国外经典教材 · 计算机科学与技术

Springer

软件开发的形式化工程方法
——结构化＋面向对象＋形式化

（日）Shaoying Liu 著

清华大学出版社

国外经典教材·计算机科学与技术

软件开发的形式化工程方法
——结构化+面向对象+形式化

（日）Shaoying Liu 著

清华大学出版社
北 京

内 容 简 介

本书首次开创了一个新技术，即形式化工程方法，把传统的形式化方法和软件工程有机结合起来。它提供了一个严密、系统、有效的软件开发方法，其实用性超过了目前所有形式化方法。这正好可以满足学术界、软件工程类学生对学习形式化工程方法和 SOFL 的迫切需求。

本书通俗易懂，实例丰富，可满足读者即学即用的需要。书中对软件开发中的形式化工程方法进行了介绍和讨论，内容涵盖 SE 2004 中关于“软件的形式化方法”的知识点，主要包括：有限状态机、Statechart、Petri 网、通信顺序进程、通信系统演算、一阶逻辑、程序正确性证明、时态逻辑、模型检验、Z、VDM 和 Larch 等。本书可作为计算机、软件工程等专业高年级本科生或研究生的教学用书，也可供相关领域的研究人员和工程技术人员参考。

图书在版编目(CIP)数据

软件开发的形式化工程方法——结构化+面向对象+形式化＝Formal Engineering for Industrial Software Development：英文/(日)刘少英著；—影印本. —北京：清华大学出版社，2008.8
(国外经典教材·计算机科学与技术)
ISBN 978-7-302-18317-4
Ⅰ.软… Ⅱ.刘… Ⅲ.软件开发－教材－英文 Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字(2008)第 116043 号

责任编辑：文开琪
装帧设计：杨玉兰
责任印制：何 芊
出版发行：清华大学出版社 **地 址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编**：100084
社 总 机：010-62770175 **邮 购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市春园印刷有限公司
经 销：全国新华书店
开 本：185×260 **印 张**：27 **字 数**：534 千字
版 次：2008 年 8 月第 1 版 **印 次**：2008 年 8 月第1次印刷
印 数：1～3500
定 价：45.00 元

前　言

本书以简洁明了的语言介绍了软件开发技术的最新成果——形式化工程方法，并在精确而简单的概念描述和必要的数学定义基础之上，以大量浅显易懂的案例系统地阐述了一个具体的系统建模和功能描述语言 SOFL(Structured Object-Oriented Formal Language)以及在此语言基础之上建立起来的并具有代表性的形式化工程方法，称为 SOFL 方法。

大家都知道，大型软件产品的开发是一个一般需要多数开发人员参加的复杂过程。它的主要环节有三个：理解、创造和确认。为了给用户提供满意而正确的软件产品，开发者首先必须正确和全面地理解用户的要求，包括功能要求、必要的数据资源、对功能与数据资源或系统行为的限制。在此基础之上，开发者必须通过系统的设计，构建软件系统的体系结构、数据结构和有效的算法，并以一定的编程语言实现其代码。这一系列的创造过程是不可能完全自动实现的，它必须依靠软件开发者来完成。然而，开发者是不能保证不出错的。实际上，大型软件系统的开发过程中一般都要出现大量的人为错误，经常造成软件产品的生产费用增大，不能按时交货，或不能保证质量。为此，完成的软件产品的质量必须得到严密的确认以后其产品才能送交用户使用。

保证软件质量的一个很重要的要素就是软件开发方法。开发方法是不是实用有效，一般取决于它是否具备三个要素：简单、可视化和精确。也就是说，该方法必须用起来简单，其表现形式要有效地发挥图形语言的可视化效果，同时所表达的内容必须精确以确保其意思能被准确理解。传统的软件开发方法，像结构化和面向对象的开发方法，基本上都用自然语言或语义不明的图形语言来描述用户的需求和系统设计文档。由于自然语言表达结构和所用术语的意思不清楚所造成的二义性，使得软件文档经常被误解或被随意解释以造成大量严重的设计或实现上的错误。这种局面也严重地影响了切实有效的支撑工具的开发和建立。

