Teaching software architecture quality using ATAM

Renato Manzan de Andrade[1], Reginaldo Arakaki[2] and José Carlos Cordeiro[3]

[1]Polytechnic School of the University of São Paulo, São Paulo, Brazil,renato.manzan@usp.br
[2]Polytechnic School of the University of São Paulo, São Paulo, Brazil,reginaldo.arakaki@poli.usp.br
[3]Institute for Technological Research, São Paulo, Brazil, jcc.martins@bol.com.br

Abstract

Software architecture quality directly affects the user's overall satisfaction, since it defines if the quality systems requirements will be achieved, especially the non-functional requirements. By this reasons, evaluating software architecture quality has become one of the most important activities in the software development cycle. Nevertheless, architecture quality concepts are not handled with due importance by the curricula of undergraduate and graduate software engineering courses. Although software architecture is part of Software Engineering undergraduate curriculum, in many cases software architecture quality classes do not explicit clearly the importance of this issue. Students have strong programming skills, but very seldom know architectural quality concepts and how the business drivers will drive architectural decisions. This paper proposes a guide for teaching software architecture quality supported by ATAM (Architecture Trade-off Analysis Method) and presents the results of applying this guide in undergraduate and graduate courses.

Keywords: *Software architecture quality, ATAM, Quality Models, Education*.

1. Introduction

Quality attributes of large software systems are principally determined by the system's software architecture. That is, in large systems, the achievement of qualities such as performance, availability, and modifiability depends more on the overall software architecture than on code-level practices such as language choice, detailed design, algorithms, data structures, testing, and so forth. [1]

So, the quality of software architecture is a critical step to ensure that the software meets its design objectives in a reliable and predictable fashion. [2] Although software architecture and its influence on quality requirements are widely regarded as one of the most important software artifacts, in many cases real-world application of software engineering concepts does not effectively map with current software engineering curriculums. Typically, a student's first real-world experience working on large-scale software development projects and software quality attributes is in his/her first full-time position. [3]

An ordinary student is used to writing small programs, from scratch in a language, without architectural quality guidance. Students have strong programming skills, but very seldom know architectural concepts and their influences on software quality and relations to business drivers. These students tend to immediately start coding once they receive a problem to be solved. [4] This result in a serious gap in software engineering curriculum: students are expected to learn how to design complex systems without the requisite intellectual tool for doing so effectively. [5]

This paper presents an approach to teach software architecture using ATAM (Architecture Tradeoff Analysis Method). The proposed approach has been applied in advanced software laboratory undergraduate discipline in Computer Engineering courses since 2003.

2. Architecture Tradeoff Analysis Method

All design, in any discipline, involves tradeoffs, this is well accepted. What is less well understood is the means for making informed and possibly even optimal tradeoffs. [1] Design decisions are often made for non-technical reasons: strategic business concerns, meeting the constraints of cost and schedule, using available personnel, and so forth.

Much of a software architect's life is spent designing software systems to meet a set of quality attribute requirements related to business drivers. [6]

To be meaningful, quality attribute requirements must be specific about how an application should achieve a given business driver. A common problem regularly found in architectural documents is a general statement such as "The application must be scalable". This is far too imprecise and really not much use to anyone. So, must this hypothetical application scale to handle increased simultaneous user connections? Or increased data volumes? Or deployment to a larger user base? Or all of these? Without elaboration, each of these statements is subject to interpretation and misunderstanding because the concept of quality is a subjective term. [7]

Defining which of these scalability measures must be supported by the system is crucial from an architectural perspective, as solutions for each differ. It's vital therefore to define concrete quality attribute requirements, such as: *It must be possible to scale the deployment from an initial 100 geographically dispersed user desktops to 10,000 without an increase in effort/cost for installation and configuration.* This is precise and meaningful. As an architect, this points to a path to a set of tactical and concrete technologies that facilitate zero-effort installation and deployment subjective term. [8]

Methods as ATAM for evaluating architecture-level designs that considers multiple quality attributes help us to identify trade-off points between these attributes, facilitates communication between stakeholders (such as user, developer, customer, maintainer) from the perspective of each attribute, clarifies and refines requirements, and provides a framework for an ongoing, concurrent process of system design and analysis

ATAM has been used for over a decade to evaluate software architectures in domains ranging from automotive to financial to defense. It is composed by nine steps, listed below [1]:

- Step 1- Present the ATAM
- Step 2 - Present Business Drivers
- Step 3 - Present Architecture
- Step 4 - Identify Architectural Approaches
- Step 5 - Generate Quality Attribute Utility Tree
- Step 6 - Analyze Architectural Approaches
- Step 7 - Brainstorm and Prioritize Scenarios
- Step 8 - Analyze Architectural Approaches
- Step 9 – Present results.

