

# Software Architecture Education Session Report

Mary Shaw<sup>1</sup>, Hans van Vliet<sup>2</sup>

<sup>1</sup>*Carnegie Mellon University, Pittsburgh, USA*

<sup>2</sup>*Vrije Universiteit, Amsterdam, The Netherlands*

*Mary.Shaw@cs.cmu.edu, hans@cs.vu.nl*

## Abstract

*In the software architecture education session, we discussed four main issues: how to make a software architecture course sufficiently realistic, how to teach non-technical competencies of software architects, the place of such a course in a university curriculum, and how to grow software architects beyond the university. The session resulted in a first sketch of software architecture knowledge areas, and the extent to which these are deemed required for certain classes of software professionals.*

## 1. Introduction

In preparation of this session, the moderators put the following summary on the wiki page for the session:

The subject of software architecture has entered computer science and software engineering curricula. One of the first such courses was developed at CMU some 10 years ago by Mary Shaw and David Garlan. Many others have followed. In this session we will exchange experiences with teaching software architecture, and try to develop some curriculum and teaching guidelines. Topics that could be addressed include: contents of a software architecture course, course material, preferred level (undergraduate, graduate), teaching methods (e.g. does one need the architecture of a big industrial system as illustration, or will a small somewhat artificial case suffice). We are especially interested in experiences of participants in teaching such a course and in good examples of well-documented architectures that could be used as teaching aids.

This summary reflects the academic background of the moderators, and industrial participants suggested some additional topics:

- How can existing curricula be extended or modified to better fit the professional community?

- Are there alternative delivery channels that could make university courses more viable for practicing architects?
- If “architecting is a combination of strategy, technology, leadership, consulting, can an interdisciplinary approach be adopted that adequately covers the broad challenges an architect faces?

Discussion focused on overall content and the form in which it could be presented to three audiences: general computer science / information technology undergraduates, software engineering students, and practicing professionals. We avoided discussion of specific courses on the grounds that individual courses are constrained by local considerations.

## 2. Discussion topics

At the start of the first session, all participants stated their concerns with respect to software architecture education. These concerns can be broadly characterized under four key issues:

- How can we make an architecture course realistic? Issues raised include: how to use examples that are large enough to be realistic, how to show students the complexity involved, how to make a stand-alone course sufficiently interesting in scope and practical aspects, how to effectively cover programming-in-the-large?
- How can we teach extra competencies, such as leadership, management, politics? What is the set of extra competencies required? How to understand the role of an architect?
- What is the place and fit of a software architecture course in the curriculum? Should there be one course, or should the topic be integrated in a number of courses? Some teachers have only one software engineering course, and want to know the key aspects of software architecture to include in that one general course, others can teach an

entire course on software architecture. Is problem-based learning appropriate for teaching software architecture? Can students from different courses serve different roles, such as architect, tester, requirements engineer, etc?

- How can we grow architects beyond universities? There is a need for formal training of practicing architects. What should a practicing architect know, and how do we bring this knowledge effectively to practicing architects? But also: is formal education really what is needed?

The first two topics are discussed in Sections 2.1 and 2.2, and the latter two topics are discussed in Section 3.

### 2.1. How to make the course realistic

Participants generally felt the tension between the limited time available and the tangibility or scale of examples used. A clear need was expressed for good examples, such as standard examples that exist in other fields. A handbook or library can be helpful in providing both good and bad examples, and also which solutions work for which problem. These examples could be critiqued in class. One could use open source projects, and for example reverse engineer the architectures thereof and work from there. Alternatively, companies could be asked to make their favorite architecture available for class use.

### 2.2. Extra-technical skills

We are used to teaching our students technical skills like abstraction, information hiding, etc. It is generally felt that architects need extra-technical skills as well, and maybe even more so than the pure technical ones. Can we use examples to introduce some of the real-world issues (e.g. Harvard Business Review examples such as [3]). To get a handle on the type of extra skills needed, we might analyze what makes architects successful, e.g. the architect competency framework from [1]. This framework combines technical skills with non-technical skills such as leadership, strategy, politics, etc.

