# Design and Implementation of a Software Testing Training Course

**Ing. Gustavo Lopez, Universidad de Costa Rica**

Gustavo Lopez is a researcher at the University of Costa Rica's Research Center on Information and Communication Technologies (CITIC), where he has worked since 2012. He has contributed to several research projects on software testing and human-computer interaction, and he has also designed and taught training courses on topics related to software testing. He received his B.Sc. in 2011 and his M. Sc in Computer and Information Science in 2015 both from University of Costa Rica. His research interests include in software testing, human-computer interaction, and computer science education.

**Ing. Francisco Cocozza, CITIC**
**Dr. Alexandra Martinez, Universidad de Costa Rica**

Alexandra Martinez is an Associate Professor in the Department of Computer Science and Informatics at the University of Costa Rica (UCR), where she has worked since 2009. She has taught graduate and undergraduate courses in Databases, Software Testing, and Bioinformatics. She has done applied research in software testing, software quality and bioinformatics at the university's Research Center on Information and Communication Technologies (CITIC). Previously, she worked as a Software Design Engineer in Test at Microsoft Corporation in Redmond, WA, and as a Software Engineer at ArtinSoft in San Jose, Costa Rica. She received her Ph.D. in Computer Engineering from the University of Florida in 2007, her M.S. in Computer Engineering from the University of Florida in 2006, and her B.S. in Computer and Information Science from the University of Costa Rica in 2000. She also received a scholarship to study in the Pre-Doctoral Program in Computer Science at the Ecole Polytechnique Fédérale de Lausanne, in Switzerland, from 2001 to 2002.

**Prof. Marcelo Jenkins, University of Costa Rica**

# Design and Implementation of a Software Testing Training Course


## Abstract

This paper presents the design and implementation of a software testing training course for software developers with little or no background on software testing. The design of this training course is modular, so that it can be adapted to different industry needs. The first module is a theoretical course that comprises the fundamentals of software testing, testing types, levels and design techniques. The second module is a practical hands-on workshop where students apply the theoretical concepts from the first module using a specialized tool that supports the entire testing process. The training was given to a small group of developers who work in a software development unit at our university. Our design and implementation of the training course was assessed from three different points of view: the trainer's, the trainee's, and the manager's. Our experience might help educational institutions and college professors in designing and implementing software testing training courses for industry.


## 1. INTRODUCTION

Software testing is a critical activity in software engineering. It is estimated that the cost of software testing exceeds half of the total cost of development and maintenance[16]. Still, we are far from producing defect-free software. In order to achieve effective testing, testers need to have good support from tools as well as sharp testing skills. The same goes for developers if they are doing part of the testing. Software testing is a technical investigation of the software under test performed to provide stakeholders with information about its quality[11]. The goal of testing is to find errors in the software, thus a test is successful if it can detect an error[12]. Testing improves the quality of the final product because most of the errors found get corrected before the software is released into production.

In a previous work we performed an evaluation of the software development process used by a systems unit from our university[8]. Such evaluation was based on the Standard CMMI Appraisal Method for Process Improvement (SCAMPI), from Carnegie Mellon's Software Engineering Institute. The results from this evaluation revealed several weaknesses in the development process of the organization, particularly in the Verification and Validation process areas (according to CMMI-DEV 1.3). The goal of the training described in this paper was to help the organization overcome some of the detected weaknesses.

This case study describes a two-part training course in software testing designed for software developers with little or no background in the software testing area. We devote half of the training to teach the fundamentals of software testing, best practices, testing types, testing levels, and test design techniques. The other half of the training is devoted to the use of a testing tool that supports the entire software testing process.

The rest of the paper is organized as follows. Section 2 mentions relevant related work in the area. Section 3 describes the design of the training course. Section 4 presents the training's implementation and assessment method. Section 5 shows the results of our assessment, and Section 6 states our conclusions and future work.

## 2. RELATED WORK

There is an extensive body of work related to teaching software testing both at the undergraduate and the graduate levels,[2, 5, 6, 13, 14, 15, 18] but little has been published on industrial training on software testing. Here we examine some of the recent published work that is most comparable to ours.

Jones and Chatmon[10] presented a framework called SPRAE, which encompasses five main principles needed to appropriately practice software quality assurance. The SPRAE framework provides a guideline to perform teaching activities and has been used several times in a senior-level elective course on software testing.

