

Ranking Task Activity in Teaching Software Engineering

Ye Tao, Guozhu Liu

School of Information Science &
Technology
Qingdao University of Science &
Technology
266061, Qingdao, China
ye.tao@qust.edu.cn
lgz_0228@163.com

Jürgen Mottok, Rudi Hackenberg

Laboratory for Safe and Secure
Systems, LaS³
Ostbayerische Technische
Hochschule (OTH) Regensburg
D-93025 Regensburg
juergen.mottok@hs-regensburg.de
rudi.hackenberg@hs-regensburg.de

Georg Hagel

Kempen University of Applied
Sciences
Bahnhofstr. 61
87435 Kempen, Germany
georg.hagel@fh-kempen.de

Abstract— In this research, we investigate the possibility of applying ranking task activity in teaching and learning software engineering courses. We introduce three types of ranking tasks, conceptual-, contextual- and sequential ranking questions, which cover most core topics such as requirement analysis, architecture design and quality validation in the course. We have also done experiments on a group of students to see if ranking tasks could increase their conceptual knowledge in specific areas. Assessments were given in order to evaluate the effectiveness of this activity, showing an obvious increase in complex conceptual understanding.

Keywords — Ranking Task; Software Engineering; Just-in-Time Teaching

I. INTRODUCTION

Ranking Task (RT) is an innovative type of conceptual exercise that asks students to make comparative judgments about a set of variations on a particular situation. It was first introduced by Thomas [1] and applied in physics course [2]. A typical RT activity consists of three parts: i) a description of the problem situation, including any constraints and the basis for ranking different arrangements; ii) a set of figures or instructions showing the different arrangements of the situation to be compared; iii) a place to record the ranking of each variation; iv) a place to explain the reason for each ranking choice.

The idea of introducing RT into software engineering was based on our previous research [3] [4], cooperated by Chinese and German universities for improving software engineering education. Recent years, the course of software engineering is listed as main courses in most Chinese colleges. As a comprehensive and introductory course, it applies both computer science and engineering principles and practices to the creation, operation, and maintenance of software systems. However, different from learning a specific programming language (C/C++/Java), most students found that most abstract concepts of software engineering are difficult to understand, due to the lack of the programming practices and realistic projects experiences. Students in the course are able to easily write some code snippets to solve a particular problem, but they had little or no understanding of the concepts behind it, e.g. design principles/patterns, module cohesion/coupling, and the types of code coverage.

Studies [5] [6] have proved that misconceptions remain highly resistant to change by traditional methods of instruction, and conceptual assignments did reduce the number of student misconceptions. Biggs [7] presented how active learning can help to get students over their misconceptions for learning.

In order to help students to gain conceptual knowledge, we consider the use of RT in teaching some topics in software engineering courses. Basically, this paper describes three types of RT that we designed for students, to clarify their misconceptions in some important areas, and help them to compare these concepts in a context environment. Section II gives a brief introduction to the course and audiences. Section III details three sample RT activities, and Section IV gives the evaluation results. Section V concludes the paper.

II. OVERVIEW OF THE COURSE

At the Qingdao University of Science and Technology (QUST), the major of software engineering is an independent, interdisciplinary program supported by both the faculty and the engineers from industrial enterprises. Graduates of this program will earn a bachelor of software engineering degree.

The course of *Introduction to Software Engineering* is scheduled in the autumn semester of the 3rd year. The problems to solve in the course of software engineering are complex and large. The knowledge area consists of those skills and concepts that are essential to programming practice. As an introductory course, it covers the topics that encompass all phases of the life cycle of a software system, including requirement analysis and specification, software artifacts modeling, quality assessment and control. The body of knowledge with core topics is listed below.

