EasyChair Preprint

№ 5491

Guidelines for Software Testing Education Objectives from Industry Practices with a Constructive Alignment Approach

Timo Hynninen, Jussi Kasurinen, Antti Knutas and Ossi Taipale

May 10, 2021

Cite as:
Hynninen, T., Kasurinen, J., Knutas, A., & Taipale, O. (2018, July). Guidelines for software testing education objectives from industry practices with a constructive alignment approach. In Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (pp. 278-283). ACM.

This is a pre-print version of the published article. Please cite the published version at: https://dl.acm.org/citation.cfm?id=3197108

Guidelines for Software Testing Education Objectives from Industry Practices with a Constructive Alignment Approach

Timo Hynninen
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Finland
timo.hynninen@lut.fi

Jussi Kasurinen
South-Eastern Finland University of Applied Sciences (XAMK)
Finland
jussi.kasurinen@xamk.fi

Antti Knutas
Lero – The Irish Software Research Centre
Ireland
antti.knutas@lero.ie

Ossi Taipale
Lappeenranta University of Technology
P.O. Box 20
FI-53851 Finland
ossi.taipale@lut.fi

ABSTRACT

Testing and quality assurance are characterized as the most expensive tasks in the software life cycle. However, several studies also indicate that the industry could enhance product quality and reduce costs by investing in developing testing practices. Software engineering educators can bridge the gap between formal education and industry practices to produce more industry-ready graduates, by observing the industry in action. To find out the current state of industry, we conducted a study in software organizations to assess how they test their products and which process models they follow. According to the survey results, the organizations rely heavily on test automation and use sophisticated testing infrastructures, apply agile practices even when working with mission-critical software, and have reduced the use of formal process reference and assessment models. Based on the results, this paper identifies a number of key learning objectives in quality assurance and software testing disciplines that the industry expects from university graduates. The principles of constructive alignment are used to present learning goals, teaching methods, and assessment methods that align with the industry requirements.

CCS CONCEPTS

• **Social and professional topics** → Software engineering education; • **Software and its engineering** → Software verification and validation

KEYWORDS

Curriculum, software testing, constructive alignment

ACM Reference format:

Timo Hynninen, Jussi Kasurinen, Antti Knutas and Ossi Taipale. 2018. Guidelines for Software Testing Education Objectives from Industry Practices with a Constructive Alignment Approach. In Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18). ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3197091.3197108

1 INTRODUCTION

Testing is an important part of software life cycle, as inadequate testing and quality assurance practices in can cause substantial immediate costs as well as poor quality and high maintenance products. Proper testing education can improve software quality, for example students more experienced in testing produce more reliable code [1], [2].

Constructive alignment is an outcomes-based approach to teaching in which the learning outcomes that students are intended to achieve are defined before teaching takes place [3]. Teaching and assessment methods are then designed to best achieve those outcomes and to assess the standard at which they have been achieved. The teaching environment, practices and evaluation should support learning goals and the student's future environment [4].

In order to align the testing education content with industry practices, the following research questions were formulated: 1) Which testing tools and technologies are most used in the industry? 2) What are the current issues related to testing in the industry? 3) How should the learning goals, teaching methods and evaluation methods in a software testing course constructively aligned with current industry practices?

Rest of the paper is structured as follows: Section 2 introduces related work in testing education and testing surveys. Section 3 describes the research process. The survey and its results are presented in Section 4. The constructive alignment model and guidelines are given in Section 5. Finally, we conclude in Section 6.

2 RELATED RESEARCH

There are several studies that implicitly investigate software testing education from a constructive or a requirements alignment perspective without explicitly citing the theory. For example, a study by Krutz et al. [5] investigates motivational issues and placing students into a more real-life like environment that supports learning. The studied issue was that students tend to think testing and quality assurance work as boring and unnecessary extra work. In the Krutz et al. study, the motivational problems were addressed by applying real open-source software projects as the course assignments to give the tasks more realistic scope and scale. Based on their results, 85 percent of their students considered this approach to be positive, with student feedback also indicating improved motivation and learning results. Similar observations were also reported in a study by Garousi and Mathur [6], which also observed that it is not uncommon for a computer science degree program to omit the concept of quality assurance and software testing from their course curricula. In another similar study Broman et al. [7] discuss aligning software testing course with real world practices, and explicitly use the theory of constructive alignment.

