1 Introduction

Nowadays, software is everywhere, from commercial enterprises to virtually all areas of our day-to-day professional, public, and private lives. Air travel, phone calls, bank transfers, or driving would all be next to impossible without software. Software-controlled components can be found in every home and in many everyday devices, from washing machines to cars [BJ+06]. Software is not usually autonomous, but is instead embedded along with hardware and electronics, or as part of the business processes that companies use to generate value [TT+00].

The value and commercial success of companies and products is increasingly determined by software and software quality (see [BM00], [SV99], [TT+00]). Software engineers are thus faced with the challenge of implementing increasingly complex requirements at ever-increasing speed using ever-decreasing budgets while maintaining a high level of software quality.

Continual increase in the size and complexity of software systems has made them some of the most complex human-made systems ever created. The best example is the Internet, which is a truly global software-based system. Internet is now available beyond the bounds of our home planet on the International Space Station (ISS).

A structured and systematic approach to design is essential for the success of software-based systems. Despite the use of established software development methods, the number of unsuccessful software projects remains alarmingly large. To counter this, we need to avoid as many errors as possible, or identify and eliminate them during the early phases of software engineering. Requirements engineering and software architecture are two of these phases. In the words of Ernst Denert, one of the fathers of methodical software development, software architecture is the “Ultimate software engineering discipline” (taken from Denert’s foreword in [Sie04]).

1.1 Software architecture as an aspect of software engineering

Problems with software projects were identified as early as the 1960s, and were referred to then as *"the software crisis"*. From 7–11 October 1968, the NATO Science Committee invited 62 internationally renowned researchers and experts to a conference in Garmisch, Germany, to address the future of software development. This conference is now regarded as the birth of modern software engineering [Dij72].

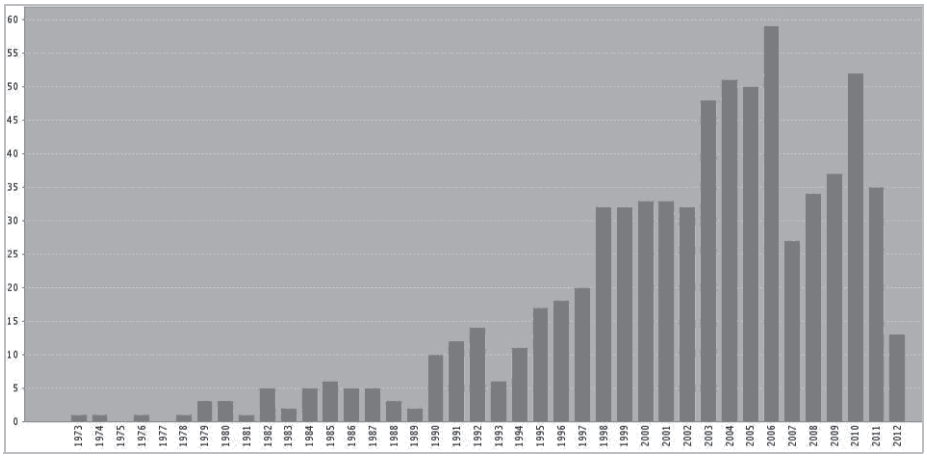

Figure 1-1 *Publications on the subject of software architecture since 1973 [Reu12]*

Compared to traditional engineering disciplines (such as construction) that can fall back on several thousand years of experience, software engineering is still an extremely young discipline. It is therefore no surprise that the sub-discipline of software architecture is even younger. Figure 1-1 shows an increasing number of publications on the subject of software architecture from the 1990s onward [Reu12]. These figures are taken from *The Web of Knowledge*—one of the largest and most renowned publication databases.

With a view to the long history of construction architecture, Marcus Vitruvius Pollio (a Roman architect from the first century BC) was an architectural pioneer. In *De architecture*—nowadays known as *Ten Books on Architecture* [Vit60]—he argued that good architecture can be achieved using a clever combination of the following elements:

- **Utilitas (usefulness):**
  The building performs its function.
- **Firmitas (solidity):**
  The building is stable and long-lasting.
- **Venustas (elegance):**
  The building is aesthetically pleasing.