3. Teaching software architecture quality using ATAM

One of the most important characteristics of ATAM is the unbroken relation between business drivers, quality attributes, tradeoffs and software architecture decisions.

The relationship between these concepts helps the students to understand how business drivers influence architectural approaches/decisions, and how to apply software metrics to assess software architecture quality.

The approach to teach software architecture using ATAM is based on software quality attributes, software metrics and inherent tradeoffs between quality attributes, captured in ATAM quality utility tree. Each leaf of the tree is strongly related to business drivers, allowing the students to connect business with architectural decisions.

The used teaching approach has the following characteristics:

- **Practical approach:** usually software architecture quality and its metrics are presented in a very theoretical and abstract way, but the students have weak intuitions about high-level architectural abstractions and quality attributes. Real world examples of software architectures metrics, collected from practical examples, lead to a better understanding of its concepts and quality tradeoffs.
- Problem-based learning: engineering education is undergoing significant changes, notably in the way engineering schools are adopting problem-based instruction to meet the changing demands of engineering practice. Mastery of technical content is no longer sufficient. Increasingly, engineering programs are requiring students to work projects that are open-ended with loosely specified requirements, produce professional quality reports and presentations, consider ethics and the impact of their field on society, and develop lifelong learning practices. An implicit goal of this shift in curricula is to produce graduates who will be ready to assume engineering tasks upon graduation—that is, with the skills to develop solutions to problems under competing constraints of functionality, cost, reliability, maintainability, and safety [9]. In problem-based learning, students are actively involved with problems coming from real practice. The following steps are applied to leverage the learning of NFRs in software architecture [10].
  - Identify concepts and parts of the problem that needs clarification;
  - Define the problem;
  - Analyze the problem, brainstorm about solutions or causes;
  - Structure solutions or causes;
  - State learning objectives;
  - Self-study directed towards learning objectives;
  - Report things learned and application to the problem.
- **Quality requirements metrics:** quality requirements impact directly on measures such as productivity and cost. Ultimately, these quantitative measures determine the justification for investment in a software development project. In view of this reality it is surprising that non-functional requirements are often ignored in the analysis process. [11]
- **Simulation techniques:** Simulation has been used in engineering disciplines for many years to great advantage. In recent years, an increasing amount of attention has been paid to using simulation to advance software engineering. There are a number of areas in which simulation can benefit software engineering, including: assessing the costs of software development, supporting metric collection, building consensus and communication, requirements management, project management, training, process improvement, risk management, and acquisition management. [12] Simulations can also produce visualizations of the architecture's execution. [13] These visualizations are particularly useful for identifying software architecture failures as bottlenecks points, resources starvation, memory leaks, low performance, etc. Simulation, done during the architecture and design stage, is also a low cost alternative to the actual implementation and execution of a real system.
- **Intensive use of proof of concepts:** proof of concept is used as evidence that the chosen software architecture is viable and capable of meeting quality attributes requirements and related business drivers.
- **Real-world projects:** incorporating real-world problems of sufficient magnitude and complexity into the classes is necessary to enable effective learning of software architecture quality skills and concepts.
- **Incremental learning:** knowledge is often hierarchical, and frequently the best way to assure performance on higher-level objectives is to identify the prerequisite skills needed for a current unit of instruction and ascertain that students have mastered them. Based on this premise and considering the richness, the complexity and the multiple dimensions of software architectures quality concepts,

the teaching approach uses an incremental approach, so the topics are presented in an increasing abstraction level.

