Chapter 3
SCALE: A Competence Analytics Framework

David Boulanger, Jérémie Seanosky, Colin Pinnell, Jason Bell, Vivekanandan Kumar and Kinshuk

Abstract This paper introduces SCALE, a Smart Competence Analytics engine on LEarning, as a framework to implement content analysis in several learning domains and provide mechanisms to define proficiency and confidence metrics. SCALE's ontological design plays a crucial role in centralizing and homogenizing disparate data from domain-specific parsers and ultimately from several learning domains. This paper shows how SCALE has been applied in the programming domain and reveals systematically how the work content of a student can be analyzed and converted to evidences to assess his/her proficiency in domain-specific competences and how SCALE can also analyze the student's interaction with a learning activity and provide a confidence metric to assess his/her behavior as he/she culminates toward goal achievements.

Keywords SCALE · Competence · Proficiency · Confidence · Learning analytics · Ontological design

3.1 Introduction

This paper proposes an evidence-based competence analytics framework called Smart Competence Analytics engine on LEarning (SCALE). The goal of this research is to feature an approach to competence-based learning applicable to any learning domain. SCALE is mainly concerned with content analysis, one of the key outcomes of learning analytics. However, content analysis is highly domain-specific. SCALE, therefore, through its ontological design attempts to reduce the customization effort and time to a minimum. Although SCALE does not intend to define proficiency and confidence metrics, it aims at providing teachers with the mechanisms to implement their own definitions of proficiency and confidence.

D. Boulanger (✉) · J. Seanosky · C. Pinnell · J. Bell · V. Kumar · Kinshuk
School of Computing and Information Systems, Athabasca University, Athabasca, Canada
e-mail: david.boulanger@athabascau.ca

Y. Li et al. (eds.), *State-of-the-Art and Future Directions of Smart Learning*,
Lecture Notes in Educational Technology, DOI 10.1007/978-981-287-868-7_3

Thus, SCALE allows real-time customization and calibration of proficiency and confidence models. This paper strives to show the applicability of the framework in any learning domain by applying it in Java programming.

Learning analytics is a study of context-aggregate and context-precise insights based on observed learning experiences that are continually validating in nature where a learning trace comprises of an instantiated network of models that lead to a measurable chunk of learning. In this setting, SCALE provides an important subset of observation data to instantiate those models in order to find causal relationships between those models' variables through a propensity score matching observational study design that will be implemented in a future version.

In the next section, a brief literature review highlights the progress made by other researchers in the employ of ontologies to track by means of e-learning technologies students' competences in programming (i.e., Java). The third section gives a high-level view of SCALE's architecture, while the fourth section shows how SCALE works in the programming domain. The paper concludes by outlining the next steps for a future full-fledged experiment with SCALE.

3.2 Literature Review

Hosseini et al. [1] conducted a study to look at how Java programs are developed. By means of the JavaParser tool, they succeeded to collect unique datasets highlighting the intermediate programming steps through series of snapshots showing how students developed their programs over time. Hosseini et al. performed a fine-grained concept-based analysis on each step to identify the most common programming paths. The results of their experiments showed that students tend to develop and debug their programs incrementally and that tracking intermediate programming steps provide invaluable opportunities to give better feedback to students. Their experiment was conducted in the Java programming domain and used the JavaParser tool "to extract a list of ontological concepts from source code using a Java ontology[1] developed by PAWS laboratory." Those concepts were extracted for each code capture.

Sosnovsky et al. [2] uphold that precise student modeling is key in the effectiveness of adaptive educational systems. Student modeling requires assessing adequately the background knowledge of students before they start using such adaptive systems. Even though this information may be available from other systems, discrepancies most often occur due to different knowledge representation, system architecture, and modeling constraints. Sosnovsky et al. argue "that the implementation of underlying knowledge models in a sharable format, as domain ontologies—along with application of automatic ontology mapping techniques for model alignment—can help to overcome the 'new-user' problem and will greatly widen opportunities for student model translation."

[1] http://www.sis.pitt.edu/~paws/ont/java.owl.

Finally, Ganapathi et al. [3] propose an approach to the practical ontology development and present their approach by designing an ontology to teach Java programming. Researchers in [4–7] rather focus on the identification of common misconceptions and pitfalls in learning programming, while Sampson [8] highlights the role that competence-based learning can play to implement the lifelong learning paradigm.

