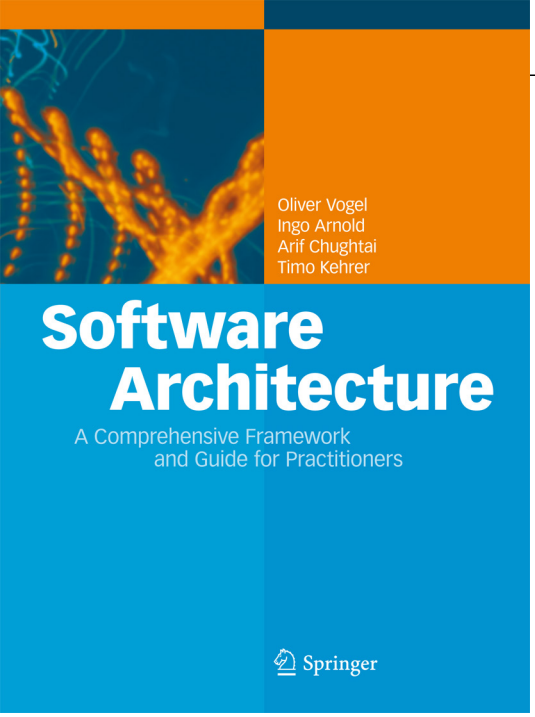

Oliver Vogel
Ingo Arnold
Arif Chughtai
Timo Kehrer

Software Architecture

A Comprehensive Framework and Guide for Practitioners

Springer

Software Architecture

Oliver Vogel • Ingo Arnold • Arif Chughtai
Timo Kehrer

Software Architecture

A Comprehensive Framework and Guide for Practitioners

Springer

All Authors
authors@software-architecture-book.org

Oliver Vogel
oliver.vogel@software-architecture-book.org

Ingo Arnold
ingo.arnold@software-architecture-book.org

Arif Chughtai
arif.chughtai@software-architecture-book.org

Timo Kehrer
timo.kehrer@software-architecture-book.org

Translator
Tracey Duffy
TSD Translations
TraceyDuffy@tsdtranslations.org

Copyright © 2009 by Spektrum Akademischer Verlag, Heidelberg, Germany.
Title of the German original: Software-Architektur. Grundlagen - Konzepte - Praxis
ISBN: 978-3-8274-1933-0
All rights reserved.

ISBN 978-3-642-19735-2 e-ISBN 978-3-642-19736-9
DOI 10.1007/978-3-642-19736-9
Springer New York Dordrecht Heidelberg London

ACM Codes: D.2, K.6

Library of Congress Control Number: 2011933921

© Springer-Verlag Berlin Heidelberg 2011
This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.
The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover Design Editor: KünkelLopka GmbH

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

"The architect should be equipped with knowledge of many branches of study and varied kinds of learning, for it is by his judgement that all work done by the other arts is put to test." Thus opens Chapter I in Marcus Vitruvius Pollio's seminal text, "*The Ten Books on Architecture*" [1]. Readers unfamiliar with Vitruvius' work may find it surprising to learn that it was published in the first century B.C., long before anyone even dreamt of such a thing as software. It is, in fact, the oldest known engineering text. Yet, this nugget of wisdom from a two thousand-year-old text resonates fully today, spanning the full continuum of technological evolution and the growth of engineering knowledge up to our modern world of software.

Vitruvius goes on: "This knowledge is the child of practice and theory" and, yet further, "[i]t follows, therefore, that architects who have aimed at acquiring manual skill without scholarship have never been able to reach a position of authority to correspond to their pains, while those who relied only upon theories and scholarship were obviously hunting the shadow, not the substance."

I cannot think of more apt set of quotations for introducing this book on software architecture. Architects, as Vitruvius tells us, must possess not only the requisite technical knowledge of their domain (i.e., the theory), but they must *understand* it, and, true understanding comes only with direct experience (i.e., practice). Moreover, architects must take a broad perspective that encompasses many varied facets of the systems they are designing: more than just the technical issues and solutions, but also the social, economic, and even psychological factors that are at play. It has been my experience in close to forty years of industrial software development that the primary difference between a competent software architect and a skilled software developer is that architects see beyond the technology. Architects perceive a software system not as a Java or C program or even as software, but as an integral part of a greater system that serves a particular business or technical purpose. Consequently, good software architects are individuals who care deeply about the system and recognize the value that it provides, which means that in the process of design they must learn to become domain experts, but ones distinguished by a deep understanding of computing technology and its capabilities.