TABLE I. CORE TOPICS IN SOFTWARE ENGINEERING

Seq.	Topic	Coverage Time (hrs)
1.	Overview	4
2.	Software processes and models	8
3.	Software requirements and specifications	8
4.	Structured analysis and design	8

5.	Object-oriented analysis and design	8
6.	Software implementation and validation	8
7.	Software project management	4

When it comes to the design of the curricula, the activating teaching method, Just-in-Time Teaching (JiTT) [8], was applied. We broke up the entire course into smaller modules, and RT activities of each topic are assigned to students in the warm-up phase. After the lecture/peer-instruction phase, a short quiz (usually 5 multi-choice questions) is used to validate the effectiveness of the RT activity.

III. RANKING TASK DESIGN

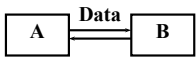
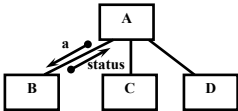
Ranking tasks are exercises that require students to compare scenarios with slightly different configurations. Unlike other types of questions such as multiple choices, short answers and true/false, RT require the student to rely more on conceptual understanding of a problem [9]. They require students to analyse a scenario from a conceptual standpoint. A typical RT will present the student with several variations on a single situation. The student's task is to rank the situation based on a certain criterion from greatest to least or vice-versa. We have designed three different types (conceptual, contextual and sequential) of RT, and each fits a specific situation that covers a certain topic of software engineering. In practice, we found that most of the important conceptions in software engineering can be categorized into the above three classes. Some typical examples are detailed below.

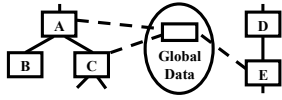
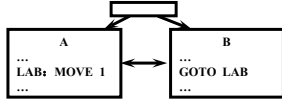
A. Conceptual Type of Ranking Task

Conceptual RT presents a group of similar concepts in software engineering that often lead to misunderstanding for most beginners. The following example is designed to test if the students understand the independence of the modules in high level design.

Question: Coupling refers to how related are two modules and how dependent they are on each other. Being low coupling would mean that changing something major in one should not affect the other. High coupling would make your code difficult to make changes as well as to maintain it. As modules are coupled closely together, making a change could mean an entire system revamp. Rank the following types of coupling, in order of *highest to lowest*.

TABLE II. SAMPLE RANKING TASK ON THE CONCEPT OF COUPLING

Seq.	Coupling Type	Sample Diagram
(A)	Data Coupling	
(B)	Control Coupling	

(C)	Common Coupling	
(D)	Content Coupling	

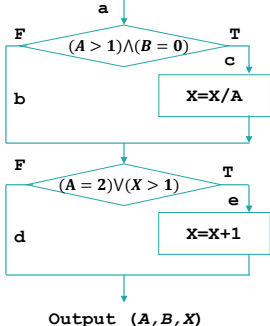
As shown above, in this example, we use diagrams instead of text descriptions to present the ranking items in an intuitive way, to help students identify the concepts that can be easily misunderstood. Similar RT questions can be designed to clarify the concepts on topics of cohesion level of two modules, and progressive refactoring with design patterns.

B. Contextual Type of Ranking Task

Contextual RT gives different conditions and constraints in a context. It provides a question with several contextual similar situations. The situations differ in the value of one or more. The student ranks the situations according to some specified criterion (code coverage scope in this example). Students are asked to give the reasons for their ranking. The ranking order and the explanation provides a window into the student's mind and helps teachers to identify different models for what students are thinking [10].

Question: Code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite. A program with high code coverage has been more thoroughly tested and has a lower chance of containing software bugs than a program with low code coverage. Rank the following test cases *w.r.t.* code coverage, in order of *highest to lowest*.

