A Recommended Feedback Model of a Programming Exercise Using Clustering-Based Group Assistance

S. Suhailan, M.S. Mat Deris, S. Abdul Samad, M.A. Burhanuddin

Abstract: ***Many studies on automated programming assessment tools with automated feedbacks have been addressed to assist students rectifying their solution's difficulty. While several studies have produced specific computational programming exercise's feedback using a static template analysis, there is still a lack of an automated programming feedback model that is dynamically enriched through a live assisted feedback from an expert. Thus, this research proposed a recommended feedback model on specific computational programming question using clustering-based live group assistance. The assisted feedback was done by an expert through a similar difficulty analysis of computer programs that were grouped together based on ordinal features using a K-Means clustering algorithm. An experiment was executed by responding to 7 program difficulty clusters that consists of 33 programs. Based on these inputs, the efficiency ratio result shows that the model can minimize expert's workload and can be effectively used as a recommender system. Furthermore, the efficiency of this model can be gradually intensified with more assisted feedbacks being provided by the expert user within other lab sessions.***

Index Terms: K-Means, programming feedback, recommender system.

I. INTRODUCTION

Mastering computer programming is a compulsory skill for computer science students. The skill can be obtained through practicing lots of programming exercises. Nowadays, many automated assessment programming tools are available. A student can self-develop and submit programming codes while the tool will instantly provide automated feedback to highlight any syntax errors found in the codes during the compilation or execution of a program. However, in certain cases, the standard error feedback is not clear and difficult to be interpreted by novice students. This indicates that the current automated feedback is still insufficient compared to the expert's (e.g. teacher, tutor) feedback. An expert can entertain student's responses in tailoring feedback to be understood. However, providing real-time programming error feedback to numerous students requires a lot of resources, especially during a lab session. There is also a need to manage previous feedback messages to be reutilized as recommended feedback on repeated similar difficulties. Thus, a new model of integrated automated programming feedback framework needs to be addressed. By having a clustering technique, computer programs can be grouped into similar difficulty in facilitating teacher to provide assisted feedback. Meanwhile, ranking technique can be used to sort computer programs based on difficulty level that can leverage teachers to pay attention to worst solution attempts. In order to reutilize this assisted feedback for future cases of similar difficulty, a recommender system needs to be designed. Thus, by having this kind of integrated feedback framework, it shall enhance automated programming feedback limitation from being insufficient and static to be more efficient and extensible in helping students rectifying programming syntax and logic errors.

II. AUTOMATED PROGRAMMING FEEDBACK MODELS

Looking from the feedback resource perspective, there are two main attributes to be considered; context and content [1], [2]. Context is the state of when, where and how the feedback is going to be activated. On the other hand, content is its associated message of what and why the feedback is activated. In general, all of programming feedback are activated using three styles; automated (machine) feedback, assisted (human) feedback and recommended (human to machine) feedback. Automated feedback refers to its predefined content and context that will be automatically activated by machine. It must be created before execution of a targeted process. Meanwhile, assisted feedback refers to its dynamic updates of content and context that will be manually activated by human through assisted-machine medium. It usually created during or after execution of a monitored process. Recommended feedback refers to its post-defined content and context of an assisted feedback that will be automatically activated by machine. It is created after execution of a new identified process. Table 1 shows the advantages of these three feedback styles in view of helping students learning computational programming.

Currently, automated feedback are lacking in activating dynamic guidelines as compared with assisted feedback from an expert [58]–[60]. The expert can provide new feedback not only in term of content (what, why and how),

Revised Manuscript Received on February 11, 2019.

S. Suhailan, Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Terengganu, Malaysia

M.S. Mat Deris., Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Terengganu, Malaysia

S. Abdul Samad, Universiti School of Information Technology, Faculty of Business & Information Science (FoBIS), UCSI University, Kuala Lumpur, Malaysia

M.A. Burhanuddin, Faculty of Information Communication Technology, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia.

but they can also interact in dynamic context (when and where) in assisting student to rectify programming problems. However, it is very challenging for an expert to respond on each student's difficulties during programming exercises [44]. It would be more practical for a teacher to give feedback after all the same difficulties are put together into certain groups. In most cases, programs' difficulty can be similarly grouped together using clustering technique.

