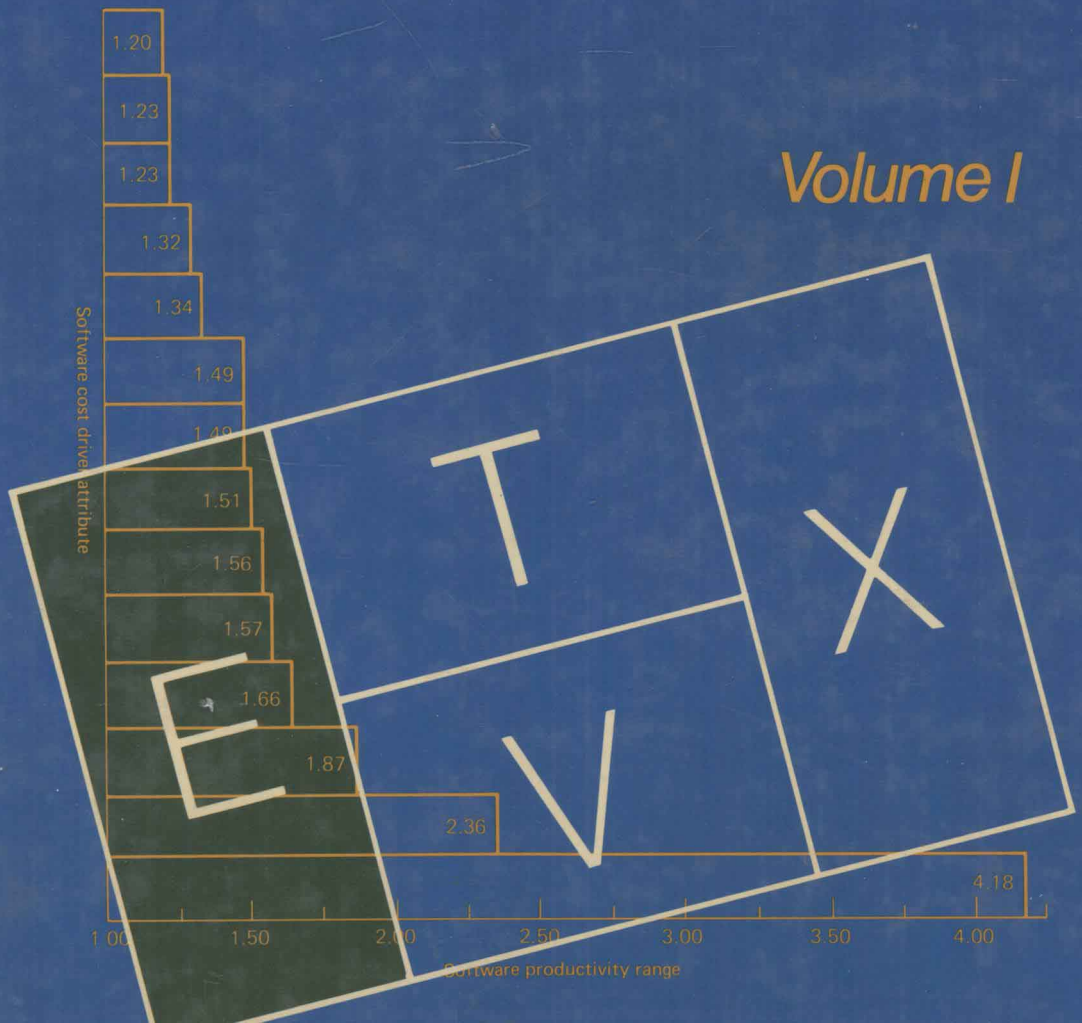


RONALD A. RADICE
RICHARD W. PHILLIPS

SOFTWARE ENGINEERING

AN INDUSTRIAL APPROACH

Volume I



Software Engineering: An Industrial Approach

Volume 1

Ronald A. Radice and Richard W. Phillips

Rensselaer Polytechnic Institute
and
International Business Machines Corporation



Prentice Hall, Englewood Cliffs, New Jersey 07632

Radice, Ronald A. (date)
Software engineering.