The authors of this book are fully cognisant of what makes a true software architect—based on their long-term experience as practitioners. They teach us not only about the fundamental technical tricks of the trade (WITH WHAT) but also

the equally important aspects (WHAT, WHERE, WHY, and WHO) and, last but not least, HOW all of these can be combined to produce a software design that hits the sweet spot. In this, the book distinguishes itself from numerous other books on software architecture—it covers the full spectrum of concerns facing an architect.

I think that we are fortunate to finally have such a comprehensive treatment of the topic at our disposal. For practicing architects, this book can serve as a handy reference—a convenient reminder and check list. For aspiring software architects, it will expose and demystify some of the less well-known but crucial aspects involved in the architectural practice and, perhaps, help identify the gaps they may need to fill to become *bona fide* heirs of Vitruvius' long-standing legacy of engineering excellence.

Bran Selic
Malina Software Corp., Ottawa, Canada

Reference

[1] Vitruvius, "The Ten Books on Architecture," (translated by Morris Hicky Morgan), Dover Publications, Inc., New York, 1960.

Foreword

Architecture skills are becoming increasingly important

For many years now I have been leading the *IT Architect Profession* program at IBM in Europe. It is my job to support the development of IT architects and to ensure that they keep their knowledge up-to-date. Increasing numbers of customers and competitors are interested in building up their own architecture skills. The Open Group, a technology-independent and provider-independent consortium, has been offering the *Open Group Information Technology Architect Certification Program* since 2006. Many of our customers and competitors already use it to evaluate the qualifications of their employees.

This book helps you to build up and expand these skills

In this context I am excited about the new edition of this book. It describes and explains very clearly and in a well-structured way what architects of IT systems do and what IT or software architecture is all about. The book therefore offers a good basis for familiarizing yourself with the topic and improving your architecture skills. It fits perfectly with the current trend that I see both in The Open Group and with our customers and competitors. It reflects the way of thinking that we have been promoting and demanding for many years at IBM.

The time is ripe to get into this exciting topic…

It is a very good time for IT architects. The trends in IT and technology are developing ever further and ever faster. A software architecture as the basis for the development of IT systems has become increasingly important in dealing with these rapid changes. Not least the whole discussion around the topic of service-oriented architecture (SOA) has made that more than clear.

…and develop an architectural awareness

I can therefore highly recommend this book for anyone who has recognized the necessity of dealing with the topic of software architecture. It provides a comprehensive starting point for conscious architectural thinking.

Karin Dürmeyer
IBM Distinguished Engineer
IBM IOT Northeast IT Architect Profession Leader

Preface

As a term, architecture is ever-present …

In everyday IT work, the term “software architecture,” or “architecture” in general, has become ever-present, and due to its enormous relevance for project success, can no longer be ignored. Business cards show job titles such as Security Architect, Data Architect, System Architect, or even Enterprise Architect. We create documents with the title “Solution architecture” for customers, for example, or customers themselves request architecture from suppliers. Although the term “architecture” is used so frequently, on closer inspection, it is clear that architects, project leaders, developers, and other stakeholders do not share a common understanding of the term.

… and interpreted in lots of different ways …

For some of us, “architecture” is the selection and use of a technology; for others, “architecture” is a process; for many, “the architecture” is a folder with drawings containing geometrical figures connected to one another; for others again, “architecture” may be everything that “the architect” produces—whatever this may be. In its practical use, the term “architecture” covers quite a broad scope—that is, it is not defined or understood uniformly. This often makes it difficult for several people to work together and communicate efficiently in the architecture domain and in daily working life.

