Jurnal Teknologi

Full Paper

A GUIDELINE-BASED APPROACH TO SUPPORT THE ASSESSMENT OF STUDENTS' ABILITY TO APPLY OBJECT-ORIENTED CONCEPTS IN SOURCE CODE

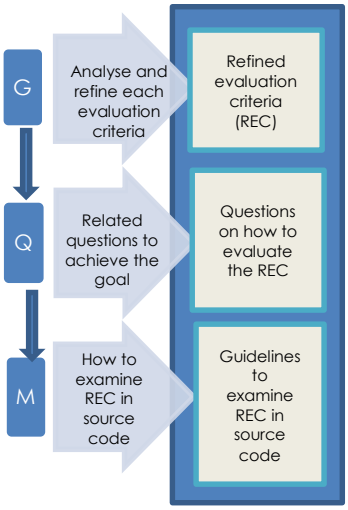
Norazlina Khamis*, Norhayati Daut

Faculty of Computing and Informatics, Universiti Malaysia Sabah, Jalan UMS, 88400 Kota Kinabalu, Sabah, Malaysia

*Corresponding author
norazlina@ums.edu.my

Article history
Received
7 July 2015
Received in revised form
7 September 20`5
Accepted
15 January 2016

Graphical abstract



Abstract

There are many approaches in assessing students' ability in object-oriented (OO) programming, but little is known on how to assess their ability in applying OO fundamental concepts in their written source codes. One major problem with programming assessment relates to variation in marks given by different assessors. Often, the grades given also does not gauge whether students know how to apply OO approaches. Thus, a new assessment approach is needed to fill these gap. The objective of this study is to construct and validate through expert consensus, a set of evaluation criteria for fundamental OO concepts together with the guidelines called *GuideSCoRE*, to help instructors assess students' ability in applying OO concepts in their program source code. The evaluation criteria are derived from fundamental OO concepts found in Malaysian OO programming syllabuses and validated by a three-round Delphi approach. The proposed evaluation criteria were mapped with related OO design heuristics and OO design principles. A guideline (*GuideSCoRE)*, constructed based on the Goal-Questions-Metrics approach together with the evaluation criteria is used by instructors when assessing students' source codes. An inter-rater reliability analysis among six instructors found moderate agreement on assessment scores (κ values of mainly between 0.421 and 0.575) indicating that whilst the guidelines do not completely eliminate variations between raters, it help reduce their occurrences.

Keywords: Object-oriented programming, object-oriented concept, programming assessment, Goal-Question-Metric approach

Abstrak

Terdapat banyak pendekatan bagi menilai keupayaan pelajar dalam pengaturcaraan berasaskan objek, tetapi sedikit sahaja diketahui bagaimana untuk menilai keupayaan mereka di dalam mengaplikasikan konsep asas berasaskan objek di dalam kod sumber mereka. Satu masalah utama dengan penilaian pengaturcaraan adalah berkaitan dengan pelbagai variasi di dalam markah yang diberi oleh penilai yang berbeza. Gred yang diberikan juga biasanya tidak melambangkan sama ada pelajar tahu bagaimana untuk mengaplikasikan pendekatan berasaskan objek. Maka, satu pendekatan penilaian baru dperlukan untuk mengisi ruang ini. Objektif kajian ini ialah membangunkan dan mengesahkan melalui persetujuan pakar, satu set kriteria penilaian konsep asas berasaskan objek bersama-sama dengan garis panduan yang dipanggil *GuideSCoRE*, untuk membantu pengajar menilai keupayaan pelajar mengaplikasikan konsep berasaskan objek di dalam kod sumber mereka. Kriteria penilaian ini diperoleh dari konsep asas berasaskan objek yang diperolehi dari silibus kursus berasaskan objek di Malaysia dan disahkan melalui pendekatan Delphi tiga langkah. Kriteria yang dicadangkan kemudian dipetakan kepada reka bentuk heuristik dan juga prinsip reka bentuk berasaskan objek. Satu garis panduan (GuideSCoRe), dibangunkan berdasarkan pendekatan Goal-Question-Metric bersama-sama dengan kriteria penilaian digunakan oleh pengajar apabila menilai kod sumber pelajar. Satu analisis kebolehpercayaan antara-penilai dikalangan enam pengajar memperolehi persetujuan sederhana ke atas skor penilaian

(niai κ diantara 0.421 dan 0.575) menunjukkan walaupun garis panduan tidak menghapuskan variasi di kalangan penilai secara keseulruhannya, namun ia membantu di dalam mengurangkan kekerapannya.