Table 1: Styles of programming feedback

Type of Feedback	Debugging Effectiveness	Techniques	References
Automated	Able to highlight specific errors (but sometimes having ambiguities)	syntax analysis	[3]–[5]
		semantic analysis	[6]–[12]
		output analysis	[13]–[20]
		visual programming	[21]–[32]
		intelligent agent	[33]–[37]
Assisted	Able to highlight general errors and mistakes (subject to human constraints)	electronic forum, social media discussion, data mining	[38]–[45]
Recommended	Able to highlight specific and general errors and mistakes (subject to previous data)	association rules, classifier, clustering and information retrieval	[46]–[49] [50]–[57]

III. ASSISTED FEEDBACK THROUGH CLUSTERING

The idea of using clustering on computer programs for automated feedback is still been explored by recent researches. In [61] use K-Means to group similar C program based on ordinal features such as number of loops, number of selection, number of modules/functions/classes, number of variables, number of jump statement instructions, and number of expression. In [62] extract Java computer program features based on number of variables, number of objects instantiation, and number of return value in order to classify its class pattern (e.g. utility, DAO, builder, bean, adapter, etc.). In [63] use this method to extract computer program features based on n-gram token sequences such as header, keywords, identifier, operators, and numerals. In [64] extract computer program features based on frequency of characters and words in a line, frequency of whitespace (in the beginning, interior and ending) of non-whitespace lines and frequency of underscore. In [65] clusters computer programs based on similar line statement with renamed variables. All of these features can be used to be associated with an assisted feedback especially in rectifying program's mistakes. However, when considering live assistance among big users in a lab session, it will be difficult for an expert to respond on each individual's mistakes. Expert may need to focus his attention towards assisting worst difficulty user rather than least difficulty users. As most of these computer program features are based on cardinal numbers, such sorting is hard to be realized. Thus, ordinal features are needed to enable the ranking-based clustering result. On the other hand, as new debugging resources is costly to be built [66] such expert's assistance should be integrated as a recommended feedback in self-enriching of an automated feedback model.

To our knowledge, there is lacking of automated programming feedback model that integrates continuous expert's feedback in rectifying difficulty of computational programming exercise. Programming difficulty is usually due to either syntax or semantic difficulty. Semantic difficulty deals with subjective problem-solving logics which requires specific feedback from expert user such as a teacher. This semantic difficulty is hard to be modeled as an automated feedback due to the uniqueness of problem specifications and diversity of solution logics.

IV. RECOMMENDED FEEDBACK USING CLUSTERED ASSISTED FEEDBACK

This proposed solution is focusing on continuous-based feedback model on computational programming exercise by integrating assisted feedback as a continuous input to the feedback recommender system. Fig. 1 shows the framework of the model.

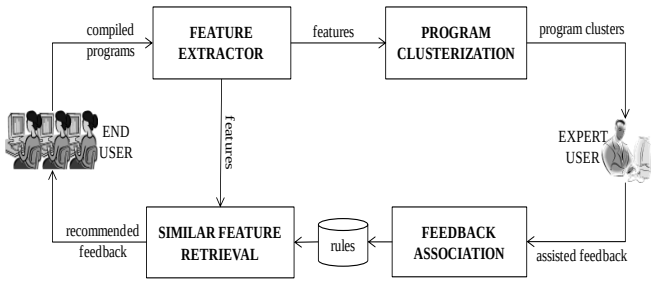

Fig. 1: Integrated computational programming feedback model

Features Extractor

Computer program features have been widely proposed by other researchers, especially in software engineering and plagiarism detection domain, but most of them are meant for a complete or working program and lack of ordinal representation. Ordinal feature is important to enable ranking of computer programs from least to worst program difficulty. In order to access the correctness of a computer program, these features need to be related with related knowledge model such as by using solution templates. The template is a set of correct program instructions (i.e. keywords, symbols, numbers) sequence that can solve a specific computational programming exercise. It can be provided more than once in satisfying variety of correct program answers. In this research, two ordinal features are used to process the incomplete program;

instruction-gram ratio (IGR) with 0 to 3 of skippable instruction and instruction count ratio(ICR) [67]. IGR is the ratio of consecutive instructions or symbols sequence in a program as compared to the template's instructions or symbols sequence with certain skippable instruction(s). Meanwhile, ICR is an average ratio of all unique instructions count in a program that matches with all unique instructions count specified in a template.

Program Clusterization

This process is aimed to sort and cluster current computer programs difficulty in solving a computational programming exercise. It helps an expert user to easily analyze new difficulty and provide feedback to rectify them. Features were selected based on IGR and ICR of computer programs. In this research, GRank K-Means algorithm [68] is used to assign similar computer programs based on their attempt difficulty into ranking-based clusters. Based on each cluster, program that is having similar Euclidean distance to the centroid within a cluster is grouped together. These groups are representing a distinctive rule to be associated with a feedback.

