JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 2, No. 1, January-February 2003

Book Review

Design by Contract, by Example

by Richard Mitchell and Jim McKim, Addison-Wesley, Boston, MA, 20002. 237 pp., $44.99(softbound). ISBN 0-201-63460-0

Reviewed by Charles Ashbacher

I continue to be surprised at the low number of developers who use design by contract principles to create software. While it does require a good deal more effort and analysis when writing the code, the benefits far outweigh the costs. When writing code using design by contract principles it seems as if you are writing code twice, one time the actual code and the "second" time the logical statements that the code must adhere to. The magnitude of extra work is summarized in the six basic principles of design by contract given at the start of the book:

Principle 1:	Separate queries from commands.
Principle 2:	Separate basic queries from derived queries.
Principle 3:	For each derived query, write a postcondition that specifies what result will be returned in terms of one or more basic queries.
Principle 4:	For each command, write a postcondition that specifies the value of every basic query.
Principle 5:	For every query and command, decide on a suitable precondition.
Principle 6:	Write invariants to define unchanging properties of objects.

The first two are not bad, but the last four can scare even excellent programmers. Developing the appropriate expressions is something that one does not learn quickly, it takes time and practice to get them right.

With all difficult tasks, the place to start is with "simple" examples, which is what is done in this book. A standard problem that is generally well understood by developers is taken and then the six principles are applied to the problem until the solution is complete. The first such problem is to implement contracts for a dictionary class, which in my opinion was an excellent first choice. It is complex enough so that the solution requires some work and effort to understand and yet is simple enough so that it is not overwhelming. The remaining problems are developing the appropriate contracts for immutable lists, queues, subclasses that inherit contracts from base classes and the contracts for an observer framework. The remainder of the book covers the benefits of design by contract and how to fulfill, test and check a precondition. The last topic is essential and completes the coverage of the topic. The example used for this is a simple counter class and it effectively demonstrates how to use the preconditions once they have been written.

The language used to demonstrate the problems is Eiffel, which is not widely used in development. However, that should not deter you in any way. Just think of it as a specialized language in which it is easy to represent contracts. If you are an experienced developer, but do not know Eiffel, it will not take a great deal more effort to learn the basics of the language while you learn how to work with software contracts.

Software developers need to take advantage of all the tools that are available, both in software and theoretical. Design by contract is one that forces you to think harder and more thoroughly about your programs, which is always a benefit. It is not easy to learn, but with this book the slope of the learning curve can be substantially flattened.