Kata kunci: Pengaturcaraan berasaskan objek, konsep berasaskan objek, penilaian, pengaturcaraan, pendekatan Goal-Question-Metric

1.0 INTRODUCTION

Object-oriented programming (OOP) has changed the practice of writing computer applications. Students who are introduced to programming need to be able to firstly, understand and verbalize the concepts involved in programming, and subsequently produce well-written, structured and understandable applications using the language involved. In most universities in Malaysia, OOP methods is introduced as an introductory course for programming, but have the reputation of being a "killer course", where the failure rate is high. The problem related to learning OOP amongst students have been discussed in literature, but no clear solutions seem to be available that is generic enough for applications in different cultural and language context. The paper aims to propose a set of evaluation criteria, which can be used by OOP instructors when assessing students' ability to apply OO concepts in their source code.

2.0 RELATED WORK

When learning OOP, students need to have a good understand of the core concepts of objects and their relationships [1]. Fleury [2] found that students constructed their own understanding of concepts in their programming assignments, and those constructions are not always complete and correct. Guzdial [3] found that students have problems in creating collaborative objects especially understanding the connections between objects. Sheetz *et al*. [4] confirmed that understanding the concepts of OOP is the hardest and most important rated by two groups of students who have completed a six-week course on OOP systems. It was suggested that instructors could increase the grasp of concepts by simplifying methods of teaching. Students in the study also indicated problems related to designing and understanding programming techniques. At a series of workshops, the COOL project at the University of Oslo, Norway [5] focused on designing the right environments and using varied delivery pedagogical approaches to offer students with richer environments and increase social interaction to solve OO problems. Lahtinen, Ala-Mutka and Jarvinen [6] surveyed 559 students who were novices to programming and those students expressed difficulty in understanding concepts and prefer to work alone on program coursework rather than in class-based practical sessions. Other studies regarding OOP [1,7-11] reported that students experienced misconceptions about object-oriented concepts.

Besides, understanding the problems experienced in learning OOP, studies have also examined the role of assessment in the learning process. After all, it cannot be assumed that students who have passed their OOP examinations have understood the OO paradigm. It has been indicated that students who have prior experience in programming perform significantly better in their university introductory programming examinations [12]. This means that novices should be introduced to OO techniques in a different way from the experienced students and the assessment given must rightly reflect the students' understanding of the concepts learnt at whatever levels.

Many assessment tools have been developed to measure students' ability in OO programming [13,14]. These tools usually focus on assessing technical and didactic quality aspect, such as the correctness of the output, appropriateness of the programming processes and the styles followed. Students' ability in applying the fundamental OO concepts in source code is not addressed. Too often, educators have to develop their own assessment instrument every time they want to examine students' learning in programming. This is due to the unavailability of validated assessment instruments in CS disciplines specifically in OOP courses [15].

It is difficult to evaluate students' grasps of fundamental concepts in programming without a valid and reliable assessment instrument, which fundamentally is closely related to the criteria used in the instrument. In assessments, educators sometime fail to consider whether students have applied the correct OOP concepts in their source codes. Often, the criteria emphasized are whether the program the student program compiles, whether the output is correct and, whether useful comments are sufficiently included. The criterion does not consider the correct and appropriate application of OO concepts. Also, different educators may use different criteria when examining students' answers and this lead to inconsistencies in assessment.

This paper aims to identify fundamental concepts covered by most OOP courses in Malaysian public universities, use the concepts to construct an evaluation criteria, and a guideline that can be used by OOP instructors when assessing students' ability to apply OO concepts in their source codes. We hope to

answer the following questions. What are the fundamental OO concepts that beginners should know? What are the evaluation criteria that can be used to assess students' ability to apply the concepts in their source codes? What guidelines can be used to further support the assessment process?