- **Learner-based teaching:** traditional education practice seems to be built on an assumption that the mind is a container, and it is the teachers' responsibility to fill this with knowledge. Learner-based teaching means that education is not viewed as a process where knowledge is transferred from the teacher to the student, but rather that knowledge is create within the students' minds. The teaching approach adopt a practice driven education model where software architecture quality concept is regarded as something which cannot be taught entirely, but must be built by each individual requiring a engaged and proactive attitude on the part of students. This approach, which goes from the concrete to the abstract, capitalize on the innate human desire to explore and learn that is characterized by "practice-pull", rather than "theory-push". [14]

In order to create a learning scenario, the project should not start from scratch but the project should start with an existing system that needs to be measure, extended, modified and measured again.

After challenging the students with of a real world project assignment, the primary strategy was to involve and motivate them in a full life cycle team project, where the teacher plays the role of "the client". This gives the students first hand personal experience in the effects of making architectural decisions on their project and on the clients' satisfaction with their product. For the last instances of the course, the Java Pet Store 2.0 has been used as system sample.

The Java Pet Store 2.0 Reference Application is a sample application designed to illustrate how the Java Enterprise Edition 5 Platform can be used to develop an AJAX-enabled Web 2.0 application, using Java Persistence APIs, applying MVC and other design patterns in an Ajax web app, using Mashups such as Google Maps service for location specific searches of pets and PayPal service for purchases, using an RSS feed as a data source. [15]

The steps in the next picture are followed to teach software architecture quality using ATAM:

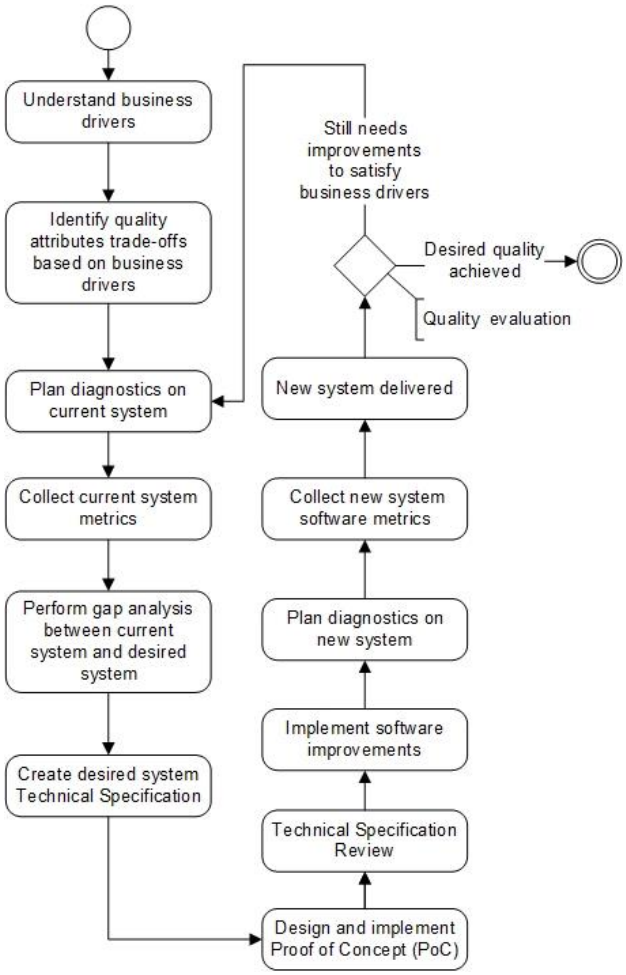


Figure 1. Steps to teach software architecture quality using ATAM.

Here is a description of each step:

- **Understand business drivers:** in this step the students understand the concept of business drivers and how it drives software architecture.
- **Identify non-functional requirements tradeoffs based on business drivers:** here ATAM and concepts are discussed and the course work assignments are distributed to teams of 2 of 3 students. It involves software improvement considering, but not limited to, the following tradeoffs:
  - Maintainability vs. usability;
  - Security vs. performance;
  - Traceability vs. performance;
  - Concurrency vs. simultaneous access;
  - Accuracy vs. concurrency;
  - Availability vs. portability.
- **Plan diagnostics on current system:** using tools as JMeter [16], an application designed to load test functional behavior and measure performance, the students plan diagnostics of measures on current system related to the course work assignment, i.e., the tradeoffs. The measures include static and dynamic metrics.
- **Collect current system metrics:** collect the metrics on the current system according to the plan defined in the previous step.
- **Perform gap analysis between current system and desired system:** with metrics results, each team performs a gap analysis between current system and the non-functional requirement attribute

desired in the new system's implementation. For instance, in the security vs. performance team, the challenging is to improve system security considering the overall system performance. The tradeoff analysis is helped by an ATAM quality utility tree. A sample of a quality utility tree is depicted bellow:

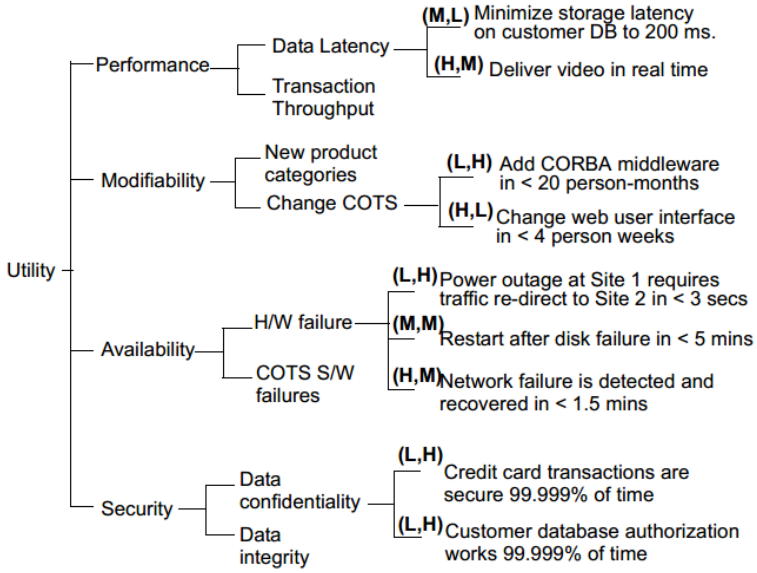


Figure 2. ATAM Quality Utility Tree.

- **Create desired system Technical Specification:** in this step, a technical specification of the desired system is created, guided by ATAM Step 6 - Analyze Architectural Approaches. The specification is a lightweight document, containing only enough information to implement system desired improvements;
- **Design and implement Proof of Concept (PoC):** the implementation of a Proof of Concept is very important in the learning process. The goal is not just confirm that an implementation is feasible, but also demonstrate its weak points. To do so, the students apply many useful technics as simulations, loopbacks, injections, etc. Here, the students learn how to choose architectural tactics. An architectural tactic is a reusable architectural building block that provides a generic solution to address issues related to quality attributes. Architectural tactics are selected based on a given set of quality attributes, and the selected tactics are composed to produce a tactic that combines the solutions of the selected tactics. The composed tactic is then instantiated to create an initial architecture of the application where the quality attributes are embodied. [17] After the choice of one or more tactics, they are implemented through a mechanism.
- **Technical Specification Review:** a formal review of Technical Specification conducted by student team supported by teachers in order to verify if the main aspects of architecture were considered. It can be considered the first test in the course.
- **Implement software improvements:** implementation of software improvements confirmed in PoC and detailed in technical specification.
- **Plan diagnostics on new system:** to verify if the software improvements were achieved a new cycle of measured must be done. At this time, in the new system.
- **Collect new system software metrics:** collect the metrics on the new system according to the plan defined in the previous step.
- **New system delivered:** new system released.
- **Quality evaluation:** presentation of new software architecture as well as the results of the new system metrics. This presentation can be considered the final test in the course. If the results were under the expectation, the process can be repeated, i.e., the system still needs improvements to satisfy business drivers. Due course time constraints, the steps were executed only once.

In the classes, the students were able to analyze in practical way the quality requirements by an architectural point of view and its importance to meeting software quality. Under this learning scenario, others essential software engineering skills were also trained as team work, software project management, software architecture design and communication skills, in a realistic environment and in architectural-centric approach.

From 2007 to 2012, this approach was applied to 85 computer engineering undergraduate students. The evaluation of teaching approach has been done by a 3 phase-survey answered by the students. The first one is applied in the beginning of course. The second one, in the middle of course (7th. class – during Technical Specification Review), and the last one in the end of course (12th. class – Quality evaluation).

