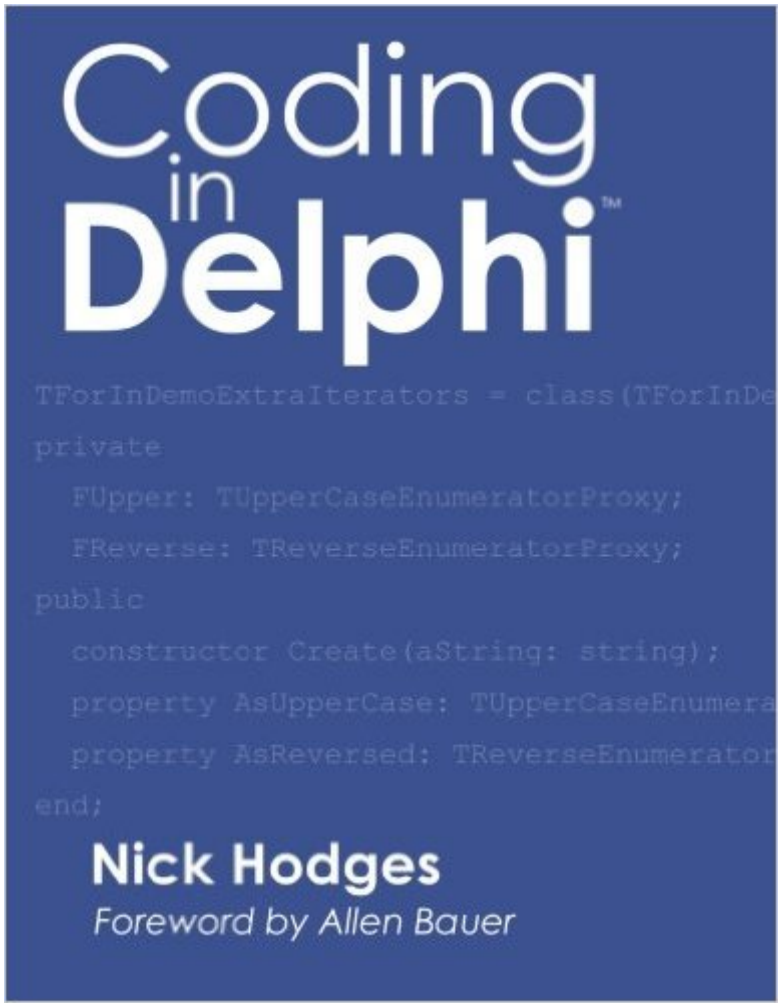
Coding In Delphi



Synopsis

Coding in Delphi is a new programming book by Nick Hodges that covers a variety of powerful Delphi programming features and techniques including Generics, Interfaces, Exception, Handling, Anonymous Methods, Collections, RTTI, Enumerators, Attributes, Dependency Injection and Unit Testing

Book Information

Paperback: 242 pages

Publisher: Nepeta Enterprises (February 22, 2014)

Language: English

ISBN-10: 1941266037

ISBN-13: 978-1941266038

Product Dimensions: 8.5 x 0.6 x 11 inches

Shipping Weight: 1.6 pounds (View shipping rates and policies)

Average Customer Review: 4.1 out of 5 starsÂ Â See all reviewsÂ (13 customer reviews)

Best Sellers Rank: #726,411 in Books (See Top 100 in Books) #7 inÂ Books > Computers & Technology > Programming > Languages & Tools > Borland Delphi #1983 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Software Development #177777 inÂ Books > Reference

