Teaching Software Architecture Quality based on run-time metrics

Renato Manzan de Andrade [1], Reginaldo Arakaki [2]

Abstract - Software architecture is strongly related to system quality, since it drives the whole development process. In large systems, the achievement of qualities such as functionality, efficiency, reliability, and maintainability depends more on the overall software architecture than on code-level practices. It is cost effective to try to determine, before a system is built, whether it will satisfy its desired qualities. By these reasons, evaluating software architecture quality has become one of the most important decisions in the software development cycle. To support this decision, many software architecture evaluation methods with distinct goals and approaches have emerged in the last few years. These methods can assist the developer in creating a software architecture that will have a potential to fulfill the requirements on the system. 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 their influences on software quality, costs and maintenance. This paper describes a practical approach to teach software architecture quality based on run-time metrics and presents the application of this approach in an advanced software laboratory undergraduate discipline.

Index Terms - Software Architecture, Software Quality, Run-time metrics, Software Engineering Education.

INTRODUCTION

Software-intensive systems play an increasingly important and central role in all aspects of everyday life becoming a critical factor to the successful operation of systems, including not only life-critical control systems, but also ordinary communication and commerce [1].

In software-intensive systems, the achievement of qualities - such as performance, availability, security, and modifiability - is dependent on the software architecture. In addition, quality attributes of large systems can be highly limited by a system's requirements and constraints [2].

Software quality can be defined as defined as the degree to which a customer or user perceives that software meets his or her composite expectations [3].

Furthermore, it is always more cost-effective to evaluate software quality as early as possible in the life cycle of the system. The obvious risk is that potentially large amounts of resources will have been put into building a system which does not fulfill its quality requirements [3]. 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[4].

Besides the Introduction, the paper is structured as follows. The next section provides background and motivation for this work. The third section presents some concepts about software metrics and its relations to software architecture quality. An approach based on a framework to teach software architecture quality created by the authors is presented in the fourth section.

An example of application of the framework in an advanced software laboratory undergraduate discipline is presented in the fifth section, emphasizing non-functional requirements, its metrics and the framework activities to teach software architecture quality. A summary conclusion is provided in the sixth section and the references used in this papers are listed in the last section.

SOFTWARE ARCHITECTURE QUALITY

Designing software architecture is a complex process, involving the creation of solutions to complex, multi-faceted problems, which often do not have a single optimal solution, but only a number of acceptable ones. One particularly difficult aspect of the architectural process is ensuring that a system will meet its quality requirements [1]. There are several reasons that explain the complexity of achieving quality attributes through software architecture including a lack of specificity in the requirements, a shortage of documented knowledge of how to design for particular quality attributes, and the trade-offs involved in achieving quality attributes [2].

While getting a system's functionality correct is important, many systems are considered to be failures because they are lacking in one or more critical non-functional qualities, such as security or scalability.

Understanding the relationship between architectural decisions and a system's quality attributes revealed software architecture validation as a useful risk-reduction strategy [3].

The idea of predicting the quality of a software product from a higher-level design description is not a new one. In 1972, Parnas described the use of modularization and information hiding as a means of high level system

[1] Renato Manzan de Andrade, Computing Engineering and Digital Systems Department - Polytechnic School of the University of São Paulo - São Paulo – SP, 05508-900, Brazil, renato.manzan@poli.usp.br
[2] Reginaldo Arakaki, Computing Engineering and Digital Systems Department - Polytechnic School of the University of São Paulo - São Paulo – SP, 05508-900, Brazil, reginaldo.arakaki@poli.usp.br

decomposition to improve flexibility and comprehensibility. In 1974, Stevens et al. introduced the notions of module coupling and cohesion to evaluate alternatives for program decomposition. During recent years, the notion of software architecture has emerged as the appropriate level for dealing with software quality.[5].

Quality permeates all aspects of software development from the initial requirements gathering process to the operation of the executable system. The quality of a system is directly related to the ability of the system to satisfy its functional, nonfunctional, implied, and specified requirements [6] A system has many characteristics such as functionality, performance, and maintainability. The quality of each of these characteristics comprises the total quality of the system. Each characteristic can be specified as an attribute of the system [7].

Quality attributes can be represented using quality models. Quality models are systems that relate various quality attributes and, in some cases, identify key engineering practices to address them and metrics appropriate for measuring or observing them. Each model uses different terminology, but they share general concepts as internal quality attributes and external quality attributes usability, efficiency), The quality metamodel can be used as a basis for describing various quality models. The Figure 1 shows a quality metamodel.