Another set of studies create requirements for software testing courses and present ways to align courses with these interventions. A study by Smith et al. [8] discusses the general requirements for developing a testing course: the university course has to be fun and competitive, allow students to learn from each other, the assignments have to demonstrate the importance of doing testing work, and provide an example of the scale and difficulty of the real-world quality assurance issues. They also present an example intervention where a course is changed to align with. These are important considerations, because for example in study aligning course curricula with the games industry [9], it was established that the academia and the industry do not share a common view on what are the necessary and important skills for the students to possess, especially when considering more theoretical topics beyond the set of taught programming languages. In this sense, it would be important to collect information on the tools and strategies applied by the industry, in the development of a course with industry-applicable experience, especially since the more refined testing tools applied by the industry require domain-related expertise [6] and which may actually be difficult or expensive to acquire without support from the degree program [10].

3 RESEARCH PROCESS

The objective of this study was to align testing education with industry practices and needs. We used surveys as the primary research method to study the industry, as surveys are used to collect information from people about their feelings and beliefs [11]. We consider the constructive alignment approach as an exploratory study for which the survey method is appropriate [12].

We used the questionnaire form introduced in Kasurinen et al. [13] and originally designed in 2005 [14] to get information about the respondents' organization profile, testing practices, test process maturity, applied process models and the tasks related to software development. The questionnaire comprised of eleven chapters containing multi-item, multi-choice questions and open-ended questions. The multi-item questions used a five-point Likert scale (1 fully disagree - 3 neutral - 5 fully agree). The reliability of the multi-item questions in the chapters were originally estimated by using the Cronbach alpha coefficient. In addition to the original 2009 questions, we added new questions about the costs of maintenance and product support.

The sampling method was convenience sampling, with as wide reach as possible within the industry. We advertised the survey in social media platforms such as LinkedIn, Facebook, Twitter and Researchgate and by direct contacts to our industrial partners and open calls for participation in several public online discussion channels. We used advertisement channels to get responses especially from our alumni, and asked them to also share the survey on social media. In order to avoid an extremely biased and small sample anyone working in the software industry was welcome to take part.

The questionnaire collected 33 responses from individuals in working in different organizations. The survey form was opened 930 times (by unique clients or IP-addresses) resulting in a response rate of 3.5 percent which is fairly normal for Internet surveys [11]. To estimate the sample size, we used publicly available statistics provided by the Ministry of Economic Affairs and Employment of Finland [15]. According to the latest estimate, there were 3360 companies in the software business sector, making the sample size approximately 1 percent of the Finnish software industry.

Finally, we used the recommendations for constructive aligned teaching [3], [4] to derive learning goals for industry practices that were collected in the survey. From there a set of teaching methods and the performances of understanding required for evaluation were designed, informed by the same recommendations. They are summarized follows:

- Learning goals should be clear, serve a purpose, and set in advance.
- Students need to be placed in situations and environments that elicit the required learnings, with declarative teaching minimized.

- Students are then required to provide evidence, either by self-set or teacher-set tasks, as appropriate, that their learning can match the stated objectives.

4 SURVEY RESULTS

Questions in the survey addressed the testing and quality assurance practices, the tools used to support these activities as well as development practices and problems.

In terms of organizational profile, very small, small and medium-sized organizations represented each about 21 % of the participants, while 36.4 % were large or very large (more than 250 employee) organizations. Approximately eighty percent of the organizations were private companies, the rest being government or nonprofit organizations. Respondents from organizations focusing primarily on national operations formed 21.2 % of the total while 39.3 % focused mostly on international business. Respondents in 51.5 % of the organizations reported that product fault could cause remarkable economic losses, and 18.2 % considered themselves primarily as open source developers. The majority of the respondents (66.7 %) were primarily software developers, 12.1% were managers and 15.2% worked in quality assurance.

The first chapter of the questionnaire was about the application level of different software testing tools. A tool was defined as "any application, framework, web service, extra library, feature of your development environment etc. whichever supports the activity in question." The four most popular tool categories include defect reporting, test automation, unit testing and defect/code tracing tools, which are used by over half of all surveyed organizations. Table 1 presents the number of used testing tools.

The second chapter of the questionnaire consisted of multi-choice questions about the severity of test and quality assurance problems. The questions covered topics such as which issues slow down the development, which issues currently restrict testing, and how well current testing tools support development needs. The issues in the questions were originally identified in 2009 [13]. The results indicate that the configurability of the testing tools is a common issue. In addition, feature development in the late phases of development can have an effect on testing schedule, and insufficient communication can slow down defect fixing. Another problem highlighted from the responses was that testing personnel do not have enough expertise in certain testing applications.

