaborating through Computing. p.177-184. IEEE

[58] MILNE, I. & ROWE, G. (2002). Difficulties in Learning and Teaching Programming—Views of Students and Tutors. Education and Information Technologies. 7 (1). p. 55-66. Springer Link

[59] Warren, P. (2001). TEACHING PROGRAMMING USING SCRIPTING LANGUAGES. Journal of Computing Sciences in Colleges. 7 (2). p.205-216. ACM Digital Library

[60] Gal-Ezer, J. & Zeldes, A. (2000). Teaching Software Designing Skills. Computer Science Education. 10 (1). p.25-38. Taylor Francis Online

[61] Deek, F. & McHugh, J. (1998). A Survey and Critical Analysis of Tools for Learning Programming. Computer Science Education. 8 (2). p.130-178. Taylor Francis Online

[62] BRUSILOVSKY, P., CALABRESE, E. & HVORECKY, J. & KOUCHNIRENKO, A. & MILLER, P. (1997). Mini-languages: a way to learn programming principles. Education and Information Technologies. 2 (1). p.65-83. Springer Link

[63] SHNEIDERMAN, B. (1976). TEACHING PROGRAMMING: A SPIRAL APPROACH TO SYNTAX AND SEMANTICS. Computer and Education. 1 (4). p.193-197. Science Direct

[64] Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. Computer Science Education, 23(3), 239-264

[65] Wolz, U., Leitner, H. H., Malan, D. J., & Maloney, J. (2009). Starting with Scratch in CS1.SIGCSE Bulletin, 41,2–3

[66] Maloney, J., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch.SIGCSE Bulletin, 40, 367–371

[67] Wood, J.M. (2003).Research Lab Guide. MICR*3260 Microbial Adaptation and Development Web Site. [Online] Available from: http://www.uoguelph.ca/mcb/teaching/micr3260/research_lab/guide.shtml. [Accessed: 22nd Jan 2015]