… initially even within the team of authors

When we decided to write a book about software architecture some years ago, we started our project by initially taking stock. We quickly learned that even within a strictly limited group of experienced software architects, it was not as easy to clearly define software architecture itself as we had expected. We realized that, even though we all had years of experience in designing, describing, or verifying software architectures, we did not have a uniform, precise understanding of the architecture domain.

Our desire for an architecture framework …

We became more and more aware of how important it was to develop a common understanding and vocabulary. An architecture framework that establishes a common, uniform terminology would allow us to look at and explain the architecture topic discriminatingly. This type of holistic framework was something we had always been looking for in our professional careers.

… and orientation

We looked back to the time when we ourselves were primarily software developers and were confronted with the term “software architecture” for the first time. At this point in time, “software architecture” was a very abstract term for us, and it was difficult for us to really grasp what it meant. There was no intuitive architecture framework available that would have enabled us to understand this

important field of topics. Theory and practice concentrated primarily on individual aspects of architecture and did not allow a holistic understanding. We therefore tried to find order amongst the architecture chaos ourselves. For a long time, we had all been subconsciously or intuitively looking for a framework that covers the important dimensions of the architecture domain. At the beginning of our journey through the IT world, we needed a lot of technical and detailed knowledge. We therefore concentrated on acquiring knowledge about techniques and technologies, process models, methods, and organizations. In the course of our professional life and thus throughout our educational journey, each one of us, constantly and partly without being aware of it, derived *his* understanding of the architecture domain from this collection of isolated individual insights. With this book project, we had finally arrived at the point where we could reconcile our individual understandings, bring them together to formulate a common understanding, and make this the core of our book.

Our architectural thinking developed over time

We all knew that there is *no one* architecture examination that gives *the one* architecture certificate that you can pass or acquire in order to then be able to call yourself a certified architect. In the course of our lives as computer scientists, we had all already worked in lots of roles. As analysts, software developers, testers, project leaders, designers, or enterprise architects, we knew that architecture has many faces and that the architecture aspect is decisively important for many roles—not solely for the role of the architect. Our experience was also that, in addition to further technical education, we first had to gather sufficient practical experience before we could start to think “architecturally.”

Our book vision

The primary goal of our book is to give readers orientation in the architecture domain. In our view, many books about architecture focus too heavily on the topic of technology. Other books concentrate on architecture documentation and nomenclatures and their related techniques. Some other books look at solution patterns for architecture problems. And finally, relevant computer magazines regularly cover reports on project experiences in which the architecture aspect of a solution presented is very often the factor that gives the article its substance. However, in our opinion at least, hardly any of these works attempt to give the reader a comprehensive orientation in the topic of architecture. Most of the books we know concentrate only on selected sub-areas of architecture. And the few books that cover architecture more broadly still lack more or less a thorough structure that provides orientation, or rather, a book architecture.

Our challenges

We thus faced two great challenges. The first challenge was to design a book structure that addressed the aspects of orientation, theory, and practice—for us, all of these aspects are equally important. Our second challenge was to develop and describe a software architecture model that then allowed us to work through

the multi-dimensional nature of this topic appropriately and to use it as a stable core for our book. The result of this initial and fundamental work was the architecture of the book itself. We describe this in detail in Chap. 2 and it is structured as follows:

> Explanation of the architecture dimensions (e.g., requirements in the context of architecture) based on a holistic architecture framework.
> Presentation of the parts of the individual architecture dimensions relevant in practice.
> Practical application of the architecture contents covered in the book.

Our book

This book is thus the result of our desire for a work that structures the topics around architecture sensibly, is based on practice, and that conveys corresponding practical experience. In particular, the book is independent of any specific technology and is timeless. For us therefore, this book belongs to that group of fundamental works that provides you with a stable and future-proof reference system that goes beyond current technological trends. The task that we set ourselves with writing this book was not easy—it required all of the authors to look at the topic of architecture intensively and in great depth beyond the otherwise usual level of considering different aspects in isolation. In the time in which we produced this book, we learned a lot. We discussed and debated with one another. As a result of working together on this book, we gained a lot of new and valuable knowledge and a common understanding of architecture.

