

Re: What books do you read / recommend

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2006-09/msg00084.html>

- *From:* "Phlip" <phlipcpp@xxxxxxxxxx>
 - *Date:* Wed, 06 Sep 2006 22:20:33 GMT
-

jwtulp wrote:

Just wondering, what books do you read or recommend regarding OO programming, OO design, OO architecture, OO patterns, in other word: what OO books :)

Within each category, I sort in order from entry-level to advanced. My first category is the C++ books, not because it's the best (it ain't), or because it's entry level (it ain't), but because it is sufficiently hard but well-explored that lots of uniquely good design books are also good C++ books.

These books introduce some topics they probably didn't teach you in school...

Firstly, books directly about C++ coding and designing:

- * Accelerated C++: Practical Programming by Example by Andrew Koenig & Barbara E. Moo
- * Effective C++ 2nd Edition by Scott Meyers
- * More Effective C++ by Scott Meyers
- * The C++ Programming Language 3rd Edition by Bjarne Stroustrup
- * Large Scale C++ Software Design by John Lakos
- * Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples by John J. Barton, Lee R. Nackman
- * C++ Coding Standards: Rules, Guidelines, and Best Practices by Herb Sutter & Andrei Alexandrescu
- * Modern C++ Design: Generic Programming and Design Patterns Applied by Andrei Alexandrescu
- * Exceptional C++ by Herb Sutter
- * More Exceptional C++ by Herb Sutter

Next, books about design:

- * Design Patterns: elements of reusable object-oriented software by Gamma, Johnson, Helm, & Vlissides
- * Refactoring: Improving the Design of Existing Code by Martin Fowler

Re: What books do you read / recommend

- * Refactoring to Patterns by Joshua Kerievsky
- * Smalltalk Best Practice Patterns by Kent Beck
- * Domain Driven Design: Tackling Complexity in the Heart of Software by Eric Evans
- * The Art of Computer Programming by Knuth

Books about the culture of programming around the source code:

- * Lean Software Development: An Agile Toolkit for Software Development Managers by Mary Poppendieck and Tom Poppendieck
- * Extreme Programming eXplained: Embrace Change, 2nd Edition, by Kent Beck
- * Agile Development: Principles Practices and Patterns by Robert C. Martin
- * AntiPatterns: refactoring software, architectures, and projects in crisis by Brown, Malveau, McCormick & Mowbray
- * The Pragmatic Programmer: From Journeyman to Master by Andy Hunt & Dave Thomas
- * Rapid Development: Taming Wild Software Schedules by Steve McConnell

How to preventing long open-ended debugging sessions:

- * Test-Driven Development: By Example by Beck
- * Code Complete 2nd Edition by Steve McConnell
- * Test Driven Development: A Practical Guide, by Dave Astels
- * How to Break Software: A Practical Guide to Testing by James A. Whittaker

Specific programming scenarios:

- * C++ GUI Programming with Qt 3 by Jasmin Blanchette & Mark Summerfield
- * Developing International Software 2nd Edition by Dr. International
- * Working Effectively with Legacy Code by Mike Feathers

/C++ GUI Programming with Qt 3/ is especially noteworthy because, unlike most other GUI Toolkits, C++ does not wrap Qt. Its inventors architected Qt directly in C++, using only the finest OO idioms.

The TDD books are especially noteworthy because, as you start writing programs larger than a couple modules, you will discover that most programmers spend most of their time debugging. The solution to this conundrum is so simple it shouldn't need a book, but our industry has a lot of "antipatterns" to overcome. The solution is to write tests for every kind of behavior you need, to only add code when you can get a test to fail, and to use Undo if a test fails unexpectedly (and you don't feel like applying a light amount of debugging). Replacing our advanced interactive debuggers with an Undo button helps code's behavior never depart from a known state. Prevention is better than a cure.

The Refactoring books are significant because, no matter how smart you are, you will always write code whose design could be improved after you read it. You could fight this effect, by planning on paper for a long time, or you can leverage it. Refactoring fits TDD like a glove.

Re: What books do you read / recommend

Re: What books do you read / recommend

Smalltalk (and Python & Ruby) is a strongly typed dynamic language. C++ is an almost-strongly typed static language that introduces generics to cover some dynamic typing abilities. Curiously, the higher level your code, the more dynamic typing you need. Dynamic typing means that `foo.bar()` will compile and execute no matter what type 'foo' is, so long as it has a method 'bar()'. This works as if all objects inherited a common interface, `Object`, that magically declared every possible method.

Curiously, the big software vendors keep pushing statically typed languages, Java & C#, while high-level code, such as GUI code or business-logic code, typically must bend over backwards to provide dynamic typing. Free and powerful languages, such as Ruby, supported entirely by their own programmer communities, are going to teach these big software vendors a thing or two.

--

Phlip

<http://c2.com/cgi/wiki?ZeekLand> <-- NOT a blog!!!

.