Feedback Association Rules

A rule is formed by the centroid's distance and its centroid's features. Under this rule, there will be a number of related computer programs for an expert to analyze and post a message as an overall assisted feedback for the rule. The scope of this assisted feedback in this proposed solution was more on highlighting missing of general logic structure to solve a specific question. Instead of alerting this message to the programs' owners, this assisted feedback will also be stored for future recommended feedback in rule data (general).

In many cases, computer programs that having nearest distance to the previous rule may be also classified under similar difficulty. In this case, the previous expert user's feedback message can be associated to the computer programs by setting the range distance threshold of the previous rule. The threshold will be used in recommended feedback to automatically detect new cases that may be covered with the similar feedback. Fig. 2 illustrates centroid's distance rule for assisted feedback implementation. If an assisted feedback is given on rule-1, it means that computer program A and B that has distance of 1 from the centroid's features will be associated with the expert user's feedback message. In case computer program C and D in rule-2 to be associated with the same feedback of rule-1, then coverage of centroid's distance need to be updated from 1 to 3. Thus, any computer programs that has distance of 1 to 3 from the centroid's features are now sharing the same of expert's assisted feedback.

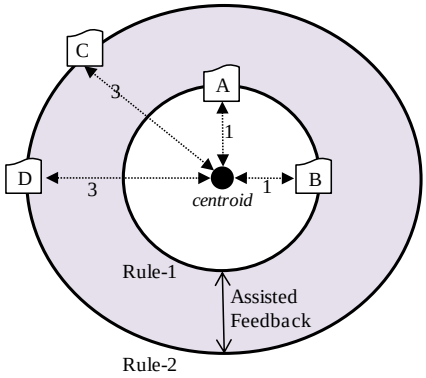

Fig. 2: Centroid-distance threshold rule

Similar feature retrieval

Similar feature retrieval is a process to retrieve recommended feedback to end user based on similar feature that was associated with previous expert's feedback in rectifying incompleteness of the program logics. Fig. 3 shows the process. It starts by receiving a request from end user to compile a computer program. Then, difficulty features of the computer program are extracted using previous IGR and ICR algorithm. These features are used to determine the closeness of the computer program to any of previous features that were already stored with expert's feedback in rule data (general). The distance of extracted features from a computer program to these cluster's features need to be measured using Euclidean distance algorithm. In case the distance is within the defined distance threshold, the associated feedback message of the rule will be presented to end user as a recommended feedback.

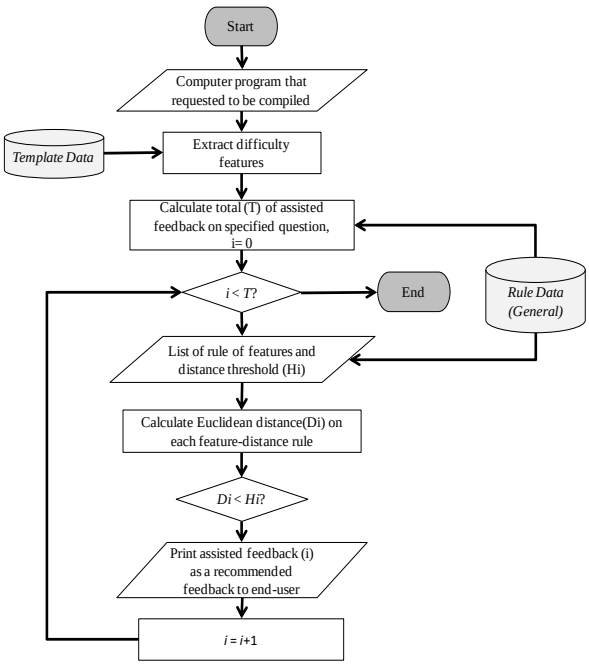

Fig. 3: Similar features feedback retrieval

V. RESULTS NAD ANALYSIS

This experiment is meant to evaluate the effectiveness of an assisted programming feedback in delivering feedback based on groups rather than individuals. It was evaluated based on its capability in enabling an assisted feedback message to be associated with a group of similar program difficulty. An experiment was done based on 67 unsuccessful computer programs answer of a Java lab test provided by [67] for dataset in year 2013. Table 2 shows the list of distinctive groups of the dataset. The groups were formed based on computer programs' similar features of their cluster number and centroid's distance. These similar features were later extracted as a distinctive rule to be associated with a general assisted feedback. In cluster-0, there were 8 computer programs that sharing a similar rule (i.e. same cluster number and same centroid's distance). Then in cluster-1, cluster2 and cluster-3, there were 3, 9 and 11 rules for 6, 17 and 36 computer programs respectively. Overall, 24 rules of distinctive features that were extracted from the 67 computer programs. So instead of individually responding to the 67 programs, expert user's assistance has been successfully minimized up to 50% by only responding to these 24 groups that represent similar program difficulty. Thus, the result shows that the method has successfully grouped computer programs into smaller groups that were possibly be assigned to an assisted feedback in rectifying insufficiency of program logics.

