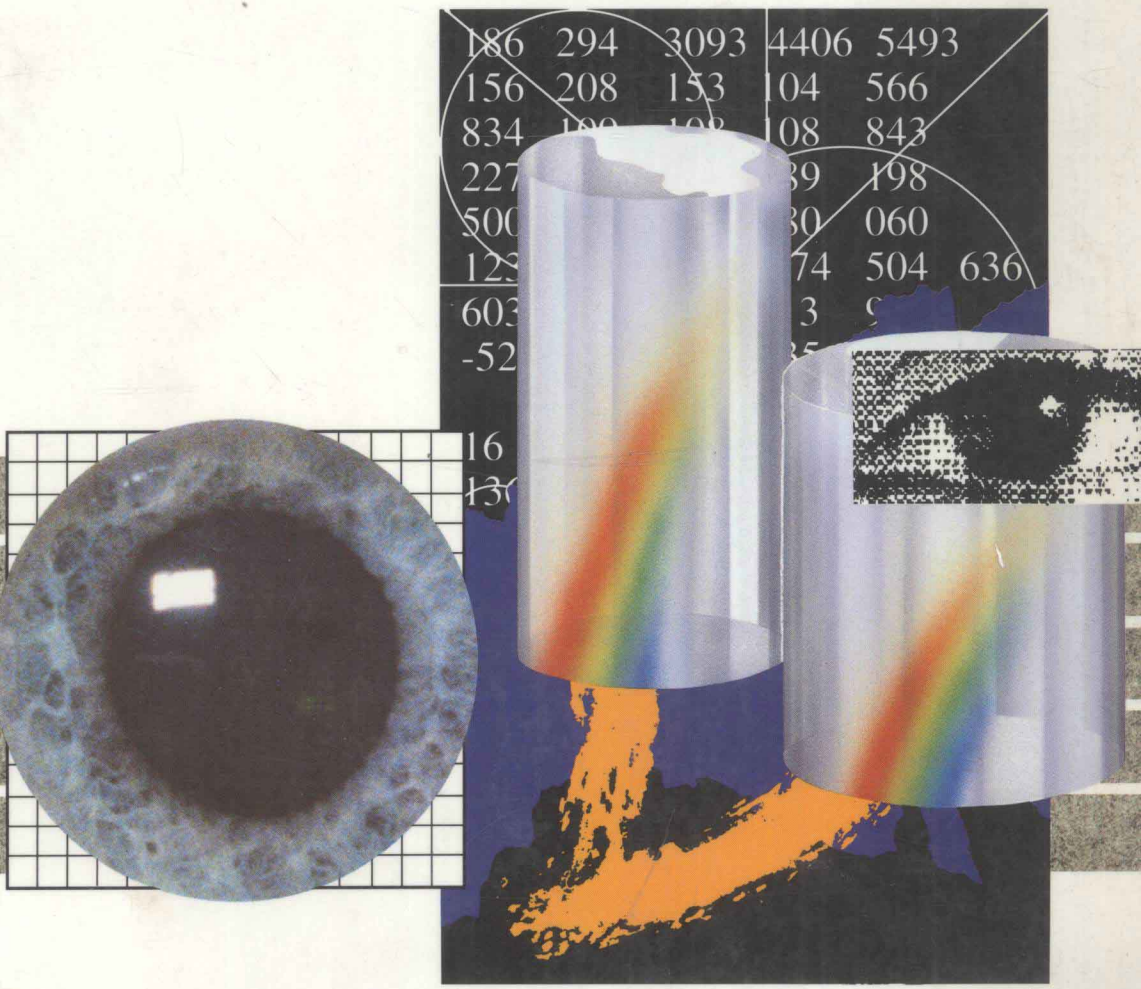
HOOD™

HIERARCHICAL OBJECT-ORIENTED DESIGN



Peter J.
ROBINSON

PRENTICE HALL
OBJECT-ORIENTED
SERIES

HIERARCHICAL OBJECT-ORIENTED DESIGN

PETER J. ROBINSON

PRENTICE HALL
NEW YORK LONDON TORONTO SYDNEY TOKYO SINGAPORE

First published 1992 by
Prentice Hall International (UK) Ltd
Campus 400, Maylands Avenue
Hemel Hempstead
Hertfordshire, HP2 7EZ
A division of
Simon & Schuster International Group