Customer Reviews

I read through the eBook version and I really like how Nick has approached the chosen topics. I'd say it's roughly comparable to Jim Coplien's legendary work, "Advanced C++ Programming Styles and Idioms". Perhaps Delphi's popularity will expand now that Nick's book is available, in the same way that C++ use expanded after Coplien published his book. ;-)The problem is, there's so much Delphi code that was written back in D4/D6 days that's still being maintained today, yet you hardly see any of the things in Nick's book being used in that old code.Aside from the fact that many of these idioms hadn't been clearly identified back then, there's a lot of resistance to modify old code to improve it in any way. A big part of that is because managers today are so paranoid about cleaning up old code, rigidly adhering to the old, "if it ain't broke, don't fix it" syndrome. There's a great solution to that, called "Unit Testing", which is one of the topics Nick covers quite adeptly. However, in order to implement Unit Testing in old Delphi code, you pretty much have to rewrite the code to support Dependency Injection (DI), which is another topic Nick discusses nicely.I'd give the book 5 stars if Nick were to add a section that digs into how you might go about modifying old code

to use the cool ideas he presents. Otherwise, it’s just a lot of interesting stuff that won’t ever be used. Another thing I’d like to see is an example that employs most of the things that Nick discusses, like a framework of some kind.On the other hand, if you’re developing NEW apps in Delphi XE4 or later (and I sure wish more people were), then this book will give you some great programming idioms and strategies to embrace in your designs that will ultimately simplify things and reduce bugs. I mean ... it’s full of stuff that should be easily noticeable in contemporary apps written these days. I look at old code any more and I want to gag. The last batch of old code I worked with, I estimated we could have reduced the number of source lines of code by 40%-60%, resulting in far more readable code that also eliminated a bunch of coding errors I found in the process (because the code would have been so much more concise).As an aside, I’d like to mention that Delphi itself has evolved into an extremely powerful language, and the IDE is still far easier to use than any other platform I’ve worked with for most things. I only wish Embarcadero would do something to redesign the IDE so it’s not rigidly locked into the same programming paradigm introduced with Dephi 1, where you have a project composed of a bunch of independent unrelated forms. How about a way to structure an MVC- or MVVM-based app the same way it handles those ancient apps that use MDI forms? (Yes, Delphi still supports MDI forms!)One thing notable about Nick’s book is it focuses exclusively on code, ignoring the IDE -- hence the title: "Coding in Delphi". Many of the things Nick covers (particularly Interfaces and Dependency Injection) are helpful for building more abstract frameworks, like MVC or MVVM style apps. You need them to design around the walls set up by the IDE because of the structure it imposes on your code.Anybody who remembers writing apps in MSC that ran in Windows knows that the best thing that Delphi did that made it stand out when it was first released was to introduce what amounted to "de-multiplexed" Windows messaging. It did this by implementing individual event handlers for every meaningful Windows event that a form needs to handle. Before Delphi, you had to hand-code the message handling loop so it would demultiplex your events and send each one out to a different (hand-coded) handler. If you added an event, you needed to modify the message handling loop and then add a handler (usually in a different unit). It was a royal PITA. Delphi was revolutionary in that it hid the Windows message loop entirely by giving you a list of events you could hook into for any object on your form. You simply double-click on the event name in the Object Inspector, and poof! You’e taken instantly to a pre-defined event handling method in the SAME form unit! It’s still the same after all these years...Ironically, when you get into designing frameworks like MVC, you really WANT to be able to handle multiplexed events through a single dispatcher and listener that multiple units (some of which may be forms) need to understand and reflect completely. We also see this in

the kind of marshalling that's used by COM objects and Interfaces, as well as the mechanisms needed to implement web services (eg., SOAP, REST, etc.).Strings are frequently used to identify the actions to be taken, and you'd typically use a case statement to handle them, except that in Delphi, case statements don't work with string variables.So you have to go out of your way to add a layer of string mapping, multiplexing, and demultiplexing to overcome Delphi's limitations if you want to work with contemporary design patterns and web services in Delphi. DI helps, but it typically obscures what's being done in the process -- which is another reason for embracing Unit Testing. (And in fairness, I'm not sure Visual Studio, Eclipse, or other programming platforms are much better in this respect. Hey guys, it's 2014 already! Why are we still programming inside of IDEs designed around programming paradigms maturing in 1990?)If you're going to go this route, you definitely want to get a copy of Nick's book because he covers a lot of things that would be useful in these situations, particularly the discussions about generics, DI, object containers, and the Spring4D container library.