3.3 SCALE Architecture

SCALE consists basically of three processing layers: parsing, inferencing, and profiling. Moreover, SCALE is built on three types of ontologies: interaction, contents, and learning trace. All learning events collected from a single student engaged in a single learning activity coming from a single learning domain will be parsed and stored in an interaction ontology. The contents ontology models the learning materials, defines learning objectives, and models and maps the skills expected from learners to the learning objectives. The learning trace ontology records measurable subsets of observations into a skill/proficiency with respect to a mapping function. Hence, the mapping between the learning outcomes in the contents ontology and their expected datasets in the learning trace ontology will define a course's expected learning outcomes in terms of proficiency levels in targeted skills.

The first processing layer in SCALE involves the expansion of the student's dataset through domain-specific parsers. For example, analyzing a student's work would imply submitting its work to a series of domain-specific parsers to extract the structure that gives meaning to the student's work. The parsing results are then ontologized and linked to the appropriate captures and constructs extracted from the student's work. Thus, the parsing layer in SCALE requires the integration of domain-specific parsers, the definition of the entities and relationships defining the domain in question, and the specification of the resulting interaction ontology in a universal data format that is an RDF ontology. It is important to highlight the fact that this process can be applied in any domain.

In the second layer of processing, SCALE identifies students' learning artifacts and builds sets of evidences to assess students' proficiency levels in domain-specific competences. SCALE performs competence assessment using pattern matching techniques (domain-specific production rules). SCALE encloses a core set of production rules corresponding to a core set of competences in the learning domain at hand and enables teachers to enhance or customize that set to address the requirements of their courses. The key technology employed by SCALE is called BaseVISor. BaseVISor is an ontological rule-based reasoning engine which extends the system's capability from mere recognition of syntactic and semantic elements in the student's work to the tracking and recognition of creative, design, and logic elements in it. BaseVISor's compact and intuitive XML syntax requires a lower level of expertise to code the production rules. Moreover, since the

rules employ the vocabulary defined in SCALE's ontologies, they can be easily written in pseudo-code and implemented by a programmer who is not an expert in the learning domain.

In the third processing layer, SCALE profiles students individually and collectively and provides students with a formative feedback that shows the progression of their proficiency and confidence at different levels of granularity (i.e., learning activity, competence, learning domain).

3.4 SCALE in Programming

This section will give an example of how SCALE can be applied in a specific learning domain. Although at the very moment of this writing SCALE is being customized and applied in English writing and mathematics, this paper will demonstrate the overall process in the programming domain where SCALE has been developed, tested through pilots, and optimized for the first time. As the analytics engine of an overall learning analytics platform, SCALE will receive data from a student enrolled in a Java programming course who is solving an assignment problem using the NetBeans IDE installed with the CODEX plug-in (a plug-in designed along with SCALE to capture the data required for competence analytics) to sense the student's work and transmit it as a learning event to SCALE. CODEX generates such learning events at a regular time interval or every time the student compiles his/her program. Figure 3.1 shows a learning event sensed by CODEX.

In order to be processed by SCALE, data must follow the format of the learning event above. This data packet will first go through the parsing layer. As previously asserted, the content of this packet will be analyzed by a series of domain-specific

StudentID	davidb
Timestamp	1428530086781 (Wed Apr 08 17:55:19 EDT 2015)
Trigger	Timer
FilePath	C:\Users\...\...\NetBeansProjects\Factorial\src\factorial\Factorial.java

Code	`package factorial;`
	`public class Factorial {`
	`  public static void main(String[] args) {`
	`  int num = 8`
	`  System.out.println(""+num+"! = "+factorial(num));`
	`  }`
	`  public static int factorial(int i) {`
	`  if (i == 0) return 1;`
	`  else return i * factorial(i-1);`
	`  }`
	`}`

Fig. 3.1 A learning event generated from the NetBeans CODEX plug-in

parsers. In the current programming domain, the content is parsed by the open source Eclipse JDT compiler (which can be handled programmatically), the single parser being tested as of this writing. The compiler breaks down the code contained in the learning event into an abstract syntax tree and reports about compile-time errors. The parsing results are then ontologized programmatically using the Apache Jena library. The abstract syntax tree along with its inherent connections and dependencies is preserved in the ontology. Figure 3.2 shows a part of the interaction ontology generated from the above learning event. The figure shows how the child/parent relationship preserves the tree data structure of the abstract syntax tree and how each node within the tree is stored along with its properties. The ontology records also the number of compile-time errors pertaining to the current piece of code. Although the most relevant pieces of information are generated in the default configuration of the Eclipse JDT compiler, the Eclipse JDT compiler's configuration could be customized to populate further the interaction ontology in order to enhance SCALE's inferencing capability. The ontologies are written in the RDF/XML format as required by BaseVISor.

