A Course on Software Architecture for Defense Applications

Paolo Ciancarini, Stefano Russo and Vincenzo Sabbatino

Abstract What is the role of a software architecture when building a large software system, for instance a command and control system for defense purposes? How it is created? How it is managed? In which way can we form software architects for this kind of systems? We describe an experience of designing and teaching several editions of a course on software architecture in the context of a large system integrator of defense mission-critical systems—ranging from air traffic control to vessel traffic control systems—namely SELEX Sistemi Integrati, a company of the Finmeccanica group. In the last few years, the company has been engaged in a comprehensive restructuring initiative for valorizing existing software assets and products and enhancing their productivity for software development. The course was intended as a baseline introduction to a School of Complex Software Systems for the many software engineers working in the company.

1 Introduction

When developing a large complex software-intensive system, a central concept is that of software architecture.

Once the requirements specification phase is complete, design can start. The main goal of the design is to describe how a system will have to be built in order to meet the specified requirements. Other than functional requirements, the design

P. Ciancarini (✉)
Dipartimento di Informatica, University of Bologna, Bologna, Italy
e-mail: paolo.ciancarini@unibo.it

S. Russo
University of Napoli Federico II, Naples, Italy

V. Sabbatino
SELEX ES, Rome, Italy

P. Ciancarini et al. (eds.), *Proceedings of 4th International Conference in Software Engineering for Defence Applications*, Advances in Intelligent Systems and Computing 422, DOI 10.1007/978-3-319-27896-4_27

must meet the specified non-functional requirements, such as performance constraints, and constraints on dependability attributes, or power consumption constraints; moreover, these goals can be met at minimum manufacturing cost [2]. The first step of the design is to produce a high-level description of the system architecture. The architectural description is a plan for the structure of the system that specifies, at high-level, which components have to be included to carry out the desired tasks and their logical interconnection.

An initial architectural description is usually a diagram that does not distinguish hardware from software features, but rather focuses on the functionality required to each block in the diagram; in a successive phase, the initial block diagram is refined into two block diagrams, one for hardware and another for software [25]. These have to be designed to meet both functional and non-functional constraints. A notation for modeling both hardware and software architectural description is SysML [13]. In this phase, it is crucial to be able to estimate the non-functional properties of components in order to compare and evaluate different architectural solutions.

Making an architectural change in later phases would results in very high costs. The adopted models, formalisms, technologies, and tools in this phase should therefore provide the necessary means to carry out such estimations in easy and inexpensive way. Once the overall architecture has been outlined, the design efforts focus on the components of the architecture.

What is the role of a software architecture when building a large system, for instance a command and control system? How it is created? How it is man-aged? These questions are especially important for large engineering companies which produce families of software-intensive command and control systems [2]. A related question is how can one introduce and teach software architecture and asset reuse to engineers who have been designing systems for years without explicitly dealing with the concept.

The increase of software size and complexity raises critical challenges to engineering companies which build systems to be integrated into larger systems of systems. Production and management problems with software-intensive systems are strictly related to requirements engineering, software design, systems' families management, and their continuous testing integration and evolution.

A number of methods and solutions to these problems are based on the introduction of software architecting in the industrial practice. However, to become effective an architecture-centric development approach needs to be assimilated and put in everyday practice by the company personnel, who need architectural education and training.

In this paper, we describe our experience in designing and developing a course on software architecture for SELEX Sistemi Integrati (also known as SELEX SI, now Selex ES), an Italian company of the Finmeccanica group. The course has been taught in several editions from 2008 to 2015, and it is one of the courses set up in the framework of the public–private cooperation between Finmeccanica and major Italian universities organized in the CINI Consortium.

The paper has the following structure: Sect. 2 presents the company and the market context for which it develops products and systems. Section 3 discusses the requirements and rationale of the course. Section 4 describes the structure of the class and the activities which were performed. Section 5 summarizes the learning results. Finally, Sect. 6 discusses the results of this initiative.

