UNIVERSITY OF OSLO
Department of Informatics

ROOP –
Robust Object-Oriented Process
Cand. Scient. Thesis

Bjørn Henrik Pedersen

16th February 2004
Before practicing Zen, mountains were mountains and rivers were rivers.

While practicing Zen, mountains are no longer mountains and rivers are no longer rivers.

After realization, mountains are mountains and rivers are rivers again.

– Diamond Sutra
Acknowledgements

This thesis is part of the Candidatus Scientiarum degree at the Department of Informatics, University of Oslo. The work of writing this thesis has been in progress since September 2002, and it is with mixed emotions that I hand over these pages to the printing office.

First of all I would like to thank my supervisor Dr. Ing. Gerhard Skagestein for helping me understand the field of object-oriented modeling in a new and exciting way. He has given me a reason for studying how objects can be understood as partly computerized and partly belonging to the real world. I am very grateful for the feedback he has given me in the process of writing this thesis. Thanks for not giving up on me when I have presented theories that are wrong. The countless hours that we have spent on discussing object-oriented modeling has really made the work of writing this thesis fun.

I would also like to thank Annita Fjuk and Håvard Hegna for giving me the opportunity to test some of the assumptions that this thesis is based on. Annita Fjuk and Håvard Hegna are both involved in the project COOL (Comprehensive Object-Oriented Learning) [Pag].

Further, I would like to thank the following people at the Hedmark University College, Rena; Dag Nylund, Bjørn-Ivar Norseth, Frode Hellerud, Martin Ø. Fevik, Marius Filtvedt, Morgan Branes, Steinar Silkeløkken and Stig R. Nygården. These are the people that helped me to test and carry out the experiment based upon some of the early results indicated in this thesis.

A final thanks goes to family and friends for supporting me all these years. A special thanks to Petter Raaberg, Øyvind Forsbak and Allan Simonsen for spending their valuable time reading through this thesis. It feels good to say that I finally did it!

Oslo, 16th February 2004
– Bjørn Henrik Pedersen
Abstract

This thesis is about how to start an Object-Oriented (OO) system development process. The research problem may be translated like this: "Does it make sense to postpone the drawing of the system boundary in an object-oriented system development process?" In the process of working out a solution to the problem described, we studied how the Unified Process (UP) recommends to start an OO system development process. UP is a well-known development process, which starts by identifying functional requirements by means of use cases. This may lead to setting up a fixed system boundary very early in the development process.

In this thesis, we suggest an OO framework for how to avoid setting up a fixed system boundary early in the development process. The focus should be on how objects collaborate, regardless of what may be perceived as real world tasks and computerized tasks. Some objects may even be split into one part belonging to the real world, and another part belonging to the computerized system.

By postponing the distribution of knowledge and responsibility between real world tasks and the computerized system, we are able to design OO models on a higher abstraction level. The objects inside our system should not be regarded as part of an OO model of the "real world", but as computerized enhancements to objects in the real world. Only after we have decided upon how knowledge and responsibility should be distributed between the two parts, we know where to draw the system boundary.

The OO framework presented in this thesis may be applied to any OO development process, where we want to delay the drawing of the system boundary.
## Contents

1 Introduction ................................................. 1
   1.1 Background and Motivation ......................... 1
   1.2 ROOP Basics ....................................... 2
   1.3 Defining the Problem Area .......................... 4
       1.3.1 Approach to the Problem ..................... 4
   1.4 Research Approach .................................. 5
       1.4.1 Literature Study ............................ 5
       1.4.2 Interactive Engagement .................... 5
       1.4.3 Observation ................................ 6
       1.4.4 Study of Collected Data / Written Material from the
            Students ........................................ 6
       1.4.5 Unstructured Interviews / Feedback Discussions discouraged
       1.4.6 Personal Reflection ........................... 6
   1.5 Possibilities and Limitations ..................... 7
   1.6 The Structure of this Thesis .................... 7

2 An Introduction to the ROOP Methodology ............ 9
   2.1 Moving Towards a Holistic Object-Oriented Approach .... 9
   2.2 The Basic Idea – Introducing the Coffee Machine ......... 13
       2.2.1 Object Selection ........................... 15
       2.2.2 Selecting the Core Objects ................ 16
       2.2.3 Collaboration .............................. 17
       2.2.4 Split the Objects and Decide upon the System Boundary 19
       2.2.5 Use cases and Activity Diagrams .......... 20
       2.2.6 Finalizing ROOP Activities in how to Start an OO
            System Development Process ................ 24

3 NextGen POS System : The Unified Solution ............ 27
   3.1 The NextGen POS System ............................ 27
   3.2 About the Solution Presented in the Book ............ 28
   3.3 Inception ......................................... 29
       3.3.1 Use Cases .................................. 30
CONTENTS

A  A ROOP Research Paper
   Based on the Rena Experiment 30-31 Oct, 2003 101

B  Glossary 127

Bibliography 140
List of Figures

1.1 Split Object in ROOP ........................................ 3
2.1 ROOP World View ........................................ 11
2.2 Main Functionality ......................................... 14
2.3 Use Case Diagram ........................................... 14
2.4 Collaboration Diagram with Classes for Generating the Computerized Part of the Objects. ............... 18
2.5 The Object Model Before Deciding upon the System Boundary ........................................ 19
2.6 Object Model of the Coffee Machine After Deciding upon the System Boundary. ...................... 19
2.7 A Collaboration Diagram of the :Customer and the :CoffeeMachine with Corresponding Real World Parts and Computerized Parts of the Objects. .......................... 20
2.8 Use Case Diagram ........................................... 21
2.9 ROOP Activity Diagram ..................................... 23
2.10 Prototype of the UI with Connected Objects ................................................................. 25
3.1 The Unified Process Life Cycle ............................... 29
3.2 Use Case Diagram ........................................... 31
3.3 System Sequence Diagram ................................... 32
3.4 Partial Domain Model ....................................... 33
3.5 Interaction Diagram and Design Classes ................. 36
3.6 Connecting the UI and Domain Layer ...................... 37
3.7 Partial Implementation Model ............................... 38
4.1 Use Case Diagram Interaction ................................ 46
4.2 Main Functionality for the NextGen POS System ......................................................... 46
4.3 ROOP Objects in the Robust Layer ......................... 49
4.4 The Person and the Loan .................................... 49
4.5 Encapsulated Smart Objects in ROOP ....................... 53
5.1 ROOP Objects with a Flexible System Boundary .......... 56
5.2 Collaboration Diagram within the Unified Process ................................................. 57
5.3 Design Classes within the Unified Process ................. 57
## LIST OF FIGURES

5.4 ROOP Collaboration Diagram with Complementary Design Classes .................................................. 58
5.5 Collaboration Diagram for enterItem() ................................................................. 62
5.6 Collaboration Diagram for getItem() ................................................................. 63
6.1 Real Part of :Sale Reads the Bar Code from the Real Part of :Item .................................................. 67
6.2 Computerized Part of :Sale Asks the Computerized Part of :Customer for Payment ............. 67
6.3 Split Object Model in ROOP ........................................................................... 67
6.4 Primary Actors and Goals at Different System Boundaries .................. 69
6.5 The Cashier as a Primary Stakeholder and as an Object Instance ....... 70
6.6 The Cashier Interacts with the Register in UP ....................................... 70
6.7 The Cashier Interacts with the Sale Object in ROOP ....................... 71
6.8 Use Case Diagram in ROOP ........................................................................ 73
6.9 ROOP Activity Diagram for the NextGen POS System .............. 76
6.10 Larman’s Design Class Diagram for the NextGen POS System ......... 79
6.11 ROOP’s Design Class Diagram for the NextGen POS System .......... 80
7.1 Context Diagram ................................................................................. 84
7.2 Unified Process World View .................................................................. 85
7.3 Enhanced Version of Ogden’s Triangle ................................................. 87
7.4 ROOP’s Holistic World View ................................................................. 88
7.5 Two Instances of :Aircraft .................................................................. 89
7.6 Interface Objects in an Open System ...................................................... 91
7.7 ROOP’s Enhanced World View ............................................................. 92
7.8 Object Knowledge in ROOP ................................................................. 93
A.1 Rena Foils (1-6) .............................................................................. 110
A.2 Rena Foils (7-12) ........................................................................ 111
A.3 Rena Foils (13-18) ........................................................................ 112
A.4 Rena Foils (19-24) ........................................................................ 113
A.5 Rena Foils (25-30) ........................................................................ 114
A.6 Rena Foils (31-36) ........................................................................ 115
A.7 Rena Foils (37-42) ........................................................................ 116
A.8 Rena Foils (43-48) ........................................................................ 117
A.9 Use Case Diagram ........................................................................... 118
A.10 Main Functionality .......................................................................... 118
A.11 ROOP Object Design ......................................................................... 119
A.12 Split Object ...................................................................................... 122
A.13 Collaboration ..................................................................................... 123
A.14 User Interface and Domain Layer ................................................... 124
A.15 Class Diagram .................................................................................. 124
# List of Tables

2.1 Use Case Description .................................. 22  
2.2 Activity Realization Diagram for Select Coffee Product .... 24  
2.3 Activity Realization Diagram for Serve Coffee .............. 24  
3.1 makePayment Contract ............................... 34  
3.2 Partial Information Expert Pattern ........................ 35  
4.1 Stakeholders Pattern .................................. 52  
4.2 No Data Movers Pattern ................................ 52  
4.3 Encapsulation Pattern .................................. 53  
5.1 CRC Card for (an Instance of) the Customer Object ....... 61  
5.2 CRC Card for (an Instance of) the Sale Object ............ 61  
5.3 CRC Card for (an Instance of) the Item Object ............ 62  
5.4 CRC Card for (an Instance of) the Product Object ....... 63  
6.1 Use Case Description of Enter Item ...................... 74  
6.2 Use Case Description of Make Payment .................... 74  
6.3 Activity Realization Diagram for Enter Item ............... 77  
6.4 Activity Realization Diagram for Make Line Item .......... 77  
6.5 Activity Realization Diagram for Make Payment .......... 77  
6.6 Activity Realization Diagram for End Sale ................. 77  
A.1 No Data Movers ....................................... 120  
A.2 Stakeholders ........................................... 121  
A.3 Encapsulation .......................................... 121  
A.4 CRC Card ............................................... 122
Chapter 1

Introduction

1.1 Background and Motivation

Object-Oriented (OO) modeling has not always fascinated me. I remember when I started studying computer science in 1998. At that time I really did not have any idea about what computer programming was, nor did I know what my friends meant when they were talking about objects. I had spent the previous two years studying psychology and economics, so changing to a completely different field was hard in the beginning. However, I appreciate the variety of knowledge that I have today. Computer science is not all about math and difficult algorithms, it is a comprehensive study of how people interact with the information technology that is available. It is the study of giving people the opportunity to involve themselves in the world we live in. In my own words, I would say it is the study of designing the future!

In 2001 my supervisor, Gerhard Skagestein, wrote an article on the subject "Are Use Cases Necessarily the Best Start of an OO System Development Process?" [Ska01].

About a year later, Skagestein introduced the idea of studying a different approach for how to start an OO system development process, as a possible thesis for the Cand. Scient degree at the University of Oslo. This was the start of a continuous discussion on how to attack the problem area. We discussed several different aspects in the real world, which in a best possible way could highlight the key features of his idea.

After a while, we ended up with studying the NextGen point-of-sale (POS) system explained in Craig Larman’s book "Applying UML and Patterns"

---

1A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor [IJR99b].
The solution provided in the book uses the Unified Process (UP)\textsuperscript{2} together with the Unified Modeling Language (UML)\textsuperscript{3}. The UP is a typical use case driven approach. Hence, it was a good starting point for working out an alternative OO system development process, and compare it against the UP. However, we were only interested in studying the field of starting a development process.

During the process of writing this thesis, we came up with the name "Robust Object-Oriented Process" (ROOP). Hence, the ROOP methodology aims at building a framework for how to start an OO system development process.

\section*{1.2 ROOP Basics}

This thesis brings forward some new OO concepts. To help you out in understanding the most essential terms, we will briefly explain them in this section. The idea is to create a mental picture of what we are trying to describe throughout this thesis.

\subsection*{ROOP’s Holistic World View}

The UP describes an open system. This means that stimulus messages are initiated from outside (the unknown) of the system boundary.

The ROOP methodology describes a closed (holistic) system. Hence, we must conduct ourselves to two boundaries. Firstly, we have the boundary that captures our universe of discourse (we choose to regard a part of the world that consists of both real world things, and computerized parts). Secondly, we have the boundary that may split the objects in two. One part belonging to reality, and another part belonging to the computerized system.

In UP the system boundary is usually chosen to be the software (and possibly hardware) system itself [Lar02]. Hence, the term system boundary is used in ROOP analogous with the boundary that may split the objects in two. It is easier to map the meaning of the word to UP, by using the term this way. The reason for applying this, is built on our assumption that use cases (in UP) fixes the boundary of the computerized system too early in the process.

\textsuperscript{2}The Unified Process (UP) is an industry standard software engineering process from the authors of UML [IJR99b].

\textsuperscript{3}The Unified Modeling Language (UML) is a general purpose visual modeling language for systems [AN02]. UML provides a visual syntax that we can use to construct models.
ROOP Basics

**ROOP Objects may be split!**

Objects in ROOP may have a computerized part and a real world part (See figure 1.1).

![Figure 1.1: Split Object in ROOP](image)

Where to draw the system boundary (which may split the objects) will later be determined by organizational influences, technological influences, working conditions of the immediate users etc. However, we feel that it is important to have an idea of how objects can be perceived in ROOP, before we explain the methodology in detail.

The ROOP methodology is divided in three phases, briefly described below.

**Phase 1 - Finding Objects**

The main objective in this phase is to find objects that fits our universe of discourse. We use special designed patterns that help us select objects of interest.

**Phase 2 - Collaboration**

The main objective in this phase is to design how objects collaborate. However, we design collaboration diagrams regardless of real world and computerized behavior.
Phase 3 - Splitting the Objects

The main objective in this phase is to decide whether the objects are split. If we choose to draw the boundary through an object, then we use different techniques to specify what happens when messages are sent from the real part to the computerized part (and vice versa).

1.3 Defining the Problem Area

Modern software development methodologies continue to grow and new standards are set to meet the customer’s high expectations. The introduction of new technology seems to control how future software will be in function. These technological features affect how the software industry thinks in the way new software is developed.

Reuse of existing software would be more cost effective than building a new solution from scratch, but only if the model of the software is built upon a theory that later changes can be integrated into the existing model. The developer’s high level goals are often related to building robust and flexible software solutions. Integrating new technology in already existing software solution is a challenge. If the integration causes major reengineering, then reuse of the existing system will be too expensive. Therefore, Object-Oriented (OO) system development processes need to focus on building system models that is able to meet new technology standards, with only minor reengineering work.

1.3.1 Approach to the Problem

Some of the ideas presented in the ROOP methodology are taken from the article "Are Use Cases Necessarily the Best Start of an OO System Development Process?" [Ska01], by Gerhard Skagestein.

This thesis is a continuation of Gerhard Skagestein’s ideas of how to start an OO system development process⁴. The approach to the problem is defined as:

"Does it make sense to postpone the drawing of the system boundary in an object-oriented system development process?"

Viewed in the light of the approach to the problem, the following assumptions will be studied:

⁴The ideas presented in this thesis are in addition influenced by several hours of discussions between Gerhard Skagestein and Bjørn Henrik Pedersen [Ska].
Research Approach

H1 The ROOP methodology will lead to a model on a relatively high abstraction level, where there is no distinction between the real part and the computerized part of the object.

H2 The objects may be split into a real world part and a computerized part in a rational way by using the ROOP methodology.

H3 If the specification of the problem to solve is changed, there is a high likelihood that the model will not have to be changed. Hence, reuse of models is easier with the use of the ROOP methodology.

1.4 Research Approach

In this section we will present the research approaches (methods) used to explore the problem statement of this thesis. Some of these research approaches are also described in appendix A on page 101.

1.4.1 Literature Study

There is an abundant number of articles and books in the field of OO system development. The challenge has been to select relevant material that substantiates the ideas described in this thesis.

However, there are two references which stand out as important for the structure and presentation of this thesis. These are Gerhard Skagestein’s article [Ska01] and Craig Larman’s book [Lar02].

The goal of the literature studies has been to find adequate material that supported or emphasized the results of this thesis.

1.4.2 Interactive Engagement

Interactive Engagement (IE) methods are designed to promote conceptual understanding through interactive engagement of students in heads-on (always) and hands-on (usually) activities which yield immediate feedback through discussion with peers and/or instructors [Hak02].

We wanted to investigate the ROOP methodology in practice, and introduced ROOP to students at the Hedmark University College, Rena. These students took part in an OO design experiment, where they applied ROOP on a given case. (See appendix A on page 101 for details)

The goal of using IE was to give the students immediate feedback of what they were doing / thinking (note that we used IE only in parts of the experiment).
1.4.3 Observation

Observation involves watching and recording behaviors within a clearly defined area. The researcher plays the role of passive observer and is, therefore, outside the action/s being observed and recorded [SA].

We used observation in parts of the Rena experiment. The goal of using observation as a research approach, was to observe how the students were able to use the ROOP methodology on a given case.

1.4.4 Study of Collected Data / Written Material from the Students

Written material from the students (Rena experiment) was analyzed and studied in the following weeks, after the experiment ended.

The goal of studying the written material was to capture details of how they applied the ROOP methodology on the case.

1.4.5 Unstructured Interviews / Feedback Discussions

Throughout the process of writing this thesis, and probably the most important research approach in terms of this thesis, has been the countless discussions with my supervisor Gerhard Skagestein.

The goal of these unstructured interviews / feedback discussions has been to get an overview of how objects can be split between the real world and the computerized system. After all, it was Gerhard’s idea from the beginning that objects may be split in two.

In addition, feedback discussions with the students in the Rena experiment gave us valuable information about weaknesses in the ROOP methodology.

1.4.6 Personal Reflection

Personal reflection requires the researcher to reflect upon, and evaluate, their own experiences, memories, values and opinions in relation to a specific issue or topic [SA].

