Perspectivas y Experiencias en el Desarrollo de un Curso de Arquitectura de Software

Perspectives and Experiences in the Development of a Software Architecture Course

Nicolás López G., MSc., Pilar Villamil G., PhD., Rubby Casallas G., PhD.
Universidad de los Andes, Bogotá, Colombia
{ni-lopez, mavillam, rcasalla} @uniandes.edu.co

Recibido para revisión 28 de Noviembre de 2007, aceptado 14 de Febrero de 2008, versión final 28 de Febrero de 2008

Resumen—**La rápida evolución de la tecnología y TI para soporte a los negocios, así mismo como los nuevos requerimientos para currículos de ingeniería que buscan el desarrollo de habilidades más allá de contenidos temáticos, imponen una serie de retos para el diseño y definición de un curso de arquitectura de software. Propuestas previas para el desarrollo de habilidades necesarias para la práctica de ingeniería de software, relacionadas a arquitectura, se han enfocado principalmente en tecnología, no son lo suficientemente flexibles a largo plazo, y no son adecuadas para el desarrollo de criterios necesarios para que los estudiantes puedan aplicar arquitectura de software en proyectos reales. Adicionalmente, las soluciones basadas exclusivamente en contenidos conceptuales no dan a los estudiantes suficiente campo para el desarrollo de habilidades de diseño en el área de arquitectura de software, particularmente, el impacto de la tecnología en la arquitectura y calidad de un sistema. En este artículo presentamos la evolución del diseño del curso de arquitectura de software para estudiantes de pregrado. Particularmente introducimos nuestra propuesta para una nueva perspectiva del diseño de este curso, usa metodologías de aprendizaje activo y su contenido está centrado en tres grandes áreas: negocio, diseño y tecnología**

Palabras Clave—**Learning Software Architecture, Active Learning, Professional Software Engineering Skills**

Abstract—**The rapid evolution of technology and IT for business support, as well as the new demands on engineering curricula for the development of skills rather than just the presentation of thematic contents, imposes a series of challenges for the design and definition of a software architecture course. Proposals for the development of skills necessary for software engineering practice, regarding architecture, focused on technology, are not flexible enough on the long term, and are not proper for the development of criteria necessary for students to apply architecture in real world projects. Conversely, solutions based solely on conceptual contents do not give enough room for students to develop design skills in the area of software architecture, and particularly, the impact that technology has on architecture and overall system quality. In this paper, we present the evolution of our design for a software architecture course for undergraduate students. Particularly we introduce a proposal for a new perspective for the design of this course; this perspective is based on three broad areas: business, design and technologies using active learning methodologies.**

Keywords—**Learning Software Architecture, Active Learning, Professional Software Engineering Skills**

I. INTRODUCTION

Educators confront many challenges while defining and evolving software architecture courses. This is due to the rapid pace in which the business, technologies and concepts related to architecture advance. The accelerated pace at which business are adopting technologies, and the increasing complexity of business application and business situations that IT must support, demand a new set of skills from software engineering professionals. Various authors [4], [6] present some of the issues in helping students develop the skills expected of software engineers.

The alarming statistics that show how technology projects fail in the real world are a clear indicator that education in software architecture still has many challenges. Usually, software architecture courses focus on the use of technologies such as JEE and .NET. Because of this, it is common that engineers are

competent in the use of specific technologies and are not aware of the business and architectural conditions that help choose one approach over another; sometimes leading to implementations that do not satisfy business needs. Software architects require skills that they must develop starting at their undergraduate education.

Our view of software architecture is that professional practice requires skills in three broad areas: business, design and technology. By business, we mean that students must have skills related to understanding business needs that are the drivers of system architecture. This includes understanding that IT as support for business processes rather than the development of applications that satisfy functional requirements. By design, we mean that students must have the basic skills to identify and understand architectural styles, paradigms and patterns. Beyond having skills to apply specific paradigms, such as Service Oriented Architecture (SOA), they must be able to apprehend a new emerging paradigm and use it in a real world context. Finally, by technologies we mean students must have at their disposal a toolbox of known technologies that support the development of applications using architectural styles, paradigms, and patterns that aim at solving specific business needs. Furthermore, we want students to develop the skill to embark in self-learning of any new technology.

We believe that a software architecture course should be a step to develop this set of skills. However, designing such a course is not an easy task, this design goes way beyond the definition of the thematic content that addresses the subjects related to these skills. We need to develop skills rather than just present a thematic content; the design of this course must be coherent with other software engineering courses, as well as with transversal spaces within a computer science/computer engineering/software engineering curriculum, and must include the definition of methodologies that support this goal [7]. So the question here is which are some appropriate methodologies to develop these skills?