An interesting study by Chan, Tang and Chen[1] performed a survey to determine the software testing practices as well as the levels of education and training in Hong Kong. They found that training support by the organizations varies considerably from one industry to another, and that most of the software testing education and training was acquired through courses offered outside the universities.

Xie, Halleux, Tillmann and Schulte[20] described their experience in teaching and training software testing techniques and tool support, in both university and industrial settings. They list the differences encountered between teaching undergraduate students and training software professionals from industry.

Another teaching experience is presented by Harrison[7], where undergraduate students are confronted with software testing from two complementary viewpoints: as developers and as independent testers. A project is developed in which students have to play both roles.

Chen, Zhang and Luo[3] proposed a new teaching approach for software testing based on diversity principles, where different behaviors of the software are exercised as far as possible. This approach was used in an advanced undergraduate course on software testing.

The use test-driven development (TDD) as part of a testing course was described by Tinkham and Kanner[19], for a combination of undergraduate and graduate students. Their main conclusion is that TDD might not be suitable to all programmers or students because they differ strongly in cognitive levels and styles of learning.

Edwards[4] also presented an approach to introduce undergraduate students to test-driven development in a programming languages course, where a considerable reduction of software defects in students programming assignments was observed.

Shepard, Lamb and Kelly[17] argue in favor of strengthening the verification and validation contents taught in undergraduate Computer Science and Software Engineering degrees, based on a long experience with a testing-focused undergraduate course.

## 3. DESIGN

The first version of this training on software testing was offered in 2011 to a group of computer engineers who developed embedded software for networking devices. This initial version was focused on unit testing with Unity, a framework for unit testing in C language. Our experience on the first version of the training was reported in a previous work[9].

Later, a software development unit from our university requested a similar training on software testing, but this time the tool they needed was Microsoft Visual Studio 2010. For this second version of the training, we reused a large portion of the material that had been developed for the first version. However, we decided to separate the practical and the theoretical parts of the training in two modules, foreseeing that other organizations asked for a similar training but using a different tool. Hence the current structure of the training comprises two modules, each of 18 hours, for a total of 36-hours of training.

The first module is called 'Introduction to Software Testing' and is a theoretical course that comprises the fundamentals and best practices of software testing, as well as testing levels, types and design techniques. The second module is called 'Using Microsoft's Test Manager, Visual Studio, and Team Foundation Server to support the Software Testing Process' and is a practical workshop where students apply the theoretical concepts from the first module using a commercial tool widely used in the industry. This workshop is taught completely in the lab for a real hands-on learning experience. Mandatory attendance is required to successfully complete the training. A more detailed description of the training modules follows.

### 3.1. Module 1: Theoretical Course

The main objective of the course was to guide the student in learning the theory and skills needed to understand and apply principles, techniques, and best practices of software testing in the context of software quality assurance. The contents of this course were organized as follows:

1. Principles of software quality and testing
    1.1. Fundamentals: quality assurance and control, verification and validation, testing process, test case, software defect
    1.2. The testing process and its activities during the software development life cycle
    1.3. Software quality best practices
2. Testing types
    2.1. Static vs. dynamic, manual vs. automated
        2.1.1. Technical Reviews
    2.2. Functional vs. Non functional
3. Testing levels
    3.1. Unit testing
    3.2. Integration testing
    3.3. System testing
    3.4. User acceptance testing
4. Test design techniques
    4.1. Black box techniques: equivalence partition, boundary value analysis, error guessing

4.2. White box techniques: coverage criteria, basis paths
4.3. Combinatory and exploratory testing

5. Defect management and quality metrics
   5.1. Defect reporting
   5.2. Defect tracking process
   5.3. Quality metrics

### 3.2. Module 2: Workshop

The main objective of the workshop was to guide the student in learning the skills needed to manage and execute a software testing process using specialized CASE tools. The contents of this workshop were organized as follows:

1. Support for manual testing management
   1.1. Brief introduction to the tools
   1.2. User story management
   1.3. Test case management
   1.4. Defects management
   1.5. Traceability among user stories, test cases, and defects
2. Support for unit testing and coverage metrics
   2.1. Unit tests in Visual Studio
   2.2. Code coverage metrics
   2.3. Test automation
3. Support for automated user interface tests
   3.1. Simple tests
   3.2. Data-driven tests
   3.3. Web and desktop application tests
4. Introduction to Team Foundation Server (TFS)
   4.1. Features of TFS
   4.2. Installation of TFS
   4.3. Backup creation and recovery
   4.4. Version control
   4.5. Reports
5. Introduction to Microsoft Test Manager
   5.1. Creating a Test Plan
   5.2. Executing the tests
   5.3. Managing the environments
6. Performance Tests in Microsoft Test Manager
   6.1. Lab Center's features and functionalities
   6.2. Recommended architecture and environment
   6.3. Virtualization with Hyper-V
   6.4. Executing and reporting performance tests

## 4. IMPLEMENTATION AND ASSESSMENT

### 4.1. Implementation

The training hereby described was given to a software development unit from our university. This unit comprises a manager, a software tester, and eight developers, for a total of ten employees. This unit develops internal software for our university.

The first module of the training was taught on March 2012 in six alternating lessons of 4 and 2 hours, 2 times per week. There were eight participants and one instructor. The second module of the training was taught during May and June 2012 in six alternating lessons of 4 and 2 hours, 2 times per week. There were nine participants and two instructors.

### 4.2. Assessment

We assessed the training course from the point of view of the trainers, the trainees and the manager. The trainers' point of view takes into account the opinion of all three instructors. One instructor taught the first module of the training and the other two taught the second module. A qualitative assessment of the training based on the observed strengths and limitations was performed, as well as an assessment of the importance (usefulness) of the main topics in the training.

The trainees' point of view includes a self-assessment of each trainee's capability (level of expertise acquired) in each of the training topics, as well as an assessment of the importance (usefulness) they gave to these topics. Both assessments were collected through a questionnaire that was completed by the trainees at the end of the course.

The manager point of view was gathered through an interview. In order to know if there had been any changes to the processes or activities performed by this unit, we allowed approximately three months between the end of the training and this interview. During the interview, we also asked the manager to rate the importance of the training topics, in order to compare his answer to that of the trainees. The next three sections discuss in detail the results of each assessment.

## 5. FINDINGS AND DISCUSSION

### 5.1. Trainers' Perspective

**On the theoretical course.** Although we were able to reuse a significant percentage of the material that had been developed for a prior training[9], we still had to reorganize, extend, and improve it. The reorganization and improvements came from the feedback given by previous trainees and trainers, while the extensions came from the particular needs of the organization that requested the new training.

The main strength of this part of the training was the active learning approach used by the instructor because it engaged trainees to actively participate throughout the classes. Some of the ways in which active learning was integrated were: in-class discussions about case studies and several other topics, exercises about test case design using different techniques, and peer instruction.

A major limitation was the class schedule, which was very tiring for the trainees. The class schedule alternated between 3 p.m. - 5 p.m. and 3 p.m. - 7 p.m. for the 2-hour and 4-hour sessions, respectively. In both cases, the trainees came directly from work, which means they were already tired when the class began. This effect was worsened in the 4-hour sessions, despite the fact that in those days we had a 30 minute break in the middle.

**On the practical workshop.** Since we began developing the workshop material, we centered on creating functional and practical guides that trainees could follow on their own in the lab. At the same time, we tried to design these lab guides to go beyond a step-by-step tutorial by including an explanation of the specific tool functionality they were learning in each guide. This approach of creating self-explanatory guides turned out to be useful when trainees arrived late at the sessions or even when they missed a significant portion of class, since they were able to catch up on their own without interfering with the rest of the class.

Although the trainees used Microsoft Visual Studio as their development tool, they had no prior knowledge of the testing features and capabilities offered by the tool. Therefore, our training helped them get the most out of the software infrastructure they already had in place. In particular, they learned about managing and linking user stories, test cases and defects as well as executing tests, automating test cases, managing test environments, and leveraging virtualization to facilitate execution of automated tests as part of a continuous integration process.

A problem we found during the implementation of the workshop was related to time management, specifically, the more advanced trainees completed the guides well before their teammates. A possible solution to this problem would be to provide extra material (more exercises or advanced topics) in the lab guides for those participants who finish early.