上世纪七十年代以来，以 Hoare 逻辑和 Dijkstra 的最弱前置谓词演算为基础发展而来的形式化开发方法为严密地开发软件系统提供了理论基础和技术，为解决上述问题迈出了重要的一步。这些技术包括形式化规范说明书写作语言，逐步求精的开发方法，以及程序正确性证明技术。但遗憾的是，这些形式化技术只重视规范说明书的精确性和软件开发的严密性，而未能在简单可行和有效的可视化方面取得可喜的进展。恰恰相反，大型软件系统的形式化规范说明书往往复杂难懂，修改花费时间，出错的可能性很大。加之，逐步求精的开发方法和正确性证明技术都对开发人员的抽象和逻辑演算能力要求很高，使得这种技术很难被一般企业的软件开发者所接受。如何才能使严密性很高的形式化开发方法融合到传统的软件工程过程和技术中以提高和促进它们的严密性、有效性以及可控制性从而最终达到提高软件生产效率和软件质量的目的就成为一个意义重大的研究课题。

形式化工程方法和 SOFL 方法就是在这种情况下从上世纪八十年代末逐渐形成并发展而来的。其目的就是建立起能把形式化开发方法有效地结合到传统的软件开发过程和方法

中的技术，使得形式化方法能够更加简单地被一般软件开发者使用，发挥其高严密性的特长以促进传统软件开发技术的有效性和系统性，并使软件开发过程能得到更加有效的工具支撑。为建立有效的系统模型和确切的需求规范说明书，SOFL 方法提供了具有简单、可视化和精确特点的形式化规范说明书语言和行之有效的建立形式化规范说明书的三步法。为有效地查出需求规范说明书中的错误，SOFL 提供了严密的复审(review)和说明书测试技术。为有效地利用结构化和面向对象设计方法的各自优势，SOFL 提供了把结构化和面向对象的设计方法有机结合的技术，使得结构化方法在需求分析和抽象设计方面发挥其优点以方便开发者和用户之间的交流和合作，而使面向对象方法在详细设计和编码方面发挥其特长以增强所开发系统的可维护性和重用性。它也提供了把结构化设计转化成面向对象的程序代码的方法和技术。为有效地发现程序中的错误，SOFL 提供了基于形式化规范说明书的严密检查(inspection)和严密测试技术。虽然这最后两种查错技术没有包括在本书之内，但它们已形成 SOFL 方法的不可分割的部分。总而言之，SOFL 是集结构化、面向对象和形式化方法于一身，并具有简单、可视化和精确的特点的形式化工程方法。它已被工业界和多数软件工程人员所使用，其效果正在引起更多软件工程师和研究者的兴趣和关注。

本书的内容反映了作者在为提高软件开发质量以及开创严密有效和实用可行的形式化工程方法这个新技术领域近 20 年来的研究成果，并为进一步彻底的解决软件危机这个一直在困扰软件工程的重大难题，提出和阐述了“智能软件工程环境”这个未来软件工程的发展方向。作为本书的作者，在 2008 年第 29 届奥运会在北京召开之际，能将此书与国内读者见面，我感到由衷的欣慰，也很感谢清华大学出版社的极大热情和努力，帮助我实现了多年来想把形式化工程方法 SOFL 介绍给国内软件工程界的大学生、研究生、研究者以及工程技术人员的夙愿。我衷心希望读者能通过学习本书的内容享受到形式化工程方法的新思想，掌握其应用的系统技术，并提高开发高质量软件产品的技巧和能力，为自己的学习、就业、研究以及工作创造新的生机。

Shaoying Liu(刘少英)
日本法政大学计算机科学系
2008 年 8 月

To my family

Foreword

In any serious engineering discipline, it would be unthinkable to construct a large system without a precise notion of what is to be built. Equally, any professional engineer must record not only his or her proposed solution to an engineering challenge, but also reasons why the solution is believed to be correct. Software engineering faces the challenge of creating very large systems and must therefore solve both of these challenges. Combined with established good practice such as inspections, *formal methods* can make a significant impact on software dependability.

