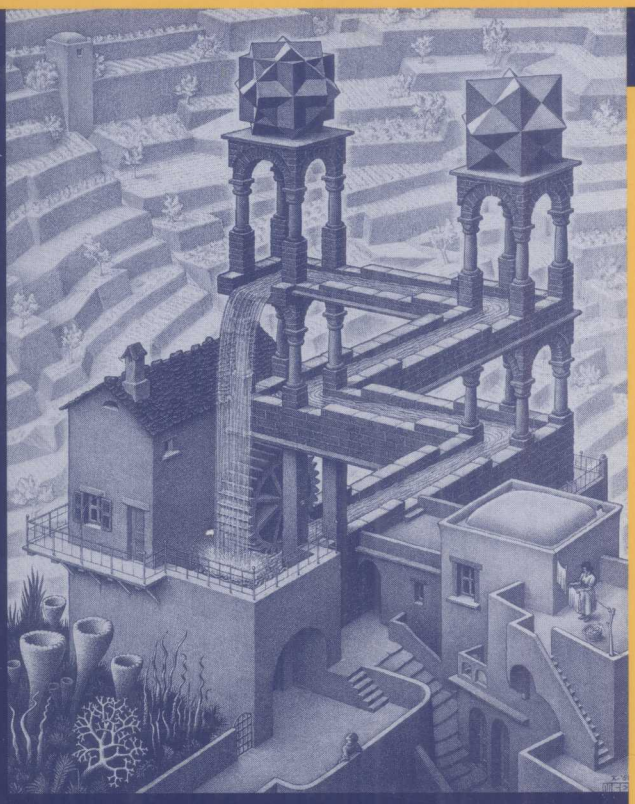
HANS VAN VLIET



SECOND EDITION

Software Engineering

Principles and Practice

WILEY

Software Engineering

Principles and Practice

SECOND EDITION

Hans van Vliet
Vrije Universiteit, Amsterdam

JOHN WILEY & SONS, LTD
Chichester ▪ New York ▪ Weinheim ▪ Brisbane ▪ Singapore ▪ Toronto

Reprinted August 2002, June 2003

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons (Canada) Ltd, 22 Worcester Road, Etobicoke, Ontario M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0–471–97508–7

Typeset in 9/13pt Palatino
Printed and bound in Great Britain by Biddles Ltd, Guildford and King's Lynn
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

To

Marjan, Jasper and Marieke

Preface

Around 1990, my wife and I discussed whether or not we should move. We were getting restless after having lived in the same house for a long time. For lack of space, my study had been changed into a child's bedroom. I definitely needed a room of my own in order to finish the first edition of this book. My wife thought her kitchen too small.

The opportunity to buy a building lot in a new development plan arose. We gathered information, went to look for an architect, subscribed to the plan. Still, we were not sure whether we really wanted to leave our house. It was situated very nicely in a dead-end street, with a garden facing south and a playground in front. When asked, our children told us they did not want to move to a new neighborhood.

So, we asked an architect about the possibility of rebuilding our house. He produced a blueprint in which the altered house had a larger kitchen and four extra rooms. We were sold immediately. We told ourselves that this rebuilding would be cheaper as well, although the architect could not yet give us a reliable cost estimate.

After giving the rebuilding plan some more thought, we decided this was the way to go. My brother, who is employed in the building industry, warned us of the mess it would create. We thought we could handle it. We started the procedure to get the permissions, which takes at least half a year in The Netherlands – and costs money (this was not accounted for).

In August, a year later, we were finally ready to start. The rebuilding was estimated to take 60 working days at most. Unfortunately, the contract did not mention any fine should this period be exceeded. We agreed on a fixed price. Certain things, such as the new electrical wiring, a new central heating unit, and the cost of plumbing were not included. We hardly knew what those 'extras' would cost in the end. We did estimate them on the back of an envelope, and felt confident.

On September 15, the first pile was driven. Counting on good weather throughout the fall, this should have meant that all would be finished by Christmas. The building contractor, however, had other urgent obligations, and progress was rather slow in the beginning. About one week's work was done during the first month.

