World Transactions on Engineering and Technology Education
Vol.16, No.2, 2018 © 2018 WIETE

Quality and testing - new teaching approaches for software engineers

Dani Almog, Hadas Chasidim & Shlomo Mark

Shamoon College of Engineering
Ashdod, Beer Sheva, Israel

ABSTRACT: System and software testing are important disciplines combining technical aspects with conceptual domains. While the reliability of systems and software is critical, many organisations involved in system and software development do not consider testing as an engineering discipline. Moreover, most testing team members do not have formal training and university curricula, generally, do not prepare graduates to have an understanding of testing design and analysis methods. So, even though proper testing has become more critical to software development, due to the software development paradigm shift (agile, DevOps and continuous applications) there is an understanding that greater proportion of quality assurance and testing is being done by the developers as part of the new development routine. Nevertheless, not much innovation in testing education or testing design is being reported. This article presents a novel approach focused on improving the learning of system and software testing, by introducing methodologies for conducting and assessing the process of education and training.

INTRODUCTION

Software engineering and software quality are relatively new as sub-disciplines. Software engineering (SE) education was designed to prepare software engineers for industry, and quality and testing are important skills needed by software engineers. Software quality contains many elements, methods and concepts that need to be integrated into the SE syllabus. Academic education often struggles to remain up to date and relevant from the industry perspective. This challenge is addressed by two major questions: first - is the current research in SE relevant, innovative? And does it contribute to the direction industry is taking? Second - does one educate and foster students to be holistic thinkers and leaders within industry? This article addresses the later aspect. First, the authors map a standard SE curriculum by looking at two sources of domain knowledge, SWEBOK and SE2014. Then, some means and practical ways to prepare an integrated approach to SE education are suggested.

The article addresses the following matters:

An attempt to investigate the current teaching curricula; map the explosion of knowledge related to software quality and testing; compare it with future needs, while considering the growing gap between what is being taught at the academia and what is needed by the industry, omitting the additional *soft skills* aspect of future SE. The authors propose to elaborate this in a future article.

Software engineers need testing knowledge - this is a proposal for a systematic top down information collection method, which will result in an orderly data set of items ready to use as teaching resources.

Conceptual learning and project orientation in teaching - introduce the topic of formative assessment in education and concept science while introducing meaning equivalence reusable learning objects (MERLO).

Ways to assess the students - methods for exercise and assess student achievement are presented here; the authors demonstrate verity of different tools and means to achieve deeper understanding and appreciation of the student’s level.

Software Engineering Curricula

The IEEE2010 [1] definition states that software engineering is *The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software*.

The SWBOK [2] and SE2014 [3] propose that SE programmes should be able to demonstrate the following qualities:

[Professional Knowledge**]** *Show mastery of software engineering knowledge and skills and of the professional standards necessary to begin practice as a software engineer*.
[Technical Knowledge] *Demonstrate an understanding of and apply appropriate theories, models, and techniques that provide a basis for problem identification and analysis, software design, development, implementation, verification, and documentation*.
[Teamwork] *Work both individually and as part of a team to develop and deliver quality software artefacts.*
[End-User Awareness] *Demonstrate an understanding and appreciation of the importance of negotiation, effective work habits, leadership, and good communication with stakeholders in a typical software development environment.*
[Design Solutions in Context] *Design appropriate solutions in one or more application domains using software engineering approaches that integrate ethical, social, legal, and economic concerns.*
[Perform Trade-Offs] *Reconcile conflicting project objectives, finding acceptable compromises within the limitations of cost, time, knowledge, existing systems, and organizations.*
[Continuing Professional Development] *Learn new models, techniques, and technologies as they emerge and appreciate the necessity of such continuing professional development.*

Figure 1 shows the current knowledge areas in the software engineering education knowledge [4].

