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*