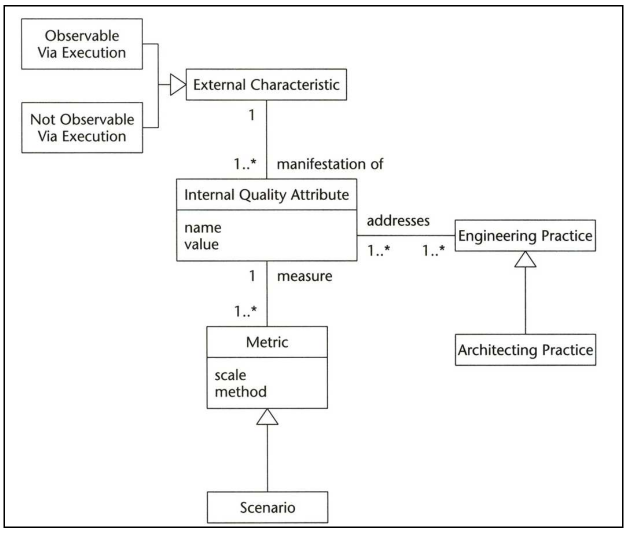


FIGURE 1 - QUALITY METAMODEL [7]

A quality model is a specific instance of the quality metamodel and defines specific characteristics, quality attributes, and metrics. In this paper, the ISO-9126 instance of the quality metamodel is used.

IS0 9126 [8] proposes a general quality model, based in McCall’s model, to specify and evaluate the quality of a software product from different perspectives or views, acquisition, development, maintenance. It considers internal quality characteristics, which are related to the software development process and environment or context and external characteristics, which are observed by the end-user on the final software product.

The view of quality can be internal or external, and it also affected by the stakeholder view in the particular stage of development.

The quality characteristics of the ISO 9126 quality model are refined into attributes, which can be measured to enrich the information about the architecture. [9].

ISO-9126 lists 6 quality characteristics: functionality, reliability, usability, efficiency, maintainability, and portability. These characteristics are refined into 8 subcharacteristics: efficiency, maintainability, portability, reliability, security, integrability, scalability and usability.

SOFTWARE METRICS

The achievement of quality attributes of a system is intimately connected with the software architecture for that system [10].

The desired combination of attributes quality shall be clearly defined; otherwise, assessment of quality is left to intuition.

Quality attributes form the basis for architectural evaluation, but simply naming the attributes by themselves is not a sufficient basis on which to judge an architecture for suitability. Often, requirements statements like the following are written [11]:

- "The system shall be robust."
- "The system shall be secure from unauthorized break-in."
- "The system shall exhibit acceptable performance."

Without elaboration, each of these statements is subject to interpretation and misunderstanding because the concept of quality is a subjective term. The use of software metrics can help to decrease the abstractness of software quality statements [11].

Defining software quality for a system is equivalent to defining a list of software quality attributes required for that system and identify a set of software metrics.

The purpose of software metrics is to make assessments throughout the software life cycle as to whether the software quality requirements are being met. The use of software metrics reduces subjectivity in the assessment and control of software quality by providing a quantitative basis for making decisions about software quality. However, the use of software metrics does not eliminate the need for human judgment in software evaluations.

The use of software metrics within an organization or project is expected to have a beneficial effect by making software quality more visible [6].

More specifically, the use of software metrics for measuring quality allows an organization to:

- Achieve quality goals;
- Establish quality requirements for a system at its outset;
- Establish acceptance criteria and standards;
- Evaluate the level of quality achieved against the established requirements;
- Detect anomalies or point to potential problems in the system;
- Predict the level of quality that will be achieved in the future;

- Monitor changes in quality when software is modified;
- Assess the ease of change to the system during product evolution;
- Validate a metric set.

Software metrics is slowly becoming an integral part of software development and are used during every phase of the software development life cycle. Research in the area of software metrics tends to focus predominantly on static metrics that are obtained by static analysis of the software artifact.

Estimating software quality attributes based on dynamic metrics for the software system are more accurate and realistic [12].

TEACHING SOFTWARE ARCHITECTURE QUALITY BASED ON RUN-TIME METRICS

Software metrics are acknowledged by both software engineering researchers and educators as being of great importance in improving the software development process. Unfortunately, the current practice in industry is to largely ignore metrics and work at an instinctive level.