The result of the trainers' assessment of the importance of each training topic is shown as a dashed line in Figure 1. This figure also contains the results of the assessments made by the trainees and their manager, so that we can easily compare the three assessments. From Figure 1, we can observe that the trainers gave more importance than the trainees to 'Microsoft Test Manager' and 'Lab Center and Virtualization' topics. This disparity might be explained because most of the trainees were developers and they possibly did not picture themselves applying those concepts or tools in their daily work (except for the software architect, but he was one out of eight). Another reason could be that these topics were covered at the end of the course, when trainees were already tired.

### 5.2. Trainees' Perspective

To ascertain the trainees' perspective, we used a two-part questionnaire. The first part of the questionnaire asked their opinion about the course and the trainers. The second part was an achievement test with questions on the course material that aim to measure trainees' learning.

One of the questions from the first part of the questionnaire was "In your opinion, what is the importance of the course topics (according to the use you could give them in your work)?" Trainees were asked to rank the importance of each topic on a 5-point Likert scale (1=lowest, 5=highest). This question measured the perceived importance (i.e., usefulness) of the topics covered in the course from the standpoint of the trainees.

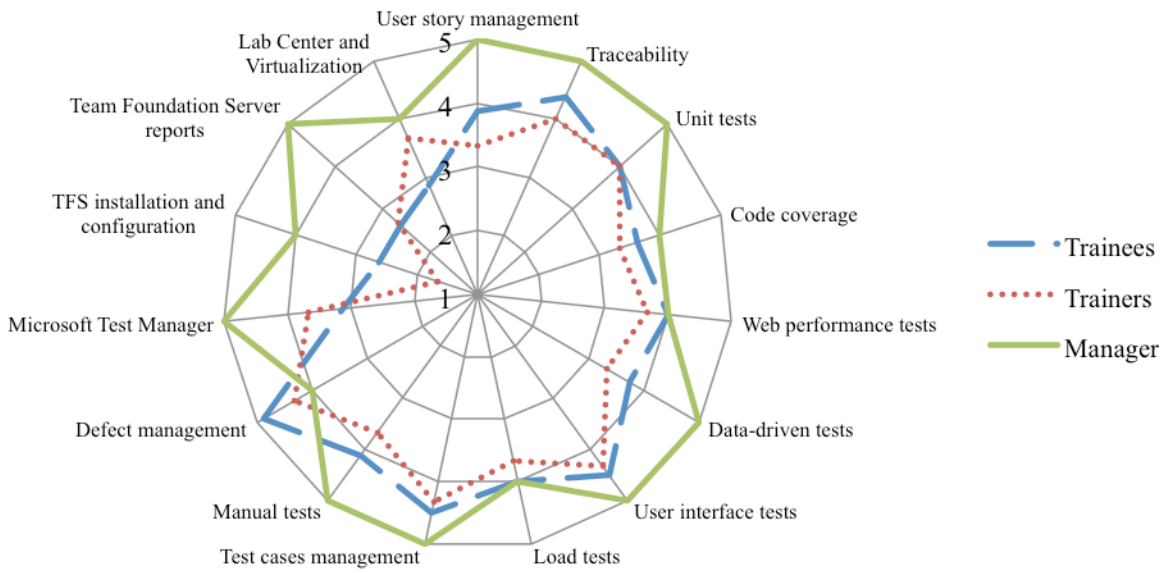


**Figure 1.** Importance of training topics as perceived by the trainers, trainees and manager.

Another question from the first part of the questionnaire was "Do you feel able to...? (a) Give a talk on the topic, (b) Explain the topic to a teammate, (c) Correctly apply the topic in your work, (d) Understand literature or talks on the topic." This question measured the trainee's perception of his ability to apply the new knowledge studied in the course (i.e., mastery of the topics). We had to convert the answers to this multiple-choice question into a 5-point numeric scale in order to be able to compare them with the answers to the first question. For this, we assigned a value of 5 to the "Give a talk on the topic" choice (meaning greater ability), a value of 1 to the "Understand literature or talks on the topic" choice (meaning lowest ability), and for the two remaining choices we use corresponding middle values.