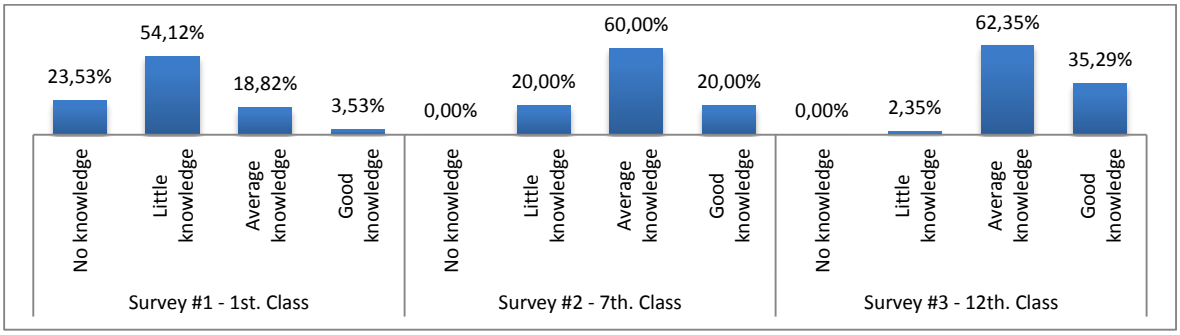
The surveys evaluate if the student has some knowledge in the following 7 topics related to software architecture:

1. General quality measures;
2. Software architecture quality measures;
3. Static quality measures;
4. Dynamic quality measures;
5. Measure usefulness to improve software architecture quality;
6. Relationship between software architecture and quality.

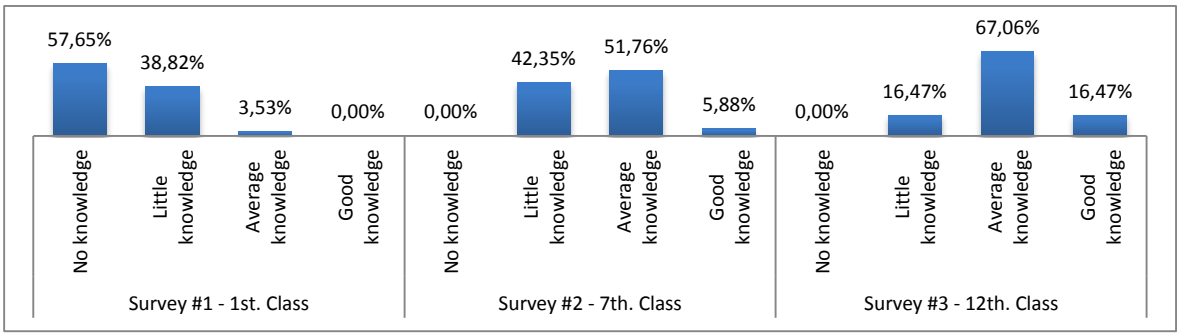
To each topic, the student should answer his/her knowledge level as following: No knowledge; Little knowledge; Average knowledge and Good knowledge.

Analyzing the students' answers, there are evidences that the concepts taught by the discipline were learned. The following pictures show the graphics of survey results in each topic evaluated.