Current software engineering curriculums emphasizes software analyses, design and construction, but do not address the complexity of a real-world. In software architecture classes the students normally build and deployed many systems, but the patterns of success and failure are not studied.

In the same way, explored theoretical frameworks for describing software architectures and processes to build them are taught, but almost no effort to evaluate them in the real world is done.

In general, software architecture classes are presented in a very theoretical and abstract way, but the students have weak intuitions about high-level architectural abstractions and quality attributes.

Students have strong programming skills, but very seldom know architectural concepts and their influences on software quality, productivity, costs, and maintenance. These students tend to immediately start coding once they receive a problem to be solved [13]. These results 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 [14].

An alternative to improve industry practice seems to be to educate current software engineering students into accepting metrics as a normal part of the software engineering process [15].

To teach to undergraduates software engineering students the importance of software metrics to achieve software quality, a framework for software architecture quality education created by the authors was used [16].

The main purpose of this framework is to help teaching students how to develop software systems from an architectural point of view, considering quality attributes issues and using software metrics to evaluate them.

The framework goals were defined in terms of what students should be able to do after successfully using the framework:

- Acquire an architectural level mental model considering quality attributes;
- Recognize the importance of an architecture-centric approach;
- Create a software metrics plan to improve system quality;
- Generate reasonable architectural alternatives for a problem and choose among them, based on functional and non-functional requirements;
- Construct proof of concepts to evaluate, improve or reject the software architecture of a system;
- Obtain real-world experience in software engineering.

The educational framework is based on the Bloom taxonomy of educational objectives (knowledge, comprehension, application, analysis, synthesis, evaluation).

Bloom's Taxonomy was first designed as a guide for measuring learning objectives in a specific field or domain.

However, it is to provide a fine-grained model for evaluating students' knowledge of software architecture quality based on run-time metrics. In this context the software metrics can be viewed as a body of knowledge (or domain) that the student employs when performing software architecture development and evolution tasks. This taxonomy provides a framework within which students' knowledge of this domain can be assessed [17].

The framework for software architecture quality education 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 trade-offs.
- **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 [18]. In problem-based learning, students are actively involved with problems coming from real practice. The framework uses the following steps to applies the problem-based learning to teach software architecture quality [19]:
  1. Identify concepts and parts of the problem that needs clarification
  2. Define the problem

3. Analyze the problem and brainstorm about solutions
4. Structure solutions
5. State learning objectives
6. Study directed towards learning objectives
7. Report things learned and application to the problem

The students do not realize the importance of non-functional requirements and its influences on software architecture quality. This is caused by the students' lack of experience in working with non-functional requirements. Due the lack of experience or even comprehension, non-functional requirements are not an interesting topic to the students. The framework uses a problem-based learning to teach software architecture quality in a more attractive way.

- **Non-functional requirements metrics:** Non-functional requirements impact directly on measures such as productivity and cost. Ultimately, it is these quantitative measures that 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 [20].
- **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 some areas in which simulation can benefit software engineering, including: assessing the costs of software development, supporting metric collection, requirements and project management, training, process improvement, risk and acquisition management [21]. Simulations can also produce visualizations of the architecture's execution. [22]. 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.
- **Real-world projects:** incorporating real-world problems of sufficient magnitude and complexity into the framework is necessary to enable effective learning of software architecture quality skills and concepts, so the framework attempts to model the "real-world" as closely as possible.
- **Project-based classes**: some courses should be set up to mimic typical projects in industry. These should include group-work, presentations, formal reviews, quality assurance, etc. It can be beneficial if such a course were to include a real-world customer or customers. Students should also be able to experience the different roles typical in a software engineering team: project manager, tools engineer, requirements engineer, etc. [23].
- **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 framework uses a 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 framework 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" [24].
- **Team work:** the Capability Maturity Model (CMM) states that, as organizations reach higher levels of maturity, individual activities become team activities [25].

In the framework for software architecture quality education, the definition of software quality metrics that must be collect is based on architecture assessment. This technique encompass a set is the activity of measuring or analyzing a system's architecture in order to understand its quality attributes. The main goal of assessment of architecture design is improve the potential quality of the system before it is implemented.

Architecture assessment also facilitates the application of design methods and provides the tools to compare design variations and eliminate them, thus reducing the potential solution field.