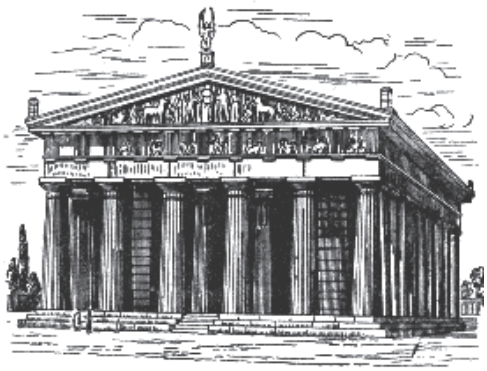

Figure 1-2 *Architecture in ancient Rome*

This hypothesis can be directly applied to the discipline of software architecture. The objective of software architecture (and thus a software architect's primary task) is to construct a system that balances the following three attributes:

- **Utilitas (usefulness):**
  The software fulfills the functional and non-functional requirements of the customer and its users.
- **Firmitas (solidity):**
  The software is stable in terms of the specified quality requirements (for example, the number of simultaneously supported users). It also has to allow future enhancements without having to completely rebuild the system.
- **Venustas (elegance):**
  The software's structure makes it intuitive to use, but also easy to maintain and develop.

1.2 iSAQB: The International Software Architecture Qualification Board

Software architecture is an extremely young discipline and, despite many publications on the subject, various opinions still exist regarding its precise scope and design in the context of computer science and information technology. The tasks and responsibilities of software architects are defined in very different ways and are subject to continual renegotiation during a project.

In contrast, software engineering disciplines such as project management, requirements engineering, and testing have a more mature knowledge base. Various independent organizations offer training curricula that clearly define the knowledge and skills required by these disciplines (for testing, visit *www.istqb.org*; for requirements engineering, visit *www.ireb.org*; for project management, visit *www.pmi.org*).

In 2008, a group of software architecture experts from business, industry, and scientific communities formed the *International Software Architecture Qualification Board* as a registered association under German law (iSAQB e.V., *www.isaqb.org*). The goal of the iSAQB is to define product- and manufacturer-independent standards for the training and certification of software architects. Certifications at *Foundation*, *Advanced*, and *Expert* levels allow software architects to certify their knowledge, experience, and skills using a recognized procedure (see figure 1-3).

Because it eliminates the terminological uncertainty referred to earlier, standardized training benefits established and aspiring software architects, companies, and training organizations. Precise training curricula are essential for the examination and certification of aspiring software, and ensure that high-quality training is available on the basis of an accepted canon of knowledge.

Certification as a *Certified Professional for Software Architecture* (CPSA) is carried out by independent bodies. CPSA *Foundation Level* certification is based on a subset of a non-public catalogue of demanding questions developed by the iSAQB and matched to the curriculum. *Advanced Level* certification also requires practical certification and participation in licensed training courses (or acknowledgement of equivalent non-iSAQB qualifications). *Expert Level* certification is currently in development.