The third and fourth chapter of the survey addressed software processes and the amount of agile practices in the organizations. In general, the results indicate that the industry is quite confident in the use of agile practices. The industry drive towards agile can also be observed from the questions concerning the use of formal process models such as SPICE (software process assessment, ISO/IEC 15504, currently part of the ISO/IEC 33000 series) [16] or software testing standard (ISO/IEC 29119) [17]. The questions covered also the utilization of capability and maturity models, such as TMMi - test maturity model integrated [18] or CMMi – capability maturity model integrated [19]. Some form of process model (formal or self-defined) was applied by only 21.2 percent of organizations, while according to the respondents none of the organizations applied capability or maturity certificates. V-model, acceptance criteria for tickets and "generic agile" were mentioned by name, all based on best practices collected from various sources and "self-defined". No standard, model or certificate program was directly named. Also, in some organizations individual employees are unsure about the application of process models or capability certificates.

Table 1: The percentage of applied testing tools in the industry.

Tool	Percentage of respondents
Bug/Defect reporting	72.7 %
Test automation	66.7 %
Unit testing	57.6 %
Bug/Code tracing	57.6 %
Performance testing	48.5 %
Test case management	45.5 %
Integration testing	45.5 %
Virtual test environment	42.4 %
Quality control	36.4 %
Automated metrics collector	36.4 %
System testing	27.3 %
Security testing	24.2 %
Test completeness	24.2 %
Test design	15.2 %
Protocol/Interface conformance tool	9.1 %

The final chapter in our survey included several questions concerning the software testing and quality assurance practices. Respondents were asked to evaluate how well different statements about development practices fit their organizational unit on a scale of 1 (fully disagree) to 5 (fully agree). The statements and survey responses in are presented in Table 2. We present mode as the primary indicator for the individual statements, as the survey used an interval Likert scale. The organizational units are more confident on their system level quality assurance (system, acceptance) testing than on the unit or integration level testing. Organizational units are also confident that they are building the product right, and at the same time, building the right product. Testing schedules may not be kept (mode 2, partially disagree) and time is not necessarily allocated enough for testing (mode 2, partially disagree). Code review practices are varying between different organizations (mode 1, fully disagree).

In addition to multi-choice questions the survey contained open-ended questions, where respondents were asked to explain how their organization manages testing and maintenance related effort. The following themes in managing testing-related work were highlighted from the open responses:

- Moving from proprietary software to open source
- Increasing the coverage of automated tests
- Focusing on service scalability in design
- Re-implementing legacy applications
- Setting up dedicated testing and development environments
- Offshoring testing work
- Establishing pre-planned maintenance time for projects, during which last defects are fixed
- Forming dedicated maintenance teams
- Emphasizing the responsibility of current developers
- Employing a risk-based testing approach to cover the most critical components rather than trying to get perfect coverage.

5 DISCUSSION AND IMPLICATIONS

To answer the first research question, *which testing tools and technologies are most used in the industry*, the most common tools in 2017 were defect-reporting, unit testing and test automation tools. Test case management and test design tools were the categories with decreasing usage. Test automation tools are popular on every level of automation (data collection, performance, general automation and tracing). Automated testing is considered cheap. However, the quality and coverage of testing is a concern to some developers.

In terms of the second research question, *issues related to testing in the industry*, the configurability of testing tools and personnel not being familiar with certain testing tools were common issues according to the survey. Although it is unclear if the respondents meant personnel not being familiar with a particular application their company uses or with tools of a particular type, this result highlights the importance of having students use a variety of tools already during their studies. The test process follows a certain path, executing the test phases regardless of the project limitations. Emphasis is put on the late phases, such as acceptance testing phase. Some form of a systematic process or method in testing is followed by 21.2% of the software companies even though over half of the companies use the most common testing tools.

Interestingly, the static testing practices are very varying between our respondents. While some organizations keep code reviews and go through checklists, about half of the responses say the opposite. One possible explanation for this result may be the fact that there were many respondents from small companies who employ extreme agile development processes and have not yet established formal processes for code reviews, walkthroughs or checklists.

The third research question, *constructively aligning a software testing education*, is addressed next. In Table 3 we present an initial design for a software testing course whose learning goals, teaching methods, and evaluation methods have been constructively aligned based on the industry survey results. In this design we aim to minimize declarative teaching, place students in environments that elicit required learnings on software testing and evaluate with "performances of understanding," as recommended in the guidelines by Biggs [3], [4]. It should be noted that the model presented is not exclusive. In other words, we recommend including listed topics in software testing education, but do not recommend excluding any topics that we do not list.