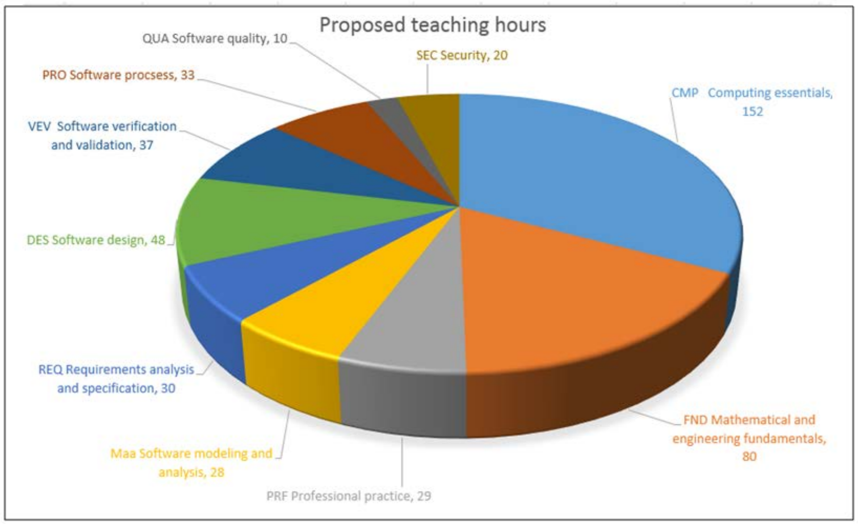


Figure 1: Software engineering education knowledge (SEEK).

QUA (software quality), which is 2% of the SEEK curricula and VEV (verification and validation, a.k.a. testing), which is 7%, are knowledge areas that cover the content being addressed in this article. Today less than 10% of the official SEEK addresses quality and testing directly. A taxonomy of software defects is available from Felderer and Beer [5]. A detailed taxonomy of risk-based testing is presented in Felderer and Schieferdecker [6].

There is a debate about the placement SQE (software quality engineering) within the Software Engineering School - is this a specialisation that merits a separate programme (degree)?, or should one treat it as fundamental knowledge needed to be embedded to all other SE specialisations. The feeling of the authors is that the modern SDLC's and software architecture tend to point out that the new software engineer should internalise all these capabilities as part of their education and the separation of quality and testing may mislead the actual need. Nevertheless, this topic should be dealt with in a separate paper and publication.

The aim of this article is to suggest some new ways to increase the proportion of quality and testing into SE curricula.

WHAT TO TEACH

The setting of an SE curriculum involves addressing several challenges:

Closing the Gap between Industry Needs and Academic Teaching

The academic ecosystem consists of systematic knowledge, composed of existing research, theories and tools that are known and validated. Similarly, the development and the approval of curricula in academia is compatible with this approach. However, in the SE discipline, the pace of technology updates and the related changes in the development processes are accelerated. Therefore, in this particular area, there is a built-in gap between academia and industry, which is relatively difficult to close. However, adapting a more dynamic approach that is fed by continuous feedback derived by both academia and industry can close this gap (See Figure 2). The authors suggest here a dynamic approach to address this gap by the integration of two feedback cycles driven by academia and industry. A common feedback session aims to formulate the mechanism of mutual coordination and adaptation for the two participating parties. Figure 2 presents a detailed view of the feedback cycles affecting industry and academia.

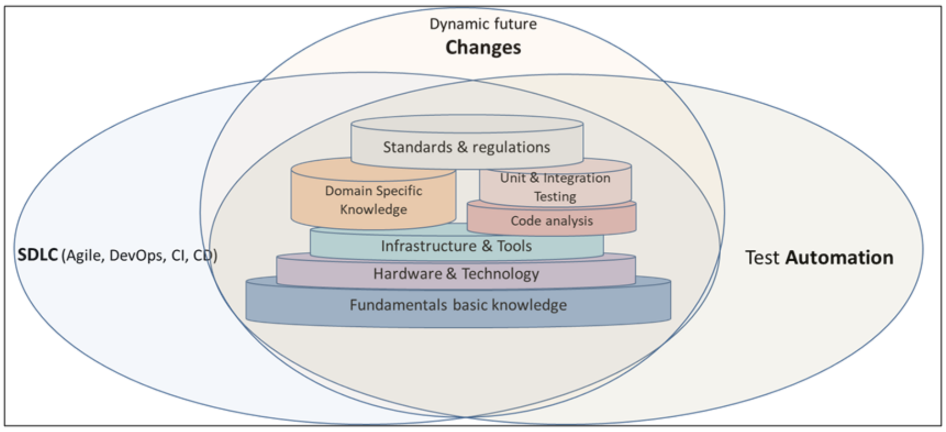


Figure 2: Scope explosion.

It is not so clear how to determine the scope of SE teaching. One might relate to each of the following lists as relevant to software quality and testing. The context of a specific course may dictate a different selection.

All the topics are subject to reconsideration facing the rapid changes the authors are experiencing in areas such as:

- software development life cycle (SDLC) - moving to agile, DevOps, CI, TIP (canary testing);
- new testing techniques and test automation;
- dynamic future changes.