In October, part of the roof had to be removed. We could interrupt work until next spring – the safe way – or continue – rather tricky in a country as wet as ours. We prayed for some dry weeks and decided to go on. The contractor started to demolish part of our house. While doing so, some surprises showed up and an even larger part of the house had to be demolished. We were really lucky – it only rained for two days while our roof was open. Our bedrooms became rather wet and the kitchen was flooded. Sometime in November, the new roof was up and we could sleep quietly again.

By the end of November, we were getting nervous. There was still a lot of work to be done but several times the workmen did not show up. In the meantime, we had made arrangements for our new kitchen to be installed the week before Christmas. Before this, a door had to be cut in an existing brick wall. The old central heating unit was placed right behind that wall and had to be removed first. The new central heating unit, unfortunately, was not available yet (fall is the peak season for central heating units).

Work continued as far as possible. A new wall was erected, after which we could enter our (old) kitchen only from the outside. For a while, we even lived with no kitchen at all. To make a long story short, the contractor made it, but only barely. The new kitchen was installed. Upstairs, however, much work remained. The project was finally finished by the end of January, only six weeks late.

During the rebuilding, life had been rather provisional. My computer was stored away in the attic. The children had virtually no space to play indoors. Dust was everywhere. These circumstances can be dealt with for a while, but we became frantic towards the end. Though the work seemed to be finished by the end of January, a lot still remained to be done: rooms had to be painted and decorated, and all that had been packed needed to be unpacked again. It took several more months before life took its normal course again.

Several months later, some of the new wooden planks on the back façade started to crack. They had expanded during the summer heat; either the tongue was

too wide or the groove too narrow. This, and various other minor problems were, eventually, rectified.

On the financial side: various tiny expenses not accounted for added up to a pretty sum. I am still not sure whether we chose the cheapest option, but I am absolutely sure that knocking down a house and rebuilding it while you still try to live in it is a nightmare. In that sense, my brother was more than right.

After the house rebuilding project and work on the first edition of this book was finished, I turned my attention to tidying up our garden. I designed a garden with various borders, terraces, a summer house and a pond. And I carried out all the work. I made one big mistake on this second project, which only manifested itself a couple of years later when I wanted to repaint the back façade. In order to do so, I had to put the legs of the ladder in the pond. So I did some rework and moved the pond.

This story is fairly typical of a software development project. Software too is often delivered late, does not meet its specifications and is, moreover, faulty. Software projects also tend to underestimate the impact of non-technical factors. The growing awareness of this in the late 1960s gave rise to the expression 'the software crisis.' Though we have made quite some progress since the term 'software engineering' was first coined back in 1968, many people are of the opinion that the software crisis is still rampant.

The field of software engineering aims to find answers to the many problems that software development projects are likely to meet when constructing large software systems. Such systems are complex because of their sheer size, because they are developed in a team involving people from different disciplines, and because they will be regularly modified to meet changing requirements, both during development and after installation.

Software engineering is still a young field compared to other engineering disciplines. All disciplines have their problems, particularly when projects reach beyond the engineers' expertise. It seems as if software development projects stretch their engineers' expertise all of the time.

The subject is rapidly moving and there are more questions than answers. Yet, a number of principles and practices have evolved in the thirty-odd years the field has existed. To foster and maintain software engineering as a professional discipline, the IEEE Computer Society and ACM have started a joint project – SWEBOK – to identify and validate the software engineering body of knowledge. This project will run from 1998 to 2001. The outcome of this project will be essential to the formulation of licensing requirements for software engineers. The licensing and accreditation of software engineers has already started, and is likely to become an important topic in the forthcoming years [Bag99].

This book addresses all of the knowledge areas identified in the SWEBOK project[1]. Of course, the relative attention paid to individual topics and the way these topics are treated reflects my own view of the field. This view can be summarized as follows:

- What is theory today may become practice tomorrow. For that reason, I have not limited myself to a discussion of well-established practices. Rather, I also pay attention to promising methods and techniques which have not yet outgrown the research environment, or hardly so: software reusability, quantitative assessment of software quality, and formal specifications, to name but a few.

- We may learn a lot from our own history. I do not only discuss techniques that have proven their worth and are in wide use today. I also discuss developments that are by now considered dead-ends. Knowing *why* a certain technique is no longer used is often valuable. My discussion of cost estimation models in chapter 7 is a case in point.