In order to make non-functional quality attributes measurable or observable, they must be reified as concrete tasks or scenarios. A technique developed at Software Engineering Institute (SEI) is to convert quality attribute requirements into concrete scenarios for users, developers, and maintainers. Scenarios can be a very powerful specification technique because they make seemingly vague or abstract requirements into tangible concrete tasks.

Scenarios can be realized through the creation of a quality attribute utility tree, which is one of the main steps of SEI's Attribute Trade-off Analysis Method (ATAM) [26]. An example utility tree is represented in Figure 2.

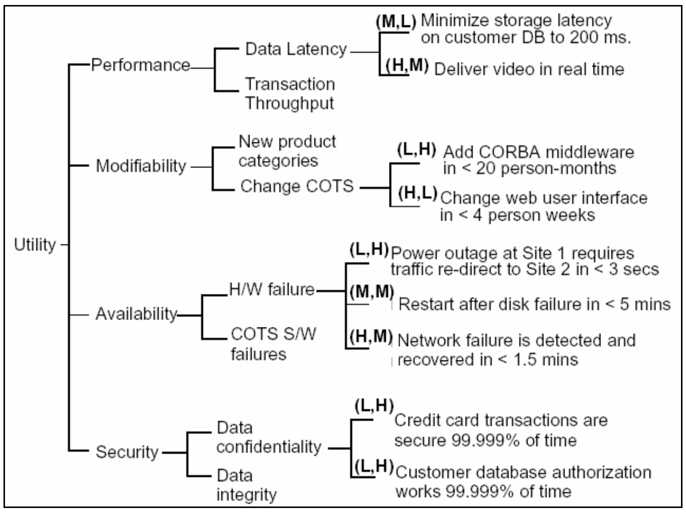

Utility
Performance
Data Latency
Transaction Throughput
(M,L) Minimize storage latency on customer DB to 200 ms.
(H,M) Deliver video in real time
Modifiability
New product categories
Change COTS
(L,H) Add CORBA middleware in < 20 person-months
(H,L) Change web user interface in < 4 person weeks
Availability
H/W failure
COTS S/W failures
(L,H) Power outage at Site 1 requires traffic re-direct to Site 2 in < 3 secs
(M,M) Restart after disk failure in < 5 mins
(H,M) Network failure is detected and recovered in < 1.5 mins
Security
Data confidentiality
Data integrity
(L,H) Credit card transactions are secure 99.999% of time
(L,H) Customer database authorization works 99.999% of time

FIGURE 2 – QUALITY UTILITY TREE [11]

The utility tree is used to translate abstract requirements into concrete scenarios used to analyze a system's architectural design.

APPLYING THE FRAMEWORK

The framework for software architecture quality education has been used to teach undergraduate students of an advanced software engineering laboratory discipline at Computing Engineering and Digital Systems Department of Polytechnic School of the University of São Paulo.

In order to create a learning scenario, the project should not start from scratch but, as in most industrial projects, the project should start with an existing system that needs to be extended, modified or measured. A short, imprecise requirements description and the legacy application are given to students.

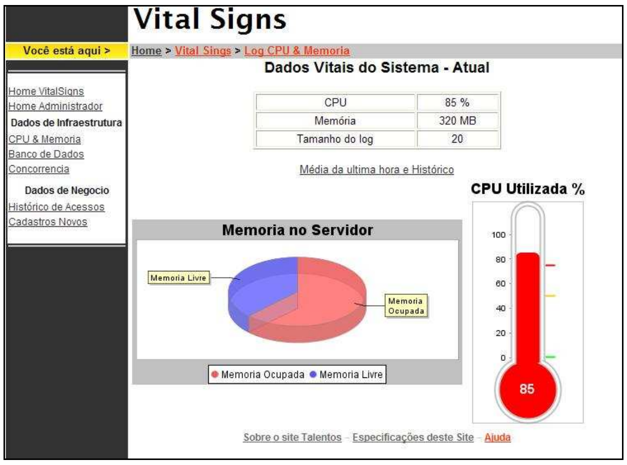
After challenging the students with of a real world project assignment, the primary strategy was to involve and motivate them in a full life cycle team project, where the teacher plays the role of "the client". This gives the students first hand personal experience in the effects of making architectural decisions on their project and on the clients' satisfaction with their product.

The concepts of non-functional requirements and software metrics were presented to the students.

Considering non-functional requirements concepts, an evaluation of the legacy system is done. Possible quality improvements are identified and some non-functional requirements, mainly those that can be measured by run-time metrics, were chosen.