Personal reflections have been important for the evolution of the ROOP methodology. The goal of using personal reflections as a research approach, has been to evaluate (and reflect upon) the usability of introducing a new methodology for how to start an object-oriented development process.
The Structure of this Thesis

1.5 Possibilities and Limitations

The framework presented in this thesis may have practical use for system developers that wish to start an OO system development process differently. The ROOP methodology is systematically structured, and may be applied to other OO processes than UP.

However, this thesis is primarily based upon studying the NextGen POS system\(^5\). Further research would tell us more about the effect of starting an OO development process with ROOP.

1.6 The Structure of this Thesis

The remaining work of this thesis has the following structure:

Chapter 2 introduces the ROOP methodology.

Chapter 3 introduces the NextGen POS System. This is the case that will be used throughout this thesis. Further, this chapter gives a brief summary of Craig Larman’s Unified Solution of the problem to solve.

Chapter 4 describes how objects are selected within ROOP.

Chapter 5 describes how objects collaborates within ROOP.

Chapter 6 describes how objects are split within ROOP. Further, this chapter explains where to draw the system boundary.

Chapter 7 discusses different model views that affect modern object-oriented methodologies.

Chapter 8 summarizes the results of this thesis and discusses directions for further research.

Appendix A includes a summary of the Rena research experiment.

Appendix B includes a glossary list of the most common expressions used in this thesis.

\(^5\)This thesis also concludes with the results found in the Rena experiment. See appendix A on page 101 for details.
Chapter 2

An Introduction to the ROOP Methodology

The Robust Object-Oriented Process (ROOP) is the name of the methodology that this thesis describes. The main focus in ROOP is to find and present a framework for how to start an Object-Oriented system development process.

This chapter starts by discussing how ROOP is influenced by other methodologies. Further, this chapter introduces the ROOP methodology by using a simple coffee machine example. The main idea is to get to know some of the ROOP techniques, and see how ROOP can be applied on a given case.

2.1 Moving Towards a Holistic Object-Oriented Approach

People have different understanding about system development processes. By that you also have different views on how people understand the basics of a system. Some people emphasize the environment [CS99] [Guy76], while others emphasize a more mechanical view of how computer systems are seen [For61] [MC00].

Object-Oriented (OO) system development processes see the world as an interaction between responsible objects. ROOP takes advantage of this paradigm by creating a holistic world view of people and automated systems, regardless of whom or what triggers a sequence of events inside the system. In other words, ROOP does not distinguish between people and automated systems. Hence, we see the system as being holistic.

In most systems it is difficult to change the distribution of knowledge and responsibility between people (real world) and computer systems. The reason for such a shift can either be organizational changes or the introduction
of future technology. In ROOP’s holistic world view, future technology and organizational changes are taken into account, thereby making the distribution of knowledge and responsibility easier than with traditional methods.

ROOP’s holistic world view is analogous with a mechanical world view. Systems that are built upon a structure of interacting feedback loops are understood as being part of a mechanical world view. A mechanical world view sees the system within the concept of a closed boundary. The system of interest states that the modes of behavior under study are created by interaction of the system components within the boundary. The boundary implies that no influences from outside of the boundary are necessary for generating the particular behavior being investigated [For67]. The result is a never ending interaction of feedback loops inside our boundary of interest.

Critics to a mechanical world view say that there are other important factors that may interact with the system, which a mechanical world view does not intercept. For example systems that interact with objects outside of our boundary are not covered in mechanical world view methodologies. In ROOP’s mechanical world we describe the world to fit our purpose. Everything outside of our area of interest is left out of consideration.

The ROOP methodology is influenced by several different systems thinking approaches. Among these approaches we find Actor Network Theory (ANT), System Dynamics, Data Flow Diagram techniques (DFD), Evolutionary development, Business Process Reengineering (BPR) and Soft Systems. Figure 2.1 on the facing page illustrates the influence these systems thinking approaches have on ROOP’s holistic world view. We will now briefly discuss these systems thinking approaches and draw some parallels to the ROOP methodology.

**ANT:** An actor network consists of and links together both technical and non-technical elements. Not only the car’s motor capacity, but also your driving training, influence your driving. Hence, ANT talks about the heterogeneous nature of actor networks [HM98].

ANT was initiated by Bruno Latour and Michael Callon [Lat87], [Lat91]. ANT was born out of ongoing efforts within the field called social studies of science and technology [HM98]. An important aspect of ANT in terms of ROOP is that ANT does not distinguish between actors outside the system and objects inside the system. Latour [Lat91] expresses this as if we abandon the divide between material infrastructure on the one hand and social superstructure on the other, a much larger dose of relativism is possible. Society and technology are not two ontologically distinct entities but more
Moving Towards a Holistic Object-Oriented Approach

Figure 2.1: ROOP World View

like phases of the same essential action.

**Systems Dynamics:** Systems dynamics combines theory, methods, and philosophy for analyzing the behavior of systems [For98].

System dynamics uses concepts from the field of feedback control to organize information into a computer simulation model. A computer acts out the roles of people in the real system. The resulting simulation reveals behavioral implications of the system represented by the model [For98]. ROOP acknowledges partly this way of thinking. How ROOP objects are capable of shifting between computerized responsibility and real world responsibility implies this. System dynamics influence ROOP directly by automatically assuming that computers are part of social systems. The challenge is to translate important policies into a computer model.

**Data Flow Diagrams:** The Data Flow Diagram is a modeling tool that allows us to picture a system as a network of functional processes, connected to one another by "pipelines" and "holding tanks" of data [You89].

Data Flow Diagrams (DFD) have been an essential part of several system development methodologies. Data flow diagrams are used to illustrate essential models of systems, which among other factors imply finding true requirements. A true requirement is a feature that the system must have no matter what technology is used to implement the system [MP84]. ROOP has brought forward some of these key aspects by saying that we must find
the systems true objects regardless of technology in use.

**Evolutionary Development:** Evolutionary development is an iterative and incremental approach to software development [Amb02].

Like most OO development processes, ROOP is based on building systems over time in an incremental and iterative development cycle. Instead of building and then delivering your system in a single “big bang” release, you instead deliver it incrementally over time [Amb02].

**Business Process Reengineering:** Business Process Reengineering (BPR) can be understood as the fundamental rethinking and radical redesign of business processes to achieve dramatic improvement in critical, contemporary measures of performance, such as cost, quality, service and speed [HC93].

Michael Hammer introduced the term BPR in 1990 [Ham90]. Since then, many companies have done some fundamental rethinking and organizational changes based on Hammer’s ideas. Like BPR, ROOP triggers organizational changes by acknowledging objects as being part of the real world and the computerized world. To achieve the desired goals, companies must break away from conventional wisdom and the constraints of organizational boundaries and should be broad and cross functional in scope [Ham90]. ROOP takes into account Hammer’s ideas of reengineering principles by designing models which are less affected by new rules in how to do business. Communication is a key word in how to do business in both BPR and ROOP. Hammer introduced the concepts of organizational changes and reengineering. ROOP takes advantage of these concepts by designing robust objects which do not necessarily change when reengineering. The only things that change are how communication is described inside the objects.

**Soft Systems Methodology** Soft Systems Methodology (SSM) [Che81] is a methodology used to support and to structure thinking about, and intervention in, complex organizational problems [Guy76].

Like SSM, ROOP acknowledges the complexity involved in any organization, especially with unstable artifacts in the real world. Both ROOP and SSM share the same aspect for understanding and dealing with this diversity of views and interests in an organization.
The Basic Idea – Introducing the Coffee Machine

2.2 The Basic Idea – Introducing the Coffee Machine

ROOP introduces a different way on how to start an object-oriented development process. ROOP is not a full methodology, in contrast with the Unified Process (UP), which describes everything from inception to transition in several iterations. The focus has been to improve the first couple of iterations on how to start an object-oriented development process. Compared with UP, ROOP ends where the second iteration in the UP elaboration phase starts.

In this section we will use a simple coffee machine as an example to illustrate some of the key concepts in the ROOP methodology. The example is meant to give the reader a quick overview of the methodology. A detailed introduction of the ROOP methodology will be given in later chapters.

The coffee machine shall have a simple User Interface (UI), and provide free coffee to all users.

The basic idea in ROOP is that you should avoid setting up a system boundary too early in the system development process. This means for example that the physical boundary of the coffee machine should not automatically be assumed to be the system boundary. Therefore, ROOP suggests that functional requirements should be described without stating what triggers an event. Hence, actors as we find them in UML use case diagrams are of no interest to us (at least initially).

What determines the system boundary is the distribution of responsibilities between the computerized system and its environment. Drawing the system boundary right through an object means that the object will be split into an object part belonging to the real world and an object part belonging to the computerized system.

By postponing the decision on where we draw the system boundary, we believe that we get a more flexible and robust object architecture. If later reuse of the model causes a shift in the system boundary, the model will hopefully still be useful. Conventional models will probably have to be changed, simply because there will be too many interface objects and data mover objects that must be redesigned.

Figure 2.2 on the next page shows the main functionality without any form of communication with the outside world. From this starting point we have a lot of possibilities designing the objects that carry out the tasks for fulfilling the systems main functionality, that is to “Satisfy the Customer’s Coffee
The Basic Idea – Introducing the Coffee Machine

Thirst”.

Figure 2.2: Main Functionality

This is in contrast with how the Unified Process (UP) starts a system development process. The UP (including RUP) starts the development process by describing use cases, where each use case is linked with one or several actors outside of the system boundary. The use cases are usually described both in terms of diagrams, and in terms of written use cases. However, the outcome of the use cases expresses the same system interaction. Thereby, setting up a system boundary is exactly what you do when you follow well-known system development methods like RUP (Rational Unified Process) that starts with Use case diagrams, because these diagrams build on an assumption of where the system boundary should go.

Figure 2.3 shows how functional requirements are described by means of use cases in RUP (Rational Unified Process) and similar methods. In addition to the use case Process Coffee, the diagram shows a direct communication with an Actor outside the system boundary. If the system in addition is interpreted as being the computerized system, then there is a chance that the distribution of work between humans and computers is determined too early in the system development process.

Figure 2.3: Use Case Diagram
2.2.1 Object Selection

Finding usable objects can be a difficult task. Since the system boundary has not been determined, ROOP opens up for a more comprehensive solution. According to ROOP, the focus should initially be on what objects you need and how they should collaborate, independent of whether they are outside or inside the computerized system, or shared between the computerized system and its environment. User interface and control objects are left out of consideration. This way of thinking yields high abstraction level models that are easy to grasp and robust against technological and organizational changes.

Before we decide upon the objects, we should make a candidate list. ROOP suggests, like several other OO methodologies, that object candidates are found using a brainstorming technique. All ideas are welcome in the initial phase of the process.

In a brainstorming session, the group\(^1\), guided by a set of principles, combats the censorship and frees team members to propose any alternative from the most logical to the most absurd [BS99]. The starting point for our brainstorming activity is the main functionality of the system. A list of possible object candidates is shown below:

- Customer
- CoffeePowderContainer
- ControlPanel
- WaterTank
- CoffeeMachine
- Product
- Ingredient
- ServicePersonell
- Cupholder

ROOP introduces three patterns to help us in the selection process. Each pattern is specially designed to find one or more ROOP objects. A ROOP object is by definition an object that correspond with the description in one or more of the following patterns:

---
\(^1\)Brainstorming is usually conducted in small groups of 3-5 people.
Stakeholders:

The main function of this pattern is to select the stakeholders of the system. ROOP defines a stakeholder as being someone or something with vested interest in the behavior of the system under discussion. For example, if we want to serve coffee, then the coffee drinker is the systems primary target.

No Data Movers:

The main function of this pattern is to leave out any objects that are inside our area of interest and do nothing more than moving data between the objects. Each object in the ROOP model has more responsibility built in, compared to objects which you would find by using the Rational Unified Process.

Typical “Clerk” objects that are operating the system are not part of the ROOP model. This is because later changes to the model could make the clerk object superfluous, thereby leaving the responsibilities of the object outside of the system. Somehow these responsibilities must be transferred to other objects to fulfill the systems main functionality. One of the major challenges in ROOP is to add responsibilities to objects, that are not affected by changes to the system. Thereby, ROOP is making the model more robust to later reuse.

Encapsulation:

The main function of this pattern is to find objects with stable definitions that encapsulates the information. These objects are often the result of two or three object candidates put together. The idea behind encapsulation is to make object reuse a more practicable task. We design to minimize the impact of change. The encapsulation pattern is closely related to Larman’s Protected Variation pattern [Lar01]².

Chapter 4.4.1 on page 51 gives a more detailed description of these patterns.

2.2.2 Selecting the Core Objects

The object list contains the objects that we want to continue working with.³ The following objects fulfill one or more of the ROOP patterns:

²Protected Variation is essentially equivalent to the Open Closed Principle [Mey88].
³The term "core" is adopted from "the CRC Card Book" [BS99]. (See glossary B on page 127 for a description of core objects.)
The Basic Idea – Introducing the Coffee Machine

- Customer
- Product
- Ingredient
- CoffeeMachine

2.2.3 Collaboration

ROOP uses Class, Responsibility and Collaboration (CRC) cards [KB89] to assign knowledge and responsibility, and decide upon how the objects collaborate. The CRC card technique is used together with an object think approach [CN93] to help us think of objects as being more like us. A detailed explanation of the CRC Card approach and the object think approach will be presented in chapter 5.1.1 on page 59.

For each object we should list its knowledge and responsibilities. A representative amount of knowledge and responsibility for the :Customer and :CoffeeMachine object is shown below. These two objects illustrate the basic idea of how objects in the ROOP methodology are designed.

**Customer:**
- Customer preferences
- Select product
- Order coffee

**CoffeeMachine:**
- Order service
- Serve product
- Recognize customer
- Process preferred choice

When giving the objects knowledge and responsibility we must think in a holistic perspective. By doing so, we add more knowledge and responsibility to the objects, which might not be implemented when we draw the boundary between the computerized part and the real part of the object. If we were about to design a traditional coffee machine model, then the customer object would only be seen as a real world part of the system. Customer preferences would for example not be included in that case. In ROOP, we call this preparing for the future. We feel that this is one of the advantages
The Basic Idea – Introducing the Coffee Machine

by using the methodology.

The collaboration diagram in figure 2.4\(^4\) shows some of the extended knowledge and responsibility that objects may have, using the ROOP methodology. How messages are sent between objects remains the same. The only difference lies in who is the initiate and receiving part of the objects. This could either be the computerized part of the object or the real part of the object.

\[\text{Boundary between the computerized part of the system and the real part of the system}\]

`Customer` \[\text{selectProduct()}\]

`CoffeeMachine`

\[\text{Classes}\]

\[\text{cashAvailable()}\]

\[\text{notifyCustomerToPay()}\]

\[\text{customerPreferences()}\]

\[\text{totalSale()}\]

\[\text{askForPayment()}\]

\[\text{processPreferredChoice()}\]

\[\text{acceptCustomer()}\]

Figure 2.4: Collaboration Diagram with Classes for Generating the Computerized Part of the Objects.

An object model of the “Satisfy Customer’s Coffee Thirst” functionality would look something like figure 2.5 on the next page. The four core objects are included in the model. The dotted line indicates a movable system boundary that can split the objects in two, leaving the objects outside the boundary or leaving the objects within the boundary.

\(^4\)Due to technical limitations, some of the figures in this thesis may slightly diverge from the UML [wKS00] standard.
2.2.4 Split the Objects and Decide upon the System Boundary

Splitting the objects and deciding upon the system boundary are determined by several different tasks. We have postponed this decision until now, so that we get full control of object knowledge and responsibility.

For a detailed description of the decision process see chapter 6.1 on page 65. For now, let us assume that we have decided that the Customer object is left outside of the system boundary. The remaining objects are split in a real world part and a computerized part. In this example we will look at different methods that may be used to express the cross-over boundary between the real part and the computerized part of the :CoffeeMachine object.

The object model will look like figure 2.6, after we have decided upon the system boundary.

---

5 A cross-over boundary is the boundary that splits an object in two. One part belonging to the real world, and one part belonging to the computerized system.
Firstly, the physical hardware in the machine, and secondly, the user interface buttons on the machine. The computerized part consists of the required software to fulfill the necessary computations to serve coffee to the customer.

The task at hand is to specify the communication between the real part of the object and the computerized part of the object. We will use a combination of activity diagrams and use cases to illustrate what happens in the boundary cross-over. The collaboration diagram in figure 2.7 illustrates the problem to solve.

Figure 2.7: A Collaboration Diagram of the :Customer and the :CoffeeMachine with Corresponding Real World Parts and Computerized Parts of the Objects.

For a more comprehensive discussion on how objects are split, see chapter 6.1 on page 65. In this example we will just illustrate how the communication between the real and computerized part can be designed, using use cases and activity diagrams.

### 2.2.5 Use cases and Activity Diagrams

An observant reader would probably ask; “What makes these use cases (See figure 2.8 on the facing page and table 2.1 on page 22) any different from any other use cases?” The answer is really nothing! However, ROOP does not define UI communication based on the use cases, which is far too often the case with the use of RUP. ROOP uses use cases to illustrate UI interaction after we have decided upon the boundary between the computerized part and the real world part of the objects. That is the difference!
The Basic Idea – Introducing the Coffee Machine

Figure 2.8: Use Case Diagram
### The Basic Idea – Introducing the Coffee Machine

#### Use case: Select coffee product

<table>
<thead>
<tr>
<th>ID: UC1</th>
</tr>
</thead>
</table>

**Actors:**
Customer

**Includes:**
- UC2: Make selected coffee
- UC3: Dispense an ingredient

**Preconditions:**
None

**Main success scenario:**
1. Customer selects coffee product
2. Coffee machine makes selected coffee
3. Customer picks up selected coffee

**Success end conditions:**
Coffee delivered

<table>
<thead>
<tr>
<th>Table 2.1: Use Case Description</th>
</tr>
</thead>
</table>

We are not saying that use cases in RUP are meant to work as a model for how objects and user interfaces are designed. We are just referring to the weaknesses involved by starting an OO development process with use cases. The theme is a known problem to developers, and it is discussed in many books and articles [wKS00] [AB03].