2 The Context

The context of this experience is one of the most important Italian companies for system engineering: Finmeccanica, a large industrial corporate group operating worldwide in the aerospace, defense, and security sectors. It is an industry leader in the fields of helicopters and defense electronics. It employs more than 70,000 people, has revenues of 15 Billion Euros, and invests 1.8 Billion Euros (12 % of turnover) a year in R&D activities.

From Jan 2013, Selex Electronic Systems (in short Selex ES) is Finmeccanica's flagship company focusing on the design of systems. Selex ES inherits the activities of SELEX Elsag, SELEX Galileo, and SELEX SI to create a unified company. The new company operates worldwide with a workforce of approximately 17,900 employers, with main operations in Italy and the UK and a customer base spanning five continents. As the company delivers mission-critical systems in the above fields by utilizing its experience and proprietary sensors and hardware technologies—radar and electro-optic sensors, avionics, electronic warfare, communication, space instruments and sensors, UAV solutions—most of its engineers work on system integration projects and products. As software is becoming an increasing part of these systems, the need raised for updating and upgrading the employees' skills on architecting large-scale complex software systems.

SELEX SI (for the rest of this paper we will use this old company name because the course we describe was commissioned by that company) spends literally millions of hours in software development and integration inside its lines of products and systems. One of the main ways for minimizing development costs and maximizing the reuse of components consists of defining a reference software architecture for each product line [6]. In general, a product line reference architecture is a generic and somewhat abstract blueprint-type view of a system that includes the systems major components, the relationships among them, and the externally visible properties of those components. To build this kind of software architecture, domain solutions have been generalized and structured for the depiction of one architecture structure based on the harvesting of a set of patterns that have been observed in a number of successful implementations. A product line reference architecture is not usually designed for a highly specialized set of requirements. Rather, architects tend to use it as a starting point and specialize it for their own requirements.

We were required to define a course on software architecture strongly emphasizing the power of reuse product line reference architectures.

3 Designing the Course

The company started some years ago a learning program called SELEX Sistemi Integrati Academy. The Academy is an operational tool to enhance and to develop the corporate knowledge heritage and improve over time. The activities are developed by means of three schools: Radar and Systems School, System Engineering School, and School of Complex Software Systems, overall accounting for more than 100 courses.

The academic institution which has contributed to designing the School of Complex Software Systems is the Consortium of Italian Universities for Informatics (Consorzio Interuniversitario Nazionale per l'Informatica, CINI). The School of Complex Software Systems consists of 22 courses. A number of computer science and engineering professors of different CINI universities cooperated to its setup, teaching, and evaluation.

Software architecture was the first course of the School. The course was designed by a small committee including two university professors from CINI (the two first authors of this paper) and a number of people from SELEX SI, including engineering staff members (among them the third author) and the human resources department.

Software structure and architecture is becoming a classic topic for educating software engineers, as can be seen for instance in the latest version of the chapter on software design of the Body of Knowledge of Software Engineering. Also the teaching in an academic environment is quite mature, see for instance [14]. However, the personnel of large companies has a variety of experience and training histories and produce very specific architectures for specific markets. Thus, in an industrial context any teaching of software architecture has to take into account the specific requirements of the company managers.

The course committee mentioned above started from a set of learning requirements given by the company:

- the course should target generically all software developers active in SELEX SI. These are a few hundreds, with a variety of ages and cultural backgrounds. The basic assumptions were some proficiency in object-oriented programming in languages like C++ and concurrent programming in languages like Ada;
- the contents of the course should be customized for such SELEX SI personnel, meaning that the examples and the exercises should focus on real software systems developed by the company. This implied that the teachers should receive some information on the software of interest for the company. Moreover, a number of testimonials from actual projects would make short presentations showing how the main topics of the course were being applied in the company itself;

- the course, overall 35 h, should last one week (five days) or two half weeks, as chosen by the company according to the workload of the different periods.
- each class would include 20 people on average. Prerequisites for attending the class were the knowledge and practice of basic notions of object-oriented programming using a language like C++ or Java.
- all material should be written in English, as customary for all Schools in SELEX Academy.