You now hold our understanding of architecture in your hands. We hope that our claim of arranging and explaining the topic of architecture for you, and anchoring it in practical examples, will help you in your dealings with this interesting and important area of your working life or your studies.

History of the book and the English version

The first edition of this book appeared on the German-speaking market in autumn 2005. In our view, the great success that the first edition enjoyed was connected to the fact that at this time, conceptual, planning, educational, or organizational contributions in IT had gained importance to the extent that specialized technical knowledge was outsourced to countries with pay structures and an expert basis that further encouraged this trend. From then on, the role of the architect, with its holistic and integrative view of the IT challenges, formed the spearhead of a new generation of training profiles within computer science and neighboring domains. This had a corresponding positive effect on the sales of our fundamental work.

The high demand for the first edition of our book meant that we were able to offer our German-speaking readers a revised and updated second edition of the book in 2008.

In the meantime, we received numerous requests from non-German-speaking colleagues to provide an English translation of our book. All of the authors work in an international, primarily English-speaking environment, and, thanks to presentations at IT conferences or university contacts, have regular exchanges with English-speaking colleagues. We therefore quickly agreed when we received a request from Springer for a further revised version of our book—this time in English. We used the opportunity of producing an English translation to improve the contents further based on reader feedback, our practical experience, and current IT developments, such as cloud computing.

Although the translation and the repeated revision of this third edition cost our translator and us as authors many hours of our free time, we are all happy that we took advantage of this opportunity. In particular, we are delighted to finally be able to offer our book to a global audience.

Our thanks

At this point we would like to thank everyone who gave us the freedom to work on this project and who supported us. This includes our partners and children, our friends and colleagues, our employers and superiors. We would like to thank all of those who gave up their time for us and constantly gave us new strength.

Our sincere thanks also go to our translator, Tracey Duffy. With her extremely professional and team-oriented approach and her great talent for technical translation, she provided us with continuous support in realizing this translation project. Her assistance enabled us to meet our high quality standards, and to do so highly efficiently and right on schedule.

Finally, we would like to thank Ralf Gerstner at Springer, who provided us with continuous and professional support in producing this third edition of our book, and who did so with great patience.

Contents

About the Authors

Oliver Vogel is a certified Enterprise and IT Architect with IBM Switzerland. He leads, coaches, and acts as consultant for international projects in architecture topics such as architectural design, implementation, evaluation, and governance. He is also the worldwide IBM Enterprise Architecture Education leader.

Ingo Arnold is a Global Enterprise Architect with Novartis, Switzerland. In addition to his role at Novartis, Ingo is an Associate Professor at the Universities of Basel, Switzerland, and Lörrach, Germany. He is also a well-known speaker at IT conferences, where he holds presentations on topics such as SOA, IT Security, and IT Governance for international audiences.

Arif Chughtai has been a successful freelance IT consultant and IT trainer for more than 10 years. His specialist fields include software architecture, service-oriented architectures, object-oriented software development, and model-driven software development. He regularly shares his expert knowledge in lectures, presentations, and technical articles.

Timo Kehrer is a scientific employee at the Software Engineering Group of the University of Siegen, Germany. He is currently researching model-based software development, model comparison, model version management, and model evolution.

1 | Introduction

This chapter positions the topic of software architecture and provides important basic information. Firstly we will explain the relevance of architecture for developing IT systems. This is fundamental information for the following chapters. We will then show what the concept “architecture” covers in IT. The chapter closes with an overview of the structure of the book, the intended target audience, and the contents of the book. After reading this chapter you will know what architecture means and comprises in IT. You will also know the main aims of our book and how to use it.

Overview

1.1 Starting Position and Aims of the Book

Software is complex and becoming even more so

The desire to implement increasingly complex requirements faster and more cost-effectively, whilst maintaining the same level of software quality, and the complexity of maintaining (global) widely ramified, interlinked IT systems, have put the topic of software architecture increasingly into the spotlight for some years now. This applies not only to commercial business software but also to all other IT domains, for example, embedded systems, mobile communication, or social networks. However, due to the unstructured way in which software is still frequently developed even today, it is difficult to deal with the complexity of software appropriately. You can only successfully overcome the challenge this complexity presents by applying a systematic process that provides structure. Architecture is a deciding factor in this process.