The fact that descriptions and correctness arguments were required was obvious to pioneers of computing as early as von Neumann and Turing, who both wrote about ways of reasoning about programs. Since their early attempts, the need has been to find *tractable ways* of coping with systems of ever increasing size. The landmark contributions of Bob Floyd and Peter Naur culminated in Tony Hoare's wonderfully clear exposition of "axioms" for reasoning about programming constructs. This in turn led to development methods like VDM, Z, and B. Such methods work well for systems which are sequential and self-contained, but extensions were required to deal with other real world problems such as concurrency and "open" systems where obtaining specifications (and recognising that the requirements will evolve over the lifetime of the system) is as challenging as developing the "closed" components which result.

This book brings together ideas from VDM and from object-oriented thinking to propose an approach to the development of realistic software systems. "SOFL" builds on some of the most pervasive ideas to come from theoretical computing science and amalgamates them into an approach which the author has used on a variety of practical applications. Such books are to be wholeheartedly welcomed because they are written with an acute understanding of the issues for designers of useful software.

The success and pervasiveness of object-oriented methods suggest that it is unnecessary to say more about their marriage with formal methods since it might appear to be an obvious step. I should however like to add some arguments in favour of this specific combination. It is frequently argued that

today's computer applications are inherently complex. I think only part of this complexity is inevitable in today's systems. Of course, the code for an online airline seat reservation system of 2003 is bound to be larger than the code for a simple batch payroll system of the 1960s. But it is also clear that much of today's software is very poorly structured: its architecture is often opaque and users find it almost impossible to form a mental model of how it works. With a WYSIWYG word processor, this can result in frustration and expensive loss of productivity for professional users; for safety-critical applications, poorly understood systems present the real danger of an operator making life threatening mistakes. The ultimate contribution of formal methods will be to help clean up the architecture of systems, and the marriage with object-oriented ideas is important in this regard.

Another key contribution of object-oriented implementations is that they offer a way of controlling interference in concurrent computing. Interference is the key characteristic of concurrent programs (whether the parallel programs share states or interact only by communication primitives). Reasoning about interference can be delicate and complex; good engineers will reduce the areas where such complexity is required to a minimum. Object-oriented implementations put the control of interference where it belongs: that is, with the designer.

The combination of formalism and object-oriented design has the potential to yield clean and accurate implementations. The reader is encouraged to understand and use SOFL.

Cliff B. Jones
University of Newcastle upon Tyne

Preface

This book aims to give a systematic introduction to SOFL (*S*tructured *O*bject-Oriented *F*ormal *L*anguage) as one of the *Formal Engineering Methods* for industrial software development. Formal engineering methods are a further development of formal methods toward industrial application. They support the integration of formal methods into the software development process, the construction of formal specifications in a user-friendly manner, and rigorous but practical verification of software systems. SOFL achieves all of these features by integrating data flow diagrams, Petri nets, VDM, and the object-oriented approach in a coherent manner for specifications construction, and by integrating formal verification with fault tree analysis and testing for reviewing and testing specifications. It also provides a way to transform formal specifications into Java programs. SOFL has been taught for many years at universities, and has also been applied to systems modelling and design both in industry and academia.

Formal methods have made significant contributions to the establishment of theoretical foundations and rigorous approaches for software development over the last 30 years. They emphasize the use of mathematical notation in writing system specifications, both functional and non-functional, and the employment of formal proofs based on logical calculus for verifying designs and programs. However, despite a few exceptions, most formal methods have met challenges lobbying for acceptance by industrial users. A lack of appropriate education may be seen as one of the major reasons for this unfortunate situation, but, apart from this, a bigger problem is that formal methods have not successfully addressed many important engineering issues related to their application in industrial environments. For example, how can formal specifications, especially for large-scale systems, be written so that they can be easily read, understood, modified, verified, validated, and transformed into designs and programs? How can the use of formal, semi-formal, and informal methods be balanced in a coherent manner to achieve the best quality assurance under practical schedule and cost constraints? How can formal proof and testing, static analysis, and prototyping techniques be combined to achieve rigorous

and effective approaches to the verification and validation of formal specifications, designs, and programs? How can the refinement from unexecutable formal specifications into executable programs be effectively supported? How can the evolution of specifications at various levels be assisted and controlled consistently and efficiently? How can software development processes be formally managed so that they can be well predicated before they are carried out, and well controlled during their implementations? And how can effective software tools supporting the use of formal methods be built so that the productivity and reliability of systems can be enhanced?