Timing and the order to teach the different topics is crucial in an SE programme. Consider for example the following dilemma: Does one first teach basic testing techniques, such as equivalent classes and boundary value testing, before presenting unit testing? This dilemma presents itself even more when test driven development (TDD) becomes an industry standard software development paradigm in which the development of the test precedes the actual coding.

Another technical issue arises by considering mobile devices - it is obvious that most of the UI now is with mobile devices. However, software development platforms are usually on desktop technologies. One may argue that teaching testing should be aimed at mobile technology. These dilemmas and others will accompany the reader throughout this article.

SOFTWARE ENGINEERS NEED TESTING KNOWLEDGE

Figure 3 demonstrates a top down approach for collecting teaching material for a specific course, considering the dynamic need for curricula adjustment [7]. The authors propose here an approach to collect and update teaching material for SE courses that includes:

- Top down approach demand to select the main topic of a course - enabling its scope to be mapped within the full picture.
- Orderly preparation of teaching materials that serves as a knowledge-sharing mechanism between the different teachers.

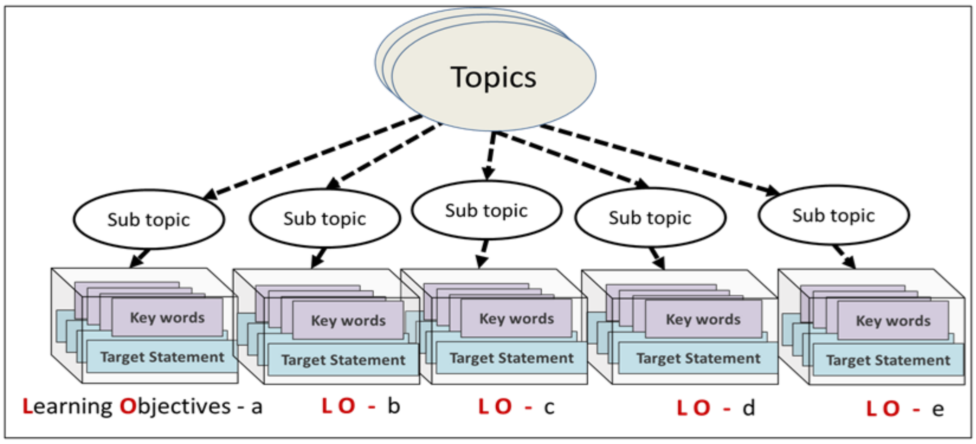


Figure 3: Knowledge collection method.

The list of course topics should be discussed internally, to convey the actual objectives to the students properly. The approach is presented graphically in Figure 3.

The preparation process of a specific item within a sub-topic indicates that this breakdown structure is not rigid and might require additional subdivision levels and a learning objectives (LO) to be added for each sub-topic. For example, when teaching domain testing one can realise the need for another subdivision, so terms such as equivalent classes and boundary values get the proper attention. This mapping is essential for determining the scope of a course and providing the basis for assessing the level of conceptual understanding reached by the students.

CONCEPTUAL LEARNING AND OTHER TEACHING METHODS

Addressing a formative assessment measurement approach is used in education to assess conceptual understanding sometimes also labelled *deep* understanding. Such assessments are used during training or education sessions to provide feedback to the instructor and directly contribute to the learning of students. It is also used for the improvement of material and delivery style. The topic of formative assessment in education and the elements of concept science is introduced first, followed by the MERLO: the meaning equivalence reusable learning objects, with an example on teaching quantitative literacy [8].

Project-Based Learning

Project-based learning (PBL) is a dynamic classroom approach in which students actively explore real-world problems and challenges, and acquire deeper knowledge [9][10]. It emphasises learning activities that are long-term, interdisciplinary and student-centred. Unlike traditional, teacher-led classroom activities, students often must organise their own work and manage their own time in a project-based class. Project-based instruction differs from traditional inquiry by its emphasis on students' collaborative or individual artefact construction to represent what is being learned.

PBL also gives the students the opportunity to explore problems and challenges that have real-world applications, increasing the possibility of long-term retention of skills and concepts. In PBL, students merge previous knowledge with new methodologies and concepts (e.g. agile) to solve problems that reflect the complexity of the real world, while working in an integrated environment. The teacher plays the role of facilitator, working with students to frame worthwhile questions, structuring meaningful tasks, coaching both knowledge development and social skills, and carefully assessing what students have learned from the experience.