### 3. Conceptual content matrix

It was felt that not all audiences could be served well with one and the same set of software architecture topics. A general computer science undergraduate should maybe know a bit about certain software architecture related topics. But a new architect should

definitely know more. So we decided to draw a matrix in which we distinguish on one hand a number of topics related to software architecture, and on the other hand different audiences to which these topics should be taught. The result of this exercise is shown in table 1.

Topic	Grad	Prog	Arch
Basic software engineering skills	1	3	5
People skills	-	3	3-5
Business skills	-	1	3
Architecture techniques	1	2	4
Requirements engineering	1	1-2	4
Software project management	1	1	3+
Programming	2	4	2
Platform technology	2	4	2
Systems engineering	-	-	-
Architecture documentation	-	1	5
Reuse and integration	1	2	4-5
Domain knowledge	1	1	5
Mentoring	-	-	-

**Table 1:**  
**Software Architecture Knowledge Areas**

The values in this table are to be interpreted according to Bloom's taxonomy [2]. Bloom's taxonomy includes three learning types: Attitude, Knowledge, and Skills. For each type there is a progression scale<sup>1</sup>. The numbers in Table 1 may be interpreted to correspond to the Bloom levels Comprehension (1) up to Evaluation (5); in the discussion, larger numbers simply meant "more". Though we did not explicitly say so in the discussions, the contents of the matrix by and large concerns the Knowledge scale. A suggestion for future work would be to present three separate matrices, one for each learning type.

<sup>1</sup> The attributes of the Bloom taxonomy [2] are:

- Knowledge: remembering previously learned material.
- Comprehension: understanding the meaning of material.
- Application: using learned material in new and concrete situations.
- Analysis: breaking down material into component parts to understand its structure.
- Synthesis: putting parts together to form new wholes.
- Evaluation: judging the value of material.

The column titles should roughly be interpreted as follows:

- Grad: someone with a BSc degree in computer science
- Prog: someone with an MSc in software engineering
- Arch: someone with an MSc in software engineering with a software architecture focus.

The “Arch” column has a dual purpose, in that it also refers to architects with some experience, in the same vein SWEBOK [4] refers to the knowledge of a software engineer with four years of experience. An entry of the form “3-5” in particular refers to this learning trajectory.

The entries in this table should be taken with utmost care. We had no time left to revisit and reconsider individual entries. A striking observation was made at the end of the session though: it seems a software architect needs to be more versatile in certain related aspects, such as people skills and integration, than in core technical software architecture aspects.

#### 4. Conclusions

Though no specific conclusions were drawn at the end of the session, the following general observations can be made: (a) in industry, software architects are senior technical leaders responsible for a great deal more than just the systems structure and (b) it would be useful to expand the table in Section 3.

#### Acknowledgements

We gratefully acknowledge the input provided by, and the lively discussions with, the participants of this working session: Art Akerman, Jesper Andersson, Laurens Blankers, Wesley Coelho, Ivica Crnkovic, Adilson Marques da Cunha, Regis Fleurquin, Wahib Hamou-Lhaoj, Anton Jansen, Tomi Männistö, Eltjo Poort, Joaoa Sousa and Judith Stafford. We especially thank Wesley Coelho for acting as wikimaster during the session.

#### 5. References

- [1] Dana Bredemeyer site: [http://www.bredemeyer.com/pdf\\_files/ArchitectCompetencyFramework.PDF](http://www.bredemeyer.com/pdf_files/ArchitectCompetencyFramework.PDF), accessed on Dec. 12, 2005.
- [2] Benjamin S. Bloom and David R. Krathwohl (ed). *Taxonomy of educational objectives: The classification of educational goals. Handbook I, cognitive domain*. Longmans, Green, 1956.
- [3] Thomas H. Davenport. The case of the soft software proposal. *Harvard Business Review*, May-June 1989.
- [4] IEEE Computer Society Professional Practices Committee. *SWEBOK: Guide to the Software Engineering Body of Knowledge, 2004 version*. IEEE Computer Society, 2004.