Bibliography: v. 1, p.
Includes index.
1. computer software—Development. I. Phillips,
Richard W. II. Title.
QA76.76.D47R327 1988 005.1 87-30037
ISBN 0-13-823220-2 (v. 1)

Editorial/production supervision: *Gloria L. Jordan*

Cover design: *Photo Plus Art*

Manufacturing buyers: *R. Washburn, C. Grant*

The publisher offers discounts on this book when ordered in bulk quantities.

For more information, write:

Special Sales/College Marketing

Prentice Hall

College Technical and Reference Division

Englewood Cliffs, N.J. 07632



© 1988 by Prentice-Hall, Inc.

A division of Simon & Schuster

Englewood Cliffs, New Jersey 07632

Ada is a registered trademark of the United States Government (Ada Joint
Program Office)

LOTUS is a registered trademark of Lotus Development Corporation

PSL/PSA is a registered trademark of ISDOS, Inc.

SADT is a registered trademark of SofTech, Inc.

SREM is a registered trademark of TRW Corporation

VisiCalc is a registered trademark of VisiCorp

Reproduced by Prentice-Hall from camera-ready copy prepared by the authors
using the IBM Publishing Systems BookMaster [®].

All rights reserved. No part of this book may be
reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-823220-2

Prentice-Hall International (UK) Limited, *London*

Prentice-Hall of Australia Pty. Limited, *Sydney*

Prentice-Hall Canada Inc., *Toronto*

Prentice-Hall Hispanoamericana, S.A., *Mexico*

Prentice-Hall of India Private Limited, *New Delhi*

Prentice-Hall of Japan, Inc., *Tokyo*

Simon & Schuster Asia Pte. Ltd., *Singapore*

Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

Dedicated to Claire and Lee

Foreword

Software Engineering is only a human generation old. At that age, civil engineering had yet to discover the right triangle. While more people are working on software engineering now than worked on civil engineering then, fundamental ideas and principles still take time to emerge. For example, it took the genius of Edsger Dijkstra several years to articulate the idea of structured programming after the advent of ALGOL 60.

Even before the discovery of structured programming, IBM undertook the largest, most complex software development project of its time, OS/360, under the leadership of Frederick Brooks. Although software systems have seldom had lifetimes of more than ten years (with changing hardware generations), OS/360 has been evolved and extended over twenty years, and no horizon is yet in sight. With requirements of hundreds of thousands of installations accepting ever new generations of hardware and operational conditions of distributed real time data processing, the methods and tools used in this evolution have been subjected to an uncompromising, relentless proving ground without precedent.

The demands of all this massive and continuing software development have led to a new understanding and crucial perspective that embeds methods and tools into a framework of intellectual and management control. This framework is called the IBM Programming Process Architecture, and it is the central and unifying theme in this book. As the authors state:

The Programming Process Architecture is the highest representation of the software process. While it includes the concept of a software engineering environment, it contains much more in a broader framework.

In short, it is an industrial strength process that has been developed and used in large scale practice with consistent results and control.

Two principles learned from large scale software development are brought forward in this book: (1) rigorous definition and management of the process beyond the levels exerted by individual tools and techniques are essential, and (2) the evolving process, as architected and practiced, must be allowed to govern the requirements for new tools.

In accordance with the scope of software development supported by the IBM Programming Process Architecture, this book provides a broad coverage of software methodologies, from formal procedures to creative heuristics, with many references to original work in the field. Classroom-tested in university courses, this book represents a first in bringing the lessons of megascale software development from the industrial crucible to the university classroom.

Harlan D. Mills

Preface

Why Another Book on Software Engineering?

There are many excellent books already available, and we have used a number of them over the years in the course we have taught at Rensselaer Polytechnic Institute (RPI). While we were developing our two-semester graduate-level course sequence in Software Engineering, we realized that none of the existing books fully met our needs for the sequence we were teaching, as none specifically represented an industrial view. The industrial view we wanted to focus upon required a book that was: (1) based on industrial experience and examples and (2) emphasized the engineering of software. We therefore found ourselves developing volumes of course notes and handouts to document our lectures for our students. The course notes for the first semester have now completed their migration into this book. This volume represents the first in a proposed series which will address the complete life cycle of programming development. The first two volumes will cover the life cycle of software development, and it is intended that succeeding volumes will cover in more detail each of the stages of development. Emphasis in this volume is placed on the methodologies within each development stage, and for each stage we note our recommended approach. This volume directly maps to the first semester as we teach it at RPI.