3.0 METHOD

3.1 Identifying Object-oriented Concepts

The initial stage of the study involved identifying fundamental OOP concepts, which are embedded in programming course syllabuses offered at Malaysian public universities and published literature. Published literature provided a variety of definitions on OO concepts and this may have increased confusions about meaning of concepts and terms. Four sources were used to identify the fundamental concepts which has been explained in [17].

The identified fundamental OO concepts will serve as a basis in constructing the evaluation criteria to assess students' ability in applying those concepts. Lecturers, who teach programming courses at Malaysian public universities, were asked to evaluate, validate and refine the concepts identified. The resulting evaluation criteria are further mapped with OO design heuristics and principles. The final output from this phase will be a set of validated evaluation criteria, which will be used in designing assessment guidelines.

Based on the analysis and comparison between Armstrong's [18] OO taxonomy and the concepts obtained from various sources described above, eight frequently occurring OO concepts are selected as the fundamental concepts relevant to this research. Those concepts are *object, class, abstraction, polymorphism, encapsulation, inheritance, message passing* and *method*. Based on these concepts, the evaluation criteria were established for assessing students' ability in applying these fundamental concepts in their source code.

3.2 Verifying Evaluation Criteria

In the second stage, Delphi approach is used to verify the set of evaluation criteria for assessing students' ability in applying fundamental OO concepts in their source code based. The criteria derived are based on experts' knowledge and experiences. We use experts to validate the content of the assessment instrument constructed. The content validation process helps indicate the degree to which the content of the items reflects the content domain as well as the representativeness and clarity of each item. The experts also offer suggestions for improving the measure. A consensus by eight experts was obtained after three iterations process. It has been found in the literature [19, 20] that three iterations are often sufficient to collect the needed information and to reach a consensus and this was applied in this study. The eight experts who participated in this study fulfilled the selection criteria as stated in [17].

In the last stage of the Delphi iteration, the experts were asked to validate the final fundamental OO concepts identified, and verify the evaluation criteria derived to assess the understanding of the OO concepts.

A total of 16 evaluation criteria is derived and the final evaluation is done by mapping the criteria with object oriented design heuristic and object-oriented design principles [22]. Details about the process of establishing evaluation criteria is found in [17] and the final evaluation criteria is presented in Table 1.

Table 1 Evaluation criteria after three iteration of Delphi approach

Identifier & Evaluation Criteria	Related OO concept
EC01: Able to identify classes at the proper level of abstraction with regards to the problem being solved (design level)	*Class Abstraction*
EC02: Able to identify the proper classes, methods and attributes to solve a particular problem (implementation level)	*Class Method Abstraction*
EC03: Able to give appropriate names/attributes (nouns) and method (verbs)	*Class*
EC04: Able to create constructors as necessary for a class	*Class*
EC05: Able to define accessor and mutator methods (i.e. getter and setter methods) as necessary for a class	*Method, Class*
EC06 :Able to send an appropriate message/method call to an object based on the type of that object and the interface of the corresponding method	*Message passing Method, Object*
EC07: Able to manipulate heterogeneous container of objects by sending appropriate polymorphic messages to each of them	*Object Message passing Polymorphism*
EC08: Able to pass correctly an object as a parameter in a message	*Message passing Object*
EC09: Able to identify the proper level of access control (e.g. private, public, protected) for the characteristics and behaviours of a class	*Encapsulation Class*
EC10: Able to identify "is-a" relationships between several related classes and implement inheritance between these classes correctly; these include super and sub-class constructors, methods that should be inherited and their access control	*Inheritance Class Method*
EC11: Able to call a method inherited from the ancestors of the class in which the call is made	*Method Class Inheritance*

Identifier & Evaluation Criteria	Related OO concept
EC12: Able to appropriately define multiple related classes as opposed to defining a single class in solving the problem in hand	*Inheritance Class*
EC13: Able to create aggregation relationships between related classes to correctly indicate whole-part relationships between the concepts/entities represented by the classes.	*Class*
EC14: Able to correctly create an object of a class using an appropriate constructor of factory method based on the documentation of the corresponding class	*Object Class*
EC15: Able to identify appropriately situations in which polymorphism can be applied	*Polymorphism*
EC16: Able to set up correctly a group of objects which work together among themselves in carrying out a certain task (vs. one object doing everything itself)	*Object*