We have now expressed by means of use cases the UI communication between the computerized part and the real part of the coffee machine. The use case diagram shows that the (real world) customer sends the stimulus message to the use case “Select coffee product”.

When it comes to object behavior, we use an UML activity diagram to illustrate the flow of messages that are sent. Activity diagrams are capable of showing both real world processes and the execution of software routines, with support for both conditional and parallel behavior [wKS00]. Figure 2.9 on the facing page shows how events are sent between the objects.

Swimlanes, as we find them in UML activity diagrams, are used to identify activities within an object. In figure 2.9 on the next page, the swimlanes are the diagram part\(^6\) within each object. The activity diagram is used to help us to identify key events/activities that describe the cross-over boundary in the :CoffeeMachine object.

From figure 2.9 on the facing page, we see two activities in the :CoffeeMachine

---

\(^6\)Each diagram part reflects one specific object in ROOP.
The Basic Idea – Introducing the Coffee Machine

Figure 2.9: ROOP Activity Diagram
The Basic Idea – Introducing the Coffee Machine

object that describe communication between the real part and the computerized part of the object. These activities are “Select Coffee Product” and “Serve Coffee”. The "Dispense Ingredient" activity in the :Ingredient object is also subject to a split activity. In short, this activity identifies the physical aspect of the ingredients that are dispensed into the coffee cup. However, in this introduction to ROOP we will focus on the two split activities in the :CoffeeMachine object.

After we have identified these key activities, we make an activity realization diagram to identify real world and computerized parts within the objects.

Table 2.2 and 2.3 show the activity realization diagrams for the split activities in the :CoffeeMachine object.

<table>
<thead>
<tr>
<th>C1 :CoffeeMachine</th>
<th>Activity1: Select Coffee Product</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real world part</strong></td>
<td><strong>Computerized part</strong></td>
</tr>
<tr>
<td>User interface buttons.</td>
<td>Recognize which button the customer has pressed.</td>
</tr>
<tr>
<td>Hardware.</td>
<td>Process preferred choice.</td>
</tr>
<tr>
<td></td>
<td>Send preferred choice to :Product.</td>
</tr>
</tbody>
</table>

Table 2.2: Activity Realization Diagram for Select Coffee Product

<table>
<thead>
<tr>
<th>C1 :CoffeeMachine</th>
<th>Activity2: Serve Coffee</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real world part</strong></td>
<td><strong>Computerized part</strong></td>
</tr>
<tr>
<td>Tap and serve coffee.</td>
<td>Receive message from :Product.</td>
</tr>
<tr>
<td>Cup.</td>
<td>Signal coffee.</td>
</tr>
<tr>
<td>Hardware.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Activity Realization Diagram for Serve Coffee

2.2.6 Finalizing ROOP Activities in how to Start an OO System Development Process

In the coffee machine example, we have briefly touched upon some of the activities that make up the ROOP methodology for how to start an OO system development process.
The Basic Idea – Introducing the Coffee Machine

The remaining work consists of building up the Design Class Diagram (DCD) with attributes, operations and associations. Before we can draw the DCD we need to gather the information we have, and see how the flow of information is sent between the objects. More collaboration diagrams are needed to fulfill the design model. Much of the information that we need can be found in the previous work, but we need to put the information into a system.

A prototype of the UI put together with a collaboration diagram gives useful information on how messages are sent in the system. A prototype of the UI in the coffee machine is shown in figure 2.10.

![Diagram of UI with connected objects](image)

Figure 2.10: Prototype of the UI with Connected Objects

We will stop with our coffee machine design here, and refer to chapter 4 on page 41, chapter 5 on page 55 and chapter 6 on page 65 for a more detailed study of the ROOP methodology.
The Basic Idea – Introducing the Coffee Machine
Chapter 3

NextGen POS System : The Unified Solution

In this chapter we will introduce the NextGen POS System. This is the case description of the system that we are about to design with ROOP. However, since we want to compare the ROOP methodology against the Unified Process, we start by giving a brief summary of Craig Larman's [Lar02] Unified Process (UP) design solution. The ROOP methodology will be presented in the following chapters.

3.1 The NextGen POS System

The following case (3.1 on the next page) is taken from the book "Applying UML and Patterns" by Craig Larman [Lar02].
About the Solution Presented in the Book

The NextGen POS system case description:

The NextGen point of sale (POS) system is a computerized application used (in part) to record sales and handle payments. An application like this can typically be used in a retail store. Such a system includes hardware components like a computer and bar code scanner, and software to run the system.

The system interfaces to various service applications, such as a third party tax calculator and inventory control. Further, it is important that these systems are relatively fault-tolerant. This means that even if remote services are temporarily unavailable, they must still be capable of capturing sales and handling at least cash payments.

A POS system must support multiple and varied client side terminals and interfaces. These include everything from a Web browser terminal, a regular personal computer running a Java Swing graphical user interface, touch screen input to a wireless PDA.

Furthermore, this is a commercial system intended for different clients with disparate needs in terms of business rule processing. This can for example be a unique set of logic to execute when a new line item is added. Therefore, the POS system will need a mechanism to provide this flexibility and customization.

3.2 About the Solution Presented in the Book

To capture a whole process in just a few pages is very difficult. Therefore, what is presented in this chapter is only a part of the process presented in the book [Lar02]. More exactly, the main focus will be on the inception phase and the first iteration of the elaboration phase.

The UP repeats over a series of iterations making up the life cycle of a system. Figure 3.1 on the facing page illustrates the the UP life cycle. Each iteration consists of four phases. The first phase : Inception launches the Project. The goal is to make the business case to the extent necessary to justify launching the project [IJR99b]

A simplified use case diagram that contains the most critical use cases are typically presented in the inception phase. Maybe only 10 - 15 percent of the Use cases are written in detail. [Lar02]. This makes it easier to move on to the elaboration phase.

During the elaboration phase, most of the products use cases are specified in detail and the system architecture is designed. Use cases are an impor-
Inception

Figure 3.1: The Unified Process Life Cycle

tant part of the Unified process, as it is an Use case driven process. This means that the development process follows a flow. It proceeds through a series of workflows that derive from the use cases [IJR99b]. At the end of the elaboration phase you have models of the system representing the most important Use cases.

The construction phase includes building the product. This is where the architectural baseline grows to become the full fledge system.

Transition is the last phase of the UP. This is where the project moves into beta release. A small number of experienced users try the product and report defects and deficiencies. Developers then correct the reported problems and incorporate some of the suggested improvements into a general release for the larger user community.

All figures in this chapter are adopted from Craig Larman’s book "Applying UML and Patterns" [Lar02].

3.3 Inception

Larman describes the inception phase as follows:

Envision the product scope, vision and business case

−Larman 2002

This includes high level goals for the project, functional and non functional requirements and a risk management plan for the following iterations. Risk management covers both technical complexity and other factors like uncertainty about the usability.

High level goals are key words that describe how the system shall behave. For the NextGen project this means that the system must be fast, robust
and handle integrated sales processing.

### 3.3.1 Use Cases

Use cases describe primarily functional requirements, and they represent a central part of the inception phase.

Some people like to distinguish between system use cases and business use cases. The terms are not precise, but the general usage is that a system use case is an interaction with the software, whereas a business use case discusses how a business responds to a customer or an event [wKS00]. In the NextGen POS system, it could be said that Larman uses what is describes as system use cases.

Essential use cases is another term that is being used, where the essential refers to essential models that are intended to capture the essence of problems through technology-free, idealized, and abstract descriptions [RBT02]. However, in this thesis we will not distinguish between any of these three terms. All use cases are referred to as use cases only\(^1\). Mainly because this is the way Larman [Lar02] expresses them.

Larman makes it clear that use cases are text documents, not diagrams. The use case model (see figure 3.2 on the facing page) is meant to be a supplement to the written descriptions.

For the NextGen POS system, the idea is to give an overview of the main use cases that are involved in the system. Further, it is important to keep user interface (UI) interactions out of the picture. Larman wants to describe the use cases at a higher level. A deeper understanding of what actions that lies beneath each use case, belongs in the elaboration phase.

To capture and describe all use cases in the inception phase would take too much time. It would also move the focus from the big picture to specifications that should be considered in later iterations. Only about 10 or 20 percent of the use cases should be written in detail at this stage. The use cases that are described in detailed will be the main focus for further work in the following phases. The use case “Process Sale” is the starting point for the NextGen POS system.

\(^1\)See glossary in appendix B on page 127 for further descriptions of these three use case terms.
Figure 3.2: Use Case Diagram
3.4 Elaboration

Moving on to iteration one in the elaboration phase, Larman has completed the planning and decided to tackle a cash only success scenario of Process Sale.

3.4.1 System Sequence Diagrams

Figure 3.3 shows input and output events related to the main success scenario of the Process Sale use case. It indicates that the cashier generates makeNewSale, enterItem, endSale and makePayment system events. The encapsulated box indicates that more items may be added. In this case, the enterItem event is performed repeatedly until no more items are added.

When identifying system events, it is necessary to be clear on the choice of the system boundary. In this context, a system event is an external event that directly stimulates the software.

The system sequence diagrams are part of the use case model. Although it is not part of the Unified Process, Larman describes them as a good way to visualize the interactions implied in the use cases.

3.4.2 Partial Domain Model

The domain model illustrates conceptual classes in a problem domain. The conceptual classes are different from software classes. They represent an
idea, thing or object. The domain model is built incrementally over several iterations during the elaboration phase. In this case, that means concepts related to the use case Process Sale. When identifying conceptual classes it is better to overspecify a domain model, than to underspecify it. If you tend to underspecify the domain model, it is easier to miss conceptual classes that later can be of importance.

Figure 3.4 shows the partial domain model for the Process Sale use case. The model contains both conceptual classes which have a purely behavioral role, and conceptual classes that have an informational role. The figure shows the domain model after associations and attributes have been considered. It is common to build the domain model incrementally, so that concepts can later be discovered and added.

Larman mentions different strategies to identify conceptual classes. A conceptual class category list links different categories to usable examples. This can for example be the category “places” which links to the example “store”. Identifying nouns and noun phrases in textual descriptions is another method.
The use cases provide good descriptions for this type of analysis.

### 3.4.3 From Use Case Details to Design

A deeper look at the *Process Sale* use case is necessary for defining system behavior. *Contract* is the term Larman uses for adding a detailed description of the use cases. Contracts describe system behavior in terms of state changes to objects in the Domain Model, after a system operation has been executed [Lar02].

The system input events from the system sequence diagram invoke system operations. The following Contracts may apply to the Process Sale use case: makeNewSale, enterItem, endSale and makePayment. An example of the makePayment Contract is shown in table 3.1.

<table>
<thead>
<tr>
<th>Contract</th>
<th>makePayment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>makePayment(amount: Money)</td>
</tr>
<tr>
<td>Cross References</td>
<td>Use Cases : Process Sale</td>
</tr>
<tr>
<td>Preconditions</td>
<td>There is a sale underway.</td>
</tr>
<tr>
<td>Postconditions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Payment instance p was created (instance creation).</td>
</tr>
<tr>
<td></td>
<td>p.amountTendered became amount (attribute modification).</td>
</tr>
<tr>
<td></td>
<td>p was associated with the current Sale (association formed); (to add it to the historical log of completed sales)</td>
</tr>
</tbody>
</table>

**Table 3.1: makePayment Contract**

It is common to discover the need to record new conceptual classes, attributes, or associations in the Domain Model when working with Contracts. Larman expresses the importance of not limiting yourself to the prior definition of the Domain Model. Enhance the model, and make new discoveries while thinking through the operation contracts.

The result from the requirements section is now taken to the design level. The goal is to create well designed objects. To get there, Larman uses some design principles called GRASP (General Responsibility Assignment Software Patterns). These patterns are a methodical approach for learning basic object design and responsibility assignment.

Larman points out five of the GRASP patterns which address very basic, common questions and fundamental design issues. These are:
Elaboration

- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Controller.

Table 3.2 shows how to apply part of the Information Expert pattern on the Process Sale use case.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Information Expert (or Expert)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution</td>
<td>Assign a responsibility to the information expert – the class that has the information necessary to fulfill the responsibility.</td>
</tr>
<tr>
<td>Problem</td>
<td>What is a general principle of assigning responsibilities to objects? A Design Model may define hundreds or thousands of software classes, and an application may require hundreds or thousands of responsibilities to be fulfilled. During object design, when the interactions between objects are defined, we make choices about the assignment of responsibilities to software classes. Done well, systems tend to be easier to understand, maintain, and there is more opportunity to reuse components in the future applications.</td>
</tr>
<tr>
<td>Example</td>
<td>In the NextGEN POS application, some class needs to know the grand total of a sale... ...</td>
</tr>
</tbody>
</table>

Table 3.2: Partial Information Expert Pattern

The further work based on this pattern consists of assigning responsibilities. For example: Who should be responsible for knowing the grand total of sale? For the Information Expert, you should look for that class of objects that has the information needed to determine the total. This can either be found in the domain model or the design model. Since this is the first iteration of the project, the domain model would be the place to look.

Together with interaction diagrams you incrementally find design classes by using the patterns. In this case it means creating a design class Sale and adding the getTotal() method. Further, you add the associated design class or classes that are in some way related to the Sale class. The interaction
(collaboration) diagrams are used to illustrate how and what sort of messages that are sent between the objects (see figure 3.5). To complete the use of the Information Expert pattern, the following items should also be considered:

- A discussion part
- Contraindications
- Benefits
- Related pattern or principles
- Also known as; Similar to other patterns

A closer look at these items and the following patterns can be found in Craig Larman's book [Lar02].

Further work consists of use case realization using GRASP patterns. This is where the object-oriented designer assigns responsibilities and establish object interactions. For the NextGen POS system, Contracts are used because they add a greater detail to the system. Further, it is up to the designer to develop partial interaction diagrams that satisfies the requirements.

### 3.4.4 Finishing Iteration 1

Adding the User Interface (UI) to the Domain Layer was partly done during the work with the GRASP pattern "Controller". In the process of constructing the Contracts, we have added some details about how the objects communicate. Figure 3.6 on the facing page shows how the UI is connected with the objects in the domain layer. It is important to know that the UI layer does not have any domain logic responsibilities. It processes only the UI tasks, and forwards the domain oriented responsibilities to the domain.
Before we can draw the design class diagram, there is a need for determining visibility. Visibility is the ability of one object to see or have reference to another [Lar02]. The idea is that for an object "A" to send a message to an object "B", then "B" must be visible to "A". This can for example be done by saying that "B" is an attribute of "A" or "B" is a parameter of a method of "A". When this has been done, you can draw the design class diagram. Larman uses navigability arrows to illustrate the associations between the classes. In fact, he states that:

Most if not all, associations in DCDs should be adorned with the necessary navigability arrows.

– Craig Larman

This is not the exact same set of associations that was generated for the class diagram in the domain model. During the creation of the interaction diagrams, the need for other associations was found. Navigability implies
Figure 3.7: Partial Implementation Model

visibility. Therefore, the work that was done in the visibility section determines in which direction the arrows should point. Figure 3.7 shows how to apply the navigability associations.

The final step of this iteration consists of designing the implementation model\(^2\). All the information needed for mapping the design to code is given in the previous work (See [Lar02] for further details). The developer must decide upon which object-oriented programming language to use. Larman uses Java for the NextGen POS system. Figure 3.7 shows how to map design to code for the Register class.

This ends iteration 1, but there is still a lot of work ahead. Larman describes further two iterations in the elaboration phase for the NextGen POS system. This chapter has provided knowledge of the most essential parts in how to start an OO system development process within the UP. The essence of this

\(^2\)The implementation model is almost analogous with the Design Class Diagram (DCD), but without java code. Larman’s DCD will be presented together with the ROOP DCD in chapter 6.4 on page 78.
Elaboration

chapter should be enough to be used as a basis of comparison with the ROOP methodology.
In this chapter, we will start our design of the NextGen POS system within the ROOP methodology. More specifically, this chapter will discuss how objects are selected in ROOP. The ROOP methodology is based on many of the same ideas that do apply for the Unified Process. However, there are also significant differences between the two methods, especially in the way we think about objects.
4.1 The Case: NextGen POS System

The NextGen POS system case description:

The NextGen point of sale (POS) system is a computerized application used (in part) to record sales and handle payments. An application like this can typically be used in a retail store. Such a system includes hardware components like a computer and bar code scanner, and software to run the system.

The system interfaces to various service applications, such as a third party tax calculator and inventory control. Further, it is important that these systems are relatively fault-tolerant. This means that even if remote services are temporarily unavailable, they must still be capable of capturing sales and handling at least cash payments.

A POS system must support multiple and varied client side terminals and interfaces. These include everything from a Web browser terminal, a regular personal computer running a Java Swing graphical user interface, touch screen input to a wireless PDA.

Furthermore, this is a commercial system intended for different clients with disparate needs in terms of business rule processing. This can for example be a unique set of logic to execute when a new line item is added. Therefore, the POS system will need a mechanism to provide this flexibility and customization.

The case description (4.1) is part of the software requirements specification, but there are also other requirements (factors) that influence the decision process. Larman’s unified solution to the problem presented in chapter 3 reflects the software requirements specification in terms of system and user functionality in the NextGen POS system. It is not ROOP’s intention to question or change these requirements. However, all methodologies are different in the way system needs are described. Therefore, the ROOP methodology will include extended requirements compared to how Larman expresses the system needs.

4.2 Introduction

_The history of OO software development has shown a trend towards higher level of abstraction, code reuse and automatic code generation._

– [DOE01]
4.2.1 Some Basic Beliefs About the Unified Process

It would be hard to suggest an alternative OO system development method (ROOP) without some basic assumptions about weaknesses in the UP. Further, it would be a bad idea to criticize one of the most well-known development methods, without having some ideas about how these weaknesses can be improved. These improvements do not necessary lead to better and more robust systems alone. Good designers make good systems even with bad methods [Ska00]. The fact is that people personalities are an important factor for the methodology chosen. Alistair Cockburn summarizes this in his dissertation for the Degree of Dr. Philos [Coc03], where one of the results was:

> Every project needs a slightly different methodology, based on those people characteristics, the project’s specific priorities, and the technologies being used. This result indicates that a team’s methodology should be personalized to the team during the project and may even change during the project.