From the chosen non-functional requirements, an ATAM quality attribute utility tree and scenarios were defined to guide the software quality evaluation and metrics.

In order to leverage the pedagogical content of the classes some mechanisms as simulators, proof of concepts, dependency injection and monitoring tools were build by the students. A computer resource monitoring page is shown in Figure 3:


Vital Signs

Você está aqui > Home > Vital Sings > Log CPU & Memoria

Home VitalSigns
Home Administrador
Dados de Infraestrutura
CPU & Memoria
Banco de Dados
Concorrencia
Dados de Negocio
Histórico de Acessos
Cadastros Novos

Dados Vitais do Sistema - Atual

CPU	85 %
Memória	320 MB
Tamanho do log	20

Média da ultima hora e Histórico

Sobre o site Talentos · Especificações deste Site · Ajuda

FIGURE 3 – COMPUTER RESOURCE MONITORING PAGE

The construction and use of these mechanisms are very important to improve the students understanding of software architecture quality and non-functional requirements. Using simulation techniques and proof of concepts is possible to explore many failure situations from real world systems as bottlenecks, resources starvation, low database performance, shared data synchronization. The results of simulations are logged and analyzed by monitoring tools [27].

When the students were introduced to these situations, they rapidly realized how important the non-functional requirements are for the overall software architecture quality.

The collected metrics were evaluated based on the utility quality tree and for the ones under an acceptable quality level, some actions to improve the software architecture were implemented.

The process is repeated until the metric value reaches the desired quality level. The framework activities are represented in an UML activity diagram, shown in Figure 4:

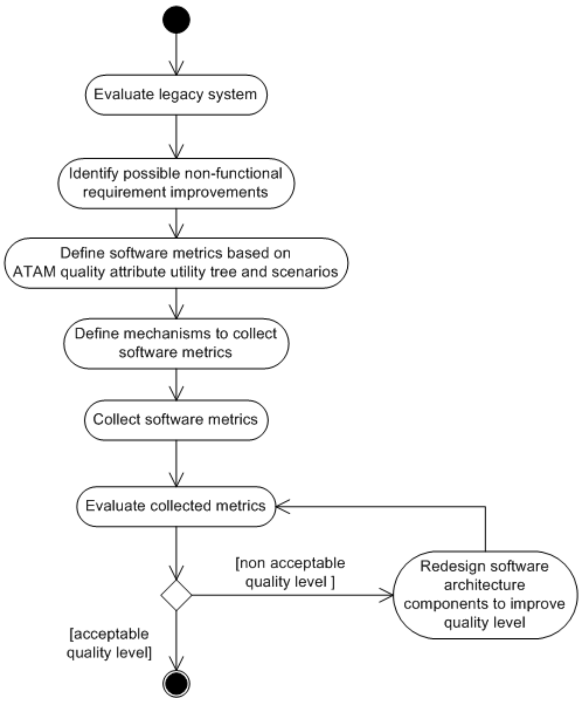

Evaluate legacy system
Identify possible non-functional requirement improvements
Define software metrics based on ATAM quality attribute utility tree and scenarios
Define mechanisms to collect software metrics
Collect software metrics
Evaluate collected metrics
[non acceptable quality level]
Redesign software architecture components to improve quality level
[acceptable quality level]

FIGURE 4 – FRAMEWORK ACTIVITIES

The (re)design and metrics evaluated were presented by each group, so the major architectural decisions were shared by all students

In the classes, the students were able to analyze in practical way the non-functional 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.

The evaluation of the application of framework was done by a learning questionnaire answered by the students. Analyzing the students' answers, the metrics values and the quality attributes of the resulting system (after the cycles of architectural redesign), there are evidences that the concepts taught by the discipline were learned.

CONCLUSIONS

This framework is been developed by the authors since 2003 and has already been applied in 20 class groups, with approximately 18 students per class, resulting in around 5550 class hours, including undergraduate and graduated disciplines and mentoring on the job activities. The students' feedback on the use of framework has been very positive.

The authors believe strongly that including practical software metrics 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.

REFERENCES

[1] Woods, E. and Rozanski, N., "Using Architectural Perspectives", *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture* (WICSA'05), Vol. 00, 2005, pp. 25-35.

[2] Bachmann, F., Bass, L., Klein, M. and Shelton, C., "Designing software architectures to achieve quality attribute requirements", *Software, IEE Proceeding*, Vol. 152, No 4, 2005, pp. 153- 165.