On the other hand, the environment at University of Los Andes has an effect on our objectives. Two main events influence our software architecture course greatly; firstly, the Software Construction group recently implemented a project to change all the basic programming courses [11][12]. As a result, the set of skills before entering the course has also changed. Previously, students entered the course with programming skills in java, skills in software process methodologies, particularly, TSP and modeling skills, specifically a complete course covering subjects including UML and state machines was part of the curriculum. However, with the changes in the curriculum students entering the course now have further developed problem analysis skills, and have more knowledge of other programming related technologies, such as XML, Servlets, JUnit, Ant, etc. Even though the software engineering course that teaches TSP still remains, the modeling course is now not compulsory, so these skills are less developed.

Secondly, not all students entering the course have the same set of skills. The course is also a leveling course for students entering the masters program that did not have an outstanding performance in the admission exam and, in some cases, these students completed an undergraduate program in some other engineering. As a result, some students entering the course only have a basic knowledge of object-oriented programming.

Our objective is to create a software architecture course that deals with all these issues. The course must cover the three broad areas: business, design and technologies, using active methodologies that help develop skills necessary for software architecture practice in the real world. Furthermore, due to the business and technological environment it has to be a course in constant evolution.

We have studied several other proposals for software architecture courses at various universities [1], [5], [8], [9], [3], [2] including the top rated universities in the US [10]. We were surprised to find that out of the top five schools only one has as part of the undergraduate curriculum a software architecture course. For the universities that do have a software architecture course, different methodologies are proposed. Most of the programs we revised focus on the conceptual and theoretical aspects of software architecture and have little emphasis on technology [1], [5], [8], [3], [2]. Some others are oriented around technology [9].

We have redesigned the software architecture course progressively over the past 2 semesters. In this process we have moved from a technology centered course, initially designed to cover thematically a series of technologies left out of other courses in the curriculum to a course based on thematic axes that directly aim at students developing skills related to software architecture. The course is supported by case studies, exercises and a central project of a small size regarding functionality. The project is divided in various cycles that emphasize certain architectural choices, their evaluation according to business conditions, their implementation using state of the art technologies, and the apprehension of the consequences that these choices have.

This paper is organized as follows. Sections 2 and 3 review the main features of the previous versions of the software architecture course. These sections present the main challenge, our solution and the most relevant problems of each approach. Section 4 presents our proposal to create an environment where students practice and acquire the skills necessary for practice in the real world. Finally, section 8 concludes the paper.

II. COURSE VERSION 1

A Challenge

Train undergraduate students in a series of technologies not covered in other courses of the curriculum in the area of Software

Engineering. The course was a walkthrough of technologies, software patterns and non-functional requirements and frameworks related to architecture, with emphasis on technology.

B. Solution

The solution focused on using technologies in which students have weaknesses.

C. Methodology

We created the first version of the course giving lecture classes with presentations; support material was limited to technology documentation pages. During lectures, we used simple examples to present technologies; these were those available as a base example by the technology providers.

Students practiced technologies by means of small workshops; the workshops proposed specific challenges that were not aligned with the material covered during class. Groups of two or three students got together for the workshops. We organized workshops around a central project of medium size with several development cycles. Every two weeks we introduced students to a new technology (or a set of technologies) necessary to implement the workshop. During the course, students developed 6 workshops.

Each workshop had its specific deliverables with specific grading criteria associated; however, there were two main projects. The first project focused on processing, transformation and presentation of information using XML, parsers and patterns like the Builder pattern and MVC. The second one was the implementation of a simple business application using J2EE.

We presented content related to architecture, analysis and methodology as a support to technology subjects. We introduced non-functional requirements each time a technology supported them. In a similar manner, we presented design patterns each time a technology is appropriate for their implementation, and students saw the pattern as the proposed solution for the workshop.

D. Problems

The course was a walkthrough of technologies and frameworks related to software architecture, but it lacked a clear focus on what architecture is and how these technologies help create, define, evaluate and implement a specific architecture.

The concept of software architecture was spread out throughout all the technology subjects. The articulation of these subjects was not clear; the introduction of one technology after another related to the workshops, and students perceived it as being abrupt.

Introducing non-technology subjects was difficult; the student was solely responsible of making necessary generalizations and abstractions of the concepts not related to a specific technology. For example, we did not introduce students with design patterns as means to solve recurring design and architectural problems, but rather as specific ways to implement a certain technology. Additionally, we introduced patterns during lectures in a high abstraction level; this made their implementation by students on workshops a 'reproduction' of how the lecturer says it should be.