Typeset in 10 on 12pt Times Roman by
Keyboard Services, Luton

Printed in Great Britain by Redwood Books, Trowbridge, Wiltshire

---

Library of Congress Cataloguing-in-Publication Data

Robinson, Peter J. (Peter Jeremy)
HOOD : hierarchical object-oriented design / Peter J. Robinson.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-390816-X (pbk.)
1. Object-oriented programming (Computer science) I. Title.
QA76.64.R63 1992
005.1'2-dc20 92-16817
CIP

British Library Cataloguing in Publication Data

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

ISBN 0-13-390816-X

---

2 3 4 5 96 95 94 93

HIERARCHICAL OBJECT-ORIENTED DESIGN

PRENTICE HALL
OBJECT-ORIENTED
SERIES

B. MEYER
Eiffel: The Language

D. MANDRIOLI AND B. MEYER
Advances in Object-Oriented Software Engineering

B. MEYER
Eiffel: The Libraries

B. HENDERSON-SELLERS
A Book of Object-Oriented Knowledge

M. LORENZ
Object-Oriented Software Development: A Practical Guide

EDITOR'S PREFACE

The potential interest of object-oriented development for real-time and process-control applications has caught many people's attention. But there remains a certain reluctance to apply the object-oriented approach in large mission-critical applications. The contribution of HOOD here is essential, as few, if any, other methods in the field have stood the test of application to sizable real-time projects.

The HOOD method (the initials stand for hierarchical object-oriented design) was commissioned by the European Space Agency and developed by CISI Ingénierie. The original version was explicitly meant for software to be developed in Ada. Probably for that reason, it did not support the full range of object-oriented concepts, focusing instead on modularity, data abstraction, information hiding, hierarchically structured abstract machines, and of course support for concurrent execution and real-time applications. The importance of classes and inheritance was later recognised, however, and the method as described in Peter Robinson's book now supports these concepts. It may be used in conjunction with object-oriented languages, while retaining its compatibility with Ada.

Although there have been a number of articles on HOOD and a tutorial by Maurice Heitz at TOOLS conference, the method has not received so far the wide coverage that it deserves. The present book should help correct this situation. Peter Robinson has for a long time been involved in HOOD, and played a major part in its evolution. By acting as consultant to many projects using HOOD and training numerous people in the method, he has gained an in-depth mastery of the concepts and of their application.

Readers will benefit from this experience through the many examples and the case study of the Appendix. They will also gain precious insights about how the method should be applied in practice, learn about the possible pitfalls, and discover what it takes to apply the object-oriented approach, with all its potential benefits, to the tricky case of real-time systems.

PREFACE

This book is based on the definition of the syntax and semantics of hierarchical object-oriented design (HOOD) as presented in the *HOOD Reference Manual Issue 3.1.1*, published in February 1992, and on the HOOD method as presented in the *HOOD User Manual Issue 3.0*, which was published in December 1989 to complement the earlier *HOOD Reference Manual Issue 3.0*.

The changes introduced in *Issue 3.1.1* as developments from *Issue 3.0* are minor from a technical viewpoint, but the document has been changed more significantly, omitting, for example, a detailed section on Ada mapping. The *HOOD User Manual*, which defines the HOOD method, has not yet been updated to reflect these changes. This book may serve as a HOOD user manual until the HOOD Technical Group completes this work.

One of the reasons for writing this book is that many readers of *the HOOD Reference Manual Issue 3.0* did not see the *HOOD User Manual Issue 3.0*, and consequently had only a partial view of HOOD, which is not only a notation but, much more importantly, also a method. This has been a serious disadvantage to these readers, and has lead to criticisms about HOOD, some of which are due to having only partial information. The author hopes that this book will remedy this lack of information for the current version of HOOD.