Software architecture has a key position

Architecture has taken up a key position in the successful development of software. The way software is developed is currently changing. In the past, the central element of a developer's role was manual programming. Now, the ability to deal with architectures and to create them is becoming an increasingly important aspect of a developer's job. This aspect is also evident from the different options that now exist for obtaining certification as an architect (see Chapter 7).

Evolution of software development

You can trace these changes in software development if you look at its evolution. During the course of this evolution, a developer first worked at the level of bits and bytes, for example. The developer's activity then shifted to increasingly abstract levels (assembler, procedural programming languages, object-oriented programming languages, etc.). These allowed the developer to perform increasingly complex tasks and implement increasingly complex requirements. As a consequence, the current evolution steps in software development contain model-based and highly architecture-centric concepts such as model-driven software development (MDSD) (see Section 6.2.6), service-oriented architectures (SOA) (see Section 6.4.11), business process modeling (BPM), and the very latest topic, cloud computing (see Section 6.4.13). The awareness for technical quality and the desire to measure it are also increasing. Modern software development tools increasingly take this desire into account and offer corresponding functionality. You can use metrics (e.g., number of dependencies between system building blocks) to check whether developers are considering architecturally significant aspects sufficiently.

Our motivation I: Give orientation for architecture

The motivation to write a book about software architecture arose from the challenges and problems in software development that we, the authors, have been encountering in our professional lives for some years. Two issues are particularly important: firstly, what exactly does architecture cover? We often see a lack of orientation when architecture is a topic on the agenda for projects. Everyone

knows that architecture is a very important topic and should therefore be "done." However, people often do not know what it means exactly, or there is no clear consensus. When people involved in the project talk about architecture, it is often the case that each person understands something different. For some, architecture is the schematic diagrams (box and line diagrams) shown on presentation slides. For others, architecture means defining the signatures of methods and functions. The lack of orientation is often expressed in the following questions:

> How can you assess whether a supposed architecture presented to you is actually architecture?
> How can you determine the quality of an architecture?
> How do you create an architecture?
> How does the thing "architecture," that you have to deliver, manifest itself?
> What do you use to create an architecture?
> What is architecture?
> What is expected of you as an architect or developer when you are asked to create an architecture?
> When and where does architecture take place?
> Who is responsible for architecture?
> Why do you need to create an architecture?

With our book, we want to give people active in the IT field orientation in the topic of architecture. This is because we have observed that many developers and architects are preoccupied with the questions listed above. Also, we have not yet been able to find a book about architecture that offers a clearly structured, comprehensive, and focused introduction to the topic—at least, not in the way that we have often wished.

Our motivation II: Improve software quality

The second important issue is the poor technical quality of software, which is the result of not considering architecture (for example, when you have to rewrite a large part of the source code to take account of new customer requirements).

Our book conveys understanding for architectural thinking

Every IT system has an architecture. But is this an architecture that has been deliberately planned, or has it arisen more or less unconsciously and randomly? The aim should be to achieve a workable architecture. However, a workable architecture does not just "happen"—it has to be developed deliberately [Bredemeyer 2002]. Due to the great importance of architecture for the software quality and the project success, it is very important to have architecture firmly fixed in thought and to thus develop an understanding for it. Helping you to establish architectural thinking and conveying the understanding required to do this are the central aims of our book.

At the beginning there is a "wish list" …

How do architects, developers, and other people involved in projects frequently experience the process of a software development project? We are sure that the following scenario will not be completely new to you. A project generally begins with recording the customer's requirements as quickly as possible in the form of a "wish list". The aim is then to convert this wish list into source code equally fast. There is not a lot of time for questioning the wish list. The focus is on a user interface that satisfies the customer's requirements and is outwardly effective (but not necessarily user-friendly). This gives the customer something tangible quickly, and you can show the customer that you are in control of the situation.

… followed by a "concept" …

Before the points on the wish list can be distributed to the individual developers for processing, the "lead developer" creates a more or less technical and accepted "concept" for the software to be developed based on the wish list. The developers use this concept as instructions.