Table 2: Rules association for assisted feedback

Rules	Cluster	Centroid's Features					Centroid's Distance	Frequency
		IGR0	IGR1	IGR2	IGR3	ICR		
1	c3	0.15	0.21	0.25	0.25	0.34	0.09	13
2	c3	0.15	0.21	0.25	0.25	0.34	0.12	8
3	c3	0.15	0.21	0.25	0.25	0.34	0.16	1
4	c3	0.15	0.21	0.25	0.25	0.34	0.17	2
5	c3	0.15	0.21	0.25	0.25	0.34	0.24	1
6	c3	0.15	0.21	0.25	0.25	0.34	0.26	5
7	c3	0.15	0.21	0.25	0.25	0.34	0.30	1
8	c3	0.15	0.21	0.25	0.25	0.34	0.32	1
9	c3	0.15	0.21	0.25	0.25	0.34	0.32	1
10	c3	0.15	0.21	0.25	0.25	0.34	0.38	2
11	c3	0.15	0.21	0.25	0.25	0.34	0.44	1
12	c2	0.42	0.67	0.74	0.75	0.67	0.23	1
13	c2	0.42	0.67	0.74	0.75	0.67	0.24	1
14	c2	0.42	0.67	0.74	0.75	0.67	0.24	1
15	c2	0.42	0.67	0.74	0.75	0.67	0.25	7
16	c2	0.42	0.67	0.74	0.75	0.67	0.33	1
17	c2	0.42	0.67	0.74	0.75	0.67	0.38	2
18	c2	0.42	0.67	0.74	0.75	0.67	0.40	2
19	c2	0.42	0.67	0.74	0.75	0.67	0.43	1
20	c2	0.42	0.67	0.74	0.75	0.67	0.50	1
21	c1	0.83	0.89	0.89	0.89	0.83	0.04	4
22	c1	0.83	0.89	0.89	0.89	0.83	0.08	1
23	c1	0.83	0.89	0.89	0.89	0.83	0.18	1
24	c0	1.00	1.00	1.00	1.00	1.00	0.00	8

Based on these rules, an assisted feedback was provided using a prototype as shown in Fig. 4. The prototype has enabled the expert user to click on any of the rules. When a rule was selected, all the computer programs that matched with the selected rule will be shown. It also highlights detail of ICR that provides an overview of missing instructions on the selected rule. This helps the expert user to quickly post assisted feedback without needing to analyze all programs individually. For instance, all the computer programs in rule-1 were missing the following instructions of template; "for", "length", "if", "else" and "charAt".

Fig. 3: Feedback association screenshot

Table 4 shows all the assisted feedbacks that were provided on selected of seven rules covering the total of 33 computer programs. From the table, rule-1 and rule-2 were sharing the same assisted feedback. In this case, the centroid's range distance was set from rule-1 (0.09415) to rule-2 (0.11575). This can be done through the prototype by inserting the rule-2's distance into the threshold value of rule-1's general assisted feedback. Another similar case of sharing assisted feedback was from rule-15 to rule-16. Thus, the centroid's range distance was set from rule-15 (0.24829) to rule-16 (0.32562). By combining rules with a same feedback, more end users can be entertained in a lab session using a minimum feedback's resource.