Although HOOD was conceived with Ada as the programming language in mind, the *HOOD Reference Manual Issue 3.1.1* is written in more general terms, and HOOD is now starting to be used with C++ as the target language. Chapter 10 of this book outlines the variations needed when thinking in C++ terms. These comments also apply to other object-oriented languages.

The book includes several examples to illustrate the didactic parts of the text. There is also a complete sample design as an appendix. As examples, they are intended to be simple, so that they can be understood by readers from diverse backgrounds.

ACKNOWLEDGEMENTS

This book has evolved from the materials and experience of the HOOD method and HOOD toolset courses that I have developed and presented for SD-Scicon (UK) Training Limited, to whom I am grateful for permission to use this material. I have also previously published papers on HOOD in two books in the Unicom Applied Information Technology Series published by Chapman and Hall.

I would like to acknowledge the excellent contributions made by the original developers of HOOD in the first ESA contract, the other members of the HOOD Working Group in 1990 and the members of the HOOD Technical Group in 1991. These include Maurice Heitz of CISI Ingenierie, Jean-François Muller of Matra Espace, Klaus Grue of CRI A/S, and Joel Bacquet and Elena Grifoni of the European Space Agency. I would like also to thank Tony Elliott of IPSYS Software plc and Simon Handley of Birmingham Polytechnic for their many useful comments on the draft of this book.

HOOD is a trademark of the HOOD User Group (HUG). This fact must be stated in any publication referencing the name of HOOD in the context of the HOOD method as the basis of the publication. A revised version of the *HOOD Reference Manual* was published as *Issue 3.1.1* in February 1992, and is available from the HOOD User Group at the following address:

Finn Hass (HOOD User Group Chairman)
CRI A/S
Bregnerodvej 144
DK-3460 Birkerod
Denmark
Tel. +45(45)822100

The HOOD User Group may also be contacted through the author.

CONTENTS

Appendices

1

INTRODUCTION

1.1 HISTORY AND OBJECTIVES

Hierarchical object-oriented design (HOOD) is an Ada design method. That is to say, HOOD was specifically developed as an architectural design method for software to be written in Ada. The main reason for HOOD's success in being adopted for a wide range of Ada projects is probably that it was developed with the clear objective of supporting the architectural design phase of the software engineering life-cycle with a specific target programming language in mind. We look into the life-cycle as defined by the European Space Agency (ESA) shortly, but first let us look at the background in ESA prior to the development of HOOD.

1.1.1 ESA activities

I joined the European Space Agency in 1976 to work as a software engineer in the Spacelab project. ESA, like NASA, is an agency that acts as a customer on behalf of the governments that provide the funds, to procure space systems and to manage research and development programmes from companies in the aerospace industry throughout Europe, Canada and the United States. My role, therefore, was to act as a customer in the project which was procuring Spacelab from industry. In this case, the customer role was to supervise requirement definition, to review technical progress in the development of the software by attending reviews at each stage of the life-cycle, and finally to accept the software by participating in the full acceptance process. Given the complexity of an embedded system in which all the hardware and software was new, the requirements generally uncertain, and the contractors spread across Europe, this approach of continuous involvement was needed to ensure that quality was developed into the end-product. The final part,

acceptance, was very important because although each of the companies involved had its own quality assurance staff, there are always divided loyalties between concern for the software quality and for the company's commercial needs for delivery to be completed, accepted and paid for. Thus the customer needs to perform a long-stop quality assurance: an ultimate 'no' if the correct procedures are not followed, tests are not performed exactly as planned, if all problems are not fully cleared and if documentation is not up-to-date.

When the software for Spacelab was completed, I moved to the Technical Directorate to take up a quality assurance role. As well as the usual concerns about standards and procedures, I wanted to develop a more general software engineering approach, and promote better technology. To this end, I first became interested in Ada in 1982, before Ada was standardised. It seemed obvious that Ada would be a good language for ESA projects since Ada was being designed specifically for embedded software, which is a major part of ESA's software, and that a lot of effort was going into Ada and producing a language which combined new and old software engineering concepts. On the other hand, critics saw a language that was too heavy, that was 'not adapted for real time', for which no validated compilers were available for a long time, and for which no industrial quality compilers were available for even longer.