… changes are suddenly necessary …

During realization—at the latest when requirements change or new requirements suddenly arise—the first shortcomings of the concept appear.

… the project has to deviate from the concept …

In the source code, the developers now have to deviate from the concept and take matters into their own hands. What they do is not documented in the concept because there, of course, nothing is changed "officially". This is because you have already "sold" the concept to the customer in perfectly designed presentations with convincing diagrams. There is also no time to change the concept and the customer would not understand or accept this.

…. the inevitable result: a big ball of mud!

The original concept and the actual source code become increasingly different. The documentation of the concept soon becomes just a pretty cover. Systematic structures that the software once contained are now covered in patchwork. Over the course of time, the software mushrooms into an unfathomable creation along the lines of the big ball of mud pattern [Foote and Yoder 1999], also known as "kludge" [Bredemeyer 2002]:

Why did the software have to end as a big ball of mud?

At some point, you reach the situation where nobody knows exactly why and how the system works. You are just happy that it does work. Maintenance and implementation of new requirements become a bigger nightmare with every version of the software and cost a lot of time and nerves. How did things get so far? After all, you had a concept! Is the wish list to blame? Is there something wrong with the concept? How can you prevent a software becoming a big ball of mud? We asked ourselves these and many other questions and searched for answers. Many of the answers that we present in our book resulted from the fact that, often, insufficient attention is given to architecture when IT systems are created.

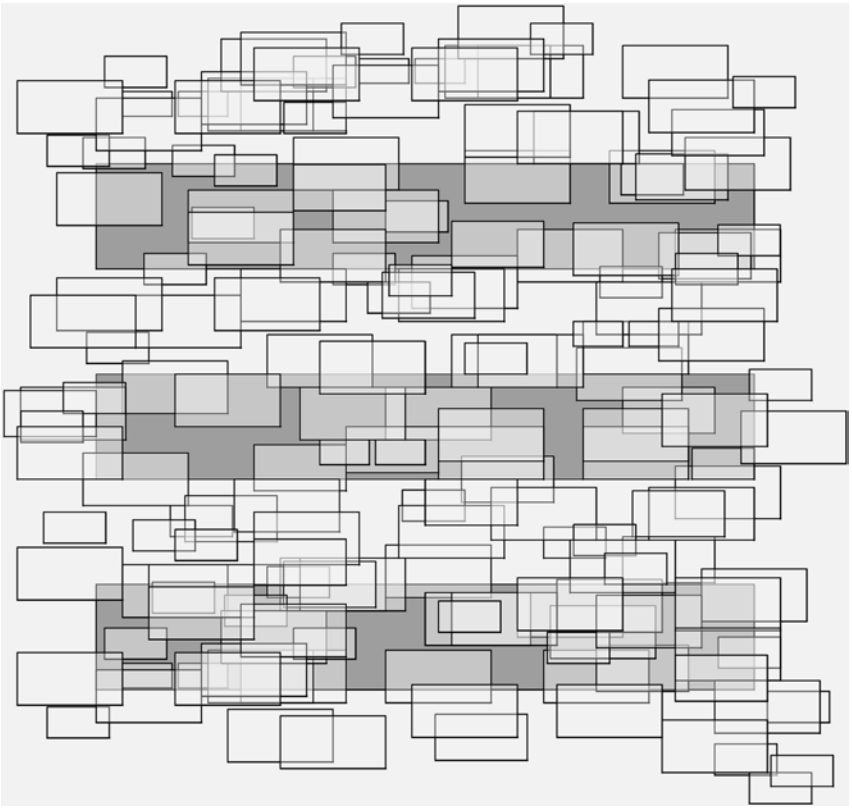

Figure 1.1-1: *Software structures out of control (big ball of mud)*

Many IT projects fail

The project scenario above is not an exaggeration—it is a widespread reality. There are also other scenarios, and they all end in a big ball of mud. Most IT projects fail to some extent. Only around 30% of these projects can claim to conclude successfully [Standish 2009] despite increasingly progressive technologies (e.g., Java EE) and concepts (e.g., SOA). The failure of a project is evident from the project exceeding the time or budget limits, or the customer being unhappy with the product delivered. Projects may even be canceled [Yourdon 2004]. Since the 1960s, this situation has been known as the “software crisis” [Dijkstra 1972]. It first became evident through the immense progress of hardware infrastructure and the related, almost infinite possibilities that opened up for software development. There are many reasons for the software crisis. They include inadequate architectures.