Enough has been written over the last twenty years about the software crisis to cause one to wonder if it will ever be solved. Clearly, since it was first uttered in 1968 at the North American Treaty Organization (NATO) conference in Garmisch, Germany, the problem has changed somewhat. Indeed much has gotten better and much will continue to further improve in the coming years. The combined and increasing focus of industry and academia on this problem will bring the evolution of Software Engineering to its required maturity in the industry and engineering disciplines. We will explore some of the historical aspects of this crisis in Chapter One. However, we do not intend to belabor this crisis in this book, but will, by example, speak more optimistically about how it can best be addressed today and what the near future holds as opportunity.

Any technical book can soon become out of date, especially in a field that is evolving as quickly as Software Engineering. Nonetheless, we believe the basic underlying principles that we highlight are necessary to both short-term and long-term Software Engineering solutions. In fact, these basic principles are evident in many of the successful project solutions we have seen. Some of the specifics we discuss in the book are undoubtedly evolving even as we go to print, but we believe the basics will persist.

It is essential that computer science and software engineering students get an industrial view in their education. It is important to reiterate here that the aca-

ademic world must do more to introduce and address the realities and the problems of industry in the Computer Science/Software Engineering curricula. It is important at this time, given the maturity level of software engineering, that the industry and academia be more widely integrated. This can be accomplished by having academia draw people from industry to teach graduate and undergraduate programs, or by having universities assign members of their teaching staffs to industrial sabbaticals.

The alternatives within Software Engineering that we have chosen as best or preferred are naturally influenced by our experiences at the International Business Machines Corporation (IBM), but we have, in almost all cases, gone beyond our immediate experiences to confirm the effectiveness of our preferences or recommendations. Inside IBM, our focus is predominantly determined from personal knowledge in System/370 software product development. This history spans a combined 55 years in the industry. Our backgrounds cover a multitude of products from operating systems, to telecommunications, to languages and applications. It includes work across all stages of the programming development life cycle. Since 1972, we have been involved in defining, implementing, and changing Software Engineering within IBM. The views and opinions we hold in this book, although influenced by our work at IBM, are ours alone and not necessarily those of IBM.

Is This Book for You?

This book was written first with our students of Software Engineering at RPI in mind. As we developed the book, we realized that it applied as well to all software workers in the industry, whether they are students, new programmers, experienced programmers, or managers of programmers; and whether their focus is systems, embedded, or application programming. The book is appropriate to both academia and industry, and, therefore, we believe we have met our original objective when we were first asked by RPI to bring an industrial view of Software Engineering into the academic environment.

What Is This Book About?

The primary focus is *software process*: what it is, what is meant by it, why it is important, what its underlying principles are, and how it is managed. The result of a good process, in general, is a good product. There are exceptions on both sides of this relationship, but we believe that a good process is required to achieve good products repeatedly. There are other important aspects in the software environment, such as tools and methodologies, but we believe that the process should determine the tools and the methodologies and not vice versa. Throughout the development life cycle any particular process is only as good as the people who work with it and believe in it. Without good programmers and

managers behind it, no process by itself is sufficient to achieve the goals of better quality, higher productivity, and shorter development cycles.

In this volume we describe the project development stages through to the code stage. In each stage we emphasize the process paradigm we recommend in Chapter Two. Again in each of these stages we focus on an industrial approach.

A Quick Outline

Chapter 1 introduces the topic of Software Engineering: its purpose, problems, and evolution.

In Chapter 2 we present the dominant theme of the book, the software process. We discuss

1. Evolution and prevalent views of software process
2. Principles relevant to software process management
3. A recommended approach for a process definition and management

In Chapter 3 we explore how the software product should be planned. This product-planning activity is central to the first portion of the Requirements and Planning stage. It is a precursor to determining the feasibility of continuing the product's actual development.

In Chapter 4 we discuss the need for alternative methods and a recommended approach to Requirements Engineering.

Chapter 5 explores a number of ideas about human factors in software, including the idea that human factors should be treated as any other functional requirement for the product, but that specific process focus must be brought to bear in order to achieve it.

Chapter 6 is focused on the planning of the project, that is, what is the best way to plan, execute, and control the project process to deliver the product.

Chapter 7 introduces the design of software and relevant concepts necessary to engineer the design.

