What Makes Teaching Software Architecture Difficult?

Matthias Galster
University of Canterbury
Christchurch, New Zealand
mgalster@ieee.org

Samuil Angelov
Fontys University of Applied Sciences
Eindhoven, The Netherlands
s.angelov@fontys.nl

ABSTRACT
The software architecture is usually the first design artifact that addresses quality issues (e.g., performance, security). Also, the architecture is reference point for other development activities, e.g., coding and maintenance. Based on our experience teaching software engineering and architecture at different institutions and levels, we discuss what makes teaching software architecture difficult, and how teaching architecture differs from teaching other software engineering topics. Our discussions can help educators design and improve software architecture curricula, and support education researchers in investigating pedagogical approaches and tools for better software architecture training.

CCS Concepts
• Software and its engineering → Software organization and properties → Software system structures → Software architectures • Social and professional topics → Professional topics → Computing education → Computing educational programs → Software engineering education.

Keywords
Software architecture; education; training; learners; students.

1. INTRODUCTION AND MOTIVATION
Tertiary software engineering education usually covers theories, concepts and practices related to requirements, architecture, design, implementation, coding, testing and maintenance. Courses are often designed around curricula recommended by ACM or IEEE. Software engineering education may not expose the very details of each software engineering area, but specific skills will be obtained while practicing in the field. Software churn, contingent workforce, the global economy and a cloud-based, connected environment require lifelong learning [8]. However, educators must ensure that graduating software engineers have a solid understanding of each area of software engineering.

The architecture is the first design artifact that embodies major design choices. These choices tend to affect key quality attributes (e.g., performance, security). Also, even when developing software based on existing systems, the architecture is the starting point for understanding a system and for evolving it. The architecture may be useless for maintenance and other activities if it is not reflected in the code. Northrop argues that all software engineering students should know the principles of architecture and how to use architecture practices [11]. However, as many topics in software engineering, teaching architecture imposes challenges on learner and educators.

Challenges related to teaching software architecture to university students have been published as part of software architecture course descriptions, such as Lago and van Vliet [9] and Mannisto et al. [10], often with the disclaimer that software architecture is still an immature discipline. We aggregate previously reported challenges and lessons learnt related to teaching software architecture and expand on them based on our own experience from teaching software engineering and architecture at undergraduate and graduate level in different countries over the last 10 years. Furthermore, we take a global view on how software architecture compares to teaching other software engineering topics (e.g., coding, testing). We discuss challenges that cannot be resolved and that are not independent of each other and therefore should inform the design of teaching and training plans.

Our findings provide educators with ideas for designing architecture courses and for regularly reflecting on their teaching. This is particularly useful for less experienced educators. Also, based on our findings, education researchers can investigate suitable teaching methods for outcome-based education. Challenges presented here apply to tertiary education but may also appear when training less experienced engineers in industry.

2. CLASSIFICATION FRAMEWORK
We adapt the semiotic triangle [12] to classify software architecture teaching challenges based on their relationship to elements in the learning space (Figure 1). To cover the learner's perspective, we introduce the "Learner" as an additional element.

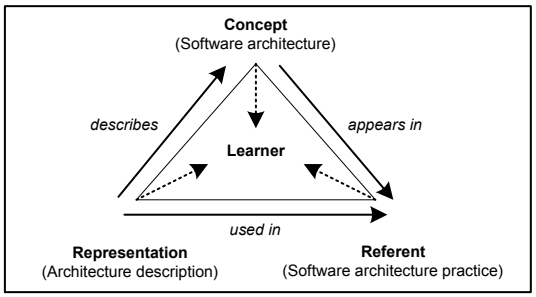

Figure 1. Elements in the learning space of architecture.

- The "Concept" is software architecture. It groups challenges related to the very nature of architecture and the discipline.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICSE'16 Companion, May 14–22, 2016, Austin, TX, USA.
Copyright is held by the owners/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4205-6/16/05 …$15.00
DOI: http://dx.doi.org/10.1145/2889160.2889187

- The "Referent" (i.e., the real world phenomenon in which the concept occurs) is architecture practice. It groups challenges that stem from how software architecture occurs in industry.
- The "Representation" (i.e., the description of the concept in the referent) groups teaching challenges related to the software architecture description and documentation.
- The "Learner" groups challenges related to characteristics and expectations of learners. Learners must be exposed to the concept (as the theory), the referent (as the space where concept is applied) and the representation (as the tangible artifact or codification of the concept). We do not include educators. Educators require general software engineering skills that also pertain in the architecture field. However, due to space limitations we do not explore specific skills in architecture educators (note that some challenges related to educators stem from other challenges presented in this paper; also, we discuss some implications for educators in Section 7).