A well-known fact in building construction

In building construction, it is a well-known fact that sooner or later, if you do not have a well-planned architecture, you will encounter problems. If you were to

build a house without first defining the architecture, you would quickly encounter problems with statics, stability, integration in the communal infrastructure (e.g., electricity and water), etc. To stay with the building construction analogy: often, when you "construct" an IT system, you start by defining the approximate overall dimensions, and then, if at all, think quickly about the allocation of rooms and the number of floors. Everything else (e.g., statics and infrastructure for power and water) is supposed to somehow just happen "during construction". The "advance planning" is documented on a scrap of paper and then "off you go". You dig out the space for the foundations, make the molds for the concrete blocks, mix the concrete, and so on. Over time, fundamental errors gradually appear and you have difficulty correcting them or you cannot correct them at all. For example, you realize that the space for the foundations is the wrong size for the concrete blocks you have made. A counterproductive operational hectic follows, in which the situation usually just gets worse.

Symptoms of poor architectures

Unfortunately, the consequences of poor architecture in IT often only appear after a considerable delay. Serious problems may only arise when you go live with a system for the first time, or when it is already in use and you have to adapt it for new requirements. An architecture that arises without being planned—i.e., that simply develops over time—leads to considerable problems in the creation, delivery, and operation of a system. The following selection of symptoms can potentially indicate a poor architecture:

> Results of the analysis are not deliberately considered.
> Overview is missing.
> Complexity runs out of control.
> Planning becomes more difficult.
> Early recognition of risk factors is barely possible.
> Reuse of knowledge and system building blocks becomes more difficult.
> Flexibility is restricted.
> Maintainability becomes more difficult.
> Problems with integration.
> Performance is bad.
> Architecture documentation is insufficient.
> Learning curve for understanding the architecture is too high.
> Functionality is redundant.
> Development cycles (e.g., translation times) are too long.
> System building blocks (e.g., classes) have numerous, unnecessary dependencies to one another.
> System building blocks that cover many different responsibilities and are therefore difficult to maintain or reuse ("monster building blocks").
> System building blocks whose implementation details are known in the entire system.
> Numerous system building blocks have to be adapted when there is a change anywhere in the system (e.g., database or user interface).

Inherent complexity

Even if you have worked out an architecture thoroughly, this is no guarantee that none of the problems listed above will occur. On one hand, this is because poor architecture is only one of many factors for the software crisis (others are, for example, users' lack of awareness for quality or an unsatisfactory IT strategy in the enterprise). On the other hand, successfully creating architectures is no easy challenge due to the inherent complexity of IT systems; on the contrary, as well as having a broad technical knowledge and well-founded experience, those responsible have to take a whole series of other aspects into account (e.g., stakeholders and requirements).

Marchitectures

To introduce and "sell" the main features of an architecture to a non-technical audience (e.g., managers and even lead architects) in an early stage of an IT project, it is often very helpful to work with so-called marchitectures (marketing architectures). These architectures usually take the form of presentation slides with a series of graphical diagrams and keywords. However, all of the other (technical) elements that make up a real architecture are missing. Marchitectures become a problem if you use them in place of a real architecture later on in the project, thus diverting the term "architecture" from its intended use. This is because the primary aim of a marchitecture is to sell something—it does not contain any definable technical "nutritional value" for software developers. You cannot use it as an adequate explanatory model for a system you are developing and the developers will therefore not accept it. In this case, during the software development, an architecture develops more or less unplanned and unconsciously depending on the abilities of the developers.