Chapter 8 discusses approaches for validating and verifying the completion of the various work items that are developed during the product cycle. We recommend preferred approaches.

Chapters 9 and 10 discuss relevant concepts, methodologies, and representations for completing software design at three discrete design levels: Product Level Design, Component Level Design, and Module Level Design.

Chapter 11 completes this book and addresses the Code stage of software development.

Subsequent stages of the software development process will be discussed in Volume 2 of this textbook series.

How to Use This Book

Our lectures follow the text, and are coupled with three other activities:

1. Course projects
2. Case studies
3. Additional readings

We follow a 14-week sequence and try to address the chapters in the following sequence:

Chapter 1: The State of Software Engineering — Week One

Chapter 2: The Process of Software Production — Week Two

Chapter 3: Planning the Product — Week Two

Chapter 4: Requirements Engineering — Weeks Three and Four

Chapter 5: Human Factors and Usability — Week Five

Chapter 6: Planning the Project — Weeks Five and Six

Chapter 7: Design of Software — Weeks Six and Seven

Chapter 8: Validation and Verification — Week Seven

Chapter 9: Product Level Design and Component Level Design —
Weeks Eight and Nine

Chapter 10: Module Level Design — Weeks Ten and Eleven

Chapter 11: Code — Week Twelve

The course projects are designed to support Software Engineering. Students work on projects that address requirements formalisms, project history repositories, project tracking, design language processing, code restructure engines, reverse engineering, code generation, test coverage, test case generation, and software metrics, among others. Projects are assigned to directly reinforce the software engineering principles we teach in our lectures.

The case studies are intended to simulate specific types of problems that the software engineer might encounter in industry, and to structure a level of project progress during the semester. While we are hesitant to do anything to subvert the software principles we teach, we do have to contend with the artificial environment of a 14-week course. The focus during this semester, therefore, is in creating a learning environment more than in requiring a fully functional project at the end of Semester One. Nonetheless, the project teams are required to demonstrate the capability of their “product” at a level of test completeness by the end of the semester. Usually the teams achieve completion at a Unit Test level, a test effort which immediately follows the completion of the code. In some cases testing has gone well beyond the Unit Test level. Basically we are simulating a product development cycle for the initial release of a product within a 14-week constrained period. During the second semester the projects are rotated, the teams are kept together as much as is practical, and the teams are asked to continue with

1. Completing Release 1
2. Using it throughout the rest of the semester
3. Maintaining their “product”
4. Developing the requirements for Release 2
5. Completing Release 2 through the System Tested level in the life cycle

Finally, additional readings are assigned to either stress major points we want the students to explore or to take advantage of current findings in the software engineering literature.

A student using this book can either do so directly in a course that uses the text or read it as stand-alone text. While it is anticipated that the text be used in a Software Engineering course, we wrote it so that it will also stand by itself.

Acknowledgments

As with all books of this nature, there are more people who deserve recognition than we can adequately acknowledge in our words here. First, we must thank our secretaries, Gloria Murphy and Anne Wendt, who helped us in so many ways with typing, editing, copying, making the graphics, and the numerous last-minute requests for changes. Fortunately, they had a well-defined process and good text processing tools to keep us under control and to help us make this text. We thank Ron Wendt for showing us the powers and intricacies of Publishing Systems BookMaster, the product we used to produce this book in camera-ready form. Next, we have to thank the many past students at RPI who were in our

classes and worked with our notes as they evolved towards this book. We also thank Herb Freeman, who initially contacted us for RPI. We want to thank IBM for giving us first the experience and then the support while we worked on this book. Finally we thank all of the reviewers who helped make this a better book than it otherwise would have been.

