Reverse Engineering of Object Oriented Code

Monographs in Computer Science

Abadi and Cardelli, **A Theory of Objects**

Benosman and Kang [editors], **Panoramic Vision: Sensors, Theory and Applications**

Broy and Stølen, **Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement**

Brzozowski and Seger, **Asynchronous Circuits**

Cantone, Omodeo, and Policriti, **Set Theory for Computing: From Decision Procedures to Declarative Programming with Sets**

Castillo, Gutiérrez, and Hadi, Expert **Systems and Probabilistic Network Models**

Downey and Fellows, **Parameterized Complexity**

Feijen and van Gasteren, **On a Method of Multiprogramming**

Herbert and Spärck Jones [editors], **Computer Systems: Theory, Technology, and Applications**

Leiss, **Language Equations**

McIver and Morgan [editors], **Programming Methodology**

McIver and Morgan, **Abstraction, Refinement and Proof for Probabilistic Systems**

Misra, **A Discipline of Multiprogramming: Program Theory for Distributed Applications**

Nielson [editor], **ML with Concurrency**

Paton [editor], **Active Rules in Database Systems**

Selig, **Geometric Fundamentals of Robotics, Second Edition**

Tonella and Potrich, **Reverse Engineering of Object Oriented Code**

Paolo Tonella
Alessandra Potrich

Reverse Engineering of Object Oriented Code

Springer

Paolo Tonella and Alessandra Potrich
ITC-irst
Via Sommarive
Povo, Trent 38050
ITALY

Cover illustration: Verona-climbing the tower. Photo courtesy Philip Greenspun, http://philip.greenspun.com.

ISBN 0-387-40295-0 e-ISBN 0-387- 23803-4 Printed on acid-free paper.

Printed in the United States of America. (BS/DH)

9 8 7 6 5 4 3 2 1 SPIN 10935804

springeronline.com

To Silvia and Chiara
– Paolo

To Bruno
– Alessandra

Contents

Foreword

There has been an ongoing debate on how best to document a software system ever since the first software system was built. Some would have us writing natural language descriptions, some would have us prepare formal specifications, others would have us producing design documents and others would want us to describe the software thru test cases. There are even those who would have us do all four, writing natural language documents, writing formal specifications, producing standard design documents and producing interpretable test cases all in addition to developing and maintaining the code. The problem with this is that whatever is produced in the way of documentation becomes in a short time useless, unless it is maintained parallel to the code. Maintaining alternate views of complex systems becomes very expensive and highly error prone. The views tend to drift apart and become inconsistent.

The authors of this book provide a simple solution to this perennial problem. Only the source code is maintained and evolved. All of the other information required on the system is taken from the source code. This entails generating a complete set of UML diagrams from the source. In this way, the design documentation will always reflect the real system as it is and not the way the system should be from the viewpoint of the documentor. There can be no inconsistency between design and implementation. The method used is that of reverse engineering, the target of the method is object oriented code in C++, C#, or Java. From the code class diagrams, object diagrams, interaction diagrams and state diagrams are generated in accordance with the latest UML standard. Since the method is automated, there are no additional costs. Design documentation is provided at the click of a button.

This approach, the result of many years of research and development, will have a profound impact upon the way IT-systems are documented. Besides the source code itself, only one other view of the system needs to be developed and maintained, that is the user view in the form of a domain specific language. Each application domain will have to come up with it's own language to describe applications from the view point of the user. These languages may range from natural languages to set theory to formal mathematical notations.

What these languages will not describe is how the system is or should be constructed. This is the purpose of UML as a modeling language. The techniques described in this book demonstrate that this design documentation can and should be extracted from the code, since this is the cheapest and most reliable means of achieving this end. There may be some UML documents produced on the way to the code, but since complex IT systems are almost always developed by trial and error, these documents will only have a transitive nature. The moment the code exists they are both obsolete and superfluous. From then on, the same documents can be produced cheaper and better from the code itself. This approach coincides with and supports the practice of extreme programming.