- Everything changes. Requirements change while development is still under way. People enter and leave the project team. The functionality of a toolset changes before the systems developed with it are replaced. And so on. Change is a recurring theme in this book.

- Human and social aspects are central. Most chapters of this book carry titles that sound fairly technical. Within these same chapters, though, I regularly touch upon human and social aspects of the trade. For example, requirements elicitation is by no means a purely technical issue, and the design of a system is heavily influenced by the prior experiences, both positive and negative, of the designer.

People actively involved in software development and maintenance — programmers, analysts, project managers — and students of computer science and software engineering alike must be aware of the problems incurred by large-scale software development, and the solutions proposed.

I firmly believe that none of the solutions proposed is a silver bullet: CASE, object-oriented software development, software reuse, architectural design, formal

[1]The Stone Man version of the SWEBOK Guide lists the following knowledge areas: software requirements analysis, software design, software construction, software testing, software maintenance, software configuration management, software quality analysis, software engineering management, software engineering infrastructure, and software engineering process. The software construction area covers both coding and unit testing; of these, only unit testing is covered in this book.

For more information on the SWEBOK project, see `http://www.swebok.org`.

specifications, process models; they each contribute their mite. The fundamental problems will, however, remain. Software systems are extremely complex artifacts. Their successful realization requires experience and talent from their designers. If applied in a thoughtful, conscientious manner, the methods and techniques discussed in this book may help you to become a professional software engineer.

Learning about Software Engineering

Most chapters of this book can be read and studied independently. In a classroom setting, the instructor has a large degree of freedom in choosing topics from this book, and the order in which to treat them. It is recommended that a first course in software engineering at least deals with the topics discussed in chapters 1–3 and 9–14 (in this order). Additional material can be chosen at will from the other chapters or be used as material for a secondary course. Two obvious clusters of material for a secondary course are chapters 4–8 and 15–19; see also figure 0.1.

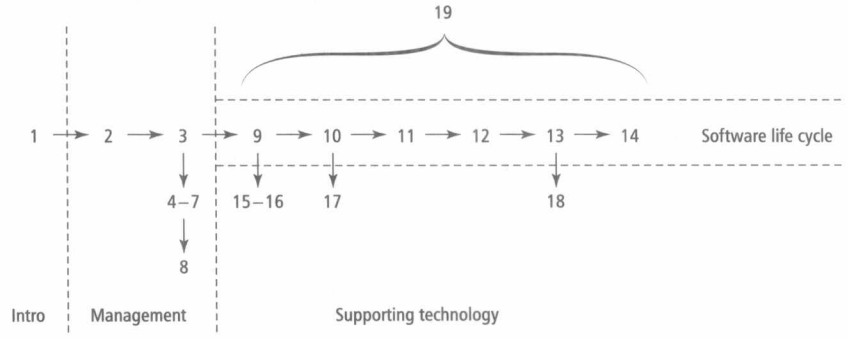


Figure 0.1 How the book is organized

A recurring problem in teaching software engineering is when and how to address project management issues. Computer science students often have difficulty in appreciating the importance of issues such as team organization and cost estimation. Software professionals know from the trenches that these non-technical issues are at least as important as the technical ones. Students of computer science or software engineering are more likely to understand the importance of management issues near the end of the course, possibly after they have been involved in some sort of practical work. However, a short treatment of the issues raised in chapters 2 and 3 should be given near the beginning of the course.

Much of what is said in this book sounds obvious. In fact it is. As one speaker at a software engineering education conference said: 'You cannot teach it, you can only preach it.' So this book is one long sermon on how to practice software development. Just as you cannot become a good hand at carpentry from reading a textbook on the subject, you cannot become a serious software engineer by merely reading and absorbing the material contained in this book. You need to practice it as well.

Doing practical work in a university setting is not easy. The many risks that real-life software development projects run cannot be realistically mimicked in a term project. Yet certain recurring problems in software development can be dealt with successfully. For example, small student teams may be asked to design, implement and test a nontrivial system, after which other teams get to maintain those systems.

