SOFTWARE ENGINEERING

ARCHITECTURE-DRIVEN SOFTWARE DEVELOPMENT

MK
MORGAN KAUFMANN

RICHARD F. SCHMIDT

Software Engineering
Architecture-Driven Software Development

Richard F. Schmidt

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Morgan Kaufmann is an imprint of Elsevier

MK
MORGAN KAUFMANN

Acquiring Editor: Todd Green
Editorial Project Manager: Lindsay Lawrence
Project Manager: Priya Kumaraguruparan
Designer: Mark Rogers

Morgan Kaufmann is an imprint of Elsevier
225 Wyman Street, Waltham, MA, 02451, USA

Library of Congress Cataloging-in-Publication Data
Schmidt, Richard, 1956-
Software engineering: architecture-driven software development / Richard Schmidt.
pages cm
Includes bibliographical references and index.
ISBN 978-0-12-407768-3
1. Software engineering. 2. Software architecture. 3. Computer software—Development. I. Title.
QA76.758.S364 2013
005.1—dc23 2013000589

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

Printed in the United States of America

13 14 15 16 17 10 9 8 7 6 5 4 3 2 1

Software Engineering

A Note from the Author

Several controversial subjects are raised by the material presented within the book. These provocative topics address the scope of "software engineering" and are central to the author's motivation for publishing this material. If the Software Engineering discipline was well established and proven to achieve successful results, then there would be no need to publish and promote this material. However, this is not the case. The success rate of projects within the software industry has hovered around 30% for the past two decades. The failure of these projects can be associated with two primary symptoms which can be observed in almost every software development project and methodology. The first symptom involves an almost complete misconception of what a software product design is and how to develop a complete design description. The second symptom involves the lack of a standard set of software engineering principles and practices which establish an appropriate scope for a software engineering discipline.

The material presented in this book provides a comprehensive set of practices which are integrated and tightly coupled. However, this material deviates with popular "best practices" which have been encouraged due to the lack of a flawless way to design software. Some of my comments may seem critical; when suggesting an approach to fix a flawed system, criticism is inevitable. The intent is to stimulate the software community into a broad dialog by which a crucial set of software engineering principles and practices can be established.

I hope that the reader can set aside his or her personal opinions concerning mainstream concepts on software engineering. Do not let these controversial topics divert your attention from the fundamental line of reasoning being discussed. This book offers a rigorous, disciplined approach to the engineering of software products. It is time for the software community at large to take action to improve its dismal performance. I hope that this material will prove beneficial to future generations of professional software engineers.

Richard Schmidt
April 15, 2013

Preface

The purpose of this book is to provide comprehensive treatment of the software engineering discipline. The material presents software engineering principles and practices that are based on systems engineering. This book provides a detailed explanation of the essential software engineering philosophy, which emphasizes a disciplined approach to designing software products. To accomplish this, Section 1, Software Engineering Fundamentals, discusses the software development framework and project constructs within which software engineering is performed. Section 2, Software Engineering Practices, presents six technical conventions that convey a philosophy for harnessing computing technologies, applying scientific principles and invoking ingenuity to architect (i.e., design) the structure of software products. Section 3, Stages of Software Engineering Application, discusses the role a software engineering team undertakes within a software development project to establish and control the software product architecture. Each stage of a typical software development project is discussed with a focus on how a software engineering team collaborates with other technical and project-related organizations to influence the architectural design and implementation of software products. These sections clarify the practices, principles, tasks, and artifacts associated with a disciplined approach to software engineering.

The fundamental concepts this material is based on were derived from systems engineering practices to achieve the objectives identified in Table 1. These objectives are achieved by applying a set of principles and practices derived from the systems engineering discipline that have been successfully applied for over 50 years to develop complex systems. The emphasis is on the establishment of a complete software architecture, which enables each element of the product to be specified for fabrication, assembly, integration, and testing (FAIT). Applying these practices to the field of software engineering provides the basis for resolving the challenges listed in Table 1.