Of course there are several drawbacks, as some types of information are not captured in the code and, therefore, reverse engineering cannot capture them. An example is that there still needs to be a test oracle – something to test against. This something is the domain specific specification from which the application-oriented test cases are derived. The technical test cases can be derived from the generated UML diagrams. In this way, the system as implemented will be verified against the system as specified. Without the UML diagrams, extracted from the code, there would be no adequate basis of comparison.

For these and other reasons, this book is highly recommendable to all who are developing and maintaining Object-Oriented software systems. They should be aware of the possibilities and limitations of automated post documentation. It will become increasing significant in the years to come, as the current generation of OO-systems become the legacy systems of the future. The implementation knowledge they encompass will most likely be only in the source and there will be no other means of regaining it other than through reverse engineering.

Trento, Italy, July 2004 *Harry Sneed*
Benevento, Italy, July 2004 *Aniello Cimitile*

Preface

Diagrams representing the organization and behavior of an Object Oriented software system can help developers comprehend it and evaluate the impact of a modification. However, such diagrams are often unavailable or inconsistent with the code. Their extraction from the code is thus an appealing option. This book represents the state of the art of the research in Object Oriented code analysis for reverse engineering. It describes the algorithms involved in the recovery of several alternative views from the code and some of the techniques that can be adopted for their visualization.

During software evolution, availability of high level descriptions is extremely desirable, in support to program understanding and to change-impact analysis. In fact, location of a change to be implemented can be guided by high level views. The dependences among entities in such views indicate the proportion of the ripple effects.

However, it is often the case that diagrams available during software evolution are not consistent with the code, or – even more frequently – that no diagram has altogether been produced. In such contexts, it is crucial to be able to reverse engineer design diagrams directly from the code. Reverse engineered diagrams are a faithful representation of the actual code organization and of the actual interactions among objects. Programmers do not face any misalignment or gap when moving from such diagrams to the code.

The material presented in this book is based on the techniques developed during a collaboration we had with CERN (Conseil Européen pour la Recherche Nucléaire). At CERN, work for the next generation of experiments to be run on the Large Hadron Collider has started in large advance, since these experiments represent a major challenge, for the size of the devices, teams, and software involved. We collaborated with CERN in the introduction of tools for software quality assurance, among which a reverse engineering tool.

The algorithms described in this book deal with the reverse engineering of the following diagrams:

Class diagram: Extraction of inter-class relationships in presence of weakly typed containers and interfaces, which prevent an exact knowledge of the actual type of referenced objects.

Object and interaction diagrams: Recovery of the associations among the objects that instantiate the classes in a system and of the messages exchanged among them.

State diagram: Modeling of the behavior of each class in terms of states and state transitions.

Package diagram: Identification of packages and of the dependences among packages.

All the algorithms share a common code analysis framework. The basic principle underlying such a framework is that information is derived *statically* (no code execution) by performing a propagation of proper data in a graph representation of the object flows occurring in a program. The data structure that has been defined for such a purpose is called the Object Flow Graph (OFG). It allows tracking the lifetime of the objects from their creation along their assignment to program variables.

UML, the Unified Modeling Language, has been chosen as the graphical language to present the outcome of reverse engineering. This choice was motivated by the fact that UML has become the standard for the representation of design diagrams in Object Oriented development. However, the choice of UML is by no means restrictive, in that the same information recovered from the code can be provided to the users in different graphical or non graphical formats.

A well known concern of most reverse engineering methods is how to filter the results, when their size and complexity are excessively high. Since the recovered diagrams are intended to be inspected by a human, the presentation modes should take into account the cognitive limitations of humans explicitly. Techniques such as focusing, hierarchical structuring and element explosion/implosion will be introduced specifically for some diagram types.

The research community working in the field of reverse engineering has produced an impressive amount of knowledge related to techniques and tools that can be used during software evolution in support of program understanding. It is the authors' opinion that an important step forward would be to publish the achievements obtained so far in comprehensive books dealing with specific subtopics.

This book on reverse engineering from Object Oriented code goes exactly in this direction. The authors have produced several research papers in this field over time and have been active in the research community. The techniques and the algorithms described in the book represent the current state of the art.

Trento, Italy — *Paolo Tonella*
July 2004 — *Alessandra Potrich*