R. A. Radice and R. W. Phillips
Poughkeepsie, New York

Reviewers

Joan Carl
Gerhard Chroust
Robert Goldberg
Jack T. Harding
Gene F. Hoffnagle
Ev Merritt
Harlan D. Mills
Robert P. Mueller

Almerin C. O'Hara, Jr.
Leonard Orzech
W. C. Peterson
A. M. Pietrasanta
Bernie Rachmales
Claire L. Radice
Hans A. Schmid
Carol A. Schneier

Contents

Foreword xv

Preface xvii

Part 1. The Basis of This Book 1

Chapter 1. The State of Software Engineering 2

- 1.1 Introduction 2
- 1.2 Beginnings of Software Engineering 3
- 1.3 Software Engineering Goals 5
- 1.4 Why Are Software Engineering Goals Important? 7
 - 1.4.1 User Satisfaction 7
 - 1.4.2 Software Management 8
 - 1.4.3 Increased Function 9
- 1.5 Effective Process Management 9
- 1.6 Meeting User Needs 11
 - 1.6.1 Requirements Engineering 11
 - 1.6.2 Human Factors 12
- 1.7 Defect Detection and Prevention Related to Software Reliability 12
 - 1.7.1 Defect Detection and Prevention 13
 - 1.7.2 Design Methodologies 14
- 1.8 Capital Investment for Software 14
 - 1.8.1 The Old-code Problem Prevention 15
 - 1.8.2 Reuse 15
- 1.9 Types of Software 18
- 1.10 Outlook for the 1990s 19
 - 1.10.1 Knowledge Engineering 19
 - 1.10.2 Artificial Intelligence and Knowledge-Based Systems 20
 - 1.10.3 Programmerless Programming 21
 - 1.10.4 Automatic Programming 21
- 1.11 Technology Transfer 22
- 1.12 The Software Engineer's Task 24
- 1.13 Summary 24
- 1.14 References 25

Chapter 2. The Process of Software Production 28

- 2.1 Introduction 28
- 2.2 An Evolution of Process in Software 29
 - 2.2.1 Cut and Run or Code and Go 29
 - 2.2.2 Design—Code—Test—Maintenance 30
 - 2.2.3 The Staged or Phased Process (the Product Life Cycle) 30

2.2.4	Waterfall Model	31
2.2.5	Incremental Development	33
2.3	Recommended Process Approach	33
2.3.1	Problems in Previous Process Approaches	35
2.3.2	Process Management Principles	35
2.3.3	Entry—Task—Validation—Exit:	36
2.4	Process Stages	42
2.4.1	Stage Definitions	44
2.5	Stage Attributes	46
2.6	Viewpoints across the Programming Process Architecture	48
2.6.1	An Example of Viewpoints Within the FVT Stage	49
2.6.2	Streamlining	51
2.7	Summary	52
2.8	References	52

Part 2. The Software Product 55

Chapter 3. Planning the Product 56

3.1	Introduction	56
3.1.1	Requirements and Planning Viewpoints	56
3.2	Key Concepts of This Chapter	57
3.2.1	Relationship to Requirements Engineering Stage	57
3.3	Entry Criteria	58
3.4	Creating the Business Proposal	60
3.4.1	High-Level and Strategic Requirements Analysis	60
3.4.2	Business Opportunity	62
3.4.3	Competitive Analysis	62
3.4.4	Description of Proposed Product	63
3.4.5	Cost and Price Objectives	64
3.4.6	Initial Sizings	64
3.4.7	Projected Profit And Loss	65
3.4.8	Risk Factors	65
3.4.9	Work Plan	66
3.5	Validation	66
3.6	Exit Criteria	66
3.7	The Decision	67
3.8	Summary	67
3.9	References	68

Chapter 4. Requirements Engineering 69

4.1	Introduction	69
4.2	Key Concepts of This Chapter	70
4.2.1	Requirements Engineering Domain	70
4.2.2	What Is a Requirement?	71

4.2.3	Sources of Requirements	72
4.3	Some Problems with Requirements Engineering	72
4.3.1	Typical Problems from a Product Perspective	73
4.3.2	Typical Problems from a Process Perspective	74
4.3.3	Some Typical Requirements Inputs	76
4.4	Entry Criteria	81
4.5	Requirements Engineering Tasks	82
4.5.1	The Requirements and PLD Specification	82
4.5.2	A Basic Requirements Engineering Approach	83
4.5.3	Planning and Design Methodology (PDM)	84
4.5.4	RPLD Model	90
4.6	Use of E-R Languages for Requirements Engineering	96
4.6.1	PSL/PSA Tutorial	97
4.6.2	SREM	104
4.6.3	SADT	106
4.6.4	An Example E-R Language Application of RPLD Model	108
4.7	Validation	125
4.7.1	Key Validation Checkpoints	125
4.7.2	Characteristics of a Good Requirements Specification	126
4.7.3	The Four Cs	129
4.7.4	Verification and Validation	129
4.8	Exit Criteria	131
4.9	Conclusions	132
4.9.1	Effects of Using Rigorous Requirements Process	132
4.9.2	Achievable Process Goals Today	133
4.9.3	Future Requirements Engineering Goals and Trends	134
4.10	References	136