The resulting interaction ontology, the output of the parsing layer, will be input in the inferencing layer for the identification of evidences that will prove the student's proficiency in a specific skill. As are the parsers in the parsing layer domain-specific, production rules that define the patterns to recognize proficiency facts will also be domain-specific. The production rules will comply with the set of competences which are mapped to the learning objectives encapsulated in the course's contents ontology. In this first application of SCALE, production rules have been written to match the control structure-related competences as found in the Java OWL ontology (see footnote 1) designed by PAWS laboratory. Proficiency facts recognized by the inferencing layer will be linked to one or more control structure competences. SCALE will also provide a set of core production rules which could be edited or customized by teachers and students at will. New production rules could also be added to enhance the number of competences to be tracked and to increase the ability of the system to track all the flavors or different

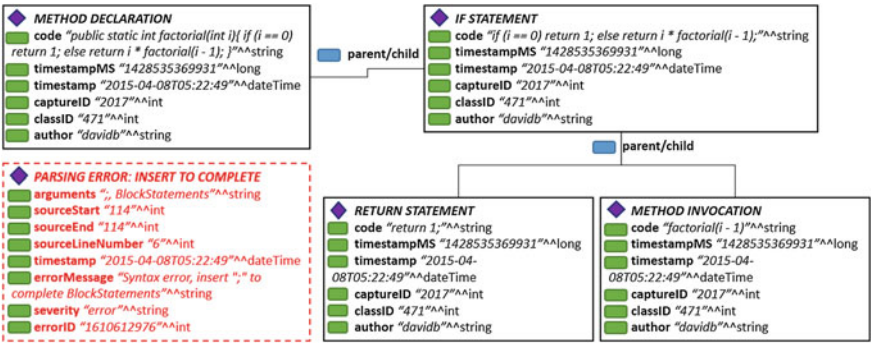


Fig. 3.2 A sample interaction ontology generated from parsing learning event in Fig. 3.1

```
1  <rule name="Recursion">
2      <body>
3          <Individual variable="Var1" rdf:type="interaction:MethodDeclaration">
4              <interaction:code variable="Proof"/>
5              <interaction:author variable="Username"/>
6              <interaction:classID variable="ClassID"/>
7              <interaction:captureID variable="CaptureID"/>
8              <interaction:timestampMS variable="TimestampMS"/>
9              <interaction:timestamp variable="Timestamp"/></Individual>
10         <Individual variable="Var2" rdf:type="interaction:SimpleName">
11             <interaction:child variable="Var1"/>
12             <interaction:code variable="MethodName"/></Individual>
13         <Individual variable="Var3" rdf:type="interaction:Block">
14             <interaction:child variable="Var1"/></Individual>
15         <Individual variable="Var4" rdf:type="interaction:ReturnStatement">
16             <interaction:child variable="Var3"/></Individual>
17         <Individual variable="Var5" rdf:type="interaction:InfixExpression">
18             <interaction:child variable="Var4"/></Individual>
19         <Individual variable="Var6" rdf:type="interaction:ParenthesizedExpression">
20             <interaction:child variable="Var5"/></Individual>
21         <Individual variable="Var7" rdf:type="interaction:MethodInvocation">
22             <interaction:child variable="Var6"/></Individual>
23         <Individual variable="Var8" rdf:type="interaction:SimpleName">
24             <interaction:child variable="Var7"/>
25             <interaction:code variable="MethodName"/></Individual>
26     </body>
27     <head>
28         <assert>
29             <Individual variable="Competence" rdf:type="interaction:Recursion">
30                 <interaction:timestamp variable="Timestamp"/>
31                 <interaction:fact variable="Proof"/></Individual>
32         </assert>
33         <updateProficiency>
34             <param>Recursion</param><param variable="TimestampMS"/>
35             <param variable="Proof"/><param variable="Username"/><param variable="ClassID"/>
36             <param variable="CaptureID"/><param>1</param>
37         </updateProficiency>
38     </head>
39 </rule>
```

Fig. 3.3 The production rule identifying basic cases of recursion in BaseVISor XML syntax

occurrences of a competence or skill. Figure 3.3 shows a production rule identifying a simple application of recursion in the student's code (although not related to control structures).