On the other hand, students studied the patterns in an order that did not ease their understanding. For example, the builder pattern, which is relatively more complex than others (i.e. delegation), was introduced first, since it was the proposed solution to the SAX workshop. We presented the pattern using the concrete elements of SAX that enabled its implementation.

Students required much more support material to guide their architectural decisions in the workshops. There were no cases or examples similar to the workshop, in terms of neither technology nor business requirements, that supported students in developing theirs. Students had at their disposition the common "hello world" example and the workshops required of them a much more complex development. These examples were far too simple, and students had to face typical implementation problems without the appropriate support tools.

The students perceived subjects related to non-functional requirements as merely informative. Around the middle of the semester, we gave a couple of lectures presenting all the typical non-functional requirements, their definition and some examples of how users perceive them. We presented tradeoffs between obvious requirements, such as performance and fault tolerance using replication, but we did not have available specific examples of decisions where architectural choices influence the quality level of various non-functional requirements, creating tradeoffs. The implementation of non-functional requirements related directly to technologies that guarantee quality levels transparently, specifically, we used J2EE. For example, for concurrence, we proposed the use of web servers and Servlets/ JSP pages with session handling. However, the evaluation, quantification, and certification of non-functional requirements with a specific architecture related to implementation technologies, and not to architectural choice.

As a general conclusion, even though the thematic content of the course was comprehensive, the methodology used posed several problems for the development of specific skills related to the subject of software architecture.

III. COURSE VERSION 2

A Challenge

In this version of the course, the main objective was to prepare students in technologies, of relevance to academy and industry, not covered in other courses of the curriculum in the area of Software Engineering. Additionally, we introduced students to concepts related to architectural styles, non-functional requirements, design and architectural patterns with the objective of developing skills in the use of these concepts.

The main objective of the course, even though it covered concepts related to the study of software architecture, was still familiarization with the technologies. From this issue rises the challenge in methodology that we confront: How can students acquire skills in the use of concepts related to architecture with a course that does not focus on teaching technology?

B. Solution

We base our focus to tackle this challenge on a series of hypotheses related to what we expect students to learn from the course, and how they learn. These are summarized as follows:

1. Students learn technologies by means of direct exposal to their implementation; the concepts learned from an initial exposal to a technology are not sufficient for students to be experts.
2. Concepts surrounding technologies can help learn how to use a technology, focusing exclusively on implementation issues can stop us from reaching this goal. In this case, students develop mechanic skills necessary for the use of specific technologies, which leads to a lack of comprehension and reflection.
3. The incremental development of a project that includes the application of concepts seen during lecture and that uses technology enables continuous evaluation of the student learning progress. Conversely, this helps students to create a consciousness of the consequences of the design and architectural choices they make during the initial phases of the project.

C. Methodology

These hypotheses guided the methodology proposed for the course. During lectures, we presented subjects with audiovisual aids, discussions of design and architectural decisions based on the development of the workshops, as well as support classes to understand the use of technologies (laboratories and support sessions). We maintained the use of two projects divided in incremental workshops developed by two or three students.

A problem presented by the previous version of the course was the difficulty to develop the workshops due to the lack of examples and support material. To tackle this issue, we introduced laboratory sessions that focus on the execution of complete examples similar to the workshops. Students received a "killer application" much more complex and complete than the ones they had to develop. They had to understand some parts of the code that related specifically to their issues in their workshop. During the laboratory, based on the specific challenges of the workshop, students had to implement a small extension of the example.

The laboratories enabled students to develop skills to apply the technology in a different context than that of the workshop. The risk of the approach is that students develop mechanical patterns for the use of a technology, without even wondering how it works.

The laboratories were used only during the second half of the course. This activity had other issues because there was no previous experience in the use of JEE technology, which we recently introduced to the course. Because of this, we had to use an example developed previously for a graduate course. We presented the example using a top-down approach, this meaning that students received a complete application including persistence, business logic and web based GUI components; we also used these three layers for the division of the workshops.

D. Problems

The main problem with the course was that there was a clear separation between the first and second half. During the first half, students had to develop workshops with little support material; this presented the same difficulties as on the previous version of the course. Additionally, during this semester the course was included as a graduate leveling course. This posed an additional challenge since some students had little experience in object oriented programming and java.

On the other hand, the decisions necessary for each workshop were too elaborate, and students had not yet developed the necessary criteria for their evaluation. If students did not make the right choices, each iteration of the project became increasingly more complex: additionally to fulfilling new objectives, students spent long hours correcting issues from previous workshops. Furthermore, it was very difficult to compare the decisions proposed by the different groups and establish discussions around them; thus, students were unable to reflect on conclusions related to design and architectural choices.