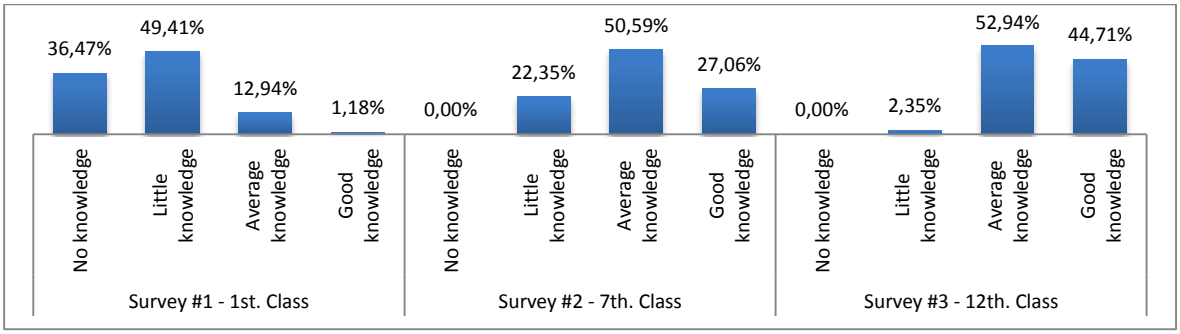
3.1. General Quality measures



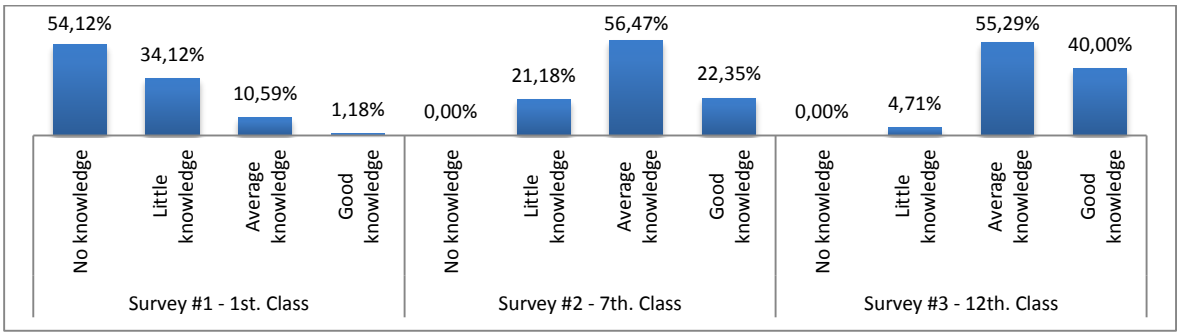
3.2. Software Architecture Quality measures



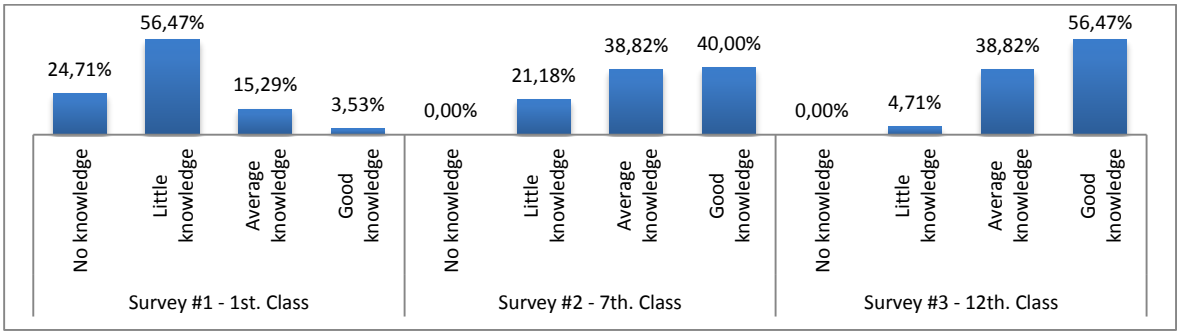
3.3 Static quality measures



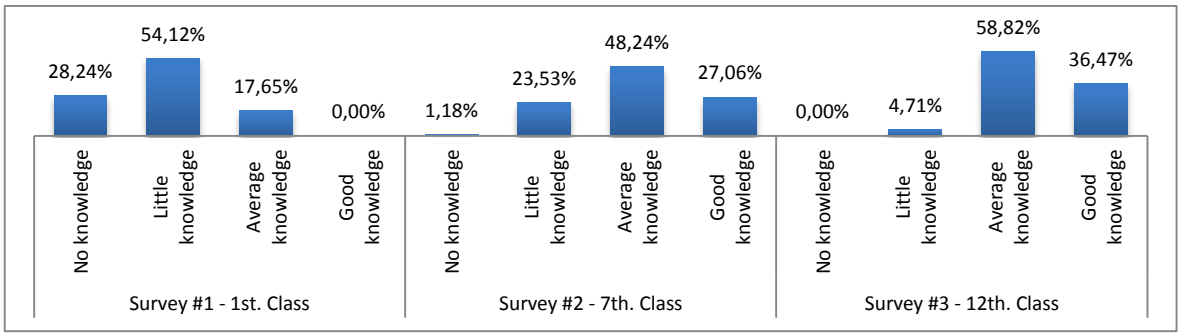
3.4 Dynamic quality measures



3.5 Measure usefulness to improve software architecture quality



3.6 Relationship between software architecture and quality



4. Conclusion

It is always more cost-effective to evaluate software quality as early as possible in the life cycle of the system. For this reason, it is important to evaluate and determine whether a system is destined to satisfy its desired qualities or not before it is built. [18] ATAM could help during evaluation if a system is destined to satisfy its desired qualities or not before it is built. In addition to that, it strengthens the relation between business drivers and software architecture decisions.