Table 1 shows the averaged results (across the trainees) for these two questions. The prevalent difference between perceived importance and ability (mastery) was about 1.5 points, but there were eight topics for which the difference was higher, namely Traceability, Unit testing, Code coverage, Data-driven tests, User interface tests, Test case management, Defect management, and TFS installation and configuration. This means that trainees deem those topics as highly important but they do not feel sufficiently confident on their mastery of the topic. The topic on TFS installation and configuration is a special case because they currently do not use the tool, so they are likely to feel overwhelmed by its complexity. In the case of Data-driven tests, the most probable reason for the high difference is that the class in which that topic was taught was overloaded with other topics, thus practice on this topic was scarce. The other topics that exhibited a high difference were probably emphasized by the trainers but practice was not sufficient to satisfy the trainees.

Trainees were also asked to provide recommendations for improving the training as well as to mention what they liked the most from the training. Some of their suggestions are summarized as follows:

- *A smaller separation between the theoretical and practical parts*: They felt that the two parts were somehow disjoint, which might be explained by the fact that they were offered two months apart from each other.
- *Time for discussion is important*: Most participants liked the opportunities for in-class discussion.
- *Offer more training hours*: The training consisted of 36 hours but some participants seemed to need more time to understand the topics and perform the exercises.
- *Improve the exercises*: Some trainees thought that the exercises were given too front up in the lectures. Perhaps providing additional guidance on the exercises would be beneficial.

**Table 1.** Perceived importance and mastery of topics by trainees.

| ***Topic*** | ***Importance*** | ***Mastery*** |
|---|---|---|
| User story management | 3.88 | 2.50 |
| Traceability | 4.38 | 2.50 |
| Unit tests | 4.00 | 2.34 |
| Code coverage | 3.63 | 1.72 |
| Web performance tests | 4.00 | 2.96 |
| Data-driven tests | 3.75 | 1.87 |
| User interface tests | 4.50 | 2.97 |
| Load tests | 4.00 | 2.65 |
| Test case management | 4.50 | 2.81 |
| Manual tests | 4.13 | 2.65 |
| Defect Management | 4.88 | 2.97 |
| Microsoft Test Manager | 3.13 | 1.87 |
| TFS installation and configuration | 2.63 | 1.09 |
| Team Foundation Server reports | 2.63 | 1.56 |
| Lab Center and Virtualization | 2.88 | 2.03 |

### 5.3. Manager's Perspective

As part of the interview to the manager of the software development unit, we asked him if he had observed any substantial change in his team's work (processes, activities, etc.) as a result of the training. His answer was that they were "working on them". This seems to imply that improvements in software processes take time to take place.

We also asked the manager his opinion about the overall training experience. His answer was that the training was a good experience for his team, and some of the tangible results he pointed

out were: (*i*) improvements on the estimation of cost and duration of the projects, thanks to the use of tools covered in the training, and (*ii*) improvements on the generated documentation, particularly separating the technical information from the customer information in use cases.

The manager also assessed the importance of each training topic, as shown in Figure 1. An interesting finding is that the manager tends to rank the importance of most of the topics higher than the trainees and even the trainers. It should be noted that the team manager did not attend the training, so his answers are based on his own prior knowledge of the topics and a brief explanation from the interviewer when needed.

## 6. CONCLUSIONS AND FUTURE WORK

We presented our experience on the design and implementation of a training course on Software Testing. The training's target audience were developers with little or no background on software testing. We described the rationale for the modular design of the course and listed the contents comprised in each module. Then we explained how the training was implemented and how it was assessed. Finally, we presented the results of the assessment (from the trainers', trainees', and manager's point of view). In what follows, we sketch some lessons learned as well as ideas for future work.

To monitor the trainee's on-the-job performance and improve the training course, a more in-depth measuring of the improvement at work of each trainee should be conducted. This could be done at a reasonable period of time after the training was conducted, say about 6 months. This would allow us to measure, either qualitatively or quantitatively, the impact of the training in the organizational performance.

We shall use more short in-class exercises in future offerings. It seems that class time is never enough in industrial training settings, particularly regarding highly technical subjects such as software testing.

We also learned that theory and practice should be interleaved and not covered separately. Thus, in future offerings we will not separate the practical and theoretical parts of the course, but instead use one session to introduce a particular subject and the second weekly session for exercises, question sessions, and closure of topics.

We learned as well that the administrative topics, although necessary in order to understand the process of software testing, were not deemed important by the team members executing the tests. Therefore, future offerings shall focus more on the technical topics.