Figure 0.2 The swimming equivalent of a correspondence course in software engineering (©*The Municipal Archives of Amsterdam. Reproduced with permission*)

Figure 0.2 depicts how schoolchildren in Amsterdam learned to swim around 1900. My father grew up in the countryside and learned it the hard way. His father simply tied a rope around his middle, threw him into the river that ran in front of the house, and shouted: 'Swim.' Nowadays, Dutch schoolchildren by and large all get their first swimming certificate before entering primary school. Their swimming lessons start off in a very gentle way, in a toddler pool, next to mamma and with lots of material to keep them floating. Gradually, the amount of floating material

is reduced and the pool gets deeper. They do not get scared and usually enjoy the swimming lesson.

A similar range of possibilities is possible in a software engineering course. The dry swimming equivalent is not to be recommended. Doing it the hard way by involving the student in a real project has its problems too. The student may, figuratively speaking, drown in the day-to-day practical intricacies of the project. Some sort of intermediate scheme involving 'real-life' aspects in a protected setting, or a sequence of educational experiences with an increasing amount of realism, seems most appropriate. Issues of software engineering education are addressed in the yearly *Conference on Software Engineering Education*. See also [Sof97c] for a state-of-the-art overview.

In addition to this practical work, the exercises at the end of each chapter provide further learning material. Exercises that are simply numbered ask relatively simple questions about the chapter just read. Exercises marked with a ♡ or ♠ require the reader to reflect seriously on major issues or study additional sources to deepen his or her understanding.[2] Exercises marked with a ♡ may require one hour to answer. Exercises marked ♠ may require more than a day. The simple exercises give but a superficial knowledge of the field. Deep knowledge of software engineering will only be developed if you cut your teeth on a number of the marked exercises. Answers to these exercises and further teaching material may be obtained from `http://www.wiley.co.uk/vanvliet`.

What's changed?

Software engineering is a rapidly evolving field. Preparing this second edition therefore necessitated changes in each and every chapter. But some chapters changed more than others. The major changes are as follows:

- I considerably extended the chapter on requirements engineering (9), especially in the area of requirements elicitation.
- I expanded sections of the chapter on software design into new chapters on software architecture (10) and object-oriented analysis and design (12).
- I replaced the chapter on software psychology by a chapter that focuses completely on user interface design (16).
- I dropped the chapter on programming languages.

[2]Rather than writing 'his or her' all the time, I will use male pronouns throughout this text for brevity.

Furthermore, the order of the chapters has been changed a bit to allow for a clustering into coherent parts.

Acknowledgements

The present text is really a fourth edition. The first two editions appeared in Dutch only. I have used this material many times in courses, both for university students and software professionals. These people have, either consciously or unconsciously, helped to shape the text as it stands. I have received many useful suggestions from Hans de Bruin, Frank Niessink, Jacco van Ossenbruggen, Bastiaan Schönhage, Victor van Swede and Martijn van Welie. Special thanks go to Gerrit van der Veer, who co-authored chapter 16. Michael Lindvall, Magnus Runesson, Kristian Sandahl and Anders Subotic from the University of Linköping in Sweden used part of the manuscript in a study-circle in scientific editing. Their remarks have been very helpful, and sometimes made me blush. Finally, I have received very useful feedback from the following reviewers: G. Edmunds (University of Southampton), Ralph F. Grove (Indiana University of Pennsylvania), Richard L. Upchurch (University of Massachusetts Dartmouth), Laurie Williams (University of Utah), Benjamin Pierce (University of Pennsylvania) and Mario Winter (FernUniversität Hagen, Germany).

Shena Deuchars of Mitcham Editorial Services did a great job as copy-editor. Many people from John Wiley & Sons have contributed to this book. Dawn Booth handled all the production work. Sandra Heath designed the cover. Katrina Arens dealt with a host of chores. Special thanks go to Gaynor, second name 'Patience', Redvers-Mutton for her support and indefatigable optimism.

The drawings that go with the chapter headings were made by Tobias Baanders. They are inspired by the artwork of Jan Snoeck that adorns the Centre of Mathematics and Computer Science in Amsterdam. The litho on the front cover is called 'Waterfall' (M.C. Escher, 1961). It is appropriate in name and message alike.

Finally I thank Marjan, Jasper and Marieke for their patience and support. The schedule overrun of this project has been worse than that of many a software development project.

Hans van Vliet
Amsterdam, August 1999

Contents