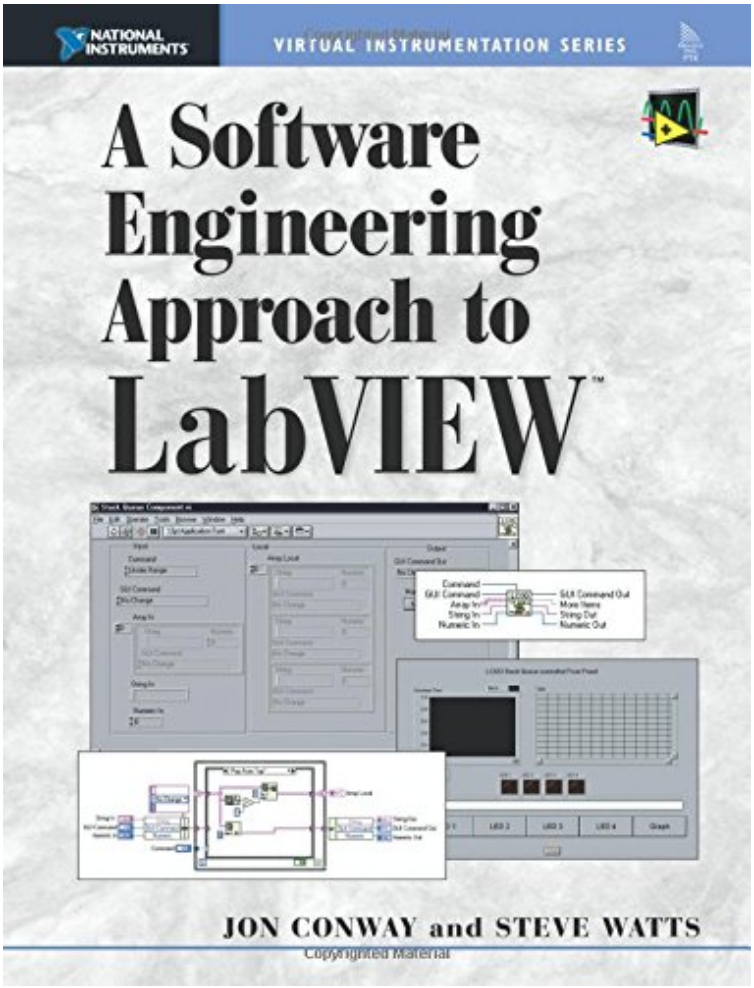
A Software Engineering Approach to LabVIEW

Jon Conway, Steve Watts
*ebooks | Download PDF | *ePub | DOC | audiobook*



#1315034 in Books 2003-05-15 2003-05-05Original language:EnglishPDF # 1 9.10 x .60 x 6.90l, 1.01 #File Name: 0130093653240 pages | File size: 46.Mb

Jon Conway, Steve Watts : A Software Engineering Approach to LabVIEW before purchasing it in order to gage whether or not it would be worth my time, and all praised A Software Engineering Approach to LabVIEW:

0 of 0 people found the following review helpful. Great Resource for LabVIEW ProgrammersBy M. SpiegelbergI'm not sure what is the cause of the bad reviews over picture quality. The pictures look great in the edition I have - if there was a problem, it's fixed now. The content of the book is great and gives LabVIEW programmers (both new and old) a quality approach to engineering good code.I would definitely get this one to add to the reference library if you are or are interested in becoming a LabVIEW programmer.3 of 6 people found the following review helpful. Excellent reference for advanced ProgrammersBy Jack HamiltonThis book is very germain for advanced LabVIEW Programmers and touches on the inportance of "Architecture" for LabVIEW code. It outlines OO Object Oriented coding and its application specifically in LabVIEW. I would have liked to see more examples - but I could easily fill in

the blanks.A recommeded reference to have for any experienced LabVIEW programmer. [...]2 of 2 people found the following review helpful. The information is still validBy Fabiola De la CuevaI love "A Software Engineering Approach to LabVIEW". It is a book intended for the intermediate to advanced LabVIEW programmer. It is a crash course on Software Design using LabVIEW as the tool. It introduces the concept of LabVIEW Component Ortiented Design (LCOD) making the development of large applications manageable. LCOD is programming based on Action Engines (a.k.a. functional global variables), but do not be discouraged by this, because the principles still apply to LVOOP and the ideas of planning/designing your application before laying the first wire still applies. It also has useful information on how to gather requirements, prototype and design the user interface.The first copies of the book were excellent quality. However some of them are not so good, making the images non readalbe (have we mentioned that LabVIEW is a "graphical programming language"!). The information is still great and you can access the code through [...]This book was invaluable when I started doing LabVIEW consulting on my own. Unfortunately given that Prentice Hall decided to move it to "on demand" printing (hence the poor quality), I don't think the authors would be encouraged to write any updates anytime soon.

A Software Engineering Approach to LabVIEW, by working programmers Jon Conway and Steve Watts, applies for the first time the techniques and principles of software design to LabVIEW programming. The LCOD technique designs flexibility into applications, making them more robust and much more easily adaptable to changes, even in large, industrial applications. Complete with examples and working code.