The learning outcomes of the course on software architecture were defined as a review course on software systems architectures and related software engineering methods for designing complex software-intensive systems. The main idea was to establish a coherent baseline of software development methods, modeling notations, techniques for software asset reuse based on architectural styles and design patterns. More specifically, the learners were expected to obtain a basic knowledge of the principles governing software architecture, and the relationship between requirements, system architectures, and software components (Component-Based Software Engineering—CBSE).

The typical academic software architecture course consists of lectures in which concepts and theories are presented, along with a small modeling project whose aim is to give learners the opportunity to put this knowledge into practice. A text reference for this kind of course is [25]. Our own experience in teaching an academic course in software architecting is described in [5].

In an industrial context, these two activities, namely ex-cathedra lectures and toy project, are not adequate because learners have already some practical experience of large software projects and, more important, there is scarce time to present, compare, and put in practice a variety architectural theories. A reference for a course in an industrial context (Philips) is [16].

In our case, the syllabus was defined as an introduction to software architecture with some reference to software product lines, putting software architecting in a perspective of software reuse, and presenting the related technologies. More precisely, the course had to recall some advanced issues in software modeling and present the main ideas in architectural design for reuse. What follows is a short description of the contents of each lecture.

1. Introduction to software architecture and techniques for software reuse. This lecture introduces the main topic: How software-intensive systems benefit form focussing all the development efforts on the software architecture, aiming at reusing the existing assets for the various product lines architectures. The reference texts for this introduction were two: [7, 18].
2. Describing software architectures. This lecture had the goal to establish a methodological and notational baseline describing an iterative, architecture-centric process able to develop and exploit architectural descriptions for software systems using UML. Thus, its contents are the principles and the practice of modeling and documenting (software) systems with UML, aiming at the ability to describe, analyze, and evaluate a software system using a modeling notation. The reference text for this lecture was [11].

3. Reusable architectural design. This lecture has the goal to discuss the reuse of architectural design ideas: After a short recall of the elementary object-oriented design patterns (Larman's GRASP [15] and the more complex and well-known Gang of Four patterns [10]), the main architectural patterns are introduced and presented in a systematic way. An architectural style can be seen as providing the softwares high-level organization. A number of architectural styles have been identified that are especially important and critical for SELEX SI systems. The reference texts for this lecture were [12, 17].
4. Component and connector views in the context of pattern-oriented software architectures. This lecture has the goal to present some advanced examples of software architecture presenting their components and behaviors. The reference texts for this lecture were [3, 20].
5. Software systems architecture. This lecture has the goal to present some advanced issues of software architecture in the context of model-driven engineering. The main topics are the model-driven architecture and SysML, as used in SELEX SI. The reference text on MDA was [23]; on SysML [8, 9, 27].

4 Teaching the Course

We started giving this course in 2008, and until 2014 there have been six editions. For each edition, the same protocol has been followed. One month before the class week, the teaching slides prepared by the instructors were examined by personnel of the company, who could suggest some changes. After the modifications, the slides were put on line in a reserved site, and access was granted to the students. Thus, the people selected for the class could have a copy of the teaching material about one week in advance.

We had two variants of the timetable: class over one week and class over two half weeks. The one-week timetable is shown in Table 1.

When the course was split over two half weeks, each period had specific entrance and final tests.