Current practices for software analysis and design stem from computer programming languages and the logical constructs by which the languages process data. This has driven software design methodologies, such as object-oriented design, that were not formulated to handle the complexity of advanced software products. By adapting systems engineering practices, this book presents a comprehensive approach to *designing* a software product by establishing rigorous software engineering principles and practices. These software engineering practices are clearly articulated to ensure that there is no uncertainty concerning their importance and applicability to software development. These practices are applied during a walkthrough of the software development process to control, revise, and manage the software architecture throughout a typical software development project context. The contents of this book are aligned with the Software Engineering Body of Knowledge[1] (SWEBOK) key process areas identified in Table 2. This alignment

[1]Institute of Electrical and Electronics Engineers (IEEE) Computer Society, *http://www.computer.org/portal/web/swebok*.

Table 1 Software Engineering Challenges and Objectives

Software Engineering Challenge	Objectives
Design must take place before coding	Know what you are building before you begin to improve cost and scheduling accuracy Reduce product complexity with design detail and precision Cost, schedule, and risk control
Delivering the software technical data package	Complete design diagrams, drawings, and specifications for software implementation (construction)
Allocate requirements among elements of the design configuration	Requirements for decomposition and allocation among software components and units Requirements traceability
Integrated product and process development (IPPD)	Concurrent design and development of product sustainment capabilities Life-cycle costs control
Preparing a software integration strategy	Planned software component integration developed during architectural design activities Efficient software implementation planning
Controlling software complexity	Reduce software maintenance/support costs Efficient, user-friendly interactions
Enabling change assimilation	Stakeholder/user satisfaction Product competitiveness
Trade-off analysis	Cost and schedule control Design optimization Product evolution/incremental release stability Increased probability of project success
Preplanned product improvement	Delayed functionality to later releases to permit on-time product delivery

with the SWEBOK demonstrates how the topics addressed in the book are arranged and associated with the topics addressed by the SWEBOK. However, the SWEBOK is based on current software development practices and does not embrace the systems engineering practices in a rigorous, technical manner.

Book outline and subject matter

The following provides a brief overview of the content of each section and chapter of this book. The sections arrange the material into three coherent topics intended to permit readers to increase their knowledge and understanding of the principles (Section 1), practices (Section 2), and application of software engineering (Section 3). By adapting systems engineering practices to the field of software engineering,

Table 2 SWEBOK Key Process Areas

Key Process Areas	Book Coverage
Software requirements knowledge areas	Section 1 Chapter 3 Section 2 Chapter 7 Chapter 9 Section 3 Chapter 17
Software design knowledge area	Section 1 Chapter 3 Chapter 6 Section 2 Chapter 10 Chapter 11 Chapter 12 Chapter 13 Chapter 14 Section 3 Chapter 18
Software construction knowledge area	Section 3 Chapter 19
Software testing knowledge area	Section 3 Chapter 19 Chapter 20
Software maintenance knowledge area	Section 1 Chapter 5 Section 3 Chapter 17 Chapter 18 Chapter 19 Chapter 20
Software configuration management knowledge area	Section 2 Chapter 9 Chapter 16 Section 3 Chapter 20, configuration audits addressed (FCA/PCA)
Software engineering management knowledge area	Section 1 Chapter 4 Section 2 Chapter 9 Chapter 16, project and technical plans addressed; work packages addressed Section 3, project and technical plans addressed; work packages addressed
Software engineering process knowledge area	Section 2 Section 3

(*Continued*)

Table 2 SWEBOK Key Process Areas (*Continued*)

Key Process Areas	Book Coverage
Software engineering methods knowledge area	Section 2 Chapter 13, Software Design Synthesis Practice object-oriented methods addressed, as applicable Chapter 14, modeling and prototyping addressed
Software quality knowledge area	Section 3, identifies software quality assurance tasks within test and evaluation subsections

this material is intended to provide an innovative, disciplined, and technically demanding approach to developing software products.

Section 1: Software engineering fundamentals

This section discusses the basic principles associated with software engineering and their execution within a software development venue. The fundamental principles, practices, and doctrine are presented to establish software engineering as a professional discipline. Software product characteristics and software development strategies are discussed to stress the challenges confronting software development projects. Software engineering, as an organizational entity, bridges the notable differences in outlook and perception that exist among technical and project management specialists. Therefore, this section addresses the integration of software engineering practices with project management responsibilities and other software development roles.