1.2 What is Software Architecture?

Architecture is difficult to comprehend

In the context of software, architecture is a relatively new discipline. Conscious architectural thinking in software development has only been around for a few decades [Shaw and Garlan 1996]. This is why there are still contradictory opinions on what exactly architecture means. Furthermore, in contrast to physical objects such as buildings, rooms, or even hardware, where it is obvious that these need and contain an architecture, this is not immediately evident for software systems. The result is that in the context of software, architecture is difficult to comprehend. In spite of this, people involved in software development projects are confronted with architecture on a daily basis even though they do not notice it. Architecture is implicitly always an aspect of software and you cannot eliminate it or ignore it—doing so leads to the negative consequences described in the previous section.

Architecture and the business side

Faced with this knowledge, the reasons why architecture has to be in a conflicting relationship with the business side become clearer. If there are numerous questions and uncertainties about architecture on the IT side, this situation is

even more strongly defined on the business side. It is often difficult to convey to the business that there is such a thing as architecture for software. In addition, it is difficult for the business to imagine what direct (financial) benefits an architecture would provide, since investments in architecture only pay off or can only be written off in the medium to long-term. This implies that architecture generally does not bring any benefits until the medium or long-term (e.g., better maintainability), and is therefore only useful for projects with a corresponding long-term time horizon for the system life cycle, corresponding complex requirements, and corresponding high risks with regard to resources, project size, etc. (see Figure 1.2-1). The business is therefore often not prepared to bear the extra costs connected to architecture (often for political reasons, for example, the creation or maintenance of artificial costs in software development). Unfortunately there is no universal solution for overcoming this challenge. Essentially, the issue is making the return on investment (ROI) of architecture tangible for the business. One option is to point out the higher financial costs (for example, due to an increased maintenance effort) caused by neglecting architecture, and which can be avoided in the medium-term, to the business at an early stage. In addition to ROI, as a result of globalization, compliance is now also at the top of the agenda for the business. Here you have to show the connection between architecture and the fulfillment of requirements with reference to IT compliance (for example, the implementation of security aspects with regard to data protection laws).

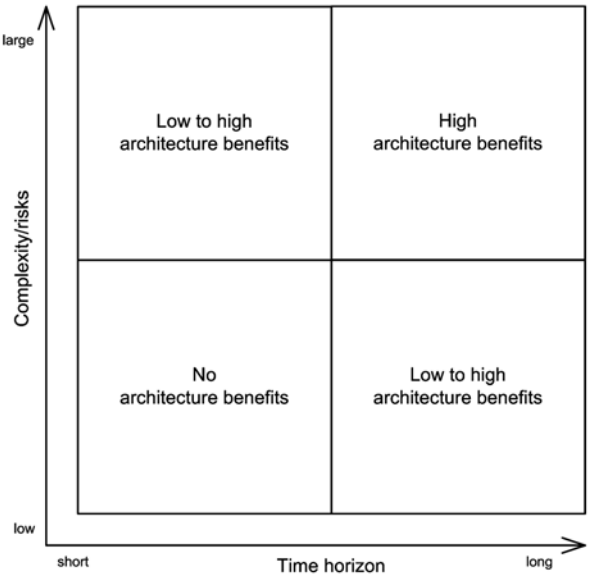

Figure 1.2-1: *Criteria for evaluating the benefits of architecture*

Focus on people

Architecture is not a purely technological issue. It also has numerous social and organizational aspects (see Chapter 7) that can influence the success of an ar-

sample content of Software Architecture: A Comprehensive Framework and Guide for Practitioners

- download online Scandalous
- **The Count Of Eleven book**
- download China Mountain Zhang
- read online The Putin Mystique: Inside Russia's Power Cult

- http://conexdxb.com/library/Selling-Hitler--The-Story-of-the-Hitler-Diaries.pdf
- http://test.markblaustein.com/library/The-Web-Designer-s-Idea-Book-Volume-2--More-of-the-Best-Themes--Trends-and-Styles-in-Website-Design.pdf
- http://musor.ruspb.info/?library/Drawing-the-Living-Figure.pdf
- http://betsy.wesleychapelcomputerrepair.com/library/Nothing-But-the-Truth--Selected-Dispatches.pdf