3.3 Development of Assessment Guideline using Goal-Questions-Metric Approach

The evaluation criteria resulting from the Delphi study is in their conceptual form and is open to interpretation by educators. Thus, each evaluation criteria resulted from the Delphi study is further analysed and refined to ensure they are presented in a measurable form. Each criterion corresponds to the desirable application properties in the source code for each of the fundamental OO concepts. Those educators who have more experience in teaching OOP will find it easy to use the evaluation criteria alone to examine students' abilities in applying fundamental OO concepts in their source code. However, educators who have less or no experience might find it difficult to understand the evaluation criteria, and may be uncertain that they are actually assessing students' application abilities from the source code being evaluated. This could inevitably lead to marking inconsistencies. Therefore, an assessment guideline on how or what to look for when examining source code is proposed to assist educators when assessing students' programs. An adaptation of the Goal Question Metric (GQM) approach [21] is used to develop the guideline.

To develop the guideline, each criterion is analysed and refined from ambiguity to determine how it should be assessed in the source code. To do this the design heuristic and principles in programming reference books were examined to ascertain what exactly needs to be looked at in the source code for each of the evaluation criteria. The assessment guideline derived from the GQM approach are presented in the GQM Profile form.

Each GQM Profile represents the assessment guideline for the respective evaluation criteria. The profile consists of four components as follows.

- **Component One:**
  Refined Evaluation Criteria
  Refers to the *Goal* of the assessment, which is the refined evaluation criteria. Each criteria resulted from previous phase is analysed and refined to ensure it is presented in a measurable form.
- **Component Two:**
  Related Design Heuristics/Principles/ References
  Refers to the related design heuristics; design principles and programming provide by reference books. This section will support the rationale of designing the assessment guidelines
- **Component Three:**
  Question
  Refers to the *Question* on how to achieve the goal, that is, how to evaluate the refined evaluation criteria in source codes.
- **Component Four:**
  Source Code Examination
  Refers to the *Metrics* of the assessment, that is, the guidelines on how to assess the refined evaluation criteria through a source code examination approach

Based on the GQM approach, two examples of the final output for one of the evaluation criteria and its assessment guideline is presented in GQM Profile in Table 2 and Table 3 respectively.

Table 2 GQM Profile for Refined Evaluation Criteria 01 (REC01)

Refined Evaluation Criteria: REC01: *Able to identify at the proper level of abstraction with regards to the problem being solved in terms of class granularity and cohesion*
Related Design Heuristic/Principles/References: Design Heuristic:[H2.8, H2.9, H2.10, H2.11, H3.1, H3.2, H3.6, H3.7, H3.8, H3.10] Design Principles: [Single Responsibility Principle]
Questions: Q1.1: How to gauge the granularity of a class in source code? Q1.2: How to gauge the cohesion of a class in source code?

Source code examination guidelines: Look at the class structure at the macro level CE1.1: Look at the number of classes in the source code. If there are too many or too few classes for the problem being solved, there is a possibility/indication of the existence of classes defined at unsuitable levels of granularity. CE1.2: Look at the number of methods for the class and their relatedness. If the class has many methods, there is a possibility that more than one responsibility have been assigned to that class. Are the methods focused on a single responsibility (cf. Single Responsibility Principle)? However, there can be situations where methods are not many, but they do not focus on a single responsibility.

Table 3 GQM Profile for Refined Evaluation Criteria 02 (REC02)

Refined Evaluation Criteria: REC02: *Able to identify proper attributes and methods of a class with respect to the problem being solved.*
Related Design Heuristic/Principles/References: Design Heuristic: [H2.1, H2.4, H2.5, H2.9, H3.9, H4.6, H4.13, H8.1] Design Principles: Single Responsibility Principle
Questions: Q2.1: Are methods and attributes highly related to the responsibilities of the class? Q2.2: Are methods defined at the appropriate granularity?
Source code examination guidelines: Look at micro level: Focus on one class. Assume the class has been appropriately defined. CE2.1: Are the attributes and methods related to their class's responsibility? The properness of attributes and methods relates to the cohesiveness of a class. (Q2.1) CE2.2: Locate methods defined which are highly related. If not appropriate, there must be one method unrelated with the responsibilities assigned (by looking at what the method does. (Q2.1) CE2.3: Look at each attribute and ascertain whether it is being used or not within the class. (Q2.1) CE2.4: Look at the size of the class's methods. If the size of a method is big, it might be an indication of improper granularity and the method should be broken up into smaller ones. (Q2.2)