3. SOFTWARE ARCHITECTURE

3.1 Fuzziness of the Concept

Software architecture as such is a fuzzy concept. High level definitions range from early notions in the 70's of Dijkstra and Parnas of the architecture as the structure of a software system, to Shaw and Garlan's notion of components and connectors as the fundamental concepts (similar to Perry and Wolf), to modern definitions of the architecture (by Bosch) as a set of architectural design decisions and things are "important" and "difficult to change". All notions agree that an architecture is about various related concepts to achieve quality concerns of software. This is different to other areas of software engineering. For example, we have a clear definition of what software testing and maintenance are and why they are important. However, how one defines architecture ultimately depends on the context, the stakeholders, the concerns, and eventually on what the purpose of the architecture is (e.g., performance assessment, workload allocation). This fuzziness causes several issues:

- Educators need to choose which notion to use in their teaching. Teaching all notions would offer a complete understanding of the concept, but introduces confusion in student understanding. Teaching one or few definitions avoids the broader view.
- It is difficult to operationalize the concept of architecture and present it to students as a tangible and useful concept. In particular, computer science students who are used to learning precise concepts perceive the "it depends" nature of what an architecture is as unsatisfying and frustrating.
- Some students (especially strong programmers) are not very strong at abstracting. An architecture is by definition a higher level picture and the abstraction gap to code is big. Students typically think about an architecture as of its documentation which is the "representation" or codification of the abstraction but not the concept.

3.2 "Wicked" Architecture Problems

In addition to the vagueness of the concept of software architecture itself, architecture problems are usually "wicked". The wicked nature of architecture requires educational approaches that deviate from the traditional educator-learner relationship [2]. Wicked means that there are often no clear problem descriptions, no clear solutions, good or bad (rather than true and false) solutions, no clear rules when to "stop" architecting, and mostly team rather than individual work (involving different personalities, opinions). Also, every problem that arises during architecting may be the symptom of another problem. In contrast, other software engineering topics such as programming lead to solutions that can be checked and measured for correctness. Architecture may be the worst-case scenario when it comes to fuzziness in software engineering [13]. This causes several issues:

- It contradicts the mental model of many (especially undergraduate) students according to which they expect well-formulated problems with clean solutions.
- It makes assessment of student work difficult since evaluations become dependent on the assessor. Assessment needs to be more detailed to ensure that the learning progress and status is assessed properly.

3.3 Practical Nature of Software Architecture

The nature of software architecture is practical rather than theoretical (like fundamental mathematics). Therefore, to provide a real practical experience, teaching architecting activities require a suitable context and problems of sufficient complexity (i.e., big enough, several quality attributes). This raises several issues:

- Unlike other topics in software engineering (e.g., programming), it is hard to learn architecture from text books or lectures only with individual self-study. Also, architecture is hard to learn through paper exercises. It is tempting for educators to present architectures as a set of components, connectors and tables, and setting assignments that are about creating different types of UML diagrams and paper reports. However, a real architecture includes code and other artefacts that cannot be captured on paper only.
- Since most architecting decisions happen in teams, learning architecting requires learning of individuals within a team. In team projects, care should be taken to differentiate project process and learning process [4]. Therefore, educators must be able to change hats, from instructor to coach to project stakeholder, etc.
- Small scale in-class examples do not reveal the significance of architecture. Using them may lead to students not appreciating the value of architecting. Similarly, short lived course projects do not reveal what it means to live with consequences of architectural decisions. Choosing a wrong pattern or technology may have no short-term impact but may require some serious refactoring of the architecture as a project moves along.
- Requirements and architecture are intertwined. Many researchers and practitioners have described their co-evolution. We touch on this again in Section 5.

4. ARCHITECTURE DESCRIPTION

The representation of the architecture depends on the definition of what an architecture is, what views are needed and the concerns and stakeholders to be addressed. Therefore, asking students to create an architecture is different to for example asking them to write a Java program. For writing a Java progam, students have a much clearer understanding of what the expected outcome is. This raises several issues:

- Practitioners use often some boxes and arrows as a starting point to visualize and document architectures. UML on the other hand is a modeling language which makes things concrete. However, sometimes UML is not suitable to describe the high level architecture. For students used clear guidelines on what and how to document this introduces confusion.
- Complete architecture exemplars are rare. This could have many reasons, e.g., confidentiality in industrial organizations or the size and complexity that make it hard to publish them (e.g., one part of the SAP ERP architecture documentation alone is more than 200 pages). Most examples (e.g., aosabook.org) provide only partial descriptions of architectures, e.g., only particular views, or only textual descriptions without the full context. Specifics are usually missing that would allow learners to follow the architecture reasoning process.