Too little has been written on Delphi, these past few years. To borrow from Sam Clemens, the death of the language has been greatly exaggerated. Nick presents a number of the more recent (and not so recent) additions to the Delphi language, and makes plain the advantages in using them. In the matter of interfaces, he harps (and admits it.)Full disclosure: I participated in the online review process during the writing, and proofread the first full draft. I'm a long-time Delphi developer, and yes, I am biased.The worst I can say of this book is that I wish Nick had gone a bit deeper in some areas. It should be in your library, and will amply repay your investment in purchase and study.

Nick's book goes through Delphi language features which have been added in the last 5-10 years, particularly Delphi XE2 onwards. It covers many of the same language features described in the excellent Delphi XE2 Foundations book but with a rather different slant. Nick's main focus is on how to use these language improvements to implement modern programming concepts, in particular unit testing. In the process Nick covers new tools which are built on these same language features such as Spring4D, DUnitX and the Delphi Mocks Framework.The result is an excellent and thought provoking introduction to new ways to develop in Delphi. Anyone who hasn't been keeping up with the latest in software development theory and the related Delphi advancements will learn a lot from reading Coding in Delphi.So why not 5 stars? After all I learned some things I didn't know and came to view several concepts in a different light as a result of reading the book. The problem was that for me the book didn't dig deeply enough and I would have liked Nick to grapple more with some of the

tricky questions. Many times while reading it I found myself thinking yes, but what about this problem, side effect or catch 22.For example Nick tries to build a case for using interfaces heavily for most, if not all, references because, as he points out, this makes for a decoupled design which is comparatively easy to unit test. The obvious problem is that interface and object references don't mix well and using both is likely to lead to AVs. Nick mentions this and suggests you never mix them. This is all very well but by necessity all components, including data access components, are object referenced and freed as objects. So what are we to do? Don't include any components in our unit tests? Or use interfaces from these components to allow easy mocking and just accept that we're going to have to be very careful to nil all interface references before we let the components be freed?Most of my concerns were along similar lines. It is easy to demonstrate interfaces, mocking, dependency injection, using a service locator just once at the root of the application to build all dependencies, etc in a simple example. It is another to then assume that all of these smoothly scale out to real world complexity without the need for compromises or new approaches.For all that the book is clearly still a 4 star book and brings a lot to the table in its own right. Perhaps it just needs a more in-depth sequel by Nick or a related party?

Nick Hodges has a clear and easy way of explaining even complex issues. I would definitely recommend this book to anyone who wants to understand coding in Delphi.

If you are looking for a real "hands-on" Delphi book, covering the areas of programming that need attention (best practice coding, etc.) this is a must have.

Very useful for learning about the newer Delphi features. I particularly liked the testing frameworks coverage.

Nick was a prior product manager for Delphi and is very well known in the Delphi world. There are good concepts covered here along with reasons to do things.

Excellent book. Too little to be written on Delphi for past few years. The best practice helps me a lot.

Definitive Delphi Problem Solver Visual Developer Developing Custom Delphi 3 Components: Master the Art of Creating Powerful Delphi 3 Software Components Delphi Developer's Guide to XML (Wordware Delphi Developer's Library) Developing Custom Delphi Components: Master the Art of Creating Powerful Delphi Software Components Secrets of Delphi 2: Exposing Undocumented Features of Delphi KickAss Delphi Programming: Cutting-edge Delphi Programming with an Attitude Programming in Delphi for Beginners CD. Textbook for High Schools / Programmirovanie v Delphi dlya nachinayushchikh CD. Uchebnoe posobie dlya VUZov Delphi 7 y Kylix 3 / Delphi 7 and Kylix 3 (Programacion / Programming) (Spanish Edition) Microsoft Directx 2 Games Programming with Delphi (Advanced Delphi Series) Delphi X Developer's Handbook [With Includes Useful Ready-To-Use Delphi Programs...] The Tao of Delphi: The Programmer's Path to Advanced Delphi Development Web Programming with Delphi (Delphi Programming) Tomes of Delphi: Alogrithm and Data Structure (Wordware Delphi Developer's Library) Delphi Complete Works of Edgar Degas (Illustrated) (Delphi Masters of Art Book 25) Coding in Delphi More Coding in Delphi The Scratch Coding Cards: Creative Coding Activities for Kids