4.0 RESULTS AND DISCUSSION

Reliability of *GuideSCoRE* in assessing students' ability in applying OO concept in their source code is being validate by development of a semi-automated web-based tool (WebSAT). The tool is develop to support the assessment of fundamental OO concepts application by integrating *GuideSCoRE*. Figure 1 indicates the structure of WebSAT. Although fully automatic assessment would ease educators' workload, it does not provide important feedback on matters that cannot be automatically assessed. Educators believe that it is not possible to automate all issues related to good programming, [25] which includes assessment of students' ability in applying fundamental OO concepts.

The assessment tool for *GuideSCoRE* developed in this research was inspired by the checklist-based evaluation proposed by Benchmarks for Science Literacy project [26] in which a set of specific, well-defined criteria was defined that can be evaluated on a uniform scale. It requires the instructor to make a decision on their level of agreement, generally on a five-point scale with a statement. The five-point Likert scale used for scoring *GuideSCoRE*'s evaluation criteria ranges from 0 = No evidence (NE), 1 = Weak evidence (WE), 2 = Average Evidence (AE), 3 = Clear evidence (CE), and 4 = Strong evidence (SE) (Table 4).

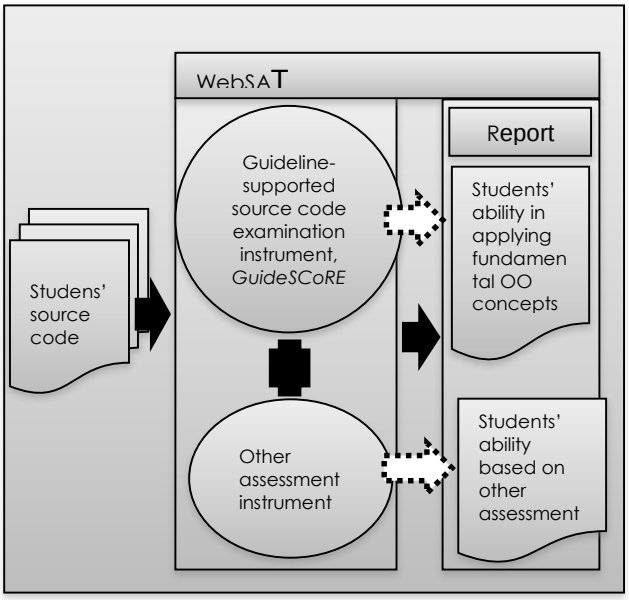


Figure 1 The Structure of WebSAT

Table 4 GuideSCoRE Assessment Template

	Assessment scale For each of the following evaluation criteria, place the most appropriate score by selecting a number from 0-5 for each specified trait to evaluate the evidence of the skills in students' source code.					Skill is not considered in the assessment (NA)	Skill is not considered in the assessment but proper application is evident (Extra marks) (EM)
Evaluation Criteria	0 = NE	1 = WE	2 = AE	3 = CE	4 = SE		
Criteria [Guidelines]							

Each rating point represents a certain degree of presence of evidence, found through examining the source code, of proper application of the skill being evaluated. According to Altman and Bland [27], "Absence of evidence is not evidence of absence". *Absence of evidence* is the absence, or lack of, any kind of evidence that may show, indicate, suggest, or be used to infer or deduce a fact. *Evidence of absence* is evidence of any kind that can be used to infer or deduce the non-existence of something. In the context of this research, the non-presence of possession of a skill in a source code (i.e. **"No evidence"**) does not infer that a student does not have that skill. It might be the case that the student possesses that skill but did not happen to apply it while writing the source code. For that matter, it is even possible for the student to score "**Strong evidence"** for that skill in a different assignment. Therefore, one can assume that *GuideSCoRE* evaluation results only represent the proper application of certain skills by students at the time the assessment is being conducted and it is based solely on evidence that can be found in their source code.