Chapter 1: Introduction to Software Engineering. This chapter provides an overview of software engineering concepts, principles, and practices that are necessary to cope with the challenges of designing and developing complex software products. Software engineering practices and tools are investigated and their relationships to project management mechanisms are identified.

Chapter 2: Generic Software Development Framework. This chapter discusses the progression of software development activities describing how the software product is defined, designed, and implemented. This chapter tracks a typical software development effort through a series of sequential stages of development separated by project milestones and reviews. The discussion addresses the relationship between the software technical and project management realms of control.

Chapter 3: Software Architecture. This chapter identifies the composition of the software architecture in terms of the software product, computing environment, and post-development processes that enable product sustainment. It relates the generation of architecture design representations, models, and documentation to technical and project-related mechanisms necessary to keep the software development effort within budget and on track for scheduled delivery. Techniques for establishing the software requirement specifications are discussed, and functional and physical architectures are aligned with the stages of software development. This chapter discusses how the software product architecture provides the structural foundation for software implementation (programmatic design, coding, integration, and testing), as well as product life-cycle support.

Chapter 4: Understanding the Software Project Environment. This chapter acquaints readers with the software product characteristics that cause software development to be convoluted and incomprehensible. It addresses the software product complexity challenges and relates those to the project constructs and practices proven to facilitate successful software development endeavors. The discussion provides insight that will help reduce project impediments, upheaval, cancellations, and failures.

Chapter 5: Software Integrated Product and Process Development (IPPD). This chapter presents the philosophy of IPPD and its impacts on project scope and post-development considerations. It attempts to substantiate the need for a well-conceived and structured software architecture to ensure that the product's useful life is extended as a result of engineering attention to life-cycle concerns during development. The simultaneous engineering of software post-development processes is examined to show how early architectural decisions can affect life-cycle and ownership costs.

Chapter 6: Impediments to Software Design. This chapter examines the underlying characteristics of software that cause its "design" praxis to be unconventional and more difficult to fathom. It investigates the characteristics of software as a design and construction material that challenges conventional engineering scrutiny. This chapter presents the software engineering principles that govern the design of software products. Finally, this chapter introduces the software design chasm to contrive a resolution which permits software products to be engineered and designed.

Section 2: Software engineering practices

This section identifies the six practices that contribute to the profession of software engineering: (1) software requirements analysis, (2) functional analysis and allocation, (3) software design synthesis, (4) software analysis, (5) software verification and validation, and (6) software control. Each practice is characterized by a number of tasks that every software engineering professional should comprehend. These practices establish a coherent set of tasks focused on the design and elaboration of *the* software product architecture.

Chapter 7: Understanding Software Requirements. This chapter presents an approach to developing software requirement specifications that are derived from stakeholder needs and expectations and contribute to determining the scope of the software development effort. Software specifications drive the definition of the software architecture, but should not infer any architectural design scheme. Software requirements serve as the point of departure for deriving the software functional and physical architectures. The architecture is engineered by formulating a functional architecture and configuring the physical architecture. Every element of the architecture must be specified and traceable back to the software specifications. The relationships among software requirements, software engineering tasks, and project and technical plans are examined.

Chapter 8: Software Requirements Analysis Practice. This chapter identifies the specific tasks that must be selectively applied to establish the software product and post-development process specifications. This practice involves the allocation of performance quotas among lower-level functional and structural elements of the software architecture. This practice begins with the effort to solicit stakeholder needs and expectations and concludes with establishing a software product requirement baseline.

Chapter 9: Software Requirements Management. This chapter discusses the importance of controlling the software architecture in a proactive manner to facilitate the assessment of proposed changes. Software requirement management tools and practices are considered that enable a software engineering team to perform pragmatic appraisals of the change impact to the software architecture and the latitude of project resources to accommodate a desired alteration. The intent is to equip the development team to react judiciously to authorized changes and to assimilate modifications into the software architecture while not disrupting project scope, plans, or progression toward a successful conclusion.