– Alistair Cockburn

However, new ideas and assumptions about people and methodologies are what make up future methodologies. If not stand-alone methodologies, then new research may help to build and provide knowledge for existing methodologies in use. In this thesis these assumptions are:

- Use cases (as they are used) causes a fixed system boundary too early in the development process
- There is a risk that too many classes derive from the domain model to the design class diagram
- Later reuse of the class diagram is difficult
- The classes do not open for fast and ready to apply changes within a class
- Later reengineering of the total system is a difficult process

To make the comparison as realistic as possible, all the assumptions mentioned above are addressed at this concrete case study. Then again, the improvements suggested in this thesis are meant to be used as a framework for all OO development processes. In the process of nesting up these ideas, new and hopefully better ways of doing the design will be presented.
4.3 Getting Started

*The players in the game are people personalities operating in an ecosystem.*  
– Alistair Cockburn

The software requirements specification is the official statement of what is required of the system. It should include both the user requirements for a system and a detailed specification of the system requirements [Som01].

The software specification is in addition to describing requirements, a good starting point for discovering objects in the ROOP methodology. Therefore, the software requirements specification makes up an important part of the whole development process. The development team should spend quality time when constructing the requirements specification.

The Unified Process is a use case driven approach [IJR99b], therefore use cases play a central part throughout the process. The first assumption in the previous section stated that use cases in the beginning of the development process causes a fixed system boundary. The Coffee Machine example in chapter 2.2 on page 13 introduced the ROOP methodology, and gave us some ideas of why use cases cause a fixed system boundary early in the development process. The main problem is that use cases describe communication between actors outside of the system and use cases inside the system.

In addition to the systems main functionalities, a list of supplementary specification is required. This includes functionality, usability, reliability, performance, supportability, implementation constraints, interfaces and information in the area of interest. All of these issues are also handled by Larman [Lar02]. For the ROOP methodology, future specification should be added to the list of supplementary specification. This will do the job of finding usable objects easier. In ROOP’s holistic world view, adding future functionality to the system is an imaginary process. We must learn to think of the possibilities that lies in a holistic model of the world! For example by introducing new technology to the system.

4.3.1 Why do Use Cases Cause a Fixed System Boundary?

Writing use cases as User Interface (UI) descriptions is a valid interpretation from the definition of use cases as "a sequence of transactions between external actors and the system." It is writing use cases at the dialog level of interaction [Coc97].
Getting Started

Writing and interpreting use cases just as UI descriptions deprives them of some of their value to the project. The design of the UI is likely to change too often for such writings to be used as contractual requirements and as system requirements. From the process or methodology perspective, the design of the UI comes later, after these goals and interactions have been named. [Coc97] [AB03]. From ROOP’s perspective, this is one of the reasons for why you should avoid starting an OO development process by writing (designing) use cases. History has shown that people often mix this dual vision of how use cases are used, and the result is often a fixed system boundary, which is hard to change later in the process [Coc97] [AB03] [RBT02].

Conventional use cases typically contain too many built-in assumptions, often hidden or implicit, about the form of the User Interface (UI) that is yet to be designed. This is problematic for UI design both because it forces design decisions to be made very early, and because it then embeds these decisions in requirements, making them difficult to modify or adapt at a later time.

– [RBT02]

Larman [Lar02] says that use cases are textual descriptions, and that use case diagrams should be used as a supplement only. Even though Larman points this out, the outcome of the use cases describes the same as a use case diagram, which is communication between actors and use cases inside the system. (See figure 4.1 on the following page).

According to Fowler [wKS00], a way to get around the actor and use case problem is simply to remove the actor from the diagram. However, how can you be sure that the use cases are found when the focus is taken away from the actor? One way to discover all the necessary use cases is to identify actors, but not worrying about the exact relationship between them. What you are after is the use cases. The actors are just a way to get there.

A use case is a set of scenarios tied together by a common user goal [wKS00]. However, who the users are do not have to be specified. By this, you may reduce the strong relationship between the actors and the use cases.

Fowler’s idea is seen as a good contribution to the ROOP methodology, and is one of the better ways to discover the systems functional requirements.

However, ROOP suggest that you focus on the systems main functionality. The main functionality gives the developer team a good starting point for discovering possible object candidates. There are no constrains related to
Getting Started

The main functionality in the NextGen POS system is shown in figure 4.2. The subsequent ROOP models in this thesis derive from this functionality alone. Note that a system can have several main functionalities. If several functionalities are found, you should work out a phase plan for each of the main functionalities. However, in this thesis we will only focus on the main functionality shown in figure 4.2.

An interesting observation is that the customer is not part of Larman’s use case model. According to ROOP, the customer is very central in the holistic system. Therefore, the main functionality reflects this idea by including the customer in the systems main functionality.
4.3.2 Agile Software Development

By means of new OO ideas, ROOP tries to build solutions to comply with later changes in the development process. Being agile is a declaration of prioritizing for project maneuverability with respect to shifting requirements, shifting technology and shifting understanding of the situation [Coc02a]. ROOP tries to adopt this way of thinking.

How can agile processes be applied to more structured development processes? Several articles have already looked at how the Rational Unified Process (RUP)\(^1\) can be enhanced to also include agile methodologies [Amb02] [Meh02]. RUP can be used in a very traditional waterfall style or in an agile manner, it all depends on how you tailor it in your environment [Meh02]. Craig Larman uses agile development techniques within the UP. Examples of this are the SCRUM process pattern and Extreme Programming (XP) practices. SCRUM is an incremental process for developing any product or managing any work. It produces a potentially, shippable set of functionality at the end of every iteration [hoS]. XP is a well-documented methodology for programmers to use. XP scores very high within its area of applicability [Coc02b]. It basically is a set of rules that should be followed. Among these rules are; Use only 3-10 programmers, let one or more customers be on site, do development in three week periods etc. Writing the test first is essential to experience the value of this approach [Lar02].

A key point in agile methodologies is the fact that different projects need different methodology trade offs [Coc02b]. It is a cooperative game of people, technology and environmental factors. All working together, trying to fit one particular project alone.

In terms of this thesis, the meaning of being agile certainly is an important aspect for being successful. Not only because of the technical complexity involved, but also for the many surrounding environmental factors that play an important part of every project. Therefore, agile development methods are highly valued, throughout the process, when it comes to developing OO software with the use of ROOP.

4.4 Finding Objects

Rebecca-Wirfs Brock et.al. [RWBW90] defines an object as part of an object-oriented approach which attempts to manage the complexity inherent in real-world problems by abstracting out knowledge, and encapsulating it within objects. Finding or creating these objects is a problem of structuring knowl-

\(^1\)RUP is a commercial variant of UP. UP and RUP are very closely related [AN02].
Finding objects is an iterative process. However, when it comes to good design it is better to discover lots of objects in the first iteration. Time spent in the first iteration, will be of advantage for the following iterations. The idea is that only minor changes are done for every iteration. If major changes are done in every iteration, then the developer team has spent less time than required on the first iteration.

Larman mentions several techniques in the book [Lar02], and they may all be applied to the ROOP methodology as well. These techniques include:

**Nouns and noun phrases:** Start by identifying nouns and noun phrases in the system requirements specification, and other relevant textual descriptions. Nouns and noun phrases could also be identified when discussing the system with the end users. All nouns are possible object candidates.

**Conceptual class category list:** Start by identifying real world concepts. In the NextGen POS system, you have the concept of sales in a store. Based on the concept, try to discover possible objects within the system model.

The object identification process is a typical group-solving task, and brainstorming is an excellent technique for identifying object candidates. The principle of brainstorming is that all ideas are welcome in the initial phase of the process. The technique can be applied to both nouns and noun phrases, and the conceptual class category list.

In the following sections, the main functionality will be our starting point for finding usable objects. These objects will be the foundation for the new system. The idea is that the ROOP objects make up the basis of the system. Figure 4.3 on the next page shows how ROOP objects fit in under Larman’s domain view of the system. The ROOP objects are shown as a separate layer, called the robust layer.

From Chinese medicine we find the concept of Yin and Yang. These two terms relate to the opposing, yet complimentary qualities, which make up all that is present in the natural world. [All]. How objects are perceived in ROOP is in some way similar to the Chinese understanding of yin and yang. Skagestein’s example of the :Person and the :Loan object illustrates this Yin and Yang behavior in the objects. Figure 4.4 on the facing page is adopted after Skagestein’s lecture notes [Ska00].
Finding Objects

Figure 4.3: ROOP Objects in the Robust Layer

Figure 4.4: The Person and the Loan
Within this yin and yang object perspective, the objects are capable of having more knowledge and responsibility, compared to more traditional methods like the RUP. From the figure 4.4 on the page before we see that the :Person object is given extended responsibilities in form of askHimForInstallation, remindHim etc.

The dotted line tells us that the system boundary has not been decided, and that the objects are capable of representing both the real world part and the computerized part. This view gives us a slightly new perspective on the objects inside the system. They should not any longer be regarded as part of an object-oriented model of the real world, but as computerized augmentations to objects in the real world [Ska01]. The challenge consists of designing these objects so that they can deal with environmental changes\(^2\) in the system. Skagestein points out that moving the boundary does not change the model. It only shifts the responsibilities between the real world part and the automated part of the object and alters the way the two parts collaborate [Ska01].

In Larman’s model of the NextGen POS system, it is our assumption that moving the object boundary will imply changes to the very basics of the system: Therefore, if we wanted to adapt Larman’s model for later reuse, this would cause major reengineering work. Trygve Reenskaug emphasizes the problem of reuse by saying; the single most highly promoted advantage of the object paradigm is its support for reuse, but this is also the area of deepest disappointments [Ree96]. In other words, if object reuse is taken into account, you will have a major advantage when changes are applied to the system.

4.4.1 ROOP Category List for the NextGen POS System

*The act of developing software is nothing but making ideas concrete in an economic context.*

– Alistair Cockburn

The candidate list is based upon the systems main functionality, which is to satisfy the customer’s need to buy products. The list is partly adopted from Larman’s candidate list for the Process Sale use case.

Object candidates for the NextGen POS system:

1. Register

\(^2\)The environment is part of ROOP’s holistic world view.
Finding Objects

2. Item
3. Store
4. Sale
5. Payment
6. Product Catalog
7. Manager
8. Product Specification
9. SalesLineItem
10. Cashier
11. Customer
12. Manager
13. Product

The object candidates are only suggestions to possible ROOP objects. The ROOP methodology uses special design patterns, which helps us to select the objects we want to continue to work with. For every object candidate, read the patterns and see if one of them applies to the object at hand. If you find a match, then mark the object as a ROOP object. The ROOP patterns are:

- Stakeholders
- No Data Movers
- Encapsulation

The Stakeholders pattern is designed to find the primary stakeholder or stakeholders in the system. The NextGen POS system is based on selling product to customers. The customer object is therefore included in the object model in ROOP. The Stakeholders pattern helps us select the objects who / which the system is intended for. The main functionality of the system indicates the importance of the customer object. (See table 4.1 on the following page)

The No Data Movers pattern helps us to select the objects that are doing something more than moving data from one object to another. Typical data movers are removed from the system. When you work with the pattern, try to find objects which has the potential to have extended responsibility. Such
Name: Stakeholders

Problem: A system may affect many people, machines or other systems. Most of these affected parties are only data movers that act on behalf of the genuine stakeholders.

Solution: Include only the stakeholder (or stakeholders) the system is intended for.

Example: In a car rental system, an important stakeholder is the customer. The clerk who is operating the system is nothing but a data mover.

Strengths: The pattern defines the stakeholder (or stakeholders) which do not change over time.

Weaknesses: Some stakeholders can act as data movers as well as being the genuine stakeholder. In some situations this could lead to problems finding the true stakeholders.

Table 4.1: Stakeholders Pattern

Objects are defined as smart objects in ROOP. By smart we mean that the objects can have extended responsibility, and operate under different states\(^3\). One state is the physical and real part of the object, and the other is the computerized (automated) part of the object. (See table 4.2).

Name: No Data Movers

Problem: In many systems, some objects have no other responsibility than moving data between objects. These objects are of no interest in ROOP. Data movers contribute to larger models, and should therefore be avoided.

Solution: Include only the objects which are inside our area of interest and do nothing more than moving data between other objects.

Example: A “clerk” is a typical data mover. Other data mover objects are typical static objects like “coin slot” and “cup holder” in a coffee machine.

Strengths: It is easier to distribute responsibilities between the objects, since there are fewer objects to deal with.

Weaknesses: Exaggerated use of the pattern can lead to too much responsibility in one or more objects.

Table 4.2: No Data Movers Pattern

The act of grouping into a single object both data and the operations that

\(^3\) A state refer to the condition of an object between events. Different states means that stimulus messages can be initiated from either the computerized part or the real world part.
Finding Objects

affect that data is known as encapsulation [RWBW90]. The Encapsulation pattern complement the No Data Movers pattern, and helps us in forming (thinking) of objects as smart objects. The idea is that every object should be responsible for their own actions. By doing this you automatically leave out the data mover objects. (See table 4.3)

<table>
<thead>
<tr>
<th>Name: Encapsulation (smart objects)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem:</strong> Expect changes in parts of the system, which will affect how the user interface is treated. We want that these changes do not force other changes to the rest of the system.</td>
</tr>
<tr>
<td><strong>Solution:</strong> Create classes with stable definitions and encapsulates the information. Later changes will then only affect the object alone.</td>
</tr>
<tr>
<td><strong>Example:</strong> In the person and loan example, it is important that the person object is given responsibilities that reflect future changes to the system. The person should for example be given the responsibility to askForPayment. All objects are considered as smart objects.</td>
</tr>
<tr>
<td><strong>Strengths:</strong> It is easy to enhance the encapsulated objects.</td>
</tr>
<tr>
<td><strong>Weaknesses:</strong> Exaggerated use of the pattern can lead to too much responsibility in one or more objects.</td>
</tr>
</tbody>
</table>

Table 4.3: Encapsulation Pattern

4.4.2 ROOP Objects in the NextGen POS System

After applying the patterns, the following four objects are selected:

**Customer:** Primary stakeholder and smart object

**Sale:** Smart object

**Item:** Smart object

**Product:** Smart object

The objects are shown in figure 4.5.

![Encapsulated Smart Objects in ROOP](image)

Figure 4.5: Encapsulated Smart Objects in ROOP

In the next phase, we will discuss how to add responsibility and knowledge to the objects. We will use a combination of Class, Responsibility and Collaboration (CRC) cards, and collaboration diagrams to complete the task.
As part of an iterative development cycle, we might need to go back and look at the candidates one more time. If any ROOP objects are left out, the CRC cards will reflect this as we play the game.
Chapter 5

Collaboration

In this chapter we will discuss how objects collaborate within the ROOP methodology. By using CRC cards we are able to simulate how messages are sent between the objects. In the process of assigning knowledge and responsibility we do not distinguish between real life behavior and computerized behavior. This is part of ROOP’s holistic (and somewhat mechanical) world view of the system. The distribution of knowledge and responsibility (between the real world and the computerized part of the objects) will be discussed in the next chapter; Splitting the Objects.

Some of the ideas applied in this chapter are adopted from the CRC Card technique [BS99] [KB89] [NS96]. We have just enhanced the ideas to fit the purpose of selecting ROOP objects.

5.1 Collaboration

How do classes fulfill their responsibilities? They can do so in two ways: by performing the necessary computation themselves, or by collaborating with other classes.

– [RWBW90]

From chapter 4.4.2 on page 53 we found that the :Customer, :Sale, :Item, and :Product objects were part of the core system architecture\(^1\) in ROOP. These are the objects that we want to continue working with, before we split the objects and decide upon the system boundary. Figure 5.1 on the next page shows a model of the four objects with a flexible system boundary\(^2\).

\(^1\)The core system architecture is a collection of (ROOP) objects that derive from the finding objects phase.

\(^2\)A flexible system boundary means that the distribution of knowledge and responsibility between the computerized part and the real world part of the objects, have not been decided yet.
In RUP, use cases can sometimes be understood as UI design specifications, even if the purpose is directed towards the design class model. From the NextGen POS system in Larman’s book [Lar02], we see that the actor “Cashier” is connected with the use case “Process Sale”. The use case builds upon an assumption of how the UI will be like. Further, this will again affect how responsibility and knowledge is distributed in the system model. Figure 3.2 on page 31 illustrates the idea.

The collaboration diagram in figure 5.2 on the next page shows the objects that are involved for fulfilling the makePayment method in UP. Responsibility and knowledge for these objects (classes) within UP are shown in figure 5.3 on the facing page. Responsibility is shown as methods, and knowledge is shown as attributes.

The distribution of knowledge and responsibility is different in ROOP. Figure 5.4 on page 58 shows ROOP’s solution for fulfilling the makePayment method. Note that even more knowledge and responsibility could be assigned to the :Customer and :Sale objects. The example is meant to illustrate the possibilities you have when distributing knowledge and responsibility in ROOP.

The main differences between Larman’s solution and the ROOP solution are as follows.
Collaboration

Assumes UI communication between :Cashier and :Register (from use case diagram).

Figure 5.2: Collaboration Diagram within the Unified Process

Figure 5.3: Design Classes within the Unified Process
Figure 5.4: ROOP Collaboration Diagram with Complementary Design Classes
In ROOP:

- The :Customer object is part of the collaboration diagram. This is in accordance with the Stakeholders pattern found in the ROOP methodology.

- The collaboration diagram does not say anything about who or what triggers the makePayment method in the :Sale object.

- The :Register and :Sale objects appears as one object.

- The objects have significant more knowledge and responsibility compared to Larman’s solution.

- The objects includes “future proofing”. Low coupling and encapsulation is part of the ROOP design. Future proofing implies that you include responsibility and knowledge, which might not be implemented when it comes to the realization of the model. How much of the responsibility and knowledge that are included will be decided after we have determined the system boundary.

Later changes to the system will most likely affect the distribution of knowledge and responsibility between the computerized part and the real part of the objects. In ROOP, objects are designed to meet these changes with minimal reengineering work within the original model. On the other hand, Larman questions the effort designers should put into “future proofing” and lowering the coupling where in fact there is no realistic motivation involved [Lar02] [Lar01].