The instrument has been designed to be flexible in terms of the criteria to be evaluated for an assessment exercise. Educators can disregard those evaluation criteria that are deemed not relevant to the current problem being solved or are not being assessed in the exercise. Prior to using the instrument for an assessment exercise, educators should determine which evaluation criteria are most relevant to assess and modify their scoring requirements as needed. The response to be given by the assessor for such evaluation criteria is any one of the following,

- **Not applicable** (NA):
  Skill is currently not considered in the assessment.
- **Extra Marks** (EM):
  Skill is not currently considered in the assessment but proper application is evident.

Upon completion of assessment using *GuideSCoRE*, a report consisting of a summary of evidences of particular skills found in a student's source code is produced. It shows the student's total score which is the sum of the individual scores given by the assessor for each evaluation criteria. It is an indicator representing the overall performance of OO skills applied by the student based on the examination of his/her source code. The calculation of the *GuideSCoRE* total score is given below. Note that it takes into account that not all evaluation criteria are being assessed by the educator.

Maximum mark for evaluation criteria i =M_i (for i=1..13)

$$\text{where } M_i = \begin{cases} 0 & if\ not\ assessed \\ 5 & if\ assessed \end{cases} \quad (1)$$

Student's mark obtained for evaluation criteria i = m_i (for i=1..13)

$$\text{where } m_i = \begin{cases} 0 & if\ not\ assessed \\ k & if\ assessed \end{cases} \quad (2)$$

and k = 1..5 (Likert point score)

$$\text{Maximum marks achievable} = \sum_{i=1}^{13} M_i \quad (3)$$

$$\text{Student's total marks assigned for GuideSCoRE approach} = \sum_{i=1}^{13} m_i \quad (4)$$

Nine Instructors evaluated the usability of the assessment tool and this number is indicated as sufficient in literature [27, 28]. An e-mail invitation was sent to 9 instructors who were involved in teaching OOP related courses from various Malaysian universities to participate in this evaluation phase. Some of them have experiences in research or development related to the OO domain. To ensure validity of this evaluation, none of these educators were involved in the Delphi study conducted earlier.

Each instructor was requested to evaluate six source codes, SC01 to SC06, for an assignment called 'My Cal' using the *GuideSCoRE instrument*. Five students from two Malaysian public universities wrote source codes, SC01 to SC05. The remaining one, SC06, was a sample answer written by an OO expert. The assignment used in this evaluation phase was designed by an OOP expert. The problem statement was designed in such a way that the fundamental OO concepts covered by *GuideSCoRE* can be applied. The five students who volunteered were enrolled in two different OOP-related courses, namely Java Programming and Data Structure. Each student was requested to produce a source code by implementing all the *GuideSCoRE*'s fundamental OO concepts in their source code. These source codes were uploaded in the WebSAT for evaluation.

4.1 Evaluating the Reliability of the GuideSCore

The evaluation phase focused on assessing the reliability of the *GuideSCoRE* in minimizing the inconsistency during OO assessment. Reliability can be defined as a scale that consistently reflect the construct it is measuring [29]. An instrument is considered reliable when students on average obtained similar score for a question that is being evaluated by more than one instructor. Reliability is indicated when two persons measuring the same construct gave the same score. An inter-rater reliability analysis using the Kappa statistic was performed to determine consistency among raters [30,31,32,33]. Tables 5 shows the percentage of agreement, Kappa value, κ for each of the source code using Fleiss Kappa calculation.