During the second half of the course, the decisions that students had to make on each workshop were specific and the laboratories were available as support. However, the decision to use a complete "killer application" example created confusion among students because there were too many concepts and technologies embedded in the example, which students had not studied and discussed. Because of this, they had problems understanding the concepts developed during lectures that hindered the development of the desired skills. In many groups, 'copy-paste' techniques prevailed over comprehension of the technology and concepts. Additionally, even though the proposed laboratory extensions were useful to develop selection criteria, in most cases, students neither elaborated the extensions nor discussed them afterwards.

IV. PROPOSAL FOR A SOFTWARE ARCHITECTURE COURSE

A Challenge

Help students develop a series of design skills in the area of software architecture, as central pillars of these skills are business needs and processes, design and software architecture concepts. Additionally, students should be able to use and understand the impact that technology.

B. Solution

The course focuses on three thematic areas (business, design and concepts, and technology) with the support of methodological tools like architectural styles, design and architectural patterns, industrial development models and specifications, and state of the art technologies. Table 1 presents the thematic axes that, using our proposed methodologically aim at the development of skills necessary for software architecture practice. The three broad areas introduced in the first section of the paper relate to these axes. The broad architecture area is divided into two separate axes: architecture and (architectural) methodology.

TABLE I
THEMATIC AXES FOR PROPOSED COURSE

Architecture
Architecural Styles
Design paradigms
• Objects
• Component Based Design (CBD)
• Application servers / containers
• SOA
Architectural Description Languages
Enterprise Application Integration
Methodology
Design Patterns
• Interface / Implementation
• Delegation
• Factory
• Build
• Observer
• Proxy
Architectural Evaluation: ATAM
Analysis
Business processes
Non Functional Requirements
• Quality scenarios
• Tradeoffs
• Sensitivity points
Technology
JEE
• EJB3
• JMS
• JSF
• JNDI
Jboss Application Server
Web Services
Other component models: Corba, .Net

C. Methodology

We attempt to shift from a passive methodology, were responsibility is left solely to the lecturer, to an active methodology where the lecturer presents problematic situations that raise questions and issues about the student's ability to solve issues with his current set of skills. The student's curiosity to solve these concerns leads him to elaborate exercises and participate in case discussions with similar issues. The discussion focus on architectural choices and the risks and non-risks associated. During discussions, the students' drive to find answers leads the process of development of the desired skills.

We still support lectures using audiovisual aids; however, short discussion worksheets are included within the lectures and we plan laboratories for all technology subjects. The worksheets focus on the main concepts and decisions related with the workshops. During class, students group and discuss the worksheets and the lecturer guides the process of defining conclusions and lessons learnt, this enables him to naturally introduce the concepts underlying and ease their comprehension, as well as show their concrete application. Additionally, we complemented the lectures with a variety of support articles and case studies that students can read before introducing a new subject.

A global project implemented iteratively now articulates the workshops, much like in the previous versions of the course. However, the big difference here is that a sole project is used and all the workshops are heavily articulated around business conditions, both in terms of technology and architectural concepts. The group discusses progressively business conditions that drive architectural decisions of the project, the lecturer introduces new concepts and restrictions as they are necessary, but unlike previous versions of the course, the comprehension of the business and the evaluation of results regarding business conditions and restrictions guide the development.

Another difference in the new workshop is that each now has clearer objectives and challenges that require students to make specific choices and evaluate their consequences. For example, in the first workshop students have to develop entity beans and plan quality scenarios for modifiability. As part of the workshop deliverables, students now have to document the results of the execution of the modifiability scenario. In this case, the decision is the definition of an appropriate entity model to represent business concepts, and the consequence is that an inappropriate model can lead to longer times required for modification.

The articulation of all the workshops creates a consciousness in students of the importance of making good architectural choices in their workshops, since these choices will have an impact on the overall quality of the project. Laboratories are the main support tools for the workshops. Contrary to the situation of the previous version of the course, students used laboratories that incrementally introduced the technologies, so their support for a specific workshop was clearer. This decision lead to an overhead of work from the course support group, nevertheless it proved to be a success to increase the probability of students correctly developing their workshops.

D. Problems

Contrary to previous versions of the course, students can now concentrate more in the relevant design and architectural choices and not just on technology. However, we could not dedicate sufficient time to some subjects; this is the case for design and architectural patterns. Additionally, many groups focused on how to copy-paste and replace code from the laboratory to implement their workshops, without reflection on what they were doing. This resulted in an increase of time dedicated to correction and debugging and many times students did not understand why the problems occurred.