Each edition leveraged upon the results of the preceding one, meaning that some adjustments were required to the course contents, usually triggered by some specific need presented by the company. Some suggestions came from the students'

Table 1 Structure of the class and version offered in one week

Time	Day 1	Day 2	Day 3	Day 4	Day 5
09–11	Test-in then lecture	Lecture	Lecture	Lecture	Lecture
11–13	Lecture	Lecture	Lecture	Lecture	Lecture
14–15	Testimonial	Testimonial	Testimonial	Testimonial	Testimonial
15–17	Hands-on	Hands-on	Hands-on	Hands-on	Lecture then test-out

comments, we gathered in class, other from people in the company covering the role of training manager. The suggestions impacted mainly the topic covered by the testimonials. In fact, these varied across the different editions of the course; the recurring topics they covered were as follows:

- The description of software architectures inside SELEX SI [19];
- Using UML for documenting SELEX SI systems;
- Typical architectural styles used inside SELEX SI;
- Experiences with MDA inside SELEX SI [1];
- Using SysML for SELEX SI systems.

All testimonial speakers were engineers of the company engaged in using the related technologies and methods in their projects.

Sometimes, some specific topics were raised and discussed during the lectures. For instance, the idea of architectural simulation [4, 24] was found very interesting and students required more information.

5 Results

The results were measured in two ways:

- Comparing the results of the entrance and final tests. A score sheet with 15 closed questions (each question had four possible answers) was given at the beginning of the first day and at the end of last day. The same set of questions and answers were used in both tests—entrance and final. In the first two editions (for which we have completed the analysis) the entrance test had a 41 % of correct answers, while the final test gave almost 80 % correct answers.
- Gathering the answers of the students to a satisfaction questionnaire. The satisfaction was measured on seven different parameters and averaged 82.7 on a scale 0–100. According to this figure, this course was one of the most appreciated in the School of Complex Software Systems.

6 Lessons Learnt and Conclusions

As mission-critical software systems become more complex, companies—in particular system integrators—feel an increasing need for improving their processes based on sound and effective techniques for software architectures definition and evaluation in specific domains [26]. This is what we observed at least at a nation-wide level in a number of public–private cooperations, which go even beyond the experience described in this paper.

When the field of software architectures emerged, it was argued it should be considered a discipline in its own [22]. After more than a decade [21], the corpus of scientific work in the field has developed consistently, but there is still a large gap between this and what is actually needed in industrial settings. Many of the achievements in the field have not matured enough; an example is architecture definition languages (ADLs) that we believe have not replaced—and are not likely anymore to replace—(extensions to) standard modeling languages. Some others are more mature, but they need to be tailored to the specific software processes, internal industrial standards, and practices.

Teaching software architectures in an industrial context requires to meet the company expectations in terms of mature knowledge and competences transferred to practitioners that they can subsequently turn into the practice of engineering complex systems. This is not easy to achieve, as architecting large-scale complex software systems—having tens of thousands of requirements and millions of lines of code—requires very high abstraction and modeling skills.

In our experience, successful teaching software architectures in an industrial context is not a unidirectional activity, but it benefits a lot from a strong cooperation between company stakeholders for the courses and teaching staff, and it included the following:

- Selecting and teaching the real foundations of software architecture, which have their roots in the very basic principles of software engineering;
- Selecting and teaching those architectural styles and patterns that can meet the real needs of practitioners in terms of design and reuse, with reference to their current target technologies; for instance, as modern middleware technologies are used in the design and development of large-scale software systems, this means defining and teaching proper guidelines that can help engineers to put architecture principles and patterns into their daily practice;
- Interleaving lectures and hands-on sessions with presentations by company testimonials, describing the practical problems, the current internal design standards, and the needs for more modern yet mature techniques;
- Tailoring the theory of software architecture to the specific domain and existing industrial software processes;
- Defining and teaching practical guidelines that engineers can apply reasonably easily to assess and evaluate software architectures based on project/product stakeholders' needs and predefined criteria.

We have also started a revision of our academic course in software architecture; a preliminary analytic comparison between the educational requirements of university students and company engineers is given in [5].

Acknowledgments The authors Ciancarini and Russo acknowledge the support of CINI. All the authors would like to personally thank Prof. P. Prinetto, director of the CINI School on Design of Complex Software Systems, and A. Mura, E. Barbi, A. Galasso, F. Marcoz, M. Scipioni, all from Finmeccanica companies, who in different stages and under different roles cooperated to the success of the various editions of the School.