Table 5 Kappa Value using Fleiss Kappa Calculation to Determine Inter-Rater Reliability

Source code	Kappa Value, κ
SC01	0.545
SC02	0.459
SC03	0.421
SC04	0.412
SC05	0.380
SC06	0.575

We use the Landis and Koch's [34] approach to interpret κ values (Table 6)

Table 6 Interpretation of the κ values

κ	Interpretation
< 0	Poor agreement
0.01 – 0.20	Slight agreement
0.21 – 0.40	Fair agreement
0.41 – 0.60	Moderate agreement
0.61 – 0.80	Substantial agreement
0.81 – 1.00	Almost perfect agreement

Table 6 indicates that the inter-rater scores range between 0.4 and 0.5 and there is moderate agreement on the assessment score for all source code being evaluated by instructors during the marking process. This indicates that the *GuideSCoRE* moderately help minimizing the marking differences among educators during the marking process. Although *GuideSCoRE* do not completely eliminate variations between raters, it does reduce the occurrence of these discrepancies and can be improved in the future.

5.0 CONCLUSION

From our preliminary study of OOP courses offered at Malaysian universities [35] we found that most Universities in Malaysia used both summative and formative assessment approaches. Students are mainly assessed through written examinations, programming assignments and oral presentations. Students are given feedback on their performance in the form of grades (A to F) and marks (00 to 100). It is assumed in this context that those who had passed can write programs, but does not indicate whether the students understand the OO concepts and therefore could reapply that knowledge in their future work places. All instructors who participated in the survey agreed that there is a need to have a framework for assessing OO skills among undergraduates. In this study we looked into how such an assessment can be implemented.

All instructors also agreed that it is important for students to have a good grasp of core OO concepts. Perusal of OO programming syllabuses offered in Malaysian universities and published literature revealed a number of fundamental concepts, which students should master. Abstraction, class, method, message passing, inheritance, object, polymorphism and encapsulation are among the most common ones. These concepts were chosen as the basis for establishing the evaluation criteria for assessing students' ability in applying fundamental object-oriented concepts in their source code as well as to validate the content using a three-round Delphi study. Sixteen validated evaluation criteria were derived based on expert consensus and were further validated by associating them with related object-oriented heuristics and principles. An instrument (*GuideSCoRE*), comprising a set of guidelines that can be used by instructors to support assessments of students' skills based on evaluation criteria is established. The contribution in this context is the *GuideSCoRE,* which, unlike other instrument in OO programming assessment, focuses on determining students' skills in applying fundamental OO concepts in their source code. Using this guidelines instructors could gauge their agreement that each criteria reflects students' mastery in applying a particular OO programming concepts in their source code. Thus, the differences during marking process due to differences in instructors' experiences are minimised.

The *GuideSCoRE* evaluation results represent the proper application of certain skills by students at the time the assessment is being conducted and it is based solely on evidence that can be found in their source code. In addition to that, the assessment approaches are independent of programming-language or environment. For example, it can be used in assessing OO source code produced using C++ or Java, or, different environment like Netbeans or Eclipse.

Acknowledgement

We are grateful for the participants and experts who involved in this study.

References

[1] Kolling. M. 1999. The Problem of Teaching Object-Oriented Programming, Part 1: Languages. *Journal of Object-Oriented Programming.*

[2] Fleury. A. E. 2000. Programming in Java: Students-Constructed Rules. In *31st SIGCSE Technical Symposium on Computer Science Education.*

[3] Guzdial, M. 2001 Centralized Mindset: A Student Problem with Object-Oriented Programming. *Journal of Computer Science Education*. 14(3&4): 28-32.

[4] Sheetz, S. D., Irwin, G., Tegarden, D. P., Nelson, H. J. and Monarchi, D. E. 1997. Exploring the Difficulties of Learning Object-oriented Techniques. *Journal of Management Information System.* 14(2): 103-131.

[5] Berge, O. and Fjuk, A. 2003. Soco-cultural Perspectives on Object-oriented Learning. *Workshop on Pedagogies and*

Tools for learning Object Oriented Concepts, European Conference on Object Oriented Programming, Darmstadt, Allermagne.

[6] Lahtiven, E., Ala-Mutka, K. and Jarvinen, H. M. 2005. A Study of the Difficulties of Novice Programmers. *ITiCSE '05*. June 27-29. Portugal.

[7] Anna, E. and Thune. M. 2005. Novice Java Programmers' Conception of "Object" and "Class", and Variation Theory, in *ITiCSE*, Monte de Caparica, Portugal. 89-93.

[8] Kate, S. *et al.* 2008. Student Understanding of Object-Oriented Programming as Expressed in Concept Maps. In *SIGCSE Bull*. 40: 332-336.