5. ARCHITECTURE PRACTICE

5.1 Relation to Other Development Activities

Software architecture affects and is affected by other software development activities. Therefore, learners need to understand these other activities. This raises the following issues:

- At what stage should software architecture be taught? Existing published course proposals teach software architecture as part of general software engineering courses, or as dedicated undergraduate or graduate courses. Since software architecture requires an understanding of software engineering in general, it is to teach it as a graduate course, rather than in the first or second year (in some current software engineering curricula, architecture is already well-established in higher courses as advanced topics at the graduate level).
- In software engineering teaching, there is often a distinction between high-level architecting and low-level implementation activities. Overcoming this requires the creation of a mindset in students to not treat architectural decisions in isolation. To intertwine architecture with other activities that are related to it, educators should allow learners to traverse requirements, architecture and implementation at the same time.

5.2 Agility and Architecting

Agile opposes "big upfront design". This is often used to justify the lack of architecting. Nevertheless, there is a need for architecting in agile projects. However, following agile "clichés" and relying on extreme views on the relationship between agile and architecture, there is a risk that naïve learners may not understand that agile approaches as used in the industry still require architecture efforts. This results in the following issues:

- Teaching architecting in agile contexts requires exemplifying a context that motivates the need for architecting activities even more than in traditional development projects. Although there is literature on how to approach architecting in agile projects, there are only few guidelines on this in education [1].
- The agile context adds complexity of teaching software architectures since architecting in agile projects introduces more choices – when to architect, how much, what and how to document. In agile contexts doing all that is necessary but only as little as necessary. For a student such a judgment is difficult given the lack of experience. The architecting freedom offered in agile methods adds to the vagueness perceived by students.

6. LEARNER

6.1 Experience of Learners

Typically, students lack substantial experience in software design and development. However, architecting activities require broad knowledge and experience in order to make informed decisions. A lot of background study may be necessary to understand a technology or a domain and to understand different choices. As a result, students face difficulties when making architectural choices. Unaware of the possibilities or overwhelmed by them, they choose naively [14], or alternatively, spend a lot of time to come to a choice. Characteristics of learners and in particular capabilities, motivation and maturity may affect how learners can reflect on theoretical aspects, deeper conceptualizations and higher abstractions [1]. The lack of experience with complex and multifaceted design decisions may also result in students not fully appreciating the importance of architecture.

A related issue is training students in an architecture course with a background in another area and maybe trained on other software engineering topics. Better insights into this could give a more accurate understanding of the learning effort required to teach students on architecture topics and the skills needed. For example, do these other skills (and skills that were learnt first) influence the mindset of future architects?

6.2 Required Cognitive Processes

The Revised Bloom's Taxonomy is a well-recognized taxonomy of educational objectives for teaching, learning and assessment of students [7]. This taxonomy is the most frequently used taxonomy for specifying learning objectives in computer science [6]. It helps educators classify what they expect students to learn, and describes learning outcomes in two dimensions: the knowledge dimension and the cognitive process dimension. Most learning objectives for software architecture can be categorized in higher cognitive categories (e.g., apply/implement/create, analyze, evaluate/critique/check) rather than lower cognitive processes (e.g., remember/recite, understand). Also, with regards to knowledge categories, most software architecture knowledge is procedural and meta-cognitive, conceptual and procedural, rather than factual [6]. Since learning objectives are usually associated with instructional and assessment activities, this has an impact on how software architecture should be taught. For example, typical methods for deep learning and to address more complex cognitive processes include collaborative learning and experiential learning. These are usually more difficult to implement than lecture or reading-based learning. Also, the behavior and cognitive processes of software architects have been subject of recent studies. An understanding of the reasoning and thinking patterns of architects could help students think more systematically. For example, is the cognitive process different in agile development compared to heavy methodologies? The difference may be more driven by a project or software development approach rather than the cognitive skills needed by a software architect.

7. IMPLICATIONS

We have discussed several implications for educators throughout this paper. Other implications for *educators* are:

- Course content and scope: Theory of architecture should be complemented with complete, practice-rooted examples, including sample problem situations and solution approaches. Furthermore, educators need to make explicit the value of architecting activities. Others have discussed industry

involvement in teaching. Not one course can address all architectural issues. Fraga and Llorens propose learning architecture based on a set of courses [3]. de Boer et al. suggest communities of learners to build architectural knowledge [2].

- Teaching and learning style: Significant more effort (compared to lecture-based courses) is required from educators to teach software architecture as a collaborative and experiential group activity. Also, rather than classroom teaching or textbook reading, the learning outcome is best if teachers are coaches and provide detailed feedback about architectural problems. Reflecting and examining one's own practices leads to better performance [5]. However, investing this effort in teaching may not always be appreciated at research-intensive universities.