Since the research to provide possible solutions to these questions addresses a different aspect of the problem; I call this area *Formal Engineering Methods.* In other words, formal methods emphasize the utilization of mathematical notation and calculus in software development, without considering the human factor (e.g., capability, skills, educational background) and other uncertainties (e.g., accuracy and completeness of requirements, changes in both specifications and programs, the scale and complexity of systems), whereas formal engineering methods advocate the incorporation of mathematical notation into the software engineering process to substantially improve the rigor, comprehensibility, and effectiveness of commonly used methods for the development of real systems in the industrial setting.

After introducing the general ideas of formal engineering methods, this book provides a tutorial on the recently developed formal engineering method SOFL. The material originally evolved from my research publications over last 15 years, from courses, and from seminars offered at universities and companies in Japan, UK, USA, and Australia. It is intended to be the basis for courses on formal engineering methods, but it also contains the latest new research results in the field. By reading through this book, the reader will find that SOFL has provided many useful ideas and techniques as solutions to many of the questions raised above. It not only makes formal methods accessible to engineers, but also makes the use of formal methods enjoyable and effective. In order to help readers study SOFL easily, I have tried to make the descriptions as precise and comprehensible as possible. I have also tried to avoid unnecessary formal semantics of SOFL constructs, to the extent that this does not affect our understanding them. Numerous examples are given throughout the book to help the explanation of the SOFL specification language and method, and many exercises are prepared for readers to improve their understanding of the material they have studied and to check their progress.

The objective of this book is to bring readers to the point where they can use SOFL to construct specifications by evolving informal specifications to semi-formal ones, and then to formal ones. It is also intended to help readers to master rigorous and practical techniques for verifying and validating specifications, to learn the process of developing software systems using SOFL, and to get new ideas for building intelligent software engineering environments.

Audience

This book is written for people who want to improve their knowledge and skills in developing complex software systems. Readers who are interested in formal methods, but frustrated by using them in practice, will benefit greatly from this book. Although I have made efforts to make the book as self-contained as possible, and have provided many exercises for individual study, the reader will need some experience in programming and basic knowledge of discrete mathematics to appreciate and digest some of the abstract material.

Using This Book

This book can be used at the second year undergraduate or above level as a computer science textbook for courses on *logic and formal specification*, *advanced software engineering*, and *software specification, verification, and validation*, respectively. According to my experience at Hosei University and other institutions, in the course on *logic and formal specification* that takes about 24 hours, the fundamental knowledge on first order logic and skills for writing comprehensible formal specifications for large-scale software systems can be introduced based on the contents of chapters 1 to 12.

The course on *advanced software engineering* usually takes 26 hours, incorporating rigorous software development techniques using a formal specification language, including skills for writing modular, hierarchical, and comprehensible formal specifications, evolving informal specifications to semi-formal and then to formal ones, transforming structured abstract design into an object-oriented detailed design, and transforming detailed design into object-oriented programs in Java. The contents of this course can contain chapters 1, 4 to 16, 19, and 20.

In the course on *software specification, verification, and validation*, which is suitable for graduate students and needs about 24 hours, the techniques for writing formal specifications and for their verification and validation can be introduced based on the contents of chapters 4 to 18.

The book can also be used as a reference book to support the study of other related courses or individual study of formal engineering methods for software development. To make the book easier to use, I have organized the materials into nine parts:

Introduction. Chapter 1 explains the motivation of formal engineering methods and describes what they are. After discussing the problems in software engineering and difficulties in using formal methods, I describe the general ideas and features of formal engineering methods and their relation with SOFL.