Chapter 10: Formulating the Functional Architecture. This chapter discusses the nature of the functional architecture and how it is developed by decomposing specified requirements into successive layers of functional elements. Each functional element is specified in an approach of continual refinement that culminates when a function is recognized to be uncomplicated and for which an implementation can be realized. The functional architecture provides a logical and coherent representation of the software product's behavior in response to stimulus, events, or conditions that arise within the computing environment.

Chapter 11: Functional Analysis and Allocation Practice. This chapter identifies the specific tasks that must be considered to ensure that a complete, consistent, and traceable functional architecture is fashioned. Analysis is performed to understand the operational and software product behaviors by examining, decomposing, classifying, and specifying the top-level functions derived from requirement specifications. Performance requirements are allocated among contributing functions to establish measures of effectiveness and performance for lower-level functional elements.

Chapter 12: Configuring the Physical Architecture. This chapter describes the purpose and strategy for arranging and specifying the software product's physical architecture. The physical architecture identifies the foundational building-blocks for software unit design, coding, and testing. The software integration strategy is developed to identify the product structure and prescribes how the software units and components are to be incrementally combined, integrated, and tested to form the complete software product.

Chapter 13: Software Design Synthesis Practice. This chapter identifies the specific tasks that must be considered to ensure that a complete, consistent, and traceable physical architecture is generated. Design synthesis is a proven systems engineering practice for transitioning from a pure functional representation of a product to a physical configuration. It involves a "make-or-buy" trade-off that corresponds to a software "implement-or-reuse" decision.

Chapter 14: Software Analysis Practice. This chapter identifies the specific tasks that must be performed to conduct design-alternative trade-off analyses and risk assessments. Architectural design decisions must be made with sufficient insight to restrain growth in application complexity and software life-cycle costs. The tasks associated with conducting a trade-off analysis and risk assessment are described to provide a basis for understanding the nature of architectural design decisions and their impact on the software development effort.

Chapter 15: Software Verification and Validation Practice. This chapter identifies the specific tasks that must be performed to ensure that the elements of the software architecture remain consistent and aligned with authorized change proposals and requests. Verification tasks must be performed to ensure that the software implementation and test and evaluation efforts are synchronized with the software architecture specifications and design documentation.

Chapter 16: Software Control Practice. This chapter identifies the specific tasks that must be selectively applied to ensure the software product architecture reflects the current design concepts and incorporates authorized change proposals, requests, and design decisions. Requirements traceability must be embedded within the software architecture and associated documentation so that the technical team can promptly and efficiently respond to decisions of the change control boards. In addition, it is necessary for authorized change proposals and requests to be reflected in project and technical plans, schedule, budgets, and work-package descriptions.

Section 3: Stages of software engineering application

This section discusses the roles and responsibilities assigned to technical organizations throughout a software development project. The participation of technical organizations in a software engineering integrated product team (IPT) is stressed.

Chapter 17: Software Requirements Definition. This chapter identifies the manner by which the software requirement specifications are generated by the software engineering IPT. The contributions of participating organizational representatives are identified as the requirements for the software product, and post-development processes are established.

Chapter 18: Software Architecture Definition. This chapter identifies the manner by which the software functional and physical architectures are defined during the preliminary and detailed architecture stages. These stages focus on an IPPD approach to facilitate the establishment of the software implementation, testing, and post-development process infrastructures necessary to facilitate the fulfillment project objectives.

Chapter 19: Software Implementation. This chapter identifies the tasks to be performed by the software implementation organization to programmatically design, code, and test software units and conduct software integration and testing. During this phase the post-development processes are implemented concurrently to support acceptance testing and the deployment readiness review.

Chapter 20: Software Acceptance Testing. This chapter identifies the tasks to be performed by the software test and evaluation organization during the conduct of software product acceptance testing. The roles of the participating organizational representatives are identified as they monitor acceptance testing, react to test failures and respond to software problem reports resulting from acceptance testing. In addition, the post-development processes must be qualified to confirm that they are ready to support software product distribution, training, and sustainment operations.

Contents