The reader should note that a production rule is the equivalent of an if/then statement. The "if" part is enclosed in the < body > tags (Lines 2–26) and the "then" part within the < head > tags (Lines 27–38). The "if" part of this recursion rule specifies the sequences and characteristics of programming constructs to identify a case of basic recursion. For example, the rule is looking for a method declaration (Lines 3–9) and stores all its properties into a set of user-defined variables. Basically, this rule checks whether the method declaration is self-invocating (Lines 21–25). As the result of the occurrence of a simple case of recursion, an evidence or fact will be asserted in the ontological model to prove the student's proficiency in that competence (Lines 28–32). Lines 33–37 call a user-defined Java function to interface and store the inferencing results into the SQL and NoSQL world. The parameters from Lines 34–36 pass data from the ontological model to the Java function for further processing or data persistency. The first and last parameters indicate the competence to which this fact corresponds and the weight of the fact

which will influence by how much the proficiency value will change according to the mathematical equation defining the competence. As for any production rule, this particular production rule will be fired for each instance of recursion found in a student's program.

The profiling layer will update the proficiency values for those skills the student has showed proper usage and will also update the confidence value of the student in the current learning activity and in the overall programming domain. Table 3.1 lists all the evidences identified from the last learning event. This learning event represents the most recent state of the corresponding learning activity (Factorial.java) recorded by SCALE.

As the next step, SCALE will retrieve all the evidences from the most recent captures of all learning activities it has so far tracked and will sum them up by competence as it can be found in Table 3.2. For example, it can be seen that for student davidb, there have been 12 cases of recursion tracked by SCALE. Using the proficiency function of the Recursion competence, it can be seen that the proficiency level of the student at using recursion is calculated to be 32.61 %. Proficiency and confidence values are always expressed as probabilities, that is, as values between 0 and 1, exclusively (0, 1). It is also interesting to note that teachers can then set a threshold for each competence to determine the learning objective in regard to that competence within their course(s). Moreover, the reader may notice that the StatementBlock and the IfElseStatement competences have proficiency levels very close to each other, while the former competence has twice the number of facts of the later competence. Given the nature of the math (proficiency) functions in Table 3.2 whose ranges lie between 0 and 1 exclusively, capturing the number of facts for each competence will also provide a measure of the persistency of a skill. In other words, it may determine in some way the decay speed of the skill if it was to be unused for a period of time or if misconception or misusage was to be detected.

SCALE also tracks the confidence of students as they work in learning activities. SCALE monitors the student's behavior that is his/her interaction with a learning tool and its learning contents. SCALE provides the mechanism to assess students' confidence both in a learning domain and in a specific activity. Again, SCALE will allow teachers and students to create or customize their own confidence model. Although some factors in the confidence model may be domain-generic such as the time taken to solve a problem, most factors will be domain-specific and will need to be tailored and customized according to the learning domain at hand. Thus, in the current learning domain, that is programming, SCALE provides a predefined confidence model consisting basically of four factors: time spent on a learning activity (writing a program), number of times the student compiles his/her code in a specific learning activity, the average number of errors per compilation for a specific learning activity, and the persistency of those errors throughout a learning activity. SCALE applies the same factors to compute the overall confidence of a student in a learning domain.

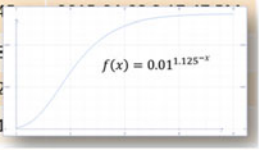
The rationale behind the calculation of a confidence value is that each factor is assessed individually according to the ideal scenario or behavior expected from a

Table 3.1 List of proficiency facts generated from the learning event in Fig. 3.1

FactID	CompetenceName	Username	ClassID	CaptureID	Timestamp	CodeSnippet (proof)	Weight
44894	StatementBlock	davidb	471	2019	2015-04-08 8:17:47 PM	{int num = 8; System.out.println (“‘ + num +”! = ” + factorial(num));}	1
44895	IfStatement	davidb	471	2019	2015-04-08 8:17:47 PM	if(i == 0) return 1; else return i * factorial(i − 1);	1
44896	IfElseStatement	davidb	471	2019	2015-04-08 8:17:47 PM	if(i ==0) return 1; else return i * factorial(i − 1);	1
44897	StatementBlock	davidb	471	2019	2015-04-08 8:17:47 PM	{if(i == 0) return 1; else return i * factorial (i − 1);}	1
44898	ReturnStatement	davidb	471	2019	2015-04-08 8:17:47 PM	return 1;	1
44899	EqualExpression	davidb	471	2019	2015-04-08 8:17:47 PM	i ==0	1
44900	ReturnStatement	davidb	471	2019	2015-04-08 8:17:47 PM	return i * factorial(i − 1);	1
44901	Recursion	davidb	471	2019	2015-04-08 8:17:47 PM	public static int factorial(int i) {if (i == 0) return 1; else return i* factorial(i − 1);}	1