TABLE III. SAMPLE RANKING TASK OF CODE COVERAGE

A process flowchart of the program to be tested	<p>Input (A, B, X)</p>  <p>Output (A, B, X)</p>																	
	<p>Test Cases</p> <table> <tr> <th>Seq.</th><th>Input/Output (A, B, X)</th><th>Description</th></tr> <tr> <td>(A)</td><td>(2,0,4) / (2,0,3)</td><td>Statement coverage</td></tr> <tr> <td rowspan="2">(B)</td><td>(2,0,4) / (2,0,3)</td><td rowspan="2">Branch coverage</td></tr> <tr> <td>(1,1,1) / (1,1,1)</td></tr> <tr> <td rowspan="2">(C)</td><td>(1,0,3) / (1,0,4)</td><td rowspan="2">Condition coverage</td></tr> <tr> <td>(2,1,1) / (2,1,2)</td></tr> <tr> <td>(D)</td><td>(2,0,4) / (2,0,3)</td><td></td></tr> </table>		Seq.	Input/Output (A, B, X)	Description	(A)	(2,0,4) / (2,0,3)	Statement coverage	(B)	(2,0,4) / (2,0,3)	Branch coverage	(1,1,1) / (1,1,1)	(C)	(1,0,3) / (1,0,4)	Condition coverage	(2,1,1) / (2,1,2)	(D)	(2,0,4) / (2,0,3)
Seq.	Input/Output (A, B, X)	Description																
(A)	(2,0,4) / (2,0,3)	Statement coverage																
(B)	(2,0,4) / (2,0,3)	Branch coverage																
	(1,1,1) / (1,1,1)																	
(C)	(1,0,3) / (1,0,4)	Condition coverage																
	(2,1,1) / (2,1,2)																	
(D)	(2,0,4) / (2,0,3)																	

	(1,1,1) / (1,1,1)	Condition/decision coverage
(E)	(2,0,4) / (2,0,3)	Multiple condition coverage
	(2,1,1) / (2,1,2)	
	(1,0,3) / (1,0,4)	
	(1,1,1) / (1,1,1)	

Contextual RT activity requires students to have relatively strong analytical capacity. In this example, a series of logical reasoning or simple numerical calculations must be done before giving a correct ranking. Another example is giving a brief description a system requirement and its constraints, and a group of applicable development life cycle models (*e.g.* waterfall, rapid prototyping, incremental and *etc.*) and let the student complete the ranking according to the appropriate degree, and give the reasons.

C. Sequential Type of Ranking Task

Sequential RT asks students to figure out the correct chronological order of several stages in software engineering. The following example is designed to test whether the sequence of requirement analysis is well understood.

Question: Processing management plays important roles in software projects. To help the student to clarify the tasks of different development stages, we designed sequential RT. The activities involved in requirements engineering vary widely, depending on the type of system being developed and the specific practices of the organization(s) involved. The following section lists some typical activities for software requirement analysis. Rank these procedures from *earliest* to *latest* on the basis of the chronological sequence.

TABLE IV. SAMPLE RANKING TASK ON THE CONCEPT OF REQUIREMENT ENGINEERING

Seq.	Stage	Description
(A)	Requirements inception	Collecting the requirements of a system from users, customers and other stakeholders.
(B)	Requirements identification	Identifying new requirements.
(C)	Requirements analysis and negotiation	Checking requirements and resolving stakeholder conflicts.
(D)	Requirements specification	Documenting the requirements in a requirements document.
(E)	System modelling	Deriving models of the system.
(F)	Requirements validation	Checking that SRS and models are consistent and meet stakeholder needs.
(G)	Requirements management	Managing changes to the requirements as the system is developed and put into use.

In some scenarios of software development, the sequence is critical, *e.g.* there are generally four recognized levels of software validation, *i.e.* unit- → integration- → system- → operational acceptance testing. A similar RT activity can be designed to help students understand the differences between levels and processes.

IV. DATA AND ANALYSIS

A total amount of 100 third-year students are required to perform a variety of RT for each topic as we discussed in previous sections. RT activities and other types of questions are combined together in a quiz, to see if the RT helped to increase conceptual knowledge in software engineering. All experiments are done on a local learning management system based on Moodle [11]. Table V shows the statistical result over the topic of module independence, as we discussed in the previous section, where 63 students have completed this RT and submitted their answers.

TABLE V. STATISTICAL RESULTS ON MODULE INDEPENDENCY

Q #	Facility index	Standard deviation	Effective weight	Discrimination index	Discriminative efficiency
1	53.51%	45.46%	20.54%	55.46%	72.22%
2	52.61%	42.61%	19.58%	53.73%	67.48%

There are 5 questions in this quiz. Question 1 asks the student to rank the *coupling* level of two modules, as described in Section III, and Question 2 was designed in the same structure and format, except that it tests the concept of *cohesion* level. Other questions are multiple choices selected randomly from the question bank (under the same topic of *module independence*).