Table 3: List of expert's assisted feedback

Rule#	Cluster	Total Program	Instruction Count Template	Feedback

1	c3	13	String=1
			.next=1
			for=0
			.length=0
			if=0
			else=0
			charAt=0
				Use a loop after reading a string input to process its characters. Here is an example to print all characters of a string using a "for" loop.
				String x = input.next();
				for (int i=0; i < x.length; i++){
				System.out.print(x.charAt(i));
				}

2	c3	8	String=1 .next=1 for=0 .length=0 if=0 else=0 charAt=1	Same as rule-1
15	c2	7	String=1 .next=1 for=1 .length=1 if=0 else=0 charAt=1	In the loop, you need to have selection statement (IF/ELSE). If the counter is 1, then you need to print the character at location-1 of the string. Else you need to print an asterisk (*) symbol.
16	c2	1	String=1 .next=1 for=1 .length=1 if=0 else=0 charAt=0	Same as rule-15
17	c2	2	String=1 .next=0 for=1 .length=1 if=0 else=0 charAt=1	Use Scanner next method to read string from user. E.g. Scanner k = new Scanner(System.in); String name = k.next();

Rule#	Cluster	Total Program	Instruction Count Template	Feedback
18	c2	2	String=1 .next=0 for=1 .length=1 if=0 charAt=1 else=0	Same as Rule-17

Based on the given assisted feedback input, recommended feedback output was tested using a prototype. The full list of recommended feedback result generated by the prototype can be downloaded at *https://figshare.com/s/3babe1f21bbeb7dcc75e*. The result showed that out of 475 total submitted programs, 181 of them were provided with recommended feedback. As the input was provided on 33 computer programs or participants, thus the input-output efficiency ratio was 181:33 or 5.5. This ratio may suggest that if an expert user were required to provide feedback assistance on each unsuccessful computer programs, his workload in assisting such similar difficulty during the lab session can be minimized up to 5.5 times. On the other hand, although only 33 participants in were directly assisted by the expert user, other unattended participants still can be automatically assisted with the same feedback through the general recommended feedback implementation. It shows that 39 participants were assisted with a recommended feedback. It means that 6 unattended participants were successfully gain the same feedback of the expert.

VI. DISCUSSION

Assisted programming feedback was effectively delivered to similar groups of computer program. The groups was represented as a distinctive rule for similar difficulties in developing required structure of solution logic. Thus, analysis time can be minimized and assisted feedback to the targeted cluster in solving their current difficulty can be promptly provided. Quick feedback is very important especially when considering the real-time environment of a lab programming session. This is because while the expert user was evaluating a current computer program's difficulty to provide an assisted feedback, the computer program may be already altered by the user few minutes later.

Although providing a general feedback can be efficient in satisfying large group of users, however, the process may be complicated as the nature of computer program solution can be diversified in term of instruction's sequence and choice. In dealing with this issue, more than one solution template need to be provided. Yet, if end user's programs were matched with too variety of templates, it may be difficult for the expert user to conclude a general feedback for a rule. Thus, this technique may not be suitable to be implemented with complex computational programming exercise as the solution logics and sequence are not straight forward. On the other hand, it is also important to create a template that consists of more than five instructions so that more distinctive rules can be generated. By having higher number of distinctive rules, more difficulty cases can be identified and rectified with expert's assisted feedback.

In term of expert user's workload, the instruction-based template was better than complete program-based template especially when considering the diversity of program solution among end users. Instruction sequence can be easily provided and does not rely on programming language syntax. Although the technique is more like a keyword sequence matching, it is more accurate and flexible in associating feedback. Rather than automatically associating feedback with certain keywords, the feedback was given based on expert's analysis on the similar mistakes of computer programs. When analyzing computer programs under certain rule, expert user can see if a computer program was clustered incorrectly due to incorrect use of existing template. Then, the expert user can create new instruction template to accommodate such program's diversity. In contrast, if a feedback already pre-associated with a missing keyword or instruction, all end user's program will be reported as having missing instruction although the user was using a correct alternative instruction. This can create confusion to the end user.

On the other hand, efficiency ratio of general recommended feedback in replicating feedback based on similar mistake may not imply on an effective feedback. As general recommended feedback was based on previous expert feedback message, the effectiveness of this approach may not be achieved if the expert's feedback was given without certain attention and guideline. For example, if general feedback message were given by an expert user to cover only certain number of computer programs in a rule, then other computer programs in the rule may perceive the feedback as unrelated.

In certain case, although the assisted feedback was provided based on similar mistake of the whole computer programs in a rule, however it still can be possibly not related to some computer programs when it comes to general recommended feedback. This inappropriate feedback on certain computer program usually happen when the program was not yet in the cluster list during the assisted feedback rule's association, but later was clustered using the same rule due to features of incorrect solution template. However, this case can be rectified by just providing new solution template to suit with the computer program so that it will be clustered into different rule. By separating the computer program, the general recommended feedback will remain appropriate under the rule. Although it seems to require some workload on expert user in the beginning for an effective general recommended feedback, but definitely the resource will be reduced and no longer needed as time goes by.