ESA Technical Directorate runs a technical research program each year, of which software is a small part. In 1985, Michel Guerin, then head of the Simulation Section of the Mathematics Division of Estec, decided to investigate the feasibility of using Ada by starting a small study in which an existing piece of operational software, in fact an attitude control system written in assembler, would be rewritten in Ada along with its accompanying simulation test environment, written in Fortran. ESA re-programmed the onboard software into Ada, using mainly the Program Design Language (PDL) provided as a design, and the industrial contractors reprogrammed the Fortran simulation into Ada. The resulting software was then run on a Data General system and produced the correct results, although rather more slowly than the corresponding Fortran simulation. The conclusion was that Ada would work, but would not be usable for operational software at that time, and that Ada would not take over from Fortran on existing projects or on existing application types for some considerable time.

In 1986, the Mathematics Division decided that if Ada were to be used, then ESA would need to know how to design Ada software, and so a study contract was awarded jointly to three companies (CISI Ingenierie (France), Matra Espace (France) and CRI A/S (Denmark)) to evaluate existing design methods for suitability, if necessary to produce a new method, and to develop a training course. The result is HOOD, which was completed in 1987.

When HOOD had been defined and the training course was being completed, it was clear that if European Space Industry were to accept HOOD as a suitable method, then a computer aided software engineering (CASE) toolset would be needed to enforce the method as well as facilitating its use. The HOOD toolset would aid the designer in the difficult task of producing and maintaining the graphics easily,

and would support the formal aspects, including checking of HOOD rules. This would therefore meet the management need for consistency and productivity, and the quality assurance needs for quality and correctness.

In 1988, a contract was awarded to Software Sciences in the UK, with support from CISI Ingenierie, Matra Espace and CRI A/S, to develop a HOOD toolset according to the ESA Software Engineering Standards PSS-05-0. In 1988, the Columbus Space Station project adopted HOOD for the architectural design phase, and in 1989 Hermes Spaceplane project also selected HOOD. As a result of this interest, other companies developed toolsets with varying technologies, but all following the standard method.

During 1989, the HOOD Working Group took account of comments about HOOD from the first users, and the need for new features to update the *HOOD Reference Manual* to Issue 3.0, and to produce the *HOOD User Manual*. ESA was keen to avoid disturbing designers, contractors and tool vendors during the early design phase, and decided not to change HOOD again for at least three years. ESA, therefore, withdrew from the development of HOOD, passing control to the HOOD User Group, which had been initiated in the first contract for the definition of HOOD.

HOOD has since been adopted by several military projects in Europe, notably the European Fighter Aircraft, by a nuclear power monitoring project in Belgium, by French electricity projects, and a large communication network project. It is fair to say that, in Europe, HOOD is considered for most major Ada projects as an established method well supported by CASE tools.

1.1.2 ESA software engineering life-cycle

HOOD was initially developed to fit into the ESA software engineering life-cycle which consists of the following phases and may be presented in the classic V shape in Figure 1.1.

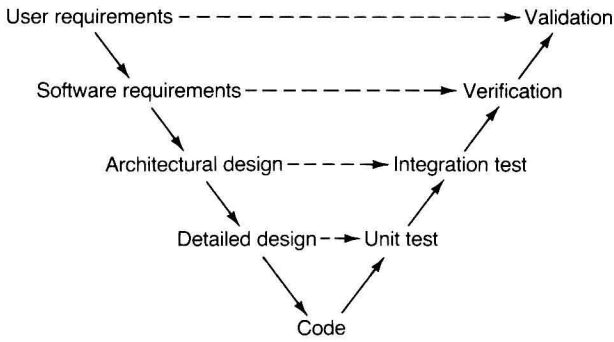


Figure 1.1 ESA software engineering life-cycle.