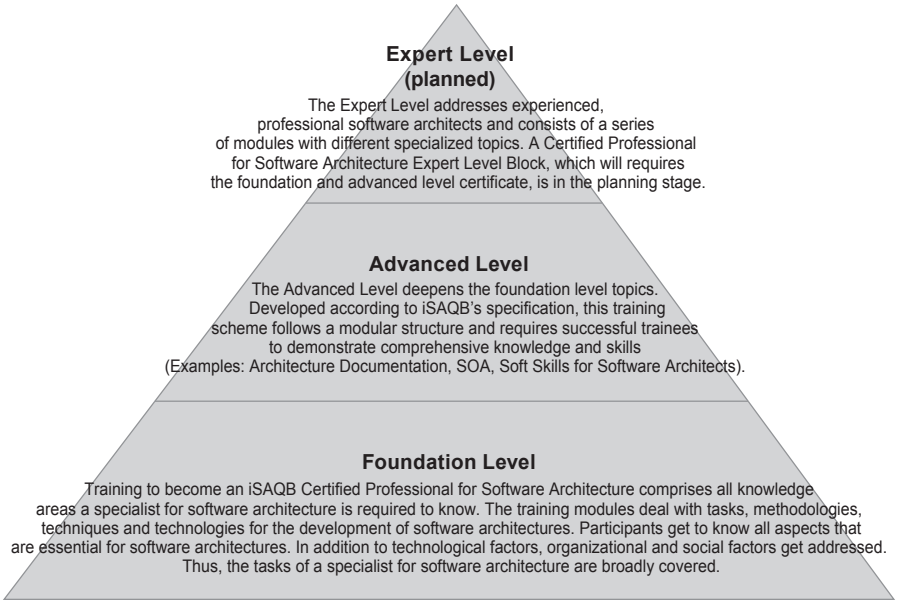

Figure 1-3 *iSAQB certification levels (www.isaqb.org)*

Various licensed training institutions offer multi-day courses designed to refresh and deepen candidates' existing knowledge in these subject areas. Participation in a course is recommended, but is not a prerequisite for registration for the certification examination.

1.3 Certified Professional for Software Architecture – Foundation and Advanced Level

The iSAQB has now defined clear certification guidelines for CPSA *Foundation Level* and *Advanced Level* certification.

Advanced Level certification is modular and consists of individual courses dedicated to specific core competences for IT professionals:

- **Methodical competence**
  Technology-independent skills for systematic approaches to IT projects
- **Technical competence**
  Skills in the use of technology for solving design tasks
- **Communicative competence**
  Communication, presentation, rhetorical, and meeting skills that increase efficiency during the software development process

Prerequisites for *Advanced Level* certification are:

- CPSA-F (*Foundation Level*) training and certification
- At least 3 years' professional experience in the IT sector
- Active participation in the design and development of at least two different IT systems
- At least 70 credit points from all three competence areas (with a minimum of 10 credit points for each)

The examination consists of solving a prescribed task and discussion of the solution with two independent examiners.

For *Foundation Level* certification is based on knowledge and skills defined in the iSAQB curriculum [isaqb-curriculum]. These are as follows:

- The definition and importance of software architecture
- The tasks and responsibilities of software architects
- The role of the software architect within a project
- State-of-the-art methods and techniques for the development of software architectures

The focus is on the acquisition of the following skills:

- Coordinating critical software architecture decisions with other parties involved in requirements management, project management, testing, and development
- Documenting and communicating software architectures on the basis of views, architectural patterns, and technical concepts
- Understanding the main steps involved in the design of software architectures and performing them independently for small and medium-sized systems

Foundation Level training provides the knowledge necessary for designing and documenting a solution-based software architecture for small and medium-sized systems, based on a sufficiently detailed requirements specification. This architecture can then serve as a template for implementation. Participants are trained to make problem-oriented design decisions on the basis of previous practical experience.

Figure 1-4 shows the content and weighting of the individual areas of the curriculum for iSAQB *Certified Professional for Software Architecture (CPSA) Foundation Level* training.

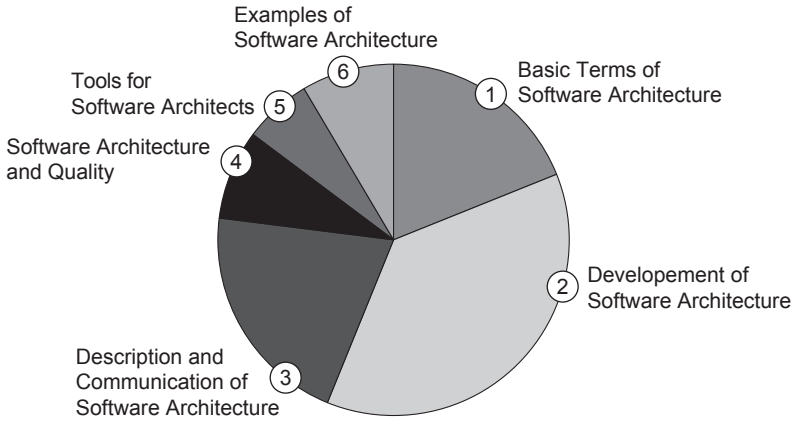

Figure 1-4 *Structure of the iSAQB curriculum for the CPSA Foundation Level training*

Various independent bodies offer certification based on the iSAQB curriculum. Examiners use standardized questions prepared by the iSAQB.

Questions are multiple-choice, so the results are objectively measurable.

The examination validates your software architecture capabilities on paper. It is up to you to prove yourself in real-world situations.