VII. CONCLUSION

This research is motivated by the challenges of automated programming feedback models to better assist students in rectifying their difficulties while completing computational programming exercises. Most of the current feedback models for specific problem-solving feedback are based on automated feedback using

pre-designed feedback's content and context. Unfortunately, these static feedback contents cannot effectively help users to rectify their real difficulties, especially in meeting the requirements of specific computational programming exercises. As a result, users are still depending on assisted feedback from expert user through tutor assistant, online discussion and simulated tutor (i.e. artificial intelligence). However, expert user dependency is not efficient when considering the large number of users to be supported. Even designing artificial intelligent is also costly to be built due to the uniqueness of each computational programming question.

Thus, as a solution to these limitations, a new model of automated feedback was proposed by integrating continuous expert user's assisted feedback as a recommender system input. It associates continuous expert user's feedback with the computer program's difficulties of specific computational programming exercise using ranking-based clustering. Computer programs are clustered using ordinal features based on instruction-template ratio. These proposed features enable computer programs to be sorted into rank so that the assisted feedback of an expert user can be effectively provided to the most struggling users within a live lab session. It then automates the assisted feedback to become a recommended feedback using a feature-range distance. The technique has not only recommended the previous feedback based on similar features, but has also successfully provided the same feedback based on the nearest features within the range distance.

ACKNOWLEDGMENT

This research was supported by Research Management and Innovation Centre, Universiti Sultan Zainal Abidin.

REFERENCES

1. J. Hullman, N. Diakorepoulos, E. Momeni, and E. Adar, "Content, context, and critique: Commenting on a data visualization blog," 18th ACM Conference on Computer Supported Cooperative Work and Social Computing, 2015, pp. 1170-1175.
2. R. Jain, And P. Sinha, "Content without context is meaningless," ACM International Conference on Multimedia, 2010, pp. 1259-1268.
3. E. R. Sykes, "Qualitative evaluation of the Java intelligent tutoring system," Journal of Systemics, Cybernetics, and Informatics, 3, 2006, pp. 49–60.
4. M. Hristova, A. Misra, M. Rutter, and R. Mercuri, "Identifying and correcting Java programming errors for introductory computer science students," ACM SIGCSE Bulletin, 35(1), 2003, pp. 153-156.
5. C. L. Jeffery, "Generating LR syntax error messages from examples," ACM Transactions on Programming Languages and Systems, 25(5), 2003, pp. 631-640.
6. J. S. Song, S. H. Hahn, K. Y. Tak, and J. H. Kim, "An intelligent tutoring system for introductory C language course," Computers and Education, 28(2), 1997, pp. 93-102.
7. W. L. Johnson, "Understanding and debugging novice programs," Artificial Intelligence, 42(1), 1990, pp. 51-97.
8. E. R. Sykes, and F. Franek, "An intelligent tutoring system prototype for learning to program JavaTM," 3rd IEEE Intl. Conf. Advanced Technologies, 2003, pp. 1-5.
9. H. C. Lane, Natural language tutoring and the novice programmer. PhD thesis, Pennsylvania: University of Pittsburgh, 2005.
10. M. Suarez, and R. Sison, "Automatic construction of a bug library for object-oriented novice Java programmer errors," International Conference on Intelligent Tutoring Systems, 2008, pp. 184-193.
11. R. Singh, S. Gulwani, and A. S. Lezama, "Automated feedback generation for introductory programming assignments," ACM SIGPLAN Conference on Programming Language Design and Implementation, 48(6), 2013, pp. 15-26.
12. B. E. Vaessen, F. J. Prins, and J. Jeuring, "University students' achievement goals and help-seeking strategies in an intelligent tutoring system," Computers and Education, 72, 2014, pp. 196-208.
13. Online Judge, Welcome to the UVa Online Judge. Available: https://uva.onlinejudge.org/.
14. Topcoder, Home. Available: https://www.topcoder.com/
15. Codechef, Programming competition, programming contest, online computer programming. Available: https://www.codechef.com/#.
16. Codeforces, Home. Available: http://codeforces.com/.
17. A. Papancea, J. Spacco, and D. Hovemeyer, "An open platform for managing short programming exercises," 9th Annual International ACM Conference on International Computing Education Research, 2013, pp. 47-52.
18. Programmr, Get good at programming! Available: http://www.programmr.com/.
19. P. Denny, A. L. Reilly, E. Tempero, and J. Hendrickx, "CodeWrite: Supporting student-driven practice of Java," 42nd ACM Technical Symposium on Computer Science Education, 2011, pp. 471-476.
20. A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel, "Scaffolding students' learning using test my code," 18th ACM Conference on Innovation and Technology in Computer Science Education, 2013, pp. 117-122.
21. W. P. Dann, S. Cooper, and R. Pausch, Learning to Program with Alice (w/CD ROM). New Jersey: Prentice Hall, 2011.
22. K. Charntaweekhun, and S. Wangsiripitak, "Visual programming using flowchart," IEEE International Symposium on Communications and Information Technologies, 2006, pp. 1062-1065.
23. M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, "RAPTOR: A visual programming environment for teaching algorithmic problem solving," ACM SIGCSE Bulletin, 37(1), 2005, pp. 176-180.
24. S. McManus, Scratch Programming in Easy Steps: Covers Versions 2.0 and 1.4. Warwickshire: In Easy Steps, 2013.
25. A. J. Mendes, and M. J. Marcelino, "Tools to support initial programming learning," Int. Conf. Computer Systems and Technologies, 2006, pp. 16-6.
26. P. Brusilovsky, and S. Sosnovsky, "Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK," Journal on Educational Resources in Computing, 5(3), 2005, pp. 1–22.
27. M. Kolling, Introduction to Programming with Greenfoot. New Jersey: Prentice Hall, 2009.
28. G. Fessakis, E. Gouli, and E. Mavroudi, "Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study," Computers and Education, 63, 2013, pp. 87-97.
29. A. Sanmorino, "Development of computer assisted instruction (CAI) for compiler model: The simulation of stack on code generation," IEEE International Conference on Green and Ubiquitous Technology, 2012, pp. 121-123.
30. M. Thompson, "Evaluating the use of flowchart-based RAPTOR programming in CS0," 45th Annual Midwest Instruction and Computing Symposium, 2012, pp. 1-10.
31. S. Xinogalos, "Using flowchart-based programming environments for