Game-based Learning

Game-based learning (GBL) [11] is a type of game that has defined learning outcomes. Game-based learning is designed to balance knowledge with gameplay and the ability of the player to retain and apply said target statement to the real world. Game-based learning describes an approach to teaching in which students explore relevant aspects of games in a learning context designed by teachers. Teachers and students collaborate in order to add depth and perspective to the experience of playing the game.

Good game-based learning applications can draw the user into virtual environments that look and feel familiar and relevant. Within an effective game-based learning environment, one works towards a goal, choosing actions and experiencing the consequences of those actions along the way. One makes mistakes in a risk-free setting, and through experimentation, one actively learns and practices the right way to do things. This keeps one highly engaged in practicing behaviours and thought processes that one can easily transfer from the simulated environment to real life. The environments of gamification also use traditional learning techniques, such as grading assignments, presenting corrective feedbacks and encouraging collaborative projects.

WAYS TO ASSESS THE STUDENTS

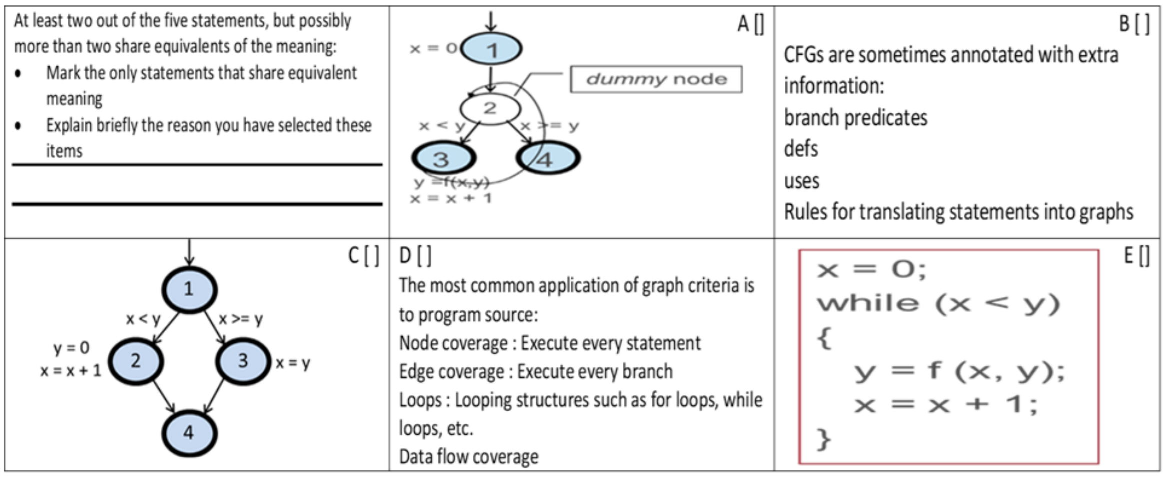
Software engineering is also about being hands-on during the course. It is suggested that a quiz or case study be prepared for each topic (or sub-topic), but preparation of these items may prove to be a tiresome mission. The authors believe in involving the student in the material collection process.

For example, a participation in the MERLO item build-up - starting with a research the Internet using keywords from IEEE standards, SWEBOK, ISTQB glossary, articles and tutorials. This research aims at the collection, development and organisation of MERLO items. The following are some suggestions to the instructor for the preparation of teaching materials. It is possible to have graduate student assignments include the collection and preparation of MERLO items. A sample assignment is:

1. Map and arrange learning material (scope - into topics) - suggested to be done as part of the curriculum preparation.
2. Divide into major sub-topics (learning chapters) - these major sub-topics will eventually become the actual learning goals and targets of the course.
3. Breakdown each chapter within the sub-topics into target statements (TS). Student participation in the preparation of TS could be done as a group assignment, so active discussions and debates on the content will become part of the learning procedures.
4. Prepare at least one MERLO items for each TS.

Formative Assessment using MERLO

The application of formative assessment measurement typically involves more than one sample MERLO item for each sub-topic taught in the course. Figures 4 is an example of MERLO item developed for a course on code analysis. It is recommended to get students involved in their preparation as an active study act.