Table 3.2 Numbers of facts for each competence that will be input in the math function as pictured in the bottom right corner along with corresponding output proficiency levels

Username	CompetenceName	Facts x	Equation $f(x)$	Proficiency	Timestamp
davidb	StatementBlock	121	$f(x) = 0.01^{1.125^{-x}}$	0.999997	2015-04-08 8:17:47 PM
davidb	IfStatement	88	$f(x) = 0.01^{1.125^{-x}}$	0.999854	2015-04-08 8:17:47 PM
davidb	IfElseStatement	63	$f(x) = 0.01^{1.125^{-x}}$	0.99724	
davidb	ReturnStatement	45	$f(x) = 0.01^{1.125^{-x}}$	0.97728	
davidb	EqualStatement	31	$f(x) = 0.01^{1.125^{-x}}$	0.88732	
davidb	Recursion	12	$f(x) = 0.01^{1.125^{-x}}$	0.32611	



very proficient student. For example, Student A and Student B have solved the same problem, have both written an excellent piece of software, and have made the same errors with the same level of persistency. However, Student A has taken twice the time of Student B. Hence, naturally, it will be expected that Student B exhibits more confidence than Student A in regard to time. It may, therefore, be inferred that taking less time to solve a problem to meet the requirements of an activity is more desirable than the contrary. The same may be true for the number of compilations a student makes. Similar conclusions could be drawn for each factor in the confidence model.

After having defined the factors making up a confidence model, one could define a threshold for each factor beyond which the confidence of a student in regard to that particular factor would start degrading more significantly. Obviously, the threshold will depend also on the characteristics of the mathematical equation used to model the confidence decay. For example, the mathematical functions defining the confidence factors time and compilations are both quadratic equations. The point here is that SCALE does not want to penalize too much students within an acceptable range of time and number of compilations made when solving a problem. There exists some trade-off between these factors. For instance, a student may decide to compile his/her code more often to reduce the time he/she takes to solve a problem or to correct errors as soon as possible to reduce the number of errors he/she will make in a problem.

Table 3.3 shows the confidence level for each factor and as a whole for the student who has sent the learning event in Fig. 3.1. Figures 3.4, 3.5, 3.6, and 3.7 display the graphs of the mathematical equations determining the value for each confidence factor. Figure 3.4 shows that the input of the equation is the amount of time in terms of minutes. The input in Fig. 3.5 is the total number of compilations made in the learning activity. The input in Fig. 3.6 is the average number of errors per compilation, while the input in Fig. 3.7 is the standard deviation of the numbers of error messages generated for all compilations for that learning activity.

Finally, the confidence measure assigned to that specific learning activity is calculated as the product of the confidence factor values. The ranges of all confidence measurements lie between 0 and 1, inclusively [0, 1]. Hence, SCALE

Table 3.3 Confidence factors along with associated raw data and derived confidence values

Confidence factors	Raw data	Confidence factor values
Duration	720,000 ms = 12 min	0.999856
Number of compilations	3	0.994
Average number of errors	0.6666666666666666	0.9916666666666667
Standard deviation of numbers of errors	0.4841229182759271	0.9096824583655185
Confidence	N/A	0.8965600373460734