Additionally, we suggest the following guidelines for constructive alignment of testing curriculum:

- Incorporate the use of the most common testing tools, defect reporting, unit testing and test automation, into the curriculum. The students will most likely require the skill to use these tools in their future workplace.

Table 2: The self-assessment of the testing and quality assurance practices (1 fully disagree – 3 neutral – 5 fully agree).

Question	Average	Mode
Our software correctly implements a specific function. We are building the product right.	4.1	4
Our software is built traceable to customer requirements. We are building the right product.	3.8	5
Our formal inspections are OK.	3.4	4
We go through checklists.	3.0	2
We keep code reviews.	3.2	1
Our unit testing (modules or procedures) is excellent.	2.9	4
Our integration testing (multiple components together) is excellent.	3.0	3
Our usability testing (adapt software to users' work styles) is excellent.	3.0	3
Our function testing (detect discrepancies between a program's functional specification and its actual behavior) is excellent.	2.9	3
Our system testing (system does not meet requirements specification) is excellent.	3.4	3
Our acceptance testing (users run the system in production) is excellent.	3.6	4
We keep our testing schedules.	3.2	2
Last testing phases are kept regardless of the project deadline.	3.0	4
We allocate enough testing time.	2.6	2

Table 3: The constructive alignment of software testing course goals and methods to industry practices.

Learning goals	Teaching methods	Assessment methods ("performances of understanding")
Learn the practice of defect reporting and the use of bug tracking tools	Individual exercises: Find and report bugs.	Demonstrate understanding through the individual projects
Implementing unit tests and evaluating test coverage	Individual exercises: Create a program and set up unit tests	
Independent implementation of test automation	Individual exercises: Set up full testing automation for a program	
Understand and apply test process design in future projects	Teamwork: Project management exercise and testing process simulation	Demonstrate understanding through equal contribution to the teamwork project (individual and group evaluation)
Integrating testing phases to software engineering practices	Teamwork: Project management exercise; acceptance testing between two teams	
Evaluating and managing technical debt; making rational compromises	Teacher-led exercise: A review of the shortcuts taken during the course, and discussion & evaluation of the long-term drawbacks of the shortcuts	Demonstrate understanding by a written assignment that reviews and evaluates technical issues
Implementing static testing: Creating checklists and performing code reviews	Teamwork: Going through checklists and reviewing each other's code. TA acts as QA manager in final projects	Demonstrate understanding by working in a simulated verification and validation review

- Use popular, widely used testing tools rather than tools designed for education, in order to teach students the correct use and configuration of real environments.
- Emphasize the importance of static testing methods as the way to improve code quality.
- Produce documentation early on to encourage a mindset for documenting the progress of the project.
- Use a variety of tools for the same purpose to give students experience of the different tools available.
- Enforce documentation practices to enhance the communication skills, for example producing and handling defect reports.

The ACM computer science curricula places testing skills in the knowledge area of software development fundamentals. Verifying program correctness is an extensive topic in the core contents of the recommendation. Testing activities in the ACM software engineering curricula are mainly under the Software Verification and Validation knowledge area, although testing themes span across multiple areas of knowledge such as Software process or Software quality. Although the ACM curricula recommendations cover testing well, they have been criticized for not providing students a rigorous enough testing mindset [20].

6 CONCLUSIONS

In this paper, we presented the alignment of software testing education goals to industry practices. We observed the industry by conducting a survey on testing tools and quality assurance practices. The survey results indicated a strong preference towards agile development practices and high use of automation. Moreover, the use of formal process reference and assessment models was in the minority. In addition, the survey results ranked the popularity of different testing tools, which directly benefits the software engineering educators.

The survey results were used to constructively align software testing education with industry practices and expectations, producing a course model that responds to industry needs. The presented model can be used as a frame of reference for the learning objectives related to testing work in computer science education. Additionally, a number of guidelines for actual course content were presented.

The study addressed a similar issue as in Krutz et al. [5] and Broman et al. [7], though from a different perspective. We took a step back and gather requirements and learning objectives for a course on software testing, rather than investigate how the requirements can be used to constructively align a course. This approach is similar to the work of Garousi and Mathur [6] who performed a review as well, though they surveyed existing degree programs instead of the industry.