1.4 The aim of this book

Members of the iSAQB developed this book during the creation of the *Certified Professional for Software Architecture, Foundation Level* curriculum. The main aim of the book is to provide a concise summary of the knowledge required to pass the CPSA *Foundation Level* examination, and thus the basic knowledge required for the creation of successful software architectures. This makes the book an ideal reference manual when preparing for the examination. In addition to reading the book, we also strongly recommend participation in the corresponding training courses, which offer practical examples of software architectures and the personal experience of our training staff, both of which go beyond the scope of this book.

The book focuses primarily on methodical skills and knowledge, so specific implementation technologies and tools are not part of the standardized training content. Specific notations and acronyms (such as UML) are to be understood only as examples. The book does not describe individual, specific procedure models or specific development processes, and instead provides various examples.

It explains important terms and concepts involved in software architecture and their relationships with other disciplines. Building on this, it provides an introduction to the fundamental methods and techniques required for design and development, description and communication, and quality assurance in software architectures. It also addresses the roles, tasks, interactions, and work environment of software architects, and describes how they integrate with company and project structures.

1.5 Prerequisites

In line with the aims described above, the book and the iSAQB curriculum assume you have previous experience in software development. The following content is neither part of the book nor the curriculum, although it forms an essential part of every software architect's skill set:

- Several years of practical experience in software development, gained by programming differing projects or systems
- Advanced knowledge of and practical experience with at least one high-level programming language
- Fundamentals of modeling, abstraction, and UML; in particular class, package, component, and sequence diagrams and how they relate to source code
- Practical experience in technical documentation; in particular the documentation of source code, system designs, and technical concepts

Knowledge and experience of object orientation is also advantageous for an understanding of some of the concepts involved. Experience in the design and implementation of distributed applications (such as client-server systems or web applications) is also desirable.

1.6 Reader's guide

The structure of this book is primarily oriented to the structure and content of the iSAQB *Foundation Level* curriculum. For more details, see figure 1-4 and [isaqb-curriculum]:

- In Chapter 2 we describe terms and software architecture basics, which are then addressed in more detail in subsequent chapters. For example, the concept of a software system "view" is introduced within the context of software architecture.

- Practical software architecture design is addressed in Chapter 3. Topics covered include variants of the architecture development procedure; important architectural patterns such as views, pipes and filters, and model view controllers; and design principles such as coupling, cohesion, and separation of concerns.
- Chapter 4 covers proven description tools and guidelines that enable you to document your software architecture and communicate it to others. This is oriented toward a specific target group. Topics covered include the iSAQB view model and cross-cutting concerns in software architectures.
- In Chapter 5 we take a look at the relationship between software architecture and quality issues. Important terms include quality, quality characteristics, ATAM (Architecture Tradeoff Analysis Method), quality tree, compromises (in the implementation of quality characteristics), qualitative architecture evaluation, and the risks involved in achieving quality assurance objectives.
- Chapter 6 lists sample support tools for modeling, generating, and documenting software architectures.
- The appendices include sample questions, a glossary, and a list of reference resources.

Chapters 2 to 5 are essential when preparing for the iSAQB examination, and the other sections are useful too. For general reading, we recommend that you thoroughly read Chapter 2 and then move on to the topics that interest you most.

1.7 Target audience

The primary target audience for this book is anyone who is preparing for iSAQB certification and/or attending iSAQB training courses. The book is also aimed at IT professionals and students who wish to familiarize themselves with the basic terms used in software architecture.

The book also provides an overview of software architecture for software project managers, product managers, and decision makers at the intermediate software development level.

1.8 Acknowledgements

We would like to take this opportunity to thank the iSAQB for its support, and in particular our iSAQB reviewers of previous editions Andreas Rothmann, Phillip Ghadir, and Stefan Zörner. Many thanks to Roger E. Rhoades for the review of the English text. We would also like to thank Ingrid Schindler from the Chair for Software Systems Engineering at Clausthal University of Technology, and the staff at ITech Progress, who provided invaluable support in the preparation of the diagrams. In particular, Christine O'Brien and Robert Kerns have been very supportive in creating the English edition of this book, thank you!

Our thanks also go to our editor Christa Preisendanz for her patience.

Finally, we particularly want to thank our families and partners who gave us the time and space to work together on this book.