The authors believe strongly that including practical software architecture formally in software engineering curricula in order to obtain software quality from an architectural point of view can help the universities to achieve their core goals in higher education, supplying the growing demand of society for high skilled system architects.

Teaching approaches related to software architecture and software quality has been developed by the authors since 2003 and has already been applied in more than 25 class groups, resulting in around 6000 class hours, including undergraduate and graduated disciplines and mentoring on the job activities. The students' feedback on the teaching approach has been very positive.

References

[1] SEI, "Architecture Tradeoff Analysis Method," *Carnegie Mellon University*, 2003. http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm. Last accessed 18 February 2014.

[2] E. Anjos and M. Zenharela, "A Framework for Classifying and Comparing Software," pp. 270–282, 2011.

[3] M. B. Blake, "A student-enacted simulation approach to software engineering education," *IEEE Trans. on Educ.*, vol. 46, no. 1, pp. 124–132, 2003.

[4] D. Rosca, W. Tepfenhart, and J. McDonald, "Software engineering education: following a moving target," *Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET'03)*\, pp. 129 – 139, 2003.

[5] M. Shaw and D. Garlan, *Software architecture: perspectives on an emerging discipline*. Prentice Hall, 1996.

[6] I. Gorton, *Essential Software Architecture*, 2a. edição. Springer-Verlag Berlin Heidelberg, 2011.

[7] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures*. Pearson Education, 2002.

[8] L. Chung, B. Nixon, and E. Yu, "An approach to building quality into software architecture," *Proceedings of the 1995 Conference of the Centre for Advanced Studies on Collaborative Research*, 1995.

[9] G. K. W. K. Chung, T. C. Harmon, and E. L. Baker, "The impact of a simulation-based learning design project on student learning," *Education, IEEE Transactions on*, vol. 44, no. 4, pp. 390–398, Nov. 2001.

[10] R. Koper and V. René, "Modeling units of learning from a pedagogical perspective." Open Universiteit Nederland, 2004.

[11] T. Pasternak, "Using trade-off analysis to uncover links between functional and non-functional requirements in use-case analysis," in *Software: Science, Technology and Engineering, 2003. SwSTE '03. Proceedings. IEEE International Conference on*, 2003, pp. 3–9.

[12] J. S. Collofello, "University/industry collaboration in developing a simulation based software project management training course," in *Software Engineering Education; Training, 2000. Proceedings. 13th Conference on*, 2000, pp. 161–168.

[13] K. S. Barber and J. Holt, "Software Architecture Correctness," *IEEE Software.*, vol. 18, no. 6, pp. 64–65, 2001.

[14] L. Ohlsson and C. Johansson, "A practice driven approach to software engineering education," *IEEE Trans. on Educ.*, vol. 38, no. 3, pp. 291–295, 1995.

[15] Oracle, "Java Pet Store 2.0," 2007. http://www.oracle.com/technetwork/java/index-136650.html. Last accessed 18 February 2014

[16] The Apache Software Foundation, “Apache JMeter,”, http://jmeter.apache.org/. Last accessed 18 February 2014.

[17] S. Kim, D.-K. Kim, L. Lu, and S.-Y. Park, “A Tactic-Based Approach to Embodying Non-functional Requirements into Software Architectures,” *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pp. 139–148, Sep. 2008.

[18] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Pearson Education, 2012.

Authors

Principal Author: Renato Manzan de Andrade holds a Master degree in Electrical Engineering from University de São Paulo. He is finishing his PhD research related to software quality and education at University de São Paulo.

Co-author: Reginaldo Arakaki holds a PhD degree in Electrical Engineering from University de São Paulo. He is presently a professor specialising in Software Architecture, Software Quality at University de São Paulo.

Co-author: José Carlos Cordeiro holds a Master degree in Software Engineering from Institute for Technological Research.