From the Back Cover Create more robust, more flexible LabVIEW applicationsthrough software design principles! Writing LabVIEW software to perform a complex task is never easyespecially when those last-minute feature requests cause a complexity explosion in your system, forcing you to rework much of your code! Jon Conway and Steve Watts offer a better solution: LCOD-LabVIEW Component Oriented Designwhich, for the first time, applies the theories and principles of software design to LabVIEW programming. The material is presented in a lighthearted, engaging manner that makes learning enjoyable, even if you're not a computer scientist. LCOD software engineering techniques make your software more robust and better able to handle complexityby making it simpler! Even large, industrial-grade applications become manageable. Design to embrace flexibility first, making changes and bug fixes much less painful Pragmatic discussion of the authors' tried and tested techniques, written byand forworking programmers Covers design principles; LCOD overview, implementation, and complementary techniques; engineering essentials; style issues; and more Complete with practical advice on requirements gathering, prototyping, user interface design, and rich with examples Work through an example LCOD project (all code included on companion Web site) to tie the lessons together This book is intended for test engineers, system integrators, electronics engineers, software engineers, and other intermediate to advanced LabVIEW programmers. None of the methods discussed are complex, so users can benefit as soon as they are proficient with the syntax of LabVIEW.Go to the companion Web site located at http://author.phptr.com/watts/ for full source code and book updates. About the Author JON CONWAY has 20 years' experience in writing software, with half of that in LabVIEW. His fieldsof expertise include real time, robotics, databases, DAQ, DSP, and multiple software languages andoperating systems; his idea for LCOD arose from his experience gained working on complex softwareprojects. Jon is a partner in Structured Software Design Consultants of Hampshire, UK. STEVE WATTS has 15 years of experience in writing test software, and has been programming inLabVIEW for 6 years. His areas of expertise include OOD, the Yourdon methodology, CASE tools,electronics, lasers, switching system design, DAQ, statistical process control, databases, userinterface design, software engineering, as well as a variety of programming languages. Steve is apartner in Structured Software Design Consultants of Hampshire, UK.Excerpt. Reprinted by permission. All rights reserved.PrefaceThere are many ways of designing and implementing a system. We are not trying to say that you should immediately adopt the techniques presented in this book in place of how you currently design and write software. Specifically, what we are saying is that this is how we design and implement software in real-world applications. We want you, the reader, to draw your own conclusions.It's important to note that the authors are working engineers who pay their mortgages by writing software, not by writing books.The Test Engineer's PerspectiveSteve Watts writes--As a normally trained test engineer I've been programming test systems for years and using many different programming languages (HPBasic, UCLA Pascal, Turbo Pascal, Visual Basic, and QuickBasic). In many of the more complex systems I have had the same experience. Doing little design up front I would plow into the coding, by the 50% stage I would normally be ahead of the game, and at the 90% stage I would be 90% complete and patting myself on the back. And then it happened!I now use the term "the complexity explosion" small changes in the software would cause problems throughout the system. The customer would throw in "unplanned-for" changes. I could no longer picture the system clearly in my head. The last 10% of the project took another 90% of the time.I knew something was wrong but didn't have the tools or training to explain what, why, or how. In the end I put it down to software being a pain.A few years ago when Jon came to the company he was touting a language called LabVIEW. This became the company standard, so I had to learn it. The first application that I wrote (in a very unpleasant style I hasten to add) was a small temperature logging effort. It became clear to me that

something was still wrong. True, G gave huge productivity increases over Pascal and Visual Basic, which I was using at the time, but the complexity explosion was still there, lurking in the background. I went back to Jon and discussed it with him and he introduced me to LCOD. I had never thought that there was a discipline called Software Engineering (I thought by writing software I was a software engineer), or heard of Coupling, Cohesion, or Information Hiding. OOD, OOA, and structured software design had all passed me by.I'm the sort of person who needs to completely understand a process beyond the words, and since we were dealing with reasonably abstract concepts I struggled in the search for this comprehension. I took postgraduate courses in Software Engineering and Object Oriented Programming. I experimented with the projects I was working on, using structured software design, CASE tools, and OOA. The inherent complexity that academia applies to all things and the embracing of this complexity (out of elitism perhaps!) by the software community, led me to believe that this whole process was harder than I thought. BUT IT'S NOT!I began to see that by applying these techniques my programs were becoming manageable, they were not increasing in complexity near the end, and I could implement late changes without reducing system robustness. Maintenance was easier and faster, customers were happy and impressed, stress levels were reduced, illness and pestilence were driven from the land, neighbor loved neighbor, and there was peace in our time.Don't get me wrong, none of this will make a complex problem any less complex, but at least by applying these techniques you won't be making it more complex.As software engineers we should be striving for the following: * Deliver what we say we are going to deliver * Deliver it when we say we are going to deliver it * Ensure that it operates predictably * Ensure that changes and bug-fixes do not harm the stability of the program or break the bank to implementWe should be in the business of managing complexity:Clever Software = BAD; Simple Software = GOODOne of our customers wrote the following testimonial (and we didn't even pay him!): "LCOD has made a complex test system simple, flexible, and futureproof."Using the analogy of a journey (as we do throughout the book), we feel we have taken enough steps forward to enable us to turn around and put up a few signposts. Hopefully, these signposts will help you in your journey.I have never regretted adding flexibility to my software, but I have always rued the times I have omitted flexibility.The techniques presented in this book are reasonably simple to understand. We feel that someone can only successfully apply something if they understand it. Our aim is to introduce and explore the concepts of software design using LabVIEW, and to do this in an understandable and applicable manner. A lot of techniques and methodologies get bogged down with computer science and forget about the design aspects; our intentions are to always concentrate on design and hopefully translate some of the computer science.