At least two out of the five statements, but possibly more than two share equivalents of the meaning:

- Mark the only statements that share equivalent meaning
- Explain briefly the reason you have selected these items

A []

B []

CFGs are sometimes annotated with extra information:
branch predicates
defs
uses
Rules for translating statements into graphs

C []

D []

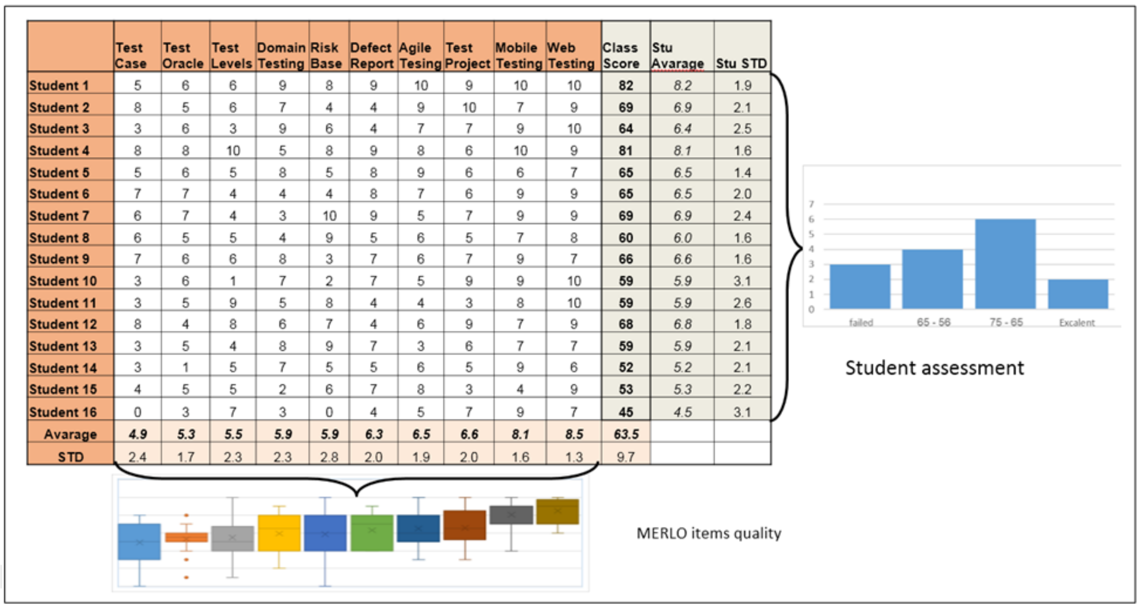
The most common application of graph criteria is to program source:
Node coverage : Execute every statement
Edge coverage : Execute every branch
Loops : Looping structures such as for loops, while loops, etc.
Data flow coverage

E []

```
x = 0;
while (x < y)
{
   y = f (x, y);
   x = x + 1;
}
```

Figure 4: MERLO examination on CFG.

The authors analysed MERLO scores as a teaching aid improvement mechanism and an opportunity for the teacher to improve his teaching skills. Figure 5 lists the results from a MERLO-style examination. Presenting the scores by topics, in addition to actual student scores, enables us to evaluate the quality of the MERLO items prepared - or alternatively to evaluate class knowledge of each sub-topic.



	Test Case	Test Oracle	Test Levels	Domain Testing	Risk Base	Defect Report	Agile Tesing	Test Project	Mobile Testing	Web Testing	Class Score	Stu Avarage	Stu STD
Student 1	5	6	6	9	8	9	10	9	10	10	**82**	*8.2*	1.9
Student 2	8	5	6	7	4	4	9	10	7	9	**69**	*6.9*	2.1
Student 3	3	6	3	9	6	4	7	7	9	10	**64**	*6.4*	2.5
Student 4	8	8	10	5	8	9	8	6	10	9	**81**	*8.1*	1.6
Student 5	5	6	5	8	5	8	9	6	6	7	**65**	*6.5*	1.4
Student 6	7	7	4	4	4	8	7	6	9	9	**65**	*6.5*	2.0
Student 7	6	7	4	3	10	9	5	7	9	9	**69**	*6.9*	2.4
Student 8	6	5	5	4	9	5	6	5	7	8	**60**	*6.0*	1.6
Student 9	7	6	6	8	3	7	6	7	9	7	**66**	*6.6*	1.6
Student 10	3	6	1	7	2	7	5	9	9	10	**59**	*5.9*	3.1
Student 11	3	5	9	5	8	4	4	3	8	10	**59**	*5.9*	2.6
Student 12	8	4	8	6	7	4	6	9	7	9	**68**	*6.8*	1.8
Student 13	3	5	4	8	9	7	3	6	7	7	**59**	*5.9*	2.1
Student 14	3	1	5	7	5	5	6	5	9	6	**52**	*5.2*	2.1
Student 15	4	5	5	2	6	7	8	3	4	9	**53**	*5.3*	2.2
Student 16	0	3	7	3	0	4	5	7	9	7	**45**	*4.5*	3.1
Avarage	***4.9***	***5.3***	***5.5***	***5.9***	***5.9***	***6.3***	***6.5***	***6.6***	***8.1***	***8.5***	***63.5***		
STD	2.4	1.7	2.3	2.3	2.8	2.0	1.9	2.0	1.6	1.3	9.7		

Figure 5: MERLO scores results.