Logic. Chapters 2 and 3 introduce mathematical logic that is adopted by SOFL. Both propositional logic and predicate logic are explained, and their application to the writing of and reasoning about SOFL specifications are discussed.

Specification. Chapters 4 to 6 cover the most important components of SOFL specifications: *module*, *hierarchy of modules*, and *explicit specifications*.

We explain the techniques of combining graphical notation and formal textual notation in writing comprehensible but formal specifications with these components.

Data types. Chapters 7 to 12 describe all the built-in data types in SOFL, which include basic types, set types, sequence and string types, composite and product types, map types, and union types. Each type is introduced by explaining its constructors and operators, and their use in specifications.

Classes. Chapter 13 is concerned with the concept of class: a user-defined data type. We discuss the structure of classes by explaining their similarity with and differences from modules, and the way to use classes in module specifications.

Software process. Chapters 14 and 15 present a software development process using SOFL from informal specifications to programs, and in particular elaborate several techniques for constructing formal specifications in an evolutionary manner.

Case study. Chapter 16 describes a case study of specifying an ATM (Automated Teller Machine) using the SOFL specification language. This case study is designed to show the entire process of developing a detailed design specification from an informal user requirements specification, and gives the reader an opportunity to review and digest the contents studied before this chapter.

Verification and validation. Chapters 17 and 18 introduce two techniques for verification and validation of specifications: rigorous reviews and specification testing. We explain how formal proof and the practical techniques like reviews and testing are integrated to provide rigorous but practical methods for verification and validation of specifications.

Transformation and software tools. Chapter 19 explains the principle and technique for the transformation of design specifications into Java programs, including data transformation and functional transformation; the last chapter, 20, discusses the potential features of an intelligent software engineering environment supporting formal engineering methods, in particular SOFL, and its importance in enhancing the productivity and reliability of software products.

All readers are recommended to read Chapter 1, but those who are experienced in programming and have sufficient knowledge about mathematical logic can skip Chapters 2 and 3. Chapters 4 to 6 present the fundamental principles and techniques for constructing specifications, and therefore are suitable for all readers. Chapters 7 to 12, concerned with abstract data types, need attention from the beginners, but can be quickly browsed by those who are familiar with VDM (Vienna Development Method), with caution because of the differences in syntax. Chapters 13 to 20 contain specific materials on SOFL and are recommended for study by all readers.

Acknowledgements

The development of SOFL benefited from numerous discussions with many people during the period 1989 to 2003. The initial research on SOFL, started

in 1989 at the University of Manchester in the UK, was motivated by Cliff B. Jones's book titled "Systematic Software Development using VDM" (first edition), and benefited from the seminars and discussions he provided for the formal methods group while I was studying for my PhD in Manchester. I am grateful to John Latham for his constructive comments on the initial work on the integration of VDM and Data Flow Diagrams, which establishes the foundation for the development of SOFL. The initial integration work also benefited from Tom DeMarco's book titled "Structured Analysis and System Specification" and from my experience of working with John A. McDermid at the University of York. My sincere thanks also go to the people whose joint work with me has impacted on the development of both the SOFL language and the method presented in this book. Chris Ho-Stuart defined an operational semantics for SOFL, and provided many suggestions on the improvement of the SOFL language. Jeff Offutt developed an approach to testing programs based on SOFL specifications. Yong Sun worked out with his research student a prototype of a graphical user interface (GUI) for SOFL. Jin Song Dong provided a denotational semantics for SOFL using Object-Z. My former students Tetsuo Fukuzaki and Koji Miyamoto developed a prototype specification testing tool and a GUI for SOFL, respectively. I would also like to express my gratitude to all the research partners and my students who have completely or partially applied SOFL to develop their software systems, or combined SOFL with other methods for software development. I appreciate very much the feedback from my students after they read the draft of the book. Financial support from the Ministry of Education, Culture, Sports, Science and Technology of Japan through several research grants is gratefully acknowledged. Finally, my thanks go to three anonymous referees for their constructive comments and suggestions, and the editor Ralf Gerstner of Springer-Verlag for his encouragement and suggestions that helped me to improve the initial draft and for his painstaking efforts in the editing of the text.

Contents