Designers have to pick their battles in lowering coupling and encapsulating things. Focus on points of realistic high instability or evolution.

– Craig Larman

From ROOP’s point of view there are minor risks involved in lowering coupling and encapsulating things, because the model has fewer (and more abstract) objects, and the system boundary has not been determined yet.

5.1.1 The CRC Card Technique

CRC cards are index cards (or computerized versions of same!), which are used to record suggested classes, the things they do, their responsibilities, and their relationship to other classes, collaboration [BS99]. CRC cards explicitly represent multiple objects simultaneously [KB89], where you use one
Collaboration

card for each class [Lar02]. However, in ROOP we consistently talk about objects, and use one index card for each object (or role). In chapter 5.1.2 on page 64 we discuss why ROOP uses object instances, and how to play the cards in the ROOP methodology. The idea of using one card for each object instance is adopted from Else Nordhagen and Gerhard Skågestein [NS96].

The purpose of "object think" is the same as using CRC cards; to get people to think that objects in computer systems know things, and know how to do things [Sr01] [Sr96]. However, ROOP does not limit the CRC card approach and "object think" approach to deal with objects inside the computerized system.

Although CRC cards are not part of the Unified Modeling Language (UML), they have successfully been used in different methodologies since Beck and Cunningham introduced the technique in 1989 [KB89]. Within ROOP, the task is to capture as much knowledge and responsibility as possible, and decide later whether it should be handled by the computerized part of the object or the real part of the object. As mentioned above, put effort into “future proofing” and lower coupling. Think of all objects as smart objects, and assign all kinds of possible future (and present) knowledge and responsibility to the objects.

The CRC cards for :Customer and :Sale are shown in table 5.1 on the next page and table 5.2 on the facing page.

From table 5.1 on the next page and table 5.2 on the facing page we see that the objects are given responsibilities to carry out different tasks. When we design these CRC cards we must learn how to simulate what kind of role the objects play in the total system. We must ask ourselves questions like; which object should be given the responsibility to carry out the task at hand, and which object is most likely to hold the necessary knowledge? Or as suggested in the object think approach, it is better to play the role of the object and say; I am a sales object. It is my responsibility to calculate prices and sales. When we learn that object are part of an holistic system, then it is easier to encapsulate the objects behavior.

We use collaboration diagrams to illustrate the collaborations that follows from responsibilities given in the CRC cards. It is better to draw several collaboration diagrams with only two or three objects involved. This is because it is difficult to capture how messages are sent in the total model. Designers who choose to employ their knowledge of the total system into one large model, should have good knowledge of how the system works before drawing
Collaboration

<table>
<thead>
<tr>
<th>C1 :Customer</th>
<th>Collaborators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge:</td>
<td>- Sale</td>
</tr>
<tr>
<td>- I am a customer in the work context of a computerized object, and a real life customer.</td>
<td></td>
</tr>
<tr>
<td>- I know my identity.</td>
<td></td>
</tr>
<tr>
<td>- I have knowledge of what I usually buy.</td>
<td></td>
</tr>
<tr>
<td>- I know how to pay for products.</td>
<td></td>
</tr>
<tr>
<td>- I know what I need.</td>
<td></td>
</tr>
<tr>
<td>- I know how much money I have.</td>
<td></td>
</tr>
<tr>
<td>- I know my billing status.</td>
<td></td>
</tr>
<tr>
<td>Responsibility:</td>
<td></td>
</tr>
<tr>
<td>- I am responsible for verifying when a sale is complete.</td>
<td></td>
</tr>
<tr>
<td>- I am responsible to remind myself to pay bills.</td>
<td></td>
</tr>
<tr>
<td>- I am responsible for completing payment.</td>
<td></td>
</tr>
<tr>
<td>- I am responsible for telling :Sale that I have purchased an item.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: CRC Card for (an Instance of) the Customer Object

<table>
<thead>
<tr>
<th>S1 :Sale</th>
<th>Collaborators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge:</td>
<td>- Item</td>
</tr>
<tr>
<td>I am a :Sale object in the work context of a computerized object, and a real life physical device.</td>
<td></td>
</tr>
<tr>
<td>I have knowledge of name, date, time, address, sale number and register id.</td>
<td></td>
</tr>
<tr>
<td>I have knowledge of identified customers.</td>
<td></td>
</tr>
<tr>
<td>I have knowledge of subtotal and total amount.</td>
<td></td>
</tr>
<tr>
<td>Responsibility:</td>
<td>- Customer</td>
</tr>
<tr>
<td>I know how to add and end a sale.</td>
<td></td>
</tr>
<tr>
<td>I am responsible to ask customer for payment.</td>
<td></td>
</tr>
<tr>
<td>I know how to ask :Item for prices.</td>
<td></td>
</tr>
<tr>
<td>I know how to calculate prices and sales.</td>
<td></td>
</tr>
<tr>
<td>I know how to process purchased product items.</td>
<td></td>
</tr>
<tr>
<td>I know how to process payment.</td>
<td></td>
</tr>
<tr>
<td>I know how to get total amount.</td>
<td></td>
</tr>
<tr>
<td>I know how to complete a sale.</td>
<td></td>
</tr>
<tr>
<td>I know how to create a new sale.</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: CRC Card for (an Instance of) the Sale Object
a complete collaboration diagram. ROOP suggests that you draw several small collaboration diagrams in the beginning, and then put them together in one large model.

Figure 5.5 shows the collaboration diagram for fulfilling the customer’s responsibility for telling :Sale that an item is purchased. The message enterItem is sent to the :Sale object. Note that enterItem is not an UI description. The message implies only the possibility of letting the :Customer object trigger the event “Item entered.” How items are entered into the system is not decided yet. After we have split the objects and decided upon the system boundary, then we can decide how to realize the UI interaction.

Figure 5.5: Collaboration Diagram for enterItem()

The CRC cards for :Item and :Product are shown in table 5.3 and table 5.4 on the next page.

<table>
<thead>
<tr>
<th>I1 :Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knowledge:</strong></td>
</tr>
<tr>
<td>I am a :Item object in the work context of a computerized object, and a real life product item.</td>
</tr>
<tr>
<td><strong>Responsibility:</strong></td>
</tr>
<tr>
<td>I know how to send price on purchased item, when :Sale asks for price.</td>
</tr>
<tr>
<td>I know how to ask :Product for my price.</td>
</tr>
<tr>
<td><strong>Collaborators:</strong></td>
</tr>
<tr>
<td>- Product</td>
</tr>
<tr>
<td>- Sale</td>
</tr>
</tbody>
</table>

Table 5.3: CRC Card for (an Instance of) the Item Object

Figure 5.6 on the facing page includes the :Item and :Product objects in the collaboration diagram. The figure shows that :Sale asks :Item for price, and :Item asks :Product for the listed price. Even more knowledge and responsibility could be assigned to each object. Anyway, the CRC cards illustrate how knowledge and responsibility may be distributed in ROOP.

No UI interactions are described in the CRC cards. An example of such
Collaboration

<table>
<thead>
<tr>
<th>P1 :Product</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knowledge:</strong></td>
</tr>
<tr>
<td>I am a :Product object in the work context of a computerized object, and real life products.</td>
</tr>
<tr>
<td>I know the identification number and name for all items that belong to me.</td>
</tr>
<tr>
<td>I have knowledge of product prices.</td>
</tr>
<tr>
<td>I know my stock of goods.</td>
</tr>
<tr>
<td><strong>Responsibility:</strong></td>
</tr>
<tr>
<td>I know how to send prices on products, when :Item asks for price.</td>
</tr>
<tr>
<td>I know how to order products</td>
</tr>
<tr>
<td>I know how to calculate changes in the stockbuilding.</td>
</tr>
<tr>
<td>I know how to register changes in the stockbuilding.</td>
</tr>
<tr>
<td><strong>Collaborators:</strong></td>
</tr>
<tr>
<td>- :Item</td>
</tr>
</tbody>
</table>

Table 5.4: CRC Card for (an Instance of) the Product Object

Figure 5.6: Collaboration Diagram for getItem()
an interaction is; I know what to do when my button is pressed. All UI interaction in the model will be described in chapter 6.3 on page 72.

5.1.2 Play the Cards

ROOP adopts Else Nordhagen and Gerhard Skagestein’s ideas of how to play the cards within a small group of people citeelseskag. the basic ideas is to use one object instance for each card.

Individual group members should claim ownership of one or more roles (cards). Put one card onto the table for each object instance [Coc99] [Coc]. This makes it much simpler to check that each role has the necessary information to take responsibility for its own part of the activities [wPL01]. You fill out the CRC cards while the group sends messages to each other. When each card has enough information, you compare instances of the same class and see if they match. If no inconsistencies are found, then all instances of that class have the same behavior.

Hanoi’s tower is a well-known game, which may be used for illustrating the principle of analyzing by means of objects instead of classes. The purpose of the game is to move a collection of rings from one side to the other, one ring at a time. The rings are placed on sticks, and there are three sticks all together. Each ring has a different size, and you cannot put a big ring on top of a small ring. From an object-oriented point of view, some of the rings (small, medium, large etc) have different responsibilities compared to the others. Therefore, it would be hard to capture these responsibilities if we only had one card for the ring class. On the other hand, if we used one card per ring instance, we would easily see that some of the rings had different responsibilities. Inconsistency between instances of the same class gives us as OO designers a hint that you should specify the class differently, or use subclasses to overcome the problem.

By means of using CRC cards in the OO solution of Hanoi’s tower, Else Nordhagen [NS96] discovered that inconsistencies between object instances of the same class, had to be designed with the use of interface objects. This is interesting because the idea behind developing the ROOP methodology derives from how interface objects are normally treated within UP. We will continue this discussion in chapter 7.3 on page 90.

---

4The idea of using object instances instead of classes is further discussed by Cockburn [Coc99]. Reenskaug’s role modeling is also based on the same idea [wPL01].

5See [NS96] for a detailed description of Else Nordhagen’s OO solution of Hanoi’s tower.
Chapter 6

Splitting the Objects

In this chapter we will discuss how to split the objects and decide upon the system boundary. Deciding upon the distribution of responsibilities between the real world part and the computerized part, are among other factors decided by the requirements specification.

Further, this chapter introduces how use cases can be applied in the process of specifying the communication point\(^1\) between the real and computerized part of the objects.

6.1 Split the Objects and Decide upon the System Boundary

The software requirements specification is used as a starting point for our discussion of where the system boundary should go. Should the NextGen POS system be smart, regular, self operated or operated by an assistant? The answers to these questions are likely to be found in the software requirements specification. If we had been using RUP, we would have designed our use cases in the beginning of the process, based on the facts in the requirements specification.

In the beginning of the development process, ROOP uses the requirements specification mainly to find the systems main functionality. Specific UI details (in the requirements specification\(^2\)) are neglected to avoid setting up a fixed system boundary early in the development process.

A list of some of the issues that influence the system boundary decision and how the objects are split, are listed below:

\(^1\) A communication point describes a user interface in the cross-over boundary.

\(^2\) For example; A cashier shall operate the register in the NextGen POS system.
The software requirements specification: The software requirements specification gives the development team a good overview of the situation.

Different solutions: Discuss each object separately and define the real part and the computerized part of the object. Small adjustments result in different solutions. Different solutions should also include future technological possibilities. Even though much of the preliminary work was carried out during the CRC card phase, we must look at the current situation and decide how much responsibility we want to give the objects.

Technical limitations, costs and delivery time: Technical limitations may affect where the boundary between the computerized and the real part of the object are drawn. In addition, more advanced solutions may influence delivery time and costs.

Usability engineering and user interface design: There is probably no skill with greater disparity between its importance to successful software and its lack of formal attention and education than usability engineering and user interface (UI) design [Lar02]. A careful study of these issues should be discussed to prevent failures to the system design. Some of these decisions are critical to where the boundary between the real and computerized part are drawn.

Figure 6.1 on the next page and figure 6.2 on the facing page illustrates some of the possibilities we have by using the ROOP methodology.

The first example illustrates (see figure 6.1 on the next page) how the real part of :Sale reads the bar code from the real part of :Item. This could for example be the case in an advanced (future) shopping center. We assume that the :Sale object has some sort of knowledge (input from :Customer) about the shopping list, and automatically enters the items while the customer shops (for example puts the items in the bag).

The other example (see figure 6.2 on the facing page) shows how the computerized part of :Sale asks the computerized part of :Customer for payment. If this scenario was realized, then the computerized part of :Customer would have to send a "remind me to pay" message to the real part of itself. This is analogous to the :Person and :Loan example in figure 4.4 on page 49.

The point is that even if distribution of knowledge and responsibility is changed between the computerized part and the real part of the object, it does not change how messages are sent between the objects. With the use of CRC cards, we have given the objects knowledge and responsibility that
Split the Objects and Decide upon the System Boundary

Figure 6.1: Real Part of :Sale Reads the Bar Code from the Real Part of :Item

Figure 6.2: Computerized Part of :Sale Asks the Computerized Part of :Customer for Payment.

may be realized when we decide upon the system boundary.

After a careful study of the four items described above, we conclude with the following object model (See figure 6.3). The object model reflects the current system that will be implemented.

Figure 6.3: Split Object Model in ROOP

In the NextGen POS example, it turns out that all objects in the object model are split in a real world part and a computerized part. No objects are
completely outside of the system boundary, nor are they completely inside of the system boundary.

The task at hand is to decide what happens in the cross-over boundary between the real world part and the computerized part of the objects. By saying that all ROOP objects are split, we automatically acknowledge two things about the objects:

The objects are perceived as being part of one or both of the following:

- The objects have physical, real part elements and automated, computerized elements.
- Some of the cross-over boundaries reflect user interface interaction in the communication point between the real and computerized part of the objects. Other interfaces communicates with technical devices.

In contrast to Larman’s solution to the problem, it was interesting to observe that the UI actually goes in the communication point of the object, and not between two objects. In the object selection process, we pointed out that the customer was not part of Larman’s original use case model. Since we are about to determine the system boundary, we will continue this discussion, and explain why Larman thinks that the customer is not the system’s primary actor.

From Larman’s perspective, the customer is not the primary actor because the NextGen POS system services the goal of the cashier. Larman uses an actor-goal list, which determines the system boundary based on the system’s goal. If the system goal had been to buy items, then the customer would have been the primary actor of the system. In other words, Larman determine the system boundary early in the development process based on the actor-goal list. Different goals, lead to different fixed system boundaries. Figure 6.4 on the next page shows how the system boundary changes dependent of the actor goal specification. The figure is adopted from Larman’s book [Lar02].

Determining the system boundary dependent of the actor-goal list is in strong contrast to the ROOP methodology. Instead of using an actor-goal list, ROOP asks: whom does the system serve? No matter how the system boundary is drawn, the system will always serve (in this case) the customer. Whom the system serves is considered a fundamental question when using the ROOP methodology.

Even in ROOP, we can find objects that are completely computerized (automated), or completely located in the real world (not automated). The
6.2 Adding Objects to the Model

In Larman’s UP solution, we have seen that a cashier operates the NextGen POS system. The ROOP methodology has so far ignored this fact, because the cashier causes a fixed (sequence) view on how the NextGen POS system is perceived in the beginning of an OO system development process. With a fixed sequence view, we mean that you (in this case Larman) decide in what way the system is meant to function. To compensate for this system functionality that we have chosen to disregard, ROOP adds necessary objects after we have decided whether the core objects are split in a computerized and real world part.

From the object selection process, we found that the cashier was a typical data mover object. Nevertheless, since the current system is to be operated by a cashier, we add the cashier object to the ROOP object model at this point. Figure 6.5 on the next page shows the cashier as a primary stakeholder and as an object instance.

---

3The cashier shall operate the NextGen POS system. This is a choice that we have taken.
How does the :Cashier object affect the ROOP model? The answer to this question is that the ROOP model is not affected at all. The :Sale object in ROOP is responsible for calculating any items entered into the NextGen POS system. In the collaboration phase, ROOP took into account several possibilities for how items were processed into the system by giving the :Sale object the necessary responsibilities for fulfilling any external behavior. In addition, we gave the :Customer object extended responsibility in form of handling parts of fulfilling payment and initiating a sale. Because of the current situation, initiate sale is now the cashier’s responsibility. The cashier takes over the communication between the two parts of :Sale, because the real part of :Sale is not supposed to operate the computerized system directly.

Figure 6.6 shows how Larman sees how the cashier communicates with the register in the NextGen POS system. The user interface is seen as the communication point between the :Cashier object and the :Register object. From Larman’s UP solution we see that the cashier is completely left outside of the system boundary, and that the register is inside of the system boundary. This system view is in accordance with Larman’s use case diagram (see figure 3.2 on page 31 for details).
Adding Objects to the Model

between the :Cashier and :Sale. We have already identified the :Sale object as being split in a real world part and a computerized part. From ROOP’s perspective the :Cashier object is also split in a real world part and a computerized part. The real part of the :Cashier object initiates a sale, while the computerized part holds on to the cashier identification. The cross-over boundary between the real and the computerized part of the :Sale object is the main user interface for the NextGen POS system. The real part of the :Sale object makes up any physical parts that the cashier needs to enter items into the NextGen POS system.

![Diagram of Cashier and Sale Interaction]

ROOP recognizes the UI as being in the communication point of the cross-over boundary between the real and computerized part of the :Sale object.

Figure 6.7: The Cashier Interacts with the Sale Object in ROOP

Now that we have decided whether the objects are split in the current version of the NextGen POS system, we need to describe how they are split. What happens in the cross-over boundary between the real part and the computerized part of the objects? ROOP find use cases to be a good tool for describing UI communication. In contrast to Larman that uses use cases for describing the overall functional requirements in the NextGen POS system, ROOP has postponed the use of use cases until now. What we really need to do, is to describe what happens when the cashier enters items into the system, and how the system deals with the input information.

In addition to use cases, ROOP uses an activity diagram together with ac-

---

4The necessary responsibilities are implemented in the :Sale object, which makes the :Register object redundant.
tivity realization diagrams to identify key activity processes that are directly involved in the cross-over boundary events.