MERLO items are a powerful tool for assessing student learning. It is standard practice to combine MERLO items with an open practical implementation assignment - there are students who express their knowledge and skills better in a problem solution questioner. A final examination may have two parts: a MERLO item quiz and a real-world practical assignment.

SUMMARY

The article covers various aspects of teaching and knowledge acquisition. It provides concrete examples and aggregated experience that can be considered as a door opener to further advances in the teaching of software engineering in general, and testing in particular:

1) Mutual dependence of academia and industry - industry/academia interactions work in two directions. One needs to synchronise efforts for mutual benefit.

2) Dynamically update content, scope and techniques - the question of what to teach was discussed in this article It is like chasing a rapidly moving target. It seems vital, but not obvious, to collect, organise and store the teaching materials. Another consideration is to consider the engineering and practical nature of software testing. This affects the practical versus conceptual educational trade-offs.
3) The MERLO approach contributes to upgrading testing teaching - adapting a deeper understanding of the scope content provides the foundation for new implementation of software testing teaching. Considering the practical demand from software testing professionals is supported by combining MERLO items' assessment together with an open practical implementation of exercises and assignments.
4) Flexibility, innovativeness, openness - all these qualities must be considered and addressed during a software testing course build-up. It is all tied up with the ability to expose and influence the rapid changes in the industry.

REFERENCES

1. IEEE 2010 (2010), ISO/IEC/IEEE 24765:2010 Systems and Software Engineering Vocabulary. Lederman, D., Affirmative Action, Innovation and the Financial Future: A Survey of Presidents, Inside Higher Ed, 1 March 2013, http://www.insidehighered.com/news/survey/affirmative-action-innovationandfinancial-future-survey-presidents.
2. SE2014 (2015), Software Engineering Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, Joint Task Force on Computing Curricula IEEE, Computer Society Association for Computing Machinery, 30 December 2017, https://www.acm.org/binaries/content/assets/education/se2014.pdf
3. SEEK (2013), Software Engineering Education Knowledge, The Educational Activities Board of the IEEE Computer Society and the ACM Education Board, 30 December 2017, http://sites.computer.org/ccse/know/FinalDraft.pdf
4. SWEBOK, Guide to the Software Engineering Body of Knowledge, Bourque, P. and Fairley, R.E. (Eds), Version 3.0, IEEE Computer Society (2014).
5. Felderer, M. and Beer, A., *Using Defect Taxonomies to Improve the Maturity of the System Test Process: Results From an Industrial Case Study*. In: Winkler, D., Biffl, S. and Bergsmann, J. (Eds), Software Quality. Increasing Value in Software and Systems Development. SWQD 2013. Lecture Notes in Business Information Processing, Berlin, Heidelberg: Springer, 133, 125-146 (2013).
6. Felderer, M. and Schieferdecker, I., A taxonomy of risk-based testing. *Inter. J. on Software Tools for Technol. Transfer*, 16, **5**, 559-568 (2014).
7. Hazzan, O., Lapidot, T. and Ragonis, N., *Guide to Teaching Computer Science: an Activity-based Approach*. Springer (2015).
8. Kenett, R.S., Ruggeri, F. and Faltin, F. (Eds), *Analytic Methods in Systems and Software Testing*. Chichester, UK: John Wiley and Sons (2018), https://www.wiley.com/en-us/Analytic+Methods+in+Systems+and+Software+Testing-p-9781119271505
9. Terenzini, P.T., Cabrera, A.F., Colbeck, C.L., Parente, J.M. and Bjorklund, S.A., Collaborative learning vs. lecture/discussion: students' reported learning gains. *J. of Engng. Educ.*, 90, **1**, 123 (2001).
10. Fosnot, C.T., *Constructivism: Theory, Perspectives, and Practice*. Teachers College Press (2013).
11. Kumar, D. (2013), 20 September 2017, http://edtechreview.in/dictionary/298-what-is-game-based-learning