Facility index represents the percent of correct answers, where we see that only half of all attempts are correct. Discrimination index is the correlation between the score for this question and the score for the whole quiz. The results show that 55%+ of the students who score highly on these two questions are the same students who score highly on the whole quiz.

We further investigate the details of the ranking items. Fig.1 shows the correct answer ratio of each ranking item in this activity. It can be seen that the concepts of *common coupling* and *content coupling* are well understood, while the rest coupling levels are often misunderstood by most students. This indicates that the instructor should put more attention to emphasize their differences in lectures.

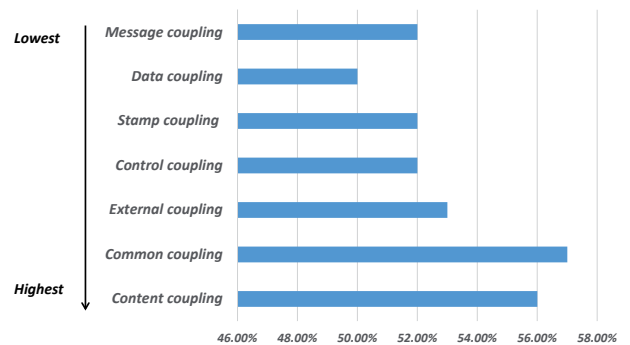


Fig. 1. Details of the ranking items.

Next, we try to evaluate the effects which were induced by RT activities, by comparing the grades/points of a same quiz on the topic of software validation, among three different groups of the attendees. The RT activity used in this evaluation was the same one described in TABLE III. The quiz, for comparison,

only consists of non-ranking-task questions, *e.g.* multiple choices within the same topic (code coverage in this experiment).

Group A did not take the RT activity, Group B partially attended the activity, which means that they did not give rankings on every required item, and Group C completed all parts of the activities, before they take the quiz. As shown in Fig. 2, Group C had an average gain of 1.21 points after the RT activities were introduced.

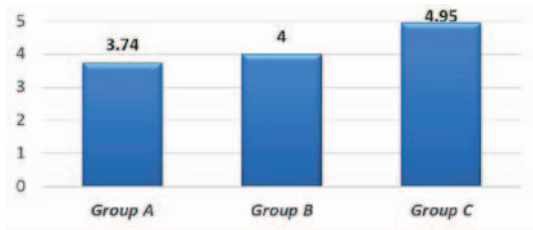


Fig. 2. Comparison between different groups on RT activity. ($N_A = 13$, $N_B = 54$, $N_C = 23$)

We have also taken a post-lecture survey on the overall effects of the RT activity application experiments, to gain an understanding of what the students thought about the RT excises in the course of software engineering, by asking students some interview questions as listed below. We believe that all participants answered the questionnaires honestly as they were aware that the score of the survey would not be included in their final assessment.

TABLE VI. SURVEY QUESTIONS ABOUT THE EFFECTIVENESS OF RT ACTIVITY

Q#	Question	Score
1	Are ranking tasks are more difficult than other types of questions within the same quiz? (Rate Scale: 1~5)	3.8/5
2	Do ranking tasks increase the understanding of conceptual topics? (Rate Scale: 1~3)	1.8/3
3	To what extent will you recommend introducing RT to other software engineering courses? (Rate Scale: 1~3)	2.0/3

Data collected from the semesters while using the RT activities does show that they can increase the conceptual knowledge of students in the study of software engineering. Over 85% students reported that RT activity helped them improve their understanding of difficult concepts of software engineering when compared with conventional question type. Those who have used RT have found that they frequently consider and analyse the scenarios in a natural and intuitive approach, rather than a memorized response, about the behaviour of a given context. However, 60%+ of students complained that some RT questions are more difficult than ordinary types of questions. This is partially because this is the first time we assign RT activities as homework, and some questions are presented in English (to compare the situations between German and Chinese students). All these factors make some students struggled with the idea of the RT. Finally, nearly 80% of the student would like to introduce RT into other courses in software engineering.