Based on the challenges listed in this paper, *education researchers* may focus on the following areas when investigating teaching methods and guidelines for educators:

- Mental models of learners in software engineering and architecture as the foundation for devising new teaching and learning approaches.
- Diagnostic, formative and summative assessment methods to properly capture learning and understanding while ensuring fairness and transparency.
- Intelligent tutoring system (which already exist for teaching topics like databases or UML design) to explore the use of computer-assisted and partially self-guided learning for software architecture.

8. CONCLUSIONS

Software architecture education has made quite some progress in the last few years. For example, Hans van Vliet's reflections on software engineering education from 2006 [15] seem to be reflected in current education. Also, a major lesson from the workshop on key considerations in teaching software architecture at CSEET in 2004 was that at that time the focus was on teaching functional attributes of a system but that we need teaching approaches that stress quality attributes. Nowadays, quality issues seem part of most courses. Teaching software architecture is now also discussed at major software architecture conferences (e.g., ECSA) and some excellent course resources are available[1]. However, given the challenges described in this paper and the human aspect of software architecting, it might be difficult to apply modern teaching approaches, such as MOOC to software architecture education (some MOOC on with "software architecture" in the title already exist but tend to focus on teaching students technologies).

Further to the challenges discussed above, there are other constraints to teaching architecture. We did not explore these since they apply to any practical subject taught at universities (e.g., constraints due to accreditation requirements, number of lectures/labs per week, expertise of staff, interests of learners, expectations from industry).

In our future work we will compare publicly available course descriptions to assess to what degree these address the challenges listed in this paper. Furthermore, we aim at building a repository of software architecture teaching material, including architecture exemplars that can be used for teaching architecture.

9. REFERENCES

[1] Angelov, S. and de Beer, P., "An Approach to Software Architecting in Agile Software Development Projects in Education," in *9th European Conference on Software Architecture* Dubrovnik/Cavtat, Croatia: Springer, 2015, pp. 157-168.

[2] de Boer, R., Farenhorst, R., and van Vliet, H., "A Community of Learners Approach to Software Architecture Education," in *22nd Conference on Software Engineering Education and Training (CSEET)* Hyderabad, India: IEEE, 2009, pp. 190-197.

[3] Fraga, A. and Llorens, J. 2007. The Challenge of Training new Architects: An Ontological and Reinforcement-learning methodology. *Journal of Software* 2, 5 (2007), 24-28.

[4] Gary, K. 2015. Project-based Learning. *IEEE Computer* 48, 9 (2015), 98-100.

[5] Hazzan, O. 2002. The Reflective Practitioner Perspective in Software Engineering Education. *Journal of Systems and Software* 63, 3 (2002), 161-171.

[6] Kiwelekar, A. and Wankhede, H., "Learning Objectives for a Course on Software Architecture," in *9th European Conference on Software Architecture* Dubrovnik/Cavtat: Springer, 2015, pp. 169-180.

[7] Krathwohl, D. R. 2002. A Revision of Bloom's Taxonomy: An Overview. *Theory into Practice* 41, 4 (2002), 212-264.

[8] Kruchten, P. 2015. Lifelong Learning for Lifelong Employment. *IEEE Software* 32, 4 (2015), 85-87.

[9] Lago, P. and van Vliet, H., "Teaching a Course on Software Architecture," in *18th Conference on Software Engineering Education & Training (CSEET)* Ottawa, Canada: IEEE, 2005, pp. 1-8.

[10] Mannisto, T., Savolainen, J., and Myllarniemi, V., "Teaching Software Architecture Design," in *7th Working IEEE/IFIP Conference on Software Architecture* Vancouver, BC: IEEE, 2008, pp. 117-124.

[11] Northrop, L., "Let's Teach Architecting High Quality Software," in *19th Conference on Software Engineering Education & Training (CSEET)* Ohau, HI: IEEE, 2006.

[12] Ogden, C. and Richards, I. 1923. *The Meaning of Meaning*. Harvest / HBJ Book.

[13] Rupakheti, C. and Chenoweth, S., "Teaching Software Architecture to Undergraduate Students: An Experience Report," in *37th International Conference on Software Engineering* Florence, Italy: IEEE, 2015, pp. 445-454.

[14] Tofan, D., Galster, M., and Avgeriou, P., "Difficulty of Architectural Decisions - A Survey with Professional Architects," in *7th European Conference on Software Architecture* Montpellier, France: Springer Verlag, 2013, pp. 192-199.

[15] van Vliet, H. 2006. Reflections on Software Engineering Education. *IEEE Software* 23, 3 (2006), 55-61.

[1] For example, http://cs.mcgill.ca/~martin/teaching/comp529-fall-2015/ or http://avandeursen.com/2013/12/30/teaching-software-architecture-with-github/