This life-cycle is primarily directed to an independent piece of software, that has a user to define the requirements in a User Requirements Document. For an embedded system, the user requirements are replaced by system requirements, subsystem requirements or equipment requirements, which define the requirements for a combined piece of hardware and software. In any case, the next step is to isolate the software requirements in a Software Requirements Document (SRD). Generally, this is produced by a contractor in a consortium, led by a prime contractor. After this SRD has been reviewed and approved by technical, quality assurance and managerial staff from the prime contractor and the agency, the design work can begin. The architectural design is intended to show the overall structure of the proposed design, whereas the detailed design is required to provide a sufficiently detailed description of the design to allow coding to follow. Both the Architectural Design Document and the Detailed Design Document are reviewed and revised to check conformance with the documented requirements. The software is then coded, and is first tested in units using the detailed design as the requirements for the test cases. The purpose of the integration test is to show that the completed software is consistent with the architectural design, with particular emphasis on testing interfaces between code components, and on external interfaces. Verification is carried out to demonstrate that the integrated software completely satisfies all the requirements documented in the SRD, and validation is performed to demonstrate that the software meets the documented user's requirements and that the software will operate successfully.

In an ideal world, the SRD is the starting point for HOOD. In practice, the requirements are not usually complete and unchanging at this stage. Often the hardware requirements are unclear because the hardware itself is also under development in a similar but parallel life-cycle. Thus the theoretical approach has to be read and understood to be carried out in a less than ideal environment. This means that changes should be expected, that top-down does not have to be applied rigidly, and that it is the principles of the method that are important. There is, therefore, still a role for the human being as software designer. In Chapter 10, we look at the interface to requirements, mainly to consider how requirements could better be defined to support the design phase.

One of the major objectives of the architectural design is to provide a clear identification of the components of the design and their interfaces. Such architecures are most clearly expressed using good diagrams, especially data flow and control flow diagrams. ESA experience was that contractors were generally unwilling to commit themselves to an architectural design until the detailed design was complete, and probably preferred to have completed coding. There was a hint of a suspicion that really coding was coming before design. Accordingly, good diagrams were to be an important part of HOOD.

HOOD does not stop there, though. In fact, a good HOOD toolset takes the designer right through the Detailed Design phase into coding and testing, by providing a text-based format for each object which can be refined step by step from a bare description of the functionality and operational interfaces, to a pseudocode

definition, to full code in the desired target language, i.e. Ada. Since most HOOD designers do the programming as well, this is eminently sensible. There is therefore no real break in the implementation from architecture to code. Currently, the break in the life-cycle is between requirements and design: between statement of the problem and development of the solution. Again, Chapter 10 looks into ways of reducing this break by providing an object life-cycle.

As a consequence, one may say that HOOD is not just an isolated method, but is a large part of the software development process. This is apparent when HOOD is extended to include requirement references for each object, thus allowing a requirement/object cross-reference table to be developed and maintained. Another approach, which is used in the Columbus Software Development Environment (SDE) for example, is to provide a relational database, which includes requirements and traces to HOOD objects. The SDE, which is an example of an integrated programming support environment (IPSE), is also used to provide configuration management, not only of HOOD objects and designs, but also of all the other documentation and code; this is the reason why HOOD does not provide any specific configuration management features.

1.1.3 Ada design method development

The purpose of the contract let by ESA in 1986/7 was to develop an Ada-oriented design method, supported by a training course, with the following additional constraints:

1. The new method had to be acceptable to a wide range of European companies, some with existing methods, and some without. It could not therefore easily be an existing national method unless this could be shown to be excellent. The experience of real-time programming languages, where Coral was developed and used in the UK, Pearl in Germany and LTR (Langage Temps Réele) in France, but each language was not accepted by other countries, was to be avoided.
2. The method should be suitable for developing software for large systems, so that they can be developed by multiple contractors in different sites and then integrated to form a complete system.
3. The method should be adaptable for software systems to run on multiple computers – these two points lead to an emphasis on interface definition and on software integration.
4. The method should follow Structured Analysis and Design Technique (SADT) and English definition of requirements since SADT was the method being considered at that stage for requirements and was being used by the French Space Agency CNES. In fact, the SADT aspect is not specifically addressed by HOOD, although the *User Manual* looks into this point.
5. The method should lead onto Program Design Language (PDL) in the Detailed Design phase, and to Ada for coding: this is seen in the formal structure of the