[3] Bazzana, G., Andersen, O., Jokela, T., "ISO 9126 and ISO 9000: friends or foes?", *Software Engineering Standards Symposium*, 1993, pp. 79 – 88.

[4] Kazman, R., Bass, L., Abowd, G., Webb, M., "SAAM: A Method for Analyzing the Properties of Software Architectures," *Proceedings of the 16th International Conference on Software Engineering*. (ICSE 94), 1994, pp. 81–90.

[5] Dobrica L.; Niemela E., "Survey on Software Architecture Analysis Methods". *IEEE Transactions on Software Engineering*, vol. 28, No. 7, 2002, pp. 638-653.

[6] IEEE, "IEEE Standard for a Software Quality Metrics Methodology", *IEEE Std 1061-1998*. (Revision of IEEE Std 1061-1992), 1998.

[7] Albin, S., *The Art of Software Architecture: Design Methods and Techniques*, Wiley, 2001.

[8] ISO/IEC 9126 International Standard. Information technology - Software product evaluation - Quality characteristics and guidelines for their use, 1991.

[9] Losavio, F., Chirinos, L. and Perez, M.A., "Quality models to design software architectures", *Technology of Object-Oriented Languages and Systems, 2001. TOOLS 38. Proceedings*, 2001, pp. 123-135.

[10] Pinna, C., "Um roteiro centrado em Arquitetura para minimização de riscos e incertezas em projetos de software", M.S. thesis, Escola Politécnica da Universidade de São Paulo, São Paulo, Brazil, 2004.

[11] Clements, P., Kazman, R. and Klein, M., *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley Professional, 2002.

[12] Gunnalan, R., Shereshevsky, M. and Ammar, H., "Pseudo dynamic metrics", *Conference on Computer Systems and Applications*, 2005, pp. 117-vii.

[13] Rosca, D.; Tepfenhart, W.; Mcdonald, J., "Software Engineering Education: Following a Moving Target", *Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET'03)*, 2003, pp. 129-139.

[14] Shaw, M., "Software engineering education: a roadmap", *International Conference on Software Engineering - Proceedings of the conference on The future of Software engineering*, 2000, pp. 371-380.

[15] Thomas, R, "A practical experiment in teaching software engineering metrics", *Software Engineering: Education and Practice, 1996. Proceedings. International Conference*, 1996, pp. 226-232.

[16] Andrade, R., "Um framework para o ensino de arquitetura de software", M.S. thesis, Escola Politécnica da Universidade de São Paulo, São Paulo, Brazil, 2005.

[17] Buckley, J. and Exton, C., "Blooms' Taxonomy: A Framework for Assessing Programmers' Knowledge of Software Systems", *11th IEEE International Workshop on Program Comprehension (IWPC'03)*, 2003, pp. 165-174.

[18] Chung, G., Harmon, T.C. and Baker, E.L., "The impact of a simulation-based learning design project on student learning", IEEE Transactions on Education, Volume: 44, No. 4, 2001, pp. 390-398.

[19] Koper, R., "Modeling Units of Study from a Pedagogical Perspective: the pedagogical meta-model behind EML". Open Universiteit Nederland, 2001.

[20] Pasternak, T., "Using trade-off analysis to uncover links between functional and non-functional requirements in use-case analysis", *IEEE International Conference on Science, Technology and Engineering*, 2003, pp.3- 9.

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

[22] Barber, K.S. and Holt, J. Software architecture correctness, *IEEE Software*, Vol. 18, No. 6, 2001, pp. 64-65.

[23] Smolander, K., "What is Included in Software Architecture? A Case Study in Three Software Organizations", *9th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, 2002, pp. 131 – 138.

[24] Ohlsson, L.; Johansson, C., "A practice driven approach to software engineering education", *IEEE Transactions on Education*, Vol. 38, No. 5, 1995, pp. 291-295.

[25] Favela, J., Pena-Mora, F., "An experience in collaborative software engineering education", *IEEE Software*,Vol.18, No. 2, 2001, pp. 47-53.

[26] "O uso de árvores de qualidade para o mapeamento de requisitos não funcionais", class notes for AQS, Mestrado Profissional em Engenharia da Computação, Instituto de Pesquisas Tecnológicas, 2005.

[27] "Explicitando requisitos não funcionais através de simulação", class notes for AQS PCS-2420, Engenharia Elétrica como ênfase em Computação – Departamento de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo , 2006.