simplifying programming and software engineering processes," IEEE Global Engineering Education Conference, 2013, pp. 1313-1322.
32. S. Parkes, C. Ramsay, and A. Spark, "Code Rocket: Improving detailed design support in mainstream software development," IEEE International Conference on Computer and Management, 2011, pp. 1-4.
33. M. A. T. Noranis, and N. S. Azuan, "A multi-agent model for information processing in computational problem solving," International Journal of Modeling and Optimization, 3(6), 2013, pp. 490–494.
34. S. A. Naser, "An agent based intelligent tutoring system for parameter passing in Java programming," Journal of Theoretical and Applied Information Technology, 4(7), 2008, pp. 585–589.
35. A. N. Kumar, "Online tutors for C++/Java programming," ACM SIGCSE Bulletin, 37, 2005, pp. 387.
36. U. Rafique, S. Y. Huang, and C. Y. Miao, "Motivation based goal adoption for autonomous intelligent agents," IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, 2011, pp. 54-57.
37. L. Jerinic, "Teaching introductory programming: Agent-based approach with pedagogical patterns for learning by mistake," International Journal of Advanced Computer Science and Applications, 5, 2014, pp. 60–69.
38. H. Hashim, and S. A. M. Noah, "A framework for semantic forum in e-learning education," IEEE International Conference on Semantic Technology and Information Retrieval, 2011, pp. 77-81.
39. N. Drljevic, and I. Boticki, "Leveraging social networks to increase motivation in learning programming," IEEE Croatian Society Electronics in Marine, 2012, pp. 341-344.
40. C. H. Lai, W. C. Lin, B. S. Jong, and Y. T. Hsia, "Java assist learning system for assisted learning on Facebook," IEEE Learning and Teaching in Computing and Engineering, 2013, pp. 77-82.
41. O. Tsur, and A. Rappoport, "What's in a hashtag? Content based prediction of the spread of ideas in microblogging communities," 5th ACM international conference on Web search and data mining, 2012, pp. 643-652.
42. A. A. Hamed, and X. Wu, "Does social media big data make the world smaller? An exploratory analysis of keyword-hashtag networks," IEEE International Congress on Big Data, 2014, pp. 454-461.
43. C. Treude, O. Barzilay, and M. A. Storey, "How do programmers ask and answer questions on the web? Nier track," IEEE 33rd International Conference on Software Engineering, 2011, pp. 804-807.
44. Y. Wang, H. Li, Y. Feng, Y. Jiang, and Y. Liu, "Assessment of programming language learning based on peer code review model: Implementation and experience report," Computers and Education, 59(2), 2012, pp. 412-422.
45. L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza, "Mining StackOverflow to turn the IDE into a self-confident programming prompter," ACM 11th Working Conference on Mining Software Repositories, 2014, pp. 102-111.
46. C. F. Medina, J. R. P. Pérez, V. M. Á. García, and M. D. P. Ruiz, "Assistance in computer programming learning using educational data mining and learning analytics," 18th ACM Conference on Innovation and Technology in Computer Science Education, 2013, pp. 237-242.
47. A. Casamayor, A. Amandi, and M. Campo, "Intelligent assistance for teachers in collaborative e-learning environments," Computers and Education, 53(4), 2009, pp. 1147-1154.
48. T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," IEEE Transactions on Software Engineering, 31(6), 2005, pp. 429-445.
49. R. M. A. E. Aziz, A. E. Aboutabl, M. S. Mostafa, "Clone detection using DIFF algorithm for aspect mining," International Journal of Advanced Computer Science and Applications, 3(8), 2012, pp. 137–140.