Chapter 5. Human Factors and Usability 139

5.1	Human Factors and Usability: What Do We Mean?	139
5.1.1	What Products Are We Talking About?	140
5.1.2	Are All Users the Same?	140
5.1.3	The Objectives of Usability	142
5.1.4	New Versus Existing Products	144
5.1.5	Cause of User Error	145
5.2	How to Approach Usability	147
5.2.1	Managing for Usability	147
5.2.2	Designing with Usability in Mind	150
5.2.3	Measuring Usability	152
5.3	How to Evaluate Usability	153
5.3.1	User Involvement	154
5.3.2	Usability Prototyping	155
5.3.3	Usability Laboratories	155
5.3.4	Testing	156
5.3.5	Readability Indices	157

5.4	User Interfaces	161
5.4.1	Commands	161
5.4.2	Messages	162
5.4.3	PF Keys	162
5.4.4	Screen Formats	164
5.4.5	Computer Assisted Instruction (CAI)	164
5.4.6	Response Time	165
5.4.7	Help Facilities	168
5.4.8	Summary	168
5.5	References	169

Part 3. Software Engineering Methods 171

Chapter 6. Planning the Project 172

6.1	Introduction	172
6.2	Key Concepts of Project Planning	172
6.3	Planning Viewpoints	173
6.4	Some Typical Planning Problems	174
6.4.1	When Size Estimates Are Inaccurate	177
6.4.2	When the Project Plan is Inadequate	179
6.5	Entry Criteria for Project Planning	180
6.6	Planning Tasks and Methods	180
6.6.1	Resource Estimation	181
6.6.2	Work Breakdown Structure	183
6.6.3	Project Process Definition	185
6.7	Validation of a Project Plan	200
6.8	Exit Criteria	201
6.9	Summary	202
6.10	References	203

Chapter 7. Design of Software 205

7.1	The Role of Design in Software Engineering	205
7.2	What Is Design?	207
7.3	Designing: What We Do	209
7.4	Design Levels	212
7.4.1	Product Level Design (PLD)	215
7.4.2	Component Level Design (CLD)	216
7.4.3	Module Level Design	217
7.5	Models and Conceptual Views	218
7.5.1	Structure	219
7.5.2	Hierarchy	220
7.5.3	Networking	222
7.5.4	Function	222
7.5.5	Finite State Machine (FSM)	223

7.6	Design Methodologies	224
7.6.1	Functional Decomposition	226
7.6.2	Data Structure	227
7.6.3	Data Flow	227
7.6.4	Prescriptive	227
7.6.5	Object Oriented	228
7.7	Design Representations	228
7.8	What Is Good Design?	229
7.8.1	Design Validation	230
7.9	Design Directions	231
7.9.1	Summary	232
7.10	References	232
Chapter 8.	Validation and Verification	234
8.1	Introduction	234
8.2	Good Programmers Make Errors	235
8.3	Alternatives for Delivering Defect-Free Product	236
8.3.1	Peer Reviews	236
8.3.2	Reviews and Walk-Throughs	238
8.3.3	Inspections	238
8.3.4	Prototyping	239
8.3.5	User Involvement	240
8.3.6	Proof of Correctness	240
8.3.7	Testing	240
8.3.8	Other Approaches	241
8.4	Recommended Approaches	241
8.4.1	Inspections	241
8.4.2	Proof of Correctness	244
8.5	What Is an Inspection in Software Development?	245
8.6	How Are Inspections Performed?	248
8.6.1	Inspection Steps	249
8.6.2	Who Participates in an Inspection?	254
8.6.3	When and Where Do Inspections Fit into the Programming Process?	256
8.7	Cost and Benefits	258
8.7.1	Cost	258
8.7.2	Benefits	259
8.8	Summary	261
8.9	References	262

Part 4. Design and Coding Stages 265

Chapter 9. Product Level Design and Component Level Design 266

9.1	From Requirements to Product Level Design (PLD)	266
-----	---	-----