V. CONCLUSION

A primary goal of this work is to provide a better way of teaching software engineering by developing a curriculum that integrates RT activities. We have developed and implemented an RT question bank that meets the demands and interests of specific topics within the course of software engineering. Assessment has been conducted for over 100 students involved in this experiment, and the results show that RT increases the students' conceptual understanding.

For the future work, there will be an extended use of those ranking tasks to a bigger group of students. Furthermore, we will develop more RT activities and continue to validate the effectiveness of RT in other computer science and software engineering courses, *e.g.* *Introduction to Database*.

ACKNOWLEDGMENT

This paper is a following research of Chinese-German empirical case study starting from 2013. The research is supported by QUST Teaching Reform Program (No. 01051242), and Shandong Province Graduate Education Innovation Projects (No.SDYC11035). The work is also supported by the Federal Republic of Germany, Federal Ministry of Education and Research, BMBF grant EVELIN “Experimentelle Verbesserung des Lernens von Software Engineering”, grant identity 01PL12022F, project sponsor DLR. The network project EVELIN is part of the “Bund-Länder-Programm für bessere Studienbedingungen und mehr Qualität in der Lehre”. The Universities of Applied Sciences Aschaffenburg, Coburg, Kempten, Landshut, NeuUlm and Regensburg are the network partners. More information under <http://www.las3.de> or <http://www.qualitätspakt-lehre.de>. Results are based on a series of experiments conducted in the autumn semester in 2015. We would like to thank our students, who participated in the presented study.

REFERENCES

- [1] T. L. O’Kuma, D. P. Maloney, and C. J. Hieggelke, *Ranking task exercises in physics* vol. 26: Prentice Hall Upper Saddle River, NJ, 2000.
- [2] J. M. Fraser, A. L. Timan, K. Miller, J. E. Dowd, L. Tucker, and E. Mazur, "Teaching and physics education research: bridging the gap," *Reports on Progress in Physics*, vol. 77, p. 032401, 2014.
- [3] Y. Tao, G. Liu, J. Mottok, R. Hackenberg, and G. Hagel, "Just-in-Time Teaching in software engineering: A Chinese-German empirical case study," in *Global Engineering Education Conference (EDUCON)*, 2014 IEEE, 2014, pp. 983-986.
- [4] Y. Tao, G. Liu, J. Mottok, R. Hackenberg, and G. Hagel, "Just-in-Time-Teaching Experience in a Software Design Pattern Course," in *Global Engineering Education Conference (EDUCON)*, Tallinn 2015, pp. 915 - 919.
- [5] A. Eryilmaz, "Effects of conceptual assignments and conceptual change discussions on students' misconceptions and achievement regarding force and motion," *Journal of Research in Science Teaching*, vol. 39, pp. 1001-1015, 2002.
- [6] S. Gönen, "A study on student teachers' misconceptions and scientifically acceptable conceptions about mass and gravity," *Journal of Science Education and Technology*, vol. 17, pp. 70-81, 2008.
- [7] A. Böttcher, A. Kämper, and V. Thurner, "On Analyzing the Effectiveness of Just-in-Time Teaching."
- [8] J. B. Biggs, *Teaching for quality learning at university: What the student does*: McGraw-Hill Education (UK), 2011.

- [9] A. Savinainen and P. Scott, "Using the Force Concept Inventory to monitor student learning and to plan teaching," *Physics Education*, vol. 37, p. 53, 2002.
- [10] D. W. Hudgins, E. E. Prather, D. J. Grayson, and D. P. Smits, "Effectiveness of collaborative ranking tasks on student understanding of key astronomy concepts," *Astronomy Education Review*, vol. 5, pp. 1-22, 2006.
- [11] J. Cole and H. Foster, *Using Moodle: Teaching with the popular open source course management system*: "O'Reilly Media, Inc.", 2007.