From the NextGen POS object model, we have identified all objects as being split. However, we will only focus on the cross-over boundary in the Sale object to illustrate the ideas behind this identification process. To complete the ROOP model for the NextGen POS system you should identify every cross-over boundary for each object by means of:

- A detailed explanation of the real part of the object.
- A detailed explanation of the computerized (automated) part of the object.
- If the cross-over boundary is identified as a UI, then describe the interface with the use of use cases, an activity diagram and activity realization diagrams.

### 6.3 Cross-Over Boundary Specification

The use case diagram in figure 6.8 on the next page shows how the cashier interacts with the NextGen POS system. The diagram identifies two stimulus messages that are sent from the cashier to the computerized part of the Sale object. This is the “Enter Item” message and the “Make Payment” message. There are of course other use cases that could be identified, but we suggest that you draw one use case diagram for every system sequence process in the system that includes UI interaction. An example of another system sequence process is “Cancel Item”.

Note that, since UI communication is identified in the cross-over boundary of the object, other objects may create user interfaces as well. In this example, we will focus on the system sequence process of entering an item into the system. The system process ends when the customer has received and paid for the item.

The use case diagram gives a good overall understanding of the user activities involved. However, we still need to specify the use case diagrams. Table 6.1 on page 74 and table 6.2 on page 74 shows the “Enter Item” and “Make Payment” use cases as written descriptions.

Use cases are good for identifying UI interaction, but they are not so good for capturing the activities that describe the data transmission from the real world part to the computerized part. Somehow we need to identify these data transmissions by giving the activities names. ROOP suggest that you
Figure 6.8: Use Case Diagram in ROOP
## Use case: Enter Item

**ID:** UC1

**Actors:**
Cashier

**Includes:**
UC2: Get Subtotal

**Preconditions:**
A customer wants to buy products

**Main success scenario:**
1. Cashier enters item price on the register.
2. Sale calculates price
3. Subtotal price is shown on the register.

**Success end conditions:**
Subtotal is displayed on the register

<table>
<thead>
<tr>
<th>Table 6.1: Use Case Description of Enter Item</th>
</tr>
</thead>
</table>

## Use case: Make Payment

**ID:** UC3

**Actors:**
Cashier

**Includes:**
UC4: Verify Payment

**Preconditions:**
UC1: Enter Item and UC2: Get Subtotal has been carried out

**Main success scenario:**
1. Cashier presses button for make payment
2. The customer pays for the products
3. Cashier enters payment received (verify payment).
4. End sale.

**Success end conditions:**
Receive payment from the customer. Sale registered.

<table>
<thead>
<tr>
<th>Table 6.2: Use Case Description of Make Payment</th>
</tr>
</thead>
</table>
use a special designed activity diagram. The activity diagram’s main function is to identify the activities that are in the communication point of the cross-over boundary. ROOP defines an activity as a more detailed explanation of the use cases involved. Identifying the key activities is analogous with looking at the cross-over boundary through a magnifying glass.

The ROOP activity diagram is shown in figure 6.9 on the following page. The activity diagram identifies key activities of all objects involved in the system sequence process of entering items into the system, until the customer receives and pays for the items.

From figure 6.9 on the next page, we see five separate diagram parts tied together. Each diagram part is analogous with swimlanes as we find them in UML activity diagrams. In addition, the objects in the diagram are split in a real world part and a computerized part. Some of the activities are part of the real world and the computerized system. These are the cross-over boundary processes in the system sequence process\(^5\), from entering items into the system, until the customer receives and pays for the items.

The activity diagram identifies a total of five cross-over boundary activities. However, we will only describe the four cross-over activities in the :Sale object for now. The cross-over activity of Verifying Payment in the :Customer object involves a more detailed study of the :Customer object, which this thesis does not have room for. Nevertheless, leaving this activity description out does not affect the total understanding and presentation of the ROOP methodology.

After we have identified the cross-over activities in the :Sale object we use an Activity Realization Diagram to describe the activity further. Table 6.3 on page 77, table 6.4 on page 77, table 6.5 on page 77 and table 6.6 on page 77 identifies real world and computerized parts in each activity. The realization diagram gives useful information about the UI prototype. When we are able to identify real world elements, then it is easier to grasp the physical and computerized aspect of the UI.

The last activity in describing the cross-over boundary is to draw a prototype of the UI, and connect the interface to the object model. A UI prototype illustrates parts of the NextGen POS user functionality. However, there is little or no incentive in designing a different UI than described by Larman in figure 3.6 on page 37. Therefore, we will use Larman’s prototype as a reference for the UI in ROOP. Using the same visual prototype does not

\(^5\)A system sequence process describes input and output events related to the main functionality under discussion.
Figure 6.9: ROOP Activity Diagram for the NextGen POS System
## Cross-Over Boundary Specification

### Activity 1: Enter Item

<table>
<thead>
<tr>
<th>Real world part</th>
<th>Computerized part</th>
</tr>
</thead>
<tbody>
<tr>
<td>User interface panel (buttons, screen etc). Hardware.</td>
<td>Recognize which button that is pressed. Capture the information that is on the screen. Send items entered to Process Task.</td>
</tr>
</tbody>
</table>

Table 6.3: Activity Realization Diagram for Enter Item

### Activity 1: Make Line Item

<table>
<thead>
<tr>
<th>Real world part</th>
<th>Computerized part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display (screen) that shows subtotal of the sale.</td>
<td>Receive information from Get Subtotal and process the information.</td>
</tr>
</tbody>
</table>

Table 6.4: Activity Realization Diagram for Make Line Item

### Activity 1: Make Payment

<table>
<thead>
<tr>
<th>Real world part</th>
<th>Computerized part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display (screen) that shows total sale. Make Payment button</td>
<td>Recognize that Make Payment button is pressed. Process task.</td>
</tr>
</tbody>
</table>

Table 6.5: Activity Realization Diagram for Make Payment

### Activity 1: End Sale

<table>
<thead>
<tr>
<th>Real world part</th>
<th>Computerized part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display (screen) that shows the total amount (cash) to pay. End sale buttons</td>
<td>Receive information from Verify Payment. Process the information and display the information on the screen.</td>
</tr>
</tbody>
</table>

Table 6.6: Activity Realization Diagram for End Sale
mean that the UI is connected to Larman’s object model. The UI in ROOP is connected to the object that is responsible for splitting the object. In this case, that means the Sale object. Later changes to the UI are considered as one of the advantages in the ROOP methodology, because there is only one object involved in creating the interfaces. That is, each object determines the UI in the cross-over boundary between the real and computerized part of the object.

6.4 The Design Class Diagram – Putting It All Together

The activity that finalizes the first iteration of the system development process is to draw the Design Class Diagram (DCD). This is where you put all the parts together into one model. In practice, the design classes are created in parallel with the collaboration diagrams as showed in the collaboration phase.

We have deliberately postponed Larman’s DCD until now, so that we can compare the model with the ROOP DCD. Figure 6.10 on the next page shows Larman’s DCD of the NextGen POS system.

Compared to the ROOP DCD in figure 6.11 on page 80, there are significant differences. In ROOP, the main differences are in the Sale class and the Customer class. The Sale class has more responsibilities, which makes some of the other objects redundant. The responsibilities for Larman’s Payment class are transferred to the Customer class in the ROOP model. In addition, the Customer class is given extended responsibility in form of customer identification. The idea is to make the Customer class ready for future changes, since it is one of the most important classes in the model.

6.5 The End of Iteration 1

This is where the first iteration of ROOP ends. Before the final model is ready for production, more planning and testing is required.

Through the ROOP methodology and the Unified Process, we have learned that different OO approaches result in different OO models. However, we are not saying that the ROOP methodology can be applied to every project out there. The ROOP methodology is one out of many contributions on how to start an OO system development process. By using different development techniques and starting with a different system view, we have highlighted some of the advantages that follow postponing the drawing of the system
Figure 6.10: Larman’s Design Class Diagram for the NextGen POS System
Figure 6.11: ROOP’s Design Class Diagram for the NextGen POS System
boundary early in the development process. Maybe some of these ideas can be applied to other OO projects in the future.
The End of Iteration 1
Chapter 7

Discussion

In this chapter, we will look at different model views that affect modern Object-Oriented (OO) methodologies. In the light of these OO perspectives, we will discuss the ROOP object world, and focus on why or what may initiate the objects inside our software model.

There are two ways of constructing a software design:

- One way is to make it so simple that there are obviously no deficiencies
- And the other is to make it so complicated that there are no obvious deficiencies

– C. A. R. Hoare

7.1 Model View Perspective

Like everything metaphysical the harmony between thought and reality is to be found in the grammar of the language.

– Ludwig Wittgenstein

7.1.1 The Model as a Copy of the Universe of Discourse

Some models capture only a static view of the universe of discourse, while other models also capture some of the behavior. Entity Relationship (ER) diagrams [Che76] and UML classes without the operations are typical examples of a static representation of reality. The model is analogous with a photo captured by a camera [Ska02]. Extending the static models by adding responsibilities to the objects will lead to a dynamic model of the real world. OO models, OO databases and CRC cards are examples of dynamic models,
where the stimulus message must be initiated from the environment. The whole idea of simulating dynamic behavior in models was the starting point for the very first OO programming language called SIMULA, developed by Ole Johan Dahl and Kristen Nygaard [OJD67].

However, common for all these models is that they are some kind of copy of the universe of discourse.

The classical context diagrams are\(^1\) (See figure 7.1) based upon a static view of the universe of discourse.

\begin{figure}[h]
  \centering
  \includegraphics[width=0.5\textwidth]{context_diagram.png}
  \caption{Context Diagram}
  \end{figure}

Tom DeMarco expresses the data model (information system) in form of essential memory. Essential memory is a simulation of things whose true existence lies outside system boundaries [DeM79]. The theme used to group the many data elements in essential memory comes from how we humans normally understand and communicate information about things. An example of essential memory is the film within an imaginary camera that we train on these objects, recording their features with data instead of silver bromide crystals or electronic pulses. At some later point, the data in essential memory will be projected through an essential access in a fundamental activity [MP84].

Object-Oriented system development methodologies are built upon different assumptions of the object model that make up the total system. However, the majority of OO models distinguish between some sort of a conceptual (domain, business) model, and the implementation (design) model [LMS98] [Ree96] [CD01] [Lar02] [RWBW90] [Kru00]. Concept or domain models represent real-world conceptual classes, not software components. Figure 7.2 on the facing page shows how traditionally OO models concretize the universe of discourse, and make a copy of the world in form of a conceptual model. The design model is again a concretization of the conceptual model.

---

\(^1\)Context diagrams derive from the Structured Analysis and design Technique (SADT), developed by Douglas T. Ross and Kenneth E. Schoman [RS77].
According to Wirfs-Brock et.al [RWB90], the main objective in the object-oriented approach is to manage the complexity of the real world by using abstraction. The knowledge of the real world is abstracted and encapsulated in objects. Lars Mathiassen expresses a similar world view model in his OO Analysis and Design methodology [LMS98], where he distinguishes between the user area and the problem area\(^2\).

The problem area is by definition the part of the environment that is administered, monitored or controlled by a computer system. The user area is defined as an organization that administers, monitors or controls the problem area. The object system expresses the users understanding of the problem area [LMS98]. The object system thereby reflects a conceptual model of the universe of discourse.

\[\text{For a large class of cases -- though not for all -- in which we employ the word "meaning" it can be defined thus: the meaning of a word is its use in the language.}\]

\[\text{-- Ludwig Wittgenstein}\]

Else Nordhagen strengthen the conceptual model of the world in the COIR architecture for Flexible Software Components and Systems [Nor95]. The COIR architecture, which pictures objects as having properties which can be laid out in four dimensions [Nor95]. The four dimensions are the Conceptual dimension, the Observable behavior dimension, the Implementation dimension and the Representation dimension. An object exists in all four dimensions, so an object has an observable behavior, an implementation and it has or can be transformed to a binary representation. Also, it has a special meaning, it models a concept within the part of the world the object system

\(^2\)Mathiassen’s problem area is analogous with the universe of discourse.
The work of how conceptual ideas are perceived by the human mind can be tracked back to the early work of C. K. Ogden and his Meaning Triangle [OR49]. Objects can be both real and artificial. Real objects are tangible things like a car or a book, while artificial objects are ideas shared by a group of people [MP84]. The Meaning Triangle (later referred to as Ogden’s Triangle) illustrates how people deal with concepts. Ogden’s Triangle as showed in figure 7.3 on the next page is an enhanced version, where one triangle is put on top of the other to illustrate the transformation process from concept to design. If we focus on the lower left triangle, we see a small universe of discourse in form of a car. The right corner of the triangle shows our area of interest as a unique symbol, which represents a real phenomenon in the universe of discourse. The top corner of the triangle represents the conceptual model of the real world. These three corners are the basic interpretation of the transformation process from real life tangible things to a copy of the real world in form of a conceptual data model. Our enhanced model also includes a second triangle that represents the next phase from concept to object. The conceptual model is presented as a unique computer object. Thereby, we have simulated the real life representation of a car as a data object.

The "symbol" corner in the triangle may be interpreted (in a figurative sense) as a combination of Else’s [Nor95] implementation and representation dimension, and the "conceptual" top corner may be interpreted as the conceptual dimension.

7.1.2 The Holistic Model

The ROOP model of the world is based on a different interpretation of the universe of discourse. Instead of concretizing and making a copy of the world, ROOP concretizes the universe of discourse in a closed object world. The ROOP model is a step in the direction of realizing a world of real life tangible things and computerized parts, in the objects that make up the total system.

With the assistance of the computer, we may enhance the behavior of an object far beyond the capabilities of its real world part. We may even bring "life" to otherwise dead objects.

– Gerhard Skagestein

Figure 7.4 on page 88 shows how ROOP capture the universe of discourse in a closed system, where the Real World represent real life things and the
Figure 7.3: Enhanced Version of Ogden’s Triangle
Enhancement represent computerized parts in the system. The objects in this system are spread around, and we may even find objects that are in the communication point between the two worlds. We must abstract our traditional way of thinking to design models on a higher level of abstraction.

Figure 7.4: ROOP’s Holistic World View

The control system in an aircraft is a perfect example of illustrating the balance between real life flying and simulation of the same task. Figure 7.5 on the next page shows two instances of the :Aircraft object. The first object to the left reflects a real aircraft, where a person may operate the plane through a computer user interface ("fly by wire").

In aircraft simulation, we find a significant amount of more computerized responsibilities. In figure 7.5 on the facing page this is illustrated by moving the boundary to the left. In this scenario, more stimulus messages are generated by the computerized part. Simulating real time flying requires a lot more computer-controlled data. For example, latitude and altitude are no longer measured as in the real world; the simulator simulates the behavior of real-time flying. The simulator has less real world parts (for example seats, motor, wings etc.) compared to the real aircraft.
7.2 System Boundary Perspective

The limits of my language are the limits of my mind. All I know is what I have words for.

– Ludwig Wittgenstein

Deciding upon what part of the environment that should be included in the system model, questions when an object belongs to the total system or not. Somewhere or somehow, we must draw the line between the world we are interested in (the Universe of Discourse) and the environment beyond that world.

We must conduct ourselves to two boundaries in the ROOP methodology. The first boundary is between the world we regard as part of the total system and the world that is beyond the universe of discourse. Secondly, there is the boundary that may split the objects in two. In this section, we will focus on what determines the boundary between what we choose to regard as a whole and everything beyond that world.

Mathiassen [LMS98] uses rich pictures as a tool for determining the world we want to design. Rich pictures are a collection of drawings that make up a good understanding of the total organization. The drawings should be on relatively high abstraction level, so that there is room for different interpretations of the same organization that we study.

Rich pictures are effective in illustrating what needs to be done, yet simple in form. The technique could be adopted as part of the ROOP methodology.
The drawings would maybe make it easier for ROOP to capture the objects in its closed environment.

Another approach described by Trygve Reenskaug [Ree96] is the OOram Role model for abstracting from the real world. A role model is part of a structure of objects that we choose to regard as a whole, separated from the rest of the structure during some period of consideration. A whole that we choose to consider as a collection of roles, each role being characterized by attributes and by actions which may involve it selves and other roles.

Through role modeling you generalize object identity, representing patterns of interacting objects performing the activities by a similar, archetypical pattern of roles performing these activities [Ree96].

For other examples in how to determine the need for information in computer systems, see Gerhard Skagestein’s book "Computer Oriented System Development" (p. 29-30) [Ska91].

### 7.3 From Open to Closed Worlds

We have so far discussed how OO methodologies realize an OO model of the world and methods for concretizing what we want to observe in the universe of discourse. However, what triggers an object to do something and perhaps send a message? Is the initial message stimulated from the environment or are the objects responsible for triggering messages themselves?

In most OO systems, a message received by an object triggers a method, and this method may send further messages. The avalanche of messages flowing between the objects must start somewhere. Some objects will spontaneously send a message without first having received one. We call the initial message a stimulus message and the resulting sequence of actions is called an activity. The object that sends a stimulus message must clearly be in the system environment [wPL01]. This way of describing how the environment interacts with the computer system is typical for open systems. They will be installed in an environment that will influence the system and be influenced by it. The environment can be a human organization as is the case for business information systems, or it can be some equipment as is the case for embedded systems [wPL01]. The NextGen POS system from a UP perspective is an example of an open system, where the cashier (environment) interacts with the computerized system.

Gerhard Skagestein and Else Nordhagen [NS96] discuss further the message stimulus theme in form of interface objects. An interface object is defined
From Open to Closed Worlds

as an object that is activated from stimulus messages outside of the system we observe [NS96]. Figure 7.6 shows an object diagram of how the interface objects are placed in the cross-over boundary between the environment and the computer system (the figure is partly adopted from [NS96]).

![Figure 7.6: Interface Objects in an Open System](image)

The opposite to open systems are closed systems. In closed systems, the stimulus messages are part of the total system we are looking into. There are no unknown sources in the universe of discourse. We recognize that cause and effect are not linear, and that often the end (effect) can influence the means (cause) [MC00]. In closed systems we choose to realize a part of the world, nothing beyond our area of interest is taken in consideration. The ROOP object model is an example of a closed system, where the system include parts of the environment of the open systems. Figure 7.7 on the following page shows the enhanced ROOP world in a closed environment. Hence, ROOP realizes the total system as a concretization of the universe of discourse.