Support articles and cases for the subjects covered in the lectures are another relevant factor for the success of class discussions. However, guaranteeing that students prepare class by reading the articles is still an issue, so we had to adjust the lectures to consider this issue.

V. CONCLUSIONS AND FUTURE WORK

We are aware that there is still a long road ahead to achieve our goals related to education of software architecture and designing the course is a continuous process that needs adjustment by the lecturers according to changes and evolution in technologies and methodologies. However, we believe that our efforts to focus the course according to thematic axes rather than technology make the course maintainable, a characteristic that guarantees its evolution and the fulfillment of its objectives

We still have as future work comparing results for this new version against the previous versions. This analysis will enable us to adjust our proposal and reinforce its weaker points. However, the results from the qualitative evaluations of the course show great improvement in student perception of the contents, methodology; furthermore, we were impressed to see that several students considered the course to be the most relevant of the curriculum for their professional practice.

REFERENCIAS

[1] 17655 Architectures for Software Systems, Course Detail Page, Carnegie Mellon University, 2007. Available: https://acis.as.cmu.edu/gale2/open/Schedule/SOCServlet?CourseNo=17655&SEMESTER=S07&Formname=Course_Detail.

[2] Advanced Analysis and Design, Course description, University College London. Available: http://www.cs.ucl.ac.uk/teaching/syllabus/mscsse/gs02.htm.

[3] Advanced Software Engineering, Course description, University College London. Available: http://www.cs.ucl.ac.uk/teaching/syllabus/ug/3015.htm.

[4] A.T. Chamillard, K. Braun, "The Software Engineering Capstone:Structure and Tradeoffs", In Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, Northern Kentucky, Kentucky, March 2002, pp. 227-231

[5] CS 15-675 Architectures of Software Systems, Course Information, Carnegie Mellon University, 1998. Available: http://www.cs.cmu.edu/afs/cs/project/tinker-arch/www/html/index.html.

[6] M. Gehrke, H. Giese, U. Nickel, T. Niere, J. Wadsack, A. Zfindorf, "Reporting about Industrial Strength Software Engineering Courses for Undergraduates", In Proceedings of the 24th International Conference on Software Engineering ICSE, 2002, pp. 395-405.

[7] A. Rugarcia et Al. , "The Future of Engineering Education. I A Vision for a new century", Chem. Engineering Education, 34(1), pp. 16-25 (2000).

[8] Software Architectures and Patterns: Course Catalogue: King's College London, Course Catalogue, Kings College London. Available: http://www.kcl.ac.uk/international/sae/sa/coursecatalogue/programme/860

[9] Software Architecture, Software Engineering Programme. Part-time Postgraduate Study, Oxford University. Available: http://www.softeng.ox.ac.uk/courses/architecture.html.

[10] US Computer Engineering School Rankings, School Rankings, Available: http://www.infozee.com/channels/ms/usa/computer-engineering-rankings.htm.

[11] J. Villalobos, R. Casallas, "Teaching/Learning a First Object-Oriented programming Course outside the CS Curriculum", Tenth Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, Nantes, France. 2006.

[12] J. Villalobos, R. Casallas, L. Osorio, "Looking for a new approach to teach/learn a first computer-programming course", International Conference on Engineering and Computer Education ICECE, Madrid, 2005.

María del Pilar Villamil G. Profesor Asistente del Departamento de Ingeniería de Sistemas y Computación de la Universidad de los Andes, en el área de sistemas distribuidos e ingeniería de información. Doctora en Informática, Institut National Polytechnique de Grenoble (INPG), Grenoble, Francia. Magister en Ingeniería de Sistemas y Computación - Universidad de los Andes. Ingeniera de Sistemas y Computación - Universidad de los Andes.

Nicolás F. López. Instructor del Departamento de Ingeniería de Sistemas y Computación de la Unviersidad de los Andes. Recibió su titulo de Maestría en Ingeniería de Sistemas y Computación de la Universidad de los Andes en el 2005. Actualmente se desempeña como instructor del curso de arquitectura de software y líder del grupo de desarrollo Qualdev.

Rubby Casallas. PhD de la Universidad Joseph Fourier (Grenoble France). Especialista en Sistemas de Información e Ingeniera de Sistemas y Computación de la Universidad de los Andes. Es Profesora Asociada del Departamento de Ingeniería de Sistemas y Computación Universidad de Los Andes. Directora de la especialización en Construcción de Software ECOS. Áreas de interés: Fábricas de software y líneas de producto basadas en modelos, enseñanza de la ingeniería de software; procesos de desarrollo de software.