By incorporating these three major improvements we believe our training course can be offered to other software development organizations, and as we do it, we will further enrich our experience as industrial trainers.

Finally, a substantial amount of time and effort was required to set up and maintain the required software and hardware infrastructures to appropriately offer such course. Paying the necessary attention to it is always a critical success factor.

## 7. ACKNOWLEDGMENTS

This work was partially supported by the Research Center on Information and Communication Technologies (CITIC) and by the Department of Computer and Information Science (ECCI) from the University of Costa Rica.

## 8. REFERENCES

1. F. T. Chan, W. H. Tang, and T. Y. Chen. Software testing education and training in Hong Kong. In *Proceedings of the Fifth International Conference on Quality Software*, pages 313–316, Washington, DC, USA, 2005. IEEE Computer Society.
2. T. Y. Chen and P.-L. Poon. Experience with teaching black-box testing in a computer science/software engineering curriculum. *IEEE Trans. on Educ.*, 47(1):42–50, Feb. 2004.
3. Z. Chen, J. Zhang, and B. Luo. Teaching software testing methods based on diversity principles. In *Proceedings of the 2011 24th IEEE-CS Conference on Software Engineering Education and Training*, pages 391–395, Washington, DC, USA, 2011. IEEE Computer Society.
4. S. H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3), Sept. 2003.
5. S. Elbaum, S. Person, J. Dokulil, and M. Jorde. Bug hunt: Making early software testing lessons engaging and affordable. *International Conference on Software Engineering*, 0:688–697, 2007.
6. V. Garousi. An open modern software testing laboratory courseware - an experience report. In *Proceedings of the 2010 23rd IEEE Conference on Software Engineering Education and Training*, pages 177–184, Washington, DC, USA, 2010. IEEE Computer Society.
7. N. B. Harrison. Teaching software testing from two viewpoints. *J. Comput. Sci. Coll.*, 26(2):55–62, Dec. 2010.
8. M. Jenkins, A. Martinez, and G. Lopez. Una experiencia de aseguramiento de la calidad en una unidad de sistemas. In *Proceedings Latin American Congress on Requirements Engineering and Software Testing*, 2012.
9. M. Jenkins, A. Martinez, J. Rodriguez, and E. Rojas. Designing a blended software testing course for embedded c software engineers. In *Proceedings of the Computers and Advanced Technology in Education*. Acta Press, 2011.
10. E. L. Jones and C. L. Chatmon. A perspective on teaching software testing. *J. Comput. Sci. Coll.*, 16(3):92–100, Mar. 2001.
11. C. Kaner and J. Bach. *Black box software testing*, 2005.
12. C. Kaner, J. Falk, and H. Q. Nguyen. *Testing Computer Software*. Wiley, 2 edition, 1999.
13. C. Kaner and S. Padmanabhan. Practice and transfer of learning in the teaching of software testing. In *Proceedings of the 20th Conference on Software Engineering Education & Training*, pages 157–166, Washington, DC, USA, 2007. IEEE Computer Society.
14. F. Kazemian and T. Howles. A software testing course for computer science majors. *SIGCSE Bull.*, 37(4):50–53, Dec. 2005.
15. C. Mao. Towards a question-driven teaching method for software testing course. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering* - Volume 05, pages 645–648, Washington, DC, USA, 2008. IEEE Computer Society.
16. M. Pezzè and M. Young. *Software Testing and Analysis: Process, Principles, and Techniques*. Wiley, 2008.
17. T. Shepard, M. Lamb, and D. Kelly. More testing should be taught. *Commun. ACM*, 44(6):103–108, June 2001.
18. B. Talon, D. Leclet, A. Lewandowski, and G. Bourguin. Learning software testing using a collaborative activities oriented platform. In *Proceedings of the 2009 Ninth IEEE International Conference on Advanced Learning Technologies*, pages 443–445, Washington, DC, USA, 2009. IEEE Computer Society.

19. A. Tinkham and C. Kaner. Experiences teaching a course in programmer testing. In *Proceedings of the Agile Development Conference*, pages 298–305, Washington, DC, USA, 2005. IEEE Computer Society.

20. T. Xie, J. de Halleux, N. Tillmann, and W. Schulte. Teaching and training developer-testing techniques and tool support. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 175–182, New York, NY, USA, 2010. ACM.