Gerhard Skagestein and Else Nordhagen’s description of the interface objects and how they are seen as being in the communication point between the environment and the information system is actually the background for developing the ROOP methodology. In later work by Gerhard Skagestein [Ska01], he questions how interface objects (as shown in figure 7.6) are treated. Skagestein wanted to include the unknown stimulus message as part of the object. This work has further developed into what we have defined as ROOP objects in this thesis, where one part of the object reflects the real world and the other part reflects the computerized system.

However, even if we know the source of the stimulus message, there is still a question of how the real part of the object interacts with the computerized part of the object, and what triggers an object to send a stimulus message.
In the ROOP methodology, we have based much of the development process on defining the main functionality of the system. Further, the main functionality is built upon satisfying a need, i.e. to satisfy the customer’s need to buy products. If we enhanced our model by including object (computerized) satisfaction, then we would have a self-served system that never ended. The product object (in the NextGen POS) could for example have been given the responsibility (functionality) of self-ordering when products are out of stock.

Forrester’s work on Systems Dynamics uses concepts from the field of feedback control to organize information into a computer simulation model. A computer acts out the roles of people in the real system. The resulting simulation reveals behavioral implications of the system represented by the model [For98]. The model of the world is thereby seen as a closed system in which each action is based on current conditions, such actions affect conditions, and the changed conditions become the basis for future action [For94]. Forrester’s closed-loop structure of the world is in many aspects similar to the ROOP model of the world, where objects may send stimulus messages based upon changed conditions. Figure 7.8 on the facing page illustrates relevant knowledge of such an object. If the state (depicted by the arrow) of the object reaches the high or the low level, then a series of actions are taken to normalize to the desired level.

The world we create relies on endogenous model (object) behavior in conformity with a mechanical world view of the system. This applies to both the ROOP methodology and systems dynamics thinking. It could be said
that a mechanical world view puts rather extreme demands on a model for generating within itself the behavior modes of interest. That is, the model boundary is to be established so that the mechanisms lie inside the boundary. The expectation of finding endogenous causes of behavior is in sharp contrast to the view often found elsewhere [Ric91].

When using the ROOP methodology it is important to be aware of the world we create. Mechanical world view models describe the world objectively and objects act rationally. Rules are necessary to control objects. When we know the rules, or perhaps better the laws, governing the functions of a machine, then we can control it [hj03].

The point is that modeling the real world is a rather complex task. Even if we simulate real life behavior as part of our model, there is still a question of the unknown that lies beyond our universe of discourse. We simply choose to disregard the unknown.

*In this world there is always danger for those who are afraid of it.*

– George Bernard Shaw
Chapter 8

Conclusion and Further Research

In this chapter we will summarize the results and contributions of this thesis\textsuperscript{1}. The main objective has been to develop an alternative OO system development process (ROOP) to investigate the research question:

"Does it make sense to postpone the drawing of the system boundary in an object-oriented system development process?"

We have compared our results with the Unified Process, and stated three assumptions about the ROOP methodology. We will discuss whether these assumptions hold in this chapter.

In the final part of this chapter we will present some suggestions for further research, which may be seen as a continuation of the work in this thesis.

8.1 Results

The results of this thesis have shown that it does make sense to postpone the drawing of the system boundary. Through analysis of how the Unified Process (UP) is applied to the NextGen POS system, we have proven that starting the design with use cases may lead to a fixed system boundary early in the development process. Further, the results have shown that the distribution of knowledge and responsibility between the classes in the design class diagram turn out to be essentially different between UP and ROOP. ROOP’s design class diagram consists of fewer and more enhanced classes than those of the UP.

\textsuperscript{1}The results are partly based on the Rena experiment. See appendix A on page 101 for details.
8.1.1 Assumption [H1]

The ROOP methodology will lead to a model on a relatively high abstraction level, where there is no distinction between the real part and the computerized part of the object.

The work within this thesis has given us strong reasons to believe that ROOP OO models tend to be on a higher abstraction level than their UP counterparts. This is further confirmed by the Rena experiment. Hence, there are strong indications that assumption [H1] holds.

By postponing the distribution of knowledge and responsibility between real world tasks and the computerized system, we have been able to reach a higher level of abstraction. How objects are perceived in ROOP confirms our assumption; the objects inside our system should not be regarded as a copy of an OO model of the "real world", but as computerized enhancements to objects in the real world.

8.1.2 Assumption [H2]

The objects may be split into a real world part and a computerized part in a rational way by using the ROOP methodology.

With the use of use cases together with the activity diagram and the activity realization diagrams, we have only partly been able to split the objects in a rational way. Hence, we have not confirmed assumption [H2]. There are still some questions related to where and how to draw the boundary that split the objects. For example, this thesis has not provided adequate evidence of the factors that influence where to draw the boundary. This thesis describes only to some extent how different factors influence the decisions we make. These factors may be the software requirements specification, different solutions (technological and social aspects), technical limitations, costs, delivery time, user interface design and usability engineering etc.

The Rena experiment also unveiled this weakness. However, The ROOP methodology was not fully developed when we carried out the experiment. Nevertheless, more research is necessary to provide more exact principles about how to split the objects in a rational way.

8.1.3 Assumption [H3]

If the specification of the problem to solve is changed, there is a high likelihood that the model will not have to be changed. Hence, reuse of models is easier with the use of the ROOP methodology.
In the Rena experiment, we changed the specification of the problem to solve to find out if the model had to be changed. The results showed that the students could use their models and accommodate the changes without doing any major reengineering.

Further, the work behind this thesis has indirectly indicated that changing the distribution of knowledge and responsibility between real world tasks and the computerized system, is easily carried out by rolling back to the collaboration phase in the process. This is shown in ROOP’s enhanced version of the CRC card approach, where we focused on of how messages are sent between objects, regardless of whether the messages was stimulated from outside or inside the computerized system. Hence, there are strong indications that assumption [H3] holds for the model designed in this thesis, and for the models designed within the Rena experiment.

8.2 Further Research

Further research is necessary to see how the ROOP methodology can be applied to other object-oriented projects. Within this thesis, we have only been able to study the ROOP effects in a relatively small scale. Hence, applying ROOP within a larger project would provide us with useful information about how robust the methodology is. First then, we would know more about whether it does make sense to postpone the drawing of the system boundary.

When it comes to reengineering and reuse of other object-oriented models (according to assumption [H3]), this thesis has not provided enough adequate information. Hence, a more comprehensive study in how to reuse models within ROOP would give us a more accurate picture of how robust ROOP models are.

In accordance to assumption [H2], further research is necessary in the field of specifying the cross-over boundary. This would provide us with more knowledge about the consequences of where and how to draw the system boundary. How do these split objects affect the user interface and the programs that are used to realize the objects? This thesis has only touched this subject, and introduced some ideas of how to proceed after we have decided upon the system boundary. However, how to realize the model by means of programming code is not discussed.

As a follow up project, it would be interesting to see if anyone continued working on the ROOP methodology. There are still a lot of questions that needs to be explored. Good luck!
Further Research
Appendix and Bibliography
Further Research
Appendix A

A ROOP Research Paper
Based on the Rena Experiment
30-31 Oct, 2003

This appendix includes the research report from the Rena experiment. The material is meant to be used as a contribution to the work provided in this thesis. The results from the experiment has given us valuable information on how to improve parts of the ROOP methodology.

The ROOP methodology has evolved since we conducted the experiment. Hence, some of the terms and ideas may distinguish from how ROOP is presented in this thesis.
Introduction

The Robust Object-Oriented Process (ROOP) has been under development for the last 16 months. The development of ROOP is the successor activity of Gerhard Skagestein’s article "Are Use Cases Necessarily the Best Start of an OO System Development Process?" [Skagestein 2001]. His thoughts on how to start a development process have been the precursor for Bjørn Henrik Pedersen’s Cand. Scient. thesis about the ROOP methodology.

In August 2003 we discussed different approaches to test the methodology in practice. We wanted to know how new users understood the methodology, and whether or not they were able to find (and use) key successor factors in ROOP. These key factors were:

- The ability to filter out higher abstract objects.
- To understand how use case diagrams can influence the system boundary decision.
- To delay the system boundary decision.
- To understand that organizational changes trigger changes in the object model.
- To treat objects as split objects.
- To give the objects more responsibility, in response to later reuse of the model.
- To see how the ROOP methodology worked as an incremental and iterative development process.

Under the preliminary work of organizing the test we came in touch with the COOL (Comprehensive Object-Oriented Learning) project in Norway. They thought the idea was an interesting contribution to a model-first approach to learning and teaching object orientation. After we had worked out the details in how to proceed with the experiment, we decided to test the methodology on first year students at the Hedmark University College, Faculty of Computer Science, Rena. The following people participated in the project execution plan:

- **Administration:**
  - Bjorn Henrik Pedersen, University of Oslo
  - Gerhard Skagestein, University of Oslo
  - Håvard Hegna, Cool Project
Participants (Group 1):
– Bjørn-Ivar Norseth, Hedmark University College
– Frode Hellerud, Hedmark University College
– Martin Ø. Fevik, Hedmark University College

Participants (Group 2):
– Marius Filtvedt, Hedmark University College
– Morgan Branes, Hedmark University College
– Steinar Sikeløkken, Hedmark University College
– Stig R. Nygården, Hedmark University College

The experiment was carried out the 30th and 31st of October 2003 at the Hedmark University College, Rena.

Assumptions

By means of the experiment, we wanted to examine the following assumptions:

H1 The ROOP methodology will lead to a model on a relatively high abstraction level.

H2 When the specification of the problem to solve is changed, there is a high likelihood that the model will not have to be changed. Hence, reuse of models is easier with the use of the ROOP methodology.

H3 The objects will be split into a real world part and a computerized part in a rational way by using the ROOP methodology.

Research Method

During the experiment, we made use of the following research methodologies:

– Active learning / Interactive engagement

Interactive Engagement (IE) methods are designed in part to promote conceptual understanding through interactive engagement of students in hands-on (always) and hands-on (usually) activities which yield immediate feedback through discussion with peers and/or instructors. Students are not passive
recipients of knowledge, but are engaged learners; and instructors are not seen as founts of information, but function more as mentors or coaches.

The researcher work together with the group to be observed, in order to achieve a first hand understanding of what is happening. The researcher may take on different roles, like group leader, consultant, subject expert, normal group member and so on.

- Observation

Observation gives the researchers an important view of the situation. Different aspects of how a methodology works and how the participants work together are gathered during the observation process.

- Study of written material from the groups

Written material produced by the group is analyzed and checked against the expected or intended output from the methodology in use. Written material gives the researchers the possibility to analyze the progress and answers in more detail.

**Carrying Out the Experiment**

Before the start of the experiment, the students were given a two-hour lecture in the ROOP methodology. The lecture notes were handed out prior to the lecture. The students were also given a compendium that they could use as a reference while solving the given tasks. (See appendix).

The participants were split into two groups, which were given the same tasks to solve. More groups would have given more well-founded conclusions, but since the participation was voluntary, we could only get 7 participants split up in two groups.

The experiment was carried out over two days. On the first day the students were given a requirement specification of a Hotel reservation system (see appendix) with the request to build an object-oriented model using the ROOP methodology. On the second day some changes to this specification were introduced (see appendix) with the request to work those changes into the model from the previous day. The time assigned to the tasks was approximately two hours each day. We felt that this would give the students sufficient time to show that they were able to use the methodology in the anticipated way. During the working process we observed how the student groups worked together. Occasionally we intervened with the students to help them on the right track. We told the students in advance to solve the
given tasks in a specific order by using the ROOP methodology. In other words, the students were not allowed to carry out the experiment by using other well-known methodologies, like for instance Rational Unified Process (RUP). With the use of interactive engagement we took control of the sequence of accomplishing the tasks using the ROOP methodology.

During the process of solving the given tasks, we saw signs of misinterpretation of the ROOP methodology. The misinterpretation was partly caused by how the students understood the lecture prior to the experiment and partly caused by the maturation period between the lecture and carrying out the experiment. The problem was that the ROOP methodology introduced a completely new way of designing object-oriented systems. Active learning during the experiment helped the students to give a better understanding of the ROOP methodology.

They started the process by finding object candidates, and by using special designed patterns they singled out the objects they wanted to continue working with. With the use of CRC (Class, Responsibility and Collaborator) cards, they assigned knowledge and responsibility to the objects and decided on how they should collaborate. The students then discussed how to split the objects in a computerized part and a real world part. At the end of the day, the written material produced by the groups was handed in for analysis.

On the second day the students were given a modified requirement specification of the Hotel reservation system. The two groups continued working with the same material they handed in the day before. The groups first had to verify whether the model from the day before could accommodate the changes, or whether some changes or additions were necessary. Then, a new discussion on where to draw the boundary between the computerized part and the real part of the objects was necessary.

After conducting the experiment, all written material produced by the groups was analyzed and cross checked with our observation notes to see whether assumptions \([H1]\), \([H2]\) and \([H3]\) had been proven or not.

**Results**

When analyzing the test material, we tried to separate between problems inherent in all object-oriented design and problems originating in the ROOP methodology alone.

The first step of the process was to find object candidates for the hotel reservation system. Both groups were able to use the ROOP patterns to find key
objects. The Customer object was of particular interest, since this object indicated the systems stakeholder. Both groups were able to identify this object. Further, they were able to design the customer object on a higher abstraction level, by giving the object extended responsibilities beyond the system specification. They expressed the same understanding of the Room object, which indicated a good understanding in how objects are perceived in the ROOP methodology. The test indicated that assumption \([H1]\) was proven.

On the second day we introduced some changes to the system. By using the ROOP methodology they were to include these changes in the original model. In general, these changes assumed more high technology solutions. In terms of ROOP, this meant a shift from real world tasks towards tasks handled by the computerized system. Since the groups during the first day had taken into account many of these changes in their hunt for objects on a high level of abstraction, only minor reengineering work had to be done. New objects and responsibilities did not conflict with the object responsibility structure discussed the day before. The test indicated that assumption \([H2]\) was proven.

On both days the groups were to discuss where to draw the boundary between the computerized part and the real part of the objects. Even though they showed a good understanding of how to distribute responsibilities between the parts of the objects, they where stuck on several occasions on where to draw the exact boundary. Some of their frustration may have come from that this part of the method was not explained very well during the presentation on the first day. The methodology as it was outlined in the lecture notes and in the compendium did neither describe in detail how to split the objects in a real part and a computerized part, nor how to document the boundary between them. Hence, the assumption \([H3]\) could not be considered proven. More work has to be done to make the ROOP methodology more specific on this point.

**Summary and Conclusions**

The test gave us an indication of how well the students were able to use the ROOP methodology in practice. Both groups were able to design a model on a relatively high abstraction level in line with assumption \([H1]\). When it came to making changes to the model, in line with assumption \([H2]\), the groups could use their models and accommodate the changes without doing any major reengineering work. With regard to assumption \([H3]\), we could observe that the groups were insecure in how to split the objects and how to document the boundaries, although they both had an overall good under-
standing of the basic principle.

The results indicated the potential of using the ROOP methodology for designing object-oriented models. To some degree, the methodology was convincing in the object selection process. Even though the objects are on a higher abstraction level, the students seemed to get a better understanding of what objects can do. The ROOP methodology truly helped in the process of selecting objects that are capable of doing more than just move data around. Further, having this object perspective naturally filtered out objects that really are nothing but attributes.

It must be said that the results of assumption [H1] and [H2] was confirmed due to a combination of interactive engagement and the student’s ability to use the ROOP methodology.

The methodology has proven to be a success in use by freshman students. Further work on explaining how objects can be split in a rational way should be considered as a major focus on the remaining work of ROOP.

As a follow up research project, it would be interesting to see how more experienced students are capable of adapting to the ROOP methodology.

Appendix

Problem Description Day 1: (in Norwegian)

Dere har fått i oppgave å lage et reservasjonssystem for Måbudalen Hotell. Hotellet har i mange år ført opp alle reservasjoner for hånd, men ønsker nå et modernisert system. Systemet skal brukes av personalet på hotellet. Reservasjon av rom foregår på hotellet eller ved at kundene ringer eller sender inn en forespørsel. Det skal altså ikke være mulig for kundene selv å reserve re (f. eks. over Internett).

Måbudalen hotell er et relativt stort hotell med 95 rom. Det finnes fire forskjellige typer rom på hotellet. Dette er etroms, toroms, treroms og fireroms.

Systemet skal håndtere både individuelle reisende og turistgrupper ved reservasjon av rom. Hotellet ønsker også å holde styr på om det er en tuoperator som bestiller rom for en hel gruppe, eller om det er en individuell reisende som bestiller ett eller flere rom. Informasjon om tuoperator eller gjest som bestiller rom skal lagres i systemet, blant annet med navn, antall personer, antall rom og romtype, samt hva slags type service som blir bestilt (rom/frokost, helpensjon, halvpensjon). Det er videre ønskelig å skrive ut
rapporter/statistikker over totalt antall overnattinger og antall overnattinger med samme nasjonalitet i løpet av en tidsperiode. Det skal også være mulig å ta ut rapporter over reisende etter formål, det vil si hvor mange gjester som var ferierende, forretningsreisende eller overnattet i sammenheng med kurs/konferanse i løpet av en tidsperiode.

Systemet skal til en hver tid inneholde en oppdatert oversikt over belegg til en hver dato, slik at når bruker klikker på ønsket ankomstdato og avreisedato, skal antall ledige og reserverte rom, samt rom på venteliste komme til synet i belegginformasjonen. Når det blir gjort endringer på en dato, ved at en reservasjon blir lagt til eller en reservasjon blir sletta, skal dette automatisert bli registrert og belegginformasjonen blir oppdatert. I tillegg til reservasjonssopplysninger skal også innsjekking og utsjekking registreres i systemet, slik at en til en hver tid har en oversikt over hvem av gjestene som har ankommet hotellet og sjekket ut. I tillegg skal det ved utsjekking komme opp pris for alle rom, samt at det skal kunne legges inn andre priser, som regninger fra restaurant, bar og telefon. Disse skal kunne knyttes til det respektive rom. Fordi det ikke alltid er slik at en gjest betaler ved utsjekking, skal ikke reservasjonen slettes. Systemet skal også kunne produsere purringer til kunder som ikke har betalt for overnattingen innen tidsfristen. Det er også nødvendig at reservasjonen blir lagret til senere for å kunne ta ut rapporter med antall overnattinger o.s.v.