References

1. Barbi E, Cantone G, Falessi D, Morciano F, Rizzuto M, Sabbatino V, Scarrone S (2012) A model-driven approach for configuring and deploying systems of systems. In: Proceedings of IEEE international conference on systems of systems engineering—SOSE. IEEE Computer Society Press, Genoa, Italy, pp 214–218
2. Bass L, Kazman R, Clements P (2012) Software architecture in practice, 3rd edn. Addison-Wesley, Reading
3. Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M (1996) Pattern-oriented software architecture: a system of patterns, vol 1. Wiley, New York
4. Ciancarini P, Franzè F, Mascolo C (2000) Using a coordination language to specify and analyze systems containing mobile components. ACM Trans Softw Eng Methodol 9(2):167–198
5. Ciancarini P, Russo S (2014) Teaching software architecture in industrial and academic contexts: similarities and differences. In: Yu L (ed) Overcoming challenges in software engineering education: delivering non-technical knowledge and skills. Advances in higher education and professional development. IGI Global, pp 397–413
6. Clements P, Northrop L (2002) Software product lines. practices and patterns. Addison-Wesley, Reading
7. Clements P et al (2010) Documenting software architectures—views and beyond, 2nd edn. Addison-Wesley, Reading
8. Delligatti L (2014) SysML distilled. AddisonWesley/The OMG Press, UK
9. Friedenthal S (2008) A practical guide to SysML: the systems modeling language. Morgan Kaufmann/The OMG Press
10. Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns. Addison-Wesley, Reading
11. Gomaa H (2005) Designing software product lines with UML. Addison-Wesley, Reading
12. Gorton I (2011) Essential software architecture. Springer, Berlin
13. Holt J, Perry S (2013) SysML for system engineering. IET
14. Lago P, van Vliet H (2005) Teaching a course on software architecture. In Proceedings of 18th IEEE conference on software engineering education and training (CSEET). IEEE Computer Society, pp 35–42
15. Larman C (2004) Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development. Prentice-Hall, Englewood Cliffs
16. Muller G (2004) Experiences of teaching systems architecting. In: Proceedings of international council on systems engineering (INCOSE) symposium, pp 1–13
17. Qian K, Fu X, Tao L, Xu C (2006) Software architecture and design illuminated. Jones and Bartlett, Burlington
18. Rozanski N, Woods E (2012) Software systems architecture, 2nd edn. Addison-Wesley, Reading
19. Sabbatino V, Arecchi A, Lanciotti R, Leardi A, Tonni V (2012) Modelling the software architecture of large systems. In: Proceedings of IEEE international conference on systems of systems engineering—SOSE. IEEE Computer Society Press, Genoa, Italy, pp 1–7
20. Schmidt D, Stal M, Rohnert H, Buschmann F (2000) Pattern-oriented software architecture: patterns for concurrent and networked objects, vol 2. Wiley, New York
21. Shaw M, Clements P (2006) The golden age of software architecture. IEEE Softw 23(2):31–39
22. Shaw M, Garlan D (1996) Software architecture. perspectives on an emerging discipline. Prentice-Hall, Englewood Cliffs
23. Stahl T, Voelter M, Czarnecki K (2006) Model-driven software development: technology, engineering, management. Wiley, New York
24. Sterling L, Ciancarini P, Turnidge T (1996) On the animation of not executable specifications by prolog. Int J Softw Eng Knowl Eng 6(1):63–88
25. Taylor R, Medvidovic N, Dashofy E (2009) Software architecture. Foundations, theory, and practice. Wiley, New York

26. Valerio A, Succi G, Fenaroli M (1997) Domain analysis and framework-based software development. ACM SIGAPP Appl Comput Rev 5(2):4–15
27. Weilkiens T (2008) Systems engineering with SysML/UML: modeling, analysis, design. Morgan Kaufmann/The OMG Press