50. G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, and Y. G. Guéhéneuc, "Is it a bug or an enhancement? A text-based approach to classify change requests," Conference of the Center for Advanced Studies on Collaborative Research, 2008, pp. 304-318.
51. M. Alvares, T. Marwala, and F. B. D. L. Neto, "Application of computational intelligence for source code classification," IEEE Congress on Evolutionary Computation, 2014, pp. 895-902.
52. S. Chaki, C. Cohen, and A. Gurfinkel, "Supervised learning for provenance-similarity of binaries," 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2011, pp. 15-23.
53. T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. A. Kofahi, and T. N. Nguyen, "Recurring bug fixes in object-oriented programs," 32nd ACM/IEEE International Conference on Software Engineering, 2010, pp. 315-324.
54. D. Lo, H. Cheng, J. Han, S. C. Khoo, and C. Sun, "Classification of software behaviors for failure detection: A discriminative pattern mining approach," 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 557-566.
55. S. Shivaji, E. J. Whitehead Jr, R. Akella, and S. Kim, "Reducing features to improve bug prediction," IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 600-604.
56. S. W. Thomas, M. Nagappan, D. Blostein, and A. E. Hassan, "The impact of classifier configuration and classifier combination on bug localization," IEEE Transactions on Software Engineering, 39(10), 2013, pp. 1427-1443.
57. X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, and H. Mei, "Can I clone this piece of code here?," 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 170-179.
58. M. R. Sánchez, P. Kinnunen, C. P. Flores, and Á. V. Iturbide, "Student perception and usage of an automated programming assessment tool," Computers in Human Behavior, 31, 2014, pp. 453-460.
59. R. A. P. Queirós, and J. P. Leal, "PETCHA: A programming exercises teaching assistant," 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, 2012, pp. 192-197.
60. H. Mungunsukh, and Z. Cheng, "An agent based programming language learning support system," IEEE International Conference on Computers in Education, 2002, pp. 148-152.
61. E. Stankov, M. Jovanov, A. M. Bogdanova, and M. Gusev, "A new model for semiautomatic student source code assessment," Journal of Computing and Information Technology, 21(3), 2013, pp. 185-194.
62. M. Mojzeš, M. Rost, J. Smolka, and M. Virius, "Feature space for statistical classification of Java source code patterns," IEEE 15th International Carpathian Control Conference, 2014, pp. 357-361.
63. S. Sharma, C. S. Sharma, and V. Tyagi, "Plagiarism detection tool "Parikshak"," IEEE International Conference on Communication, Information and Computing Technology, 2015, pp. 1-7.
64. U. Bandara, and G. Wijayarathna, "Source code author identification with unsupervised feature learning," Pattern Recognition Letters, 34(3), 2013, pp. 330-334.

65. E. L. Glassman, "Interacting with massive numbers of student solutions," 27th Annual ACM Symposium on User Interface Software and Technology, 2014, pp. 17-20.
66. K. Tam, "Debugging debugging," IEEE 35th Annual Computer Software and Applications Conference Workshops, 2011, pp. 512-515.
67. S. Safei, A. S. Shibghatullah, B. M. Aboobaider, and E. F. H. S. Abdullah, “Automated ranking assessment based on completeness and correctness of a computer program solution,” International Journal of Engineering and Technology, 7(28), 2018, pp. 278-283.
68. S. Safei, A. S. Shibghatullah, B. M. Aboobaider, and M. Makhtar, “A hybrid model of ordinal ranking-based clustering using G+Rank K-Means,” International Journal of Engineering and Technology, 7(2), 2018, pp. 41-44.