**Problem Description Day 2: (in Norwegian)**

Reservasjonssystemet for Måbudalen hotell har nå vært i bruk i to år. Hotellet har vært svært fornøyd med den første utgaven av systemet. De henvender seg derfor til dere da de ønsker å gjøre noen endringer i reservasjonssystemet.


Hotellet ønsker også å kunne gi mer personlig service til kundene sine. Ved hjelp av ny teknologi sørger nå rommene selv for at alt er på plass i forhold til kundens preferanser. Dette kan være alt fra spesielle varer i minibaren, såpe eller håndkle på badet. Dessuten er systemet i stand til å bestille nye varer (f. eks mer øl i minibaren) hvis det skulle bli tomt.

Hvilke endringer må gjøres i modellen for at det nye systemet skal kunne tas i bruk? Ta utgangspunkt i gårslagens modell og gjør de endringer som er
nødvendig for systemet, ved å bruke ROOP.


Hint 2: Id-kortet kan betraktes å inngå i den datamaskinbaserte delen av :Person objektet.

The ROOP Compendium and Lecture Notes (in Norwegian)

The ROOP (Robust Object-Oriented Process) methodology is a contribution to the ongoing discussion on how to teach and learn object-oriented modeling. The ROOP methodology is still under development.

The basic idea is that you should avoid setting up a system boundary too early in the system development process. However, setting up system boundaries is exactly what you do when you follow well-known system development methods like RUP (Rational Unified Process) that starts with Use case diagrams, because these diagrams must build on an assumption of where the system boundary should go.

Figure A.9 on page 118 shows how functional requirements are described by means of Use cases in RUP and similar methods. In addition to the use case Process Coffee, the diagram shows a direct communication with an Actor outside the system boundary. If the system in addition is interpreted as being the computerized system, then there is a chance that the distribution of work between humans and computers are determined too early in the system development process.

ROOP suggest that functional requirements should be described without stating what triggers an event. This means that actors should be left out of the use case diagrams! Figure A.10 on page 118 shows the main functionality without any form of communication with the outside world. From this starting point we have a lot of possibilities designing the objects that carry out the tasks for fulfilling the event Process Coffee.

Finding usable objects can be a difficult task. Since the system boundary has not been determined, ROOP opens up for a more comprehensive solution. According to ROOP, the focus should initially be on what objects you need and how they should communicate, independent of whether they are outside or inside the computerized system, or shared between the computerized system and its environment. ROOP, in the initial phase, does not distinguish between actors outside the system and objects inside. User interface and
Appendix A. A ROOP Research Paper
Based on the Rena Experiment 30-31 Oct, 2003

Figure A.1: Rena Foils (1-6)
Figure A.2: Rena Foils (7-12)
Figure A.3: Rena Foils (13-18)
Figure A.4: Rena Foils (19-24)
APPENDIX A. A ROOP RESEARCH PAPER
BASED ON THE RENA EXPERIMENT 30-31 OCT, 2003

![Diagram](image_url)

**Figure A.5:** Rena Foils (25-30)
Figure A.6: Rena Foils (31-36)
Figure A.7: Rena Foils (37-42)
Figure A.8: Rena Foils (43-48)
Figure A.9: Use Case Diagram

Figure A.10: Main Functionality
control objects are left out of consideration. This way of thinking yields high abstraction level models that are easy to grasp and robust against technological and organizational changes.

Let us look further into the example. An object model of the Process Coffee functionality would look something like figure A.11. The model consists of only four objects. Each object has the potential to trigger an event, simply because we have not determined the system boundary yet. The dotted line indicates a movable system boundary, which can split the object in two, leave the object outside of the boundary or leave the object inside of the boundary.

What determines the system boundary is the distribution of responsibilities between the computerized system and its environment. Drawing the system boundary right through an object means that the object will be split into an object part belonging to the real world and an object part belonging to the computerized system, and only then can use cases be considered as a convenient tool for describing communication between the object parts.

By postponing the decision of where we draw the system boundary, we believe that we get a more flexible and robust object architecture. If later development of the total system causes a shift in the system boundary, the high abstraction level model will hopefully still stand. Conventional models will probably have to be reworked, simply because there will be too many interface objects and data mover objects that must be redesigned.
A Guide to the ROOP Methodology

1. - Main Functionality

Read through the system specification and start by identifying the systems main functionality.

Example: If you are going to design a coffee machine, the systems main functionality would be to *satisfy the customer’s coffee thirst*.

2. - Candidate List

Find object candidates from the system specification. The list of objects should reflect the systems main functionality. Try to list up all possible candidates.

3. - Patterns

The ROOP methodology take in use special designed patterns, which helps us select the objects we want to continue working with. These patterns are: No Data Movers, Stakeholders and Encapsulation.

<table>
<thead>
<tr>
<th>Name: No Data Movers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem:</strong> In many systems, some objects have no other responsibility than moving data between objects. These objects are of no interest in ROOP. Data movers contribute to larger models, and should therefore be avoided.</td>
</tr>
<tr>
<td><strong>Solution:</strong> Find the objects which are inside our area of interest and do something more than moving data between the objects</td>
</tr>
<tr>
<td><strong>Example:</strong> A “clerk” is a typical data mover. Other data mover objects are typical static objects like “coin slot” and “cup holder” in a coffee machine.</td>
</tr>
<tr>
<td><strong>Strengths:</strong> It is easier to distribute responsibilities between the objects, since there are fewer objects to deal with.</td>
</tr>
<tr>
<td><strong>Weaknesses:</strong> Exaggerated use of the pattern can lead to too much responsibility on one or more objects as a result of fewer objects.</td>
</tr>
</tbody>
</table>

Table A.1: No Data Movers

4. - Object List

We now have the objects that we want to continue working with. List the objects.

- Object1
**Name:** Stakeholders

**Problem:** A system may affect many people, machines or other systems. Most of these affected parties are only data movers that act on behalf of the genuine stakeholders.

**Solution:** Include only the stakeholder (or stakeholders) the system is intended for.

**Example:** In a car rental system the only stakeholder of interest is the customer. The clerk who is operating the system is nothing but a data mover.

**Strengths:** The pattern defines the stakeholder (or stakeholders) which do not change over time.

**Weaknesses:** Some stakeholders can act as data movers as well as being the genuine stakeholder. In some situations this could lead to problems finding the true stakeholders.

<table>
<thead>
<tr>
<th>Table A.2: Stakeholders</th>
</tr>
</thead>
</table>

**Name:** Encapsulation (smart objects)

**Problem:** Expect changes in parts of the system, which will affect how the user interface is treated. We want that these changes do not force other changes to the rest of the system.

**Solution:** Create classes with stable definitions and encapsulates the information. Later changes will then only affect the object alone.

**Example:** In the person and loan example, it is important that the person object is given responsibilities that reflect future changes to the system. The person should for example be given the responsibility to askForPayment. All objects are considered as smart objects.

**Strengths:** It is easy to enhance the encapsulated objects.

**Weaknesses:** Exaggerated use of the pattern can lead to too much responsibility on one or more objects.

| Table A.3: Encapsulation |
5. - CRC Cards and the Object Think Approach

Assign one card for each object. Distribute the cards around the table and play the game – i.e. send messages to each other. Assign knowledge and responsibilities as you go along. Use the object think approach as a tool for assigning knowledge and responsibilities. For example if one person plays the role of Per (an instance of the class Person), he should say:

- I am Per
- I know what my name is
- I am a smart object
- It is my responsibility to ask for payment on the loan

<table>
<thead>
<tr>
<th>Object: Class</th>
<th>Knowledge:</th>
<th>Responsibility:</th>
<th>Collaborator:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.4: CRC Card

6. - Split the Objects and Decide upon the System Boundary

Split the objects and discuss different solutions before making the final decision. The system specification gives you an idea on how automated the system is going to be.
7. - Iterative Development

Go through the model and check if there is necessary to do any changes.

8. - Adding Objects to the Model

If necessary, add objects to the system design. In some cases the system we make are operated by what we saw as data movers. If so, add the objects to the system. One example would be the clerk in the car rental system. Remember that the responsibilities of the customer do not change even if we add the clerk object. Ask for payment is still the customer’s responsibility, but it is triggered from the clerk.

9. - Attributes, Methods and Associations

Use collaboration diagrams as a tool for how objects send messages. Assign attributes, operations and association as you work with the diagrams. Use the information you have from the CRC cards.

![Collaboration Diagram](image)

Figure A.13: Collaboration

10. - Connect the User Interface and the Domain Layer

Draw a prototype of the user interface and assign the interface to the domain layer. You do not have to think about the interface layer for now.

11. - Class Diagram

Put the parts together and draw the class diagram.
Figure A.14: User Interface and Domain Layer

Figure A.15: Class Diagram
References

Appendix B

Glossary

The glossary presented here is adopted from Craig Larman’s book "Applying UML and Patterns" [Lar02]. In addition, ROOP expressions are added to the list.

**abstract class** A class that can be used only as a superclass of some other class; no objects of an abstract class may be created except as instances of subclasses

**abstraction** The act of concentrating the essential or general qualities of similar things. Also, the resulting essential characteristics of a thing

**actor** Anyone or anything with behavior. [Coc00]

**analysis** An investigation of a domain that results in models describing its static and dynamic characteristics. It emphasizes questions of "what," rather than "how."

**architecture** Informally, a description of the organization, motivation, and structure of a system. Many different levels of architectures are involved in developing software systems, from physical hardware architecture to the logical architecture of an application framework.

**association** A description of a related set of links between objects of two classes.

**attribute** A named characteristic or property of a class.

**business use case** A business use case discusses how a business respond to a customer or an event [wKS00].

**class** In the UML, "The descriptor of a set of objects that share the same attributes, operations, methods, relationships, and behavior" [IJR99a].

**collaboration** Two or more objects that participate in a client/server relationship in order to provide a service.
communication point  A communication point describes a user interface in the cross-over boundary.

composition  The definition of a class in which each instance is comprised of other objects.

computerized part  The part of a ROOP object that is automated.

concept  A category of ideas or things. In this book [Lar02], used to designate real-world things rather than software entities. A concept’s intension is a description of its attributes, operations and semantics. A concept’s extension is the set of instances or example objects that are members of the concept. Often defined as a synonym for domain class.

contract  Defines the responsibilities and postconditions that apply to the use of an operation or method. Also used to refer to the set of all conditions related to an interface.

core objects  Core objects is the list of objects that you have after evaluating the items in the candidate list. ROOP uses special designed patterns to help us in the process of selecting core objects. The term "core" is adopted from "The CRC Card Book" [BS99]. The core objects make up the core system architecture in ROOP.

coupling  A dependency between elements (such as classes, packages, subsystems), typically resulting from collaboration between the elements to provide a service.

crc card  Class, Responsibility and Collaboration (CRC) cards are index cards, which are used to record suggested objects (classes), the things they do, their responsibilities, and their relationship to other objects (classes).

cross-over boundary  A cross-over boundary is the boundary that split an object in two. One part belonging to the real world, and one part belonging to the computerized system.

design  A process that uses the products of analysis to produce a specification for implementing a system. A logical description of how a system will work.

dia  Dia [ha] is a gtk+ based diagram creation program released under the GPL license. All figures in this thesis are drawn in Dia.

domain  A formal boundary that defines a particular subject or area of interest.
**encapsulation** A mechanism used to hide the data, internal structure, and implementation details of some element, such as an object or subsystem. All interaction with an object is through a public interface of operations.

**essential use case** An essential use case is a structured narrative, expressed in the language of the application domain and of users, comprising a simplified, generalized, abstract, technology-free and implementation independent description of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction [CL99].

**event** A noteworthy occurrence.

**extension** The set of objects to which a concept applies. The objects in the extension are the examples or instances of the concept.

**framework** A set of collaborating abstract and concrete classes that may be used as a template to solve a related family of problems. It is usually extended via subclassing for application specific behavior.

**generalization** The activity of identifying communality among concepts and defining a superclass (general concept) and subclass (specialized concept) relationships. It is a way to construct taxonomic classifications among concepts which are then illustrated in class hierarchies. Conceptual subclasses conform to conceptual superclasses in terms of intension and extension.

**holistic world view** In ROOP, a holistic world view is a model of the universe of discourse. Such models lead to a mechanical world view, i.e. systems that are built upon a structure of interacting feedback loops.

**inheritance** A feature of object-oriented programming languages by which classes may be specialized from more general superclasses. Attributes and method definitions from superclasses are automatically acquired by the subclass.

**instance** An individual member of a class. In the UML, called an object.

**intension** The definition of a concept.

**interface** A set of signatures of public operations.

**latex** \LaTeX \cite{Lam94} is a typesetting system which is very suitable for producing scientific and mathematical documents of high typographical quality. This thesis is written in \LaTeX.
main functionality The main functionality reflects the system intension. Ask yourselves the following: Who or what do (shall) the system serve. The main functionality is the starting point for the ROOP methodology. (Note that there may be more than on main functionality in a system).

mechanical world view See description of holistic world view.

message The mechanism by which objects communicate; usually a request to execute a method.

metamodel A model that defines other models. The UML metamodel defines the element types of the UML, such as Classifier.

method In the UML, the specific implementation or algorithm of an operation for a class. Informally, the software procedure that can be executed in response to a message.

model A description of static and/or dynamic characteristics of a subject area, portrayed through a number of views (usually diagrammatic or textual).

multiplicity The number of objects permitted to participate in an association.

object In the UML, a instance of a class that encapsulates state and behavior. More informally, an example of a thing.

object identity The feature that the existence of an object is independent of any values associated with the object.

object-oriented analysis The investigation of a problem domain or system in terms of domain concepts, such as conceptual classes, associations, and state changes.

object-oriented design The specification of a logical software solution in terms of software objects, such as their classes, attributes, methods, and collaborations.

object-oriented programming language A programming language that supports the concepts of encapsulation, inheritance, and polymorphism.

operation In the UML, "a specification of transformation or query that an object may be called to execute" [IJR99a]. An operation has a signature, specified by its name and parameters, and it is invoked via a message. A method is an implementation of an operation with a specific algorithm.
pattern A pattern is a named description of a problem, solution, when to
apply the solution, and how to apply the solution in new context.

persistence The enduring storage of the state of an object.

persistent object An object that can survive the process or thread that
created it. A persistent object exists until it is explicitly deleted.

polymorphism The concept that two or more classes of objects can re-
spond to the same message in different ways, using polymorphic oper-
ations. Also, the ability to define polymorphic operations.

real world part The part of a ROOP object that is real (not automated).

responsibility A knowing or doing service or group of services provided by
an element (such as a class or subsystem); a responsibility embodies
one or more of the purposes or obligations of an element.

role A named end of an association to indicate its purpose.

roop The Robust Object-Oriented Process (ROOP) is a software engineer-
ing framework for how to start an OO system development process.
ROOP is developed by the author of this thesis [Ped04]. Further, the
ROOP framework is strongly influenced by Gerhard Skagestein’s ideas
of how to start an OO system development process [Ska01].

roop object ROOP goes further in its definition of objects. The object
can represent the real world, or represent the computerized world, or
represent both worlds. Although, in ROOP’s initial phase, all objects
may represent both real world features and computerized world en-
hancements of these features.

rup The Rational Unified Process is a commercial variant of the Unified
Process. UP and RUP are very closely related [AN02].

split object A description of a ROOP object, where the object is split into
a computerized part and a real world part.

stakeholder Someone or something with vested interest in the behavior of
the system under discussion (SuD) [Coc00].

state The condition of an object between events.

subclass A specialization of another class (the superclass). A subclass in-
herits the attributes and methods of the superclass.

subtype A conceptual superclass. A specialization of another type (the
supertype) that conforms to the intension and extension of the super-
type.
superclass A class from which another class inherits attributes and methods.

supertype A conceptual superclass. In a generalization-specialization relation, the more general type; an object that has subtypes.

system boundary The system boundary is usually chosen to be the software (and possibly hardware) system itself [Lar02]. In ROOP, there are two boundaries. Firstly, you have the boundary that captures the universe of discourse (holistic world view). Secondly, you have the boundary that may split the objects in two. However, the term system boundary is used (in ROOP) analogous with the boundary that may split the objects in two. It is easier to map the meaning of the word to UP, by using the term this way.

system sequence process A system sequence process describes input and output events related to the main functionality under discussion. A system sequence process must have a start and stop process.

system use case A system use case is an interaction between a user and the system [wKS00].

total system The total system includes everything (that we choose to study) in the universe of discourse.

transition A relationship between states that is traversed if the specified event occurs and the guard condition met.

uml The Unified Modeling Language (UML) is a general purpose visual modeling language for systems.

unified process The Unified Process (UP) is an industry standard software engineering process from the authors of UML [IJR99b].

universe of discourse A description of the world we choose to study. The universe of discourse may consist of real life elements and computerized parts.

use case A use case specifies a sequence of actions, including variants, that the system can perform and that yields an observable result of value to a particular actor [IJR99b].

use case diagram A use case diagram describes part of the use case model and shows a set of use cases and actors with an association between each interacting pair of actor and use case [IJR99b].

user interface A user interface is the part of the system with which the users interact. It includes the screen displays that provide navigation
through the system, the screens and forms that capture data, and the reports that the system produces (whether on paper, on the screen, or via some other media) [DW00].

**visibility** The ability to see or have reference to an object.
Bibliography


[DOE01] Devesh Bhatt David Oglesby, Kirk Schloegel and Eric Engstrom, *A pattern-based framework to address abstraction, reuse, and cross-domain aspects in domain specific visual languages*.


BIBLIOGRAPHY

[ha] http://www.lysator.liu.se/ alla/dia/, Dia is a gtk+ based diagram creation program released under the gpl license.


BIBLIOGRAPHY


[Meh02] Shveta Mehtani, Lets connect the dots..., Lecture hold at O’Shaughnessy Education Center, University of St Thomas (2002).


BIBLIOGRAPHY


[Ska] Gerhard Skagestein, *Conversations with gerhard skagestein in the period from september 2002 to february 2004*.