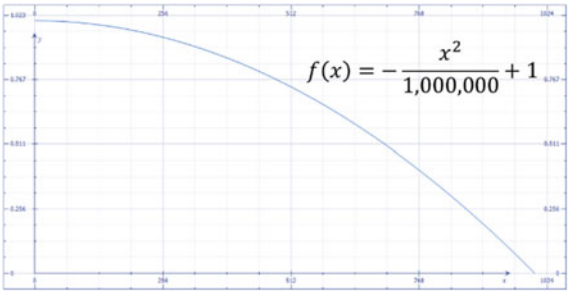


Fig. 3.4 Equation for time confidence factor

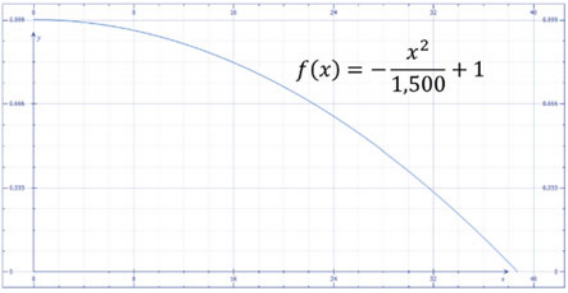


Fig. 3.5 Equation for compilation confidence factor

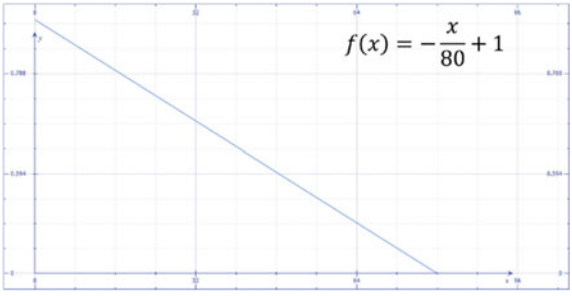


Fig. 3.6 Equation for error confidence factor

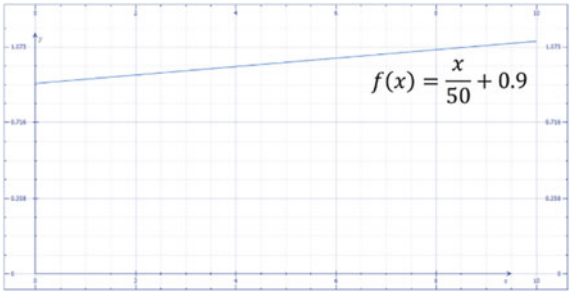


Fig. 3.7 Equation for error persistency confidence factor

assesses that the confidence of student davidb in the learning activity Factorial.java at the moment of the processing of the learning event is as follows:

$$\text{confidence} = 0.9999 \times 0.994 \times 0.9917 \times 0.9097 = 0.8966 \qquad (3.1)$$

The lowest confidence factor will impose a top limit on the overall confidence value.

3.5 Future Work and Conclusion

The scalability and robustness of SCALE are currently being tested at a partner institution in India. SCALE includes mechanisms to monitor thoroughly all the critical processing and input/output (I/O) operations in order to assess its scalability and robustness. In the near future, SCALE will be extended to a full Java course and will be applied in the English writing and mathematics learning domains. Once SCALE will meet all the robustness and scalability requirements to support a full-fledged experiment with a full Java course, a longitudinal propensity score matching observational study will be conducted to evaluate the effectiveness of SCALE over students' learning experiences.

This paper thoroughly described SCALE, an architecture to analyze the proficiency and confidence of students in any learning domain by providing teachers with mechanisms to define their own proficiency and confidence metrics. The paper explains in a step-by-step fashion how to apply SCALE in a learning domain by showing an example in the programming domain.

References

1. Hosseini, R., Vihavainen, A., & Brusilovsky, P. (2014). Exploring problem solving paths in a java programming course. In *Proceedings of the 2014 Annual Conference on Psychology of Programming (PPIG)* (pp. 65–76).

2. Sosnovsky, S., Dolog, P., Henze, N., Brusilovsky, P., & Nejdl, W. (2007). Translation of overlay models of student knowledge for relative domains based on domain ontology mapping. In *Proceedings of the 2007 AIED 13th International Conference on Artificial Intelligence in Education*.
3. Ganapathi, G., Lourdusamy, R., & Rajaram V. (2011). Towards ontology development for teaching programming language. In *Proceedings of the World Congress on Engineering 2011 (WCE)* (pp. 198–210).
4. Chen, C. -L., Cheng, S. -Y., & Lin, J. M. -C. (2012). A study of misconceptions and missing conceptions of novice java programmers. In *Proceedings of the 2012 International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)* (pp. 307–313).
5. Hristova, M., Misra, A., & Rutter, M. (2003). Identifying and correcting java programming errors for introductory computer science students. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE)*.
6. Sirkiä, T. (2012). Recognizing programming misconceptions: An analysis of the data collected from the UUhistle program simulation tool (Master's thesis). Retrieved from …. (UMI No. 1234).
7. Mow, I. T. C. (2012). Analyses of student programming errors in java programming courses. *Journal of Emerging Trends in Computing and Information Sciences, 3*(5), 739–749.
8. Sampson, D. G. (2009). Competence-related metadata for educational resources that support lifelong competence development programmes. *Educational Technology and Society, 12*(4), 149–159.