The limitations of the study warrant some discussion. The sampling of our survey was limited to a one country, and for this reason the results are not strong and confirmatory. However, we consider the survey results as exploratory from which estimates can be drawn.

In future work the actual learning activities and course organization should be addressed. One topic of interest could be the alignment of actual software testing activities with the different phases of software life cycle.

ACKNOWLEDGMENTS

This work was partially funded by the Technology Development center of Finland (TEKES), as part of the .Maintain project

(project number 1204/31/2016). The work of the third author was supported by the Ulla Tuominen foundation.

REFERENCES

[1] O. A. L. Lazzarini Lemos, C. Cutigi Ferrari, F. Fagundes Silveira, and A. Garcia, "Experience report: Can software testing education lead to more reliable code?," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 2015, pp. 359–369.

[2] O. A. Lazzarini Lemos, F. Fagundes Silveira, F. Cutigi Ferrari, and A. Garcia, "The impact of Software Testing education on code reliability: An empirical assessment," *Journal of Systems and Software*, Mar. 2017.

[3] J. Biggs, "Constructive alignment in university teaching," *HERDSA Review of Higher Education*, vol. 1, no. 5, pp. 5–22, 2014.

[4] J. Biggs, "Enhancing teaching through constructive alignment," *Higher education*, vol. 32, no. 3, pp. 347–364, 1996.

[5] D. E. Krutz, S. A. Malachowsky, and T. Reichlmayr, "Using a Real World Project in a Software Testing Course," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, New York, NY, USA, 2014, pp. 49–54.

[6] V. Garousi and A. Mathur, "Current State of the Software Testing Education in North American Academia and Some Recommendations for the New Educators," in *2010 23rd IEEE Conference on Software Engineering Education and Training*, 2010, pp. 89–96.

[7] D. Broman, K. Sandahl, and M. A. Baker, "The company approach to software engineering project courses," *IEEE Transactions on Education*, vol. 55, no. 4, pp. 445–452, 2012.

[8] J. Smith, J. Tessler, E. Kramer, and C. Lin, "Using Peer Review to Teach Software Testing," in *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, New York, NY, USA, 2012, pp. 93–98.

[9] M. M. McGill, "Defining the Expectation Gap: A Comparison of Industry Needs and Existing Game Development Curriculum," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, New York, NY, USA, 2009, pp. 129–136.

[10] F. Kazemian and T. Howles, "A Software Testing Course for Computer Science Majors," *SIGCSE Bull.*, vol. 37, no. 4, pp. 50–53, Dec. 2005.

[11] A. Fink, *How to Conduct Surveys: A Step-by-Step Guide*. Sage Publications, 2012.

[12] B. A. Kitchenham *et al.*, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on software engineering*, vol. 28, no. 8, pp. 721–734, 2002.

[13] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: empirical observations," *Advances in Software Engineering*, vol. 2010, 2010.

[14] O. Taipale, K. Smolander, and H. Kälviäinen, "Finding and Ranking Research Directions for Software Testing," in *Software Process Improvement: 12th European Conference, EuroSPI 2005, Budapest, Hungary, November 9-11, 2005. Proceedings*, I. Richardson, P. Abrahamsson, and R. Messnarz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 39–48.

[15] T. Metsä-Tokila, "Kasvun mahdollistajat - toimialaraportti ohjelmistoalasta ja teknisestä konsultoinnista" [Enablers of growth – sector report on the software industry and technical consulting], Työ- ja elinkeinoministeriö [Ministry of Economic Affairs and Employment], 2014. [Online]. Available: http://urn.fi/URN:NBN:fi-fe2017102550274. [Accessed: 4-Apr-2018].

[16] "ISO/IEC 15504-1: Information technology — Process assessment — Part 1: Concepts and vocabulary." International Organization for Standardization, 2004.

[17] ISO/IEC, "ISO/IEC 29119-1 Software and systems engineering - Software testing - Part 1: Concepts and definitions." 2013.

[18] E. van Veenendaal and B. Wells, *Test Maturity Model Integration TMMi*. The Netherlands: Uitgeverij Tutein Nolthenius, 2012.

[19] R. Kneuper, *CMMI: Improving Software and Systems Development Processes Using Capability Maturity Model Integration*. Rocky Nook, 2008.

[20] P. H. D. Valle, E. F. Barbosa, and J. C. Maldonado, "CS curricula of the most relevant universities in Brazil and abroad: Perspective of software testing education," in *Computers in Education (SIIE), 2015 International Symposium on*, 2015, pp. 62–68.