[9] Kate, S. and Lynda, T. 2007 Checklists for Grading Object Oriented CS1 Programs: Concepts and Misconceptions. *SIGCSE Bull.* 39: 166-170.

[10] Simon, H. *et al.* 1997. Avoiding Object Misconceptions. *SIGCSE Bull.* 29: 131-134.

[11] Brown, G. *et al.* 1997. *Assessing Student Learning in Higher Education.* London: Routledge, Taylor and Francis.

[12] Hagan, D. and Markham, S. 2000. Does it Help to Have Some Programming Experience Before Beginning a Computing Degree Program? In *Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and Technology in Computer Science Education*. New York: ACM. 25-28.

[13] McCracken, M. *et al.* 2001. A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. In *SIGCSE Bull.* 33: 125-180.

[14] Raymond, L and John, L. 2003. First Year Programming: Let All the Flowers Bloom. In *Proceedings of the fifth Australasian Conference on Computing Educatio*n. Volume 20, Adelaide, Australia.

[15] Allison, E. T and Mark, G. 2010. Developing a Validated Assessment of Fundamental CS1 Concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Milwaukee, Wisconsin, USA.

[16] ACM. 2013. ACM Computing Curricula 2013. Available:https://www.acm.org/education/curricula-recommendations.

[17] Norazlina K. 2016. Establishing Evaluation Criteria for Assessing Novices' Ability in Applying Object-oriented Concept Using Delphi Approach. *International Journal of Information and Education Technology*. ISSN 2010-3689.

[18] Deborah J. Armstrong. 2006. The Quarks of Object-Oriented Development. Commun. ACM 49. 2(February 2006): 123-128.

[19] Hsu, C. C and Sandford, B.A. 2007. The Delphi Technique: Making Sense of Consensus, *Practical Assessment, Research & Evaluation.* 12: 1-8.

[20] Yousuf, M. I. 2007. Using Experts' Opinion Through Delphi Technique. *Practical Assessment , Research & Evaluation.* 12: 9-18.

[21] Victor B, Gianluigi C. and Dieter R. 1994. The Goal Question Metric Approach.

[22] Martin, R. C. 1996. Design Principles. Available: http://www.objectmentor.com/resources/publishedArticles.html.

[23] Meyer, B. 1997. *Object-oriented Software Construction.* Prentice Hall.

[24] Liskov, B. 1987. Keynote Address-Data Abstraction and Hierarchy. In *proceedings on Object-oriented programming systems, languages and applications* (Addendum), Orlando, Florida, United States.

[25] Kirsti, A. M. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*. 15: 83-102.

[26] 1993. B*enchmarks for Science Literacy*: Oxford University Press.

[27] Altman, D. G and Bland, J. M. 1995. Statistics Notes: Absence of Evidence is Not Evidence of Absence. *BMJ.* 485.

[28] Creswell, J. 1998. *Qualitative Inquiry and Research Design: Choosing Among Five Traditions*. Thousand Oaks, CA: SAGE.

[29] Morse, J. M. 1995. The Significance of Saturation. *Qualitative Health Research*. 5: 147-149.

[30] Trochim, W. M. K. 2006. Research Method Knowledge Base. *Measurement: Reliability*.

[31] Nichols,T. R *et al.* 2010. Putting the Kappa Statistic to Use. *The Quality Assurance Journal.* 13: 57-61.

[32] Moskal, B. M and Leydens, J. A. 2012. Scoring Rubric Development: Validity and Reliability. *Practical Assessment, Research & Evaluation.* 7(10).

[33] Fleiss, L. 1971. Measuring Nominal Scale Agreement Among Many Raters. *Psychological Bulletin.* 76: 378-382.

[34] Landis, J.R and Koch,G. G. 1997. The Measurement of Observer Agreement for Categorical Data. *Biometrics.* 33: 159-174.

[35] Norazlina K. and Idris, S. 2007. Investigating Current Object-oriented Programming Assessment Mehod In Malaysia's Universities. In *Proceedings of the International Conference on Electrical Engineering and Informatics*, June 17-19, Bandung, Institut Teknologi Bandung. 666-668.