Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

Jolt Awards: The Best Books

The best books of the past 12 months.

September 23, 2014
URL:http://www.drdobbs.com/joltawards/jolt-awards-the-best-books/240169070

As we do every year, *Dr. Dobb's* recognizes the best books of the last 12 months via the Jolt Awards — our cycle of product awards given out every two months in each of six categories. No category gets more entrants than books, and this year was no exception with more than 30 nominees submitted by publishers, vendors, readers, and the Jolt judges themselves. The award covers all books published during the twelve months ending June 30th of this year.

The Jolt judges did an initial triage that reduced the field to a readable number of entrants. Then, there followed a second pass in which the top books were selected and ranked — after long deliberation and considerable discussion. As always, the best book of the year receives the Jolt Award; the two runners up each receive a Jolt Productivity award; and the remaining three books are awarded Jolt Finalist status. Reviews of these six volumes are included in this article.

The judges for this category were Andrew Binstock, Jonathan Harley, Gastón Hillar, David Mulcihy, Larry O'Brien, Gary Pollice, Roland Racko, Mike Riley, and Gigi Sayfan. Given the many nominees and the large number of judges, you can have high confidence that the award winners represent the very best of the books published in the 12-month award period.

We thank Jolt sponsor, Rackspace, for providing virtual machines for the judges' use in examining and testing the code and tools discussed in these books.

And now, to the winners, starting with the finalists…

by Brian Hogan

Given enough time, every language develops its own build utility. C and C++ have the many make variants. Java has Ant and Maven, Groovy has Gant, Ruby has Rake, and so on. While in theory, one tool could be used across multiple languages, the reality is that each language's ecosystem has different conventions and different ways of delivering the final distributable executable package.

Now that JavaScript is both a client and server solution, it needs a sophisticated build tool as well. While there are several products that have tried to address this need, the leading one is surely Grunt. Grunt performs many of the JS-specific tasks you'd encounter in normal Web app production: linting, minification, testing, and so on. In addition, Grunt has a growing ecosystem of plugins that help to integrate it into modern toolchains.

What Grunt has lacked is a useful manual with documentation of not only common tasks, but common build situations. This book, published by the Pragmatic Programmers, is precisely that guide. It's a practical guide to building applications in JavaScript, a manual for Grunt, and a reference that provides recipe-like solutions. Explanations of how to work with plugins and tutorials on how to write your own task templates complete this intensely hands-on book. In addition, it's well written. It is, in sum, precisely the book you wish you had for each of the major tools you use in software development.

In discussing this book, one of the Jolt Award judges lamented that there were not more inexpensive, single-topic, hands-on manuals that give you all you need to know in a direct and approachable style. This observation is spot on in this era of carelessly written manuals and defective documentation. However, Automate With Grunt is one of the few manuals that amply fulfills this simple mission. If you write apps primarily with JavaScript, this is the book to get and keep within reach.

— Andrew Binstock

by Francis daCosta

The Internet of Things (IoT) is coming. This book investigates in surprising depth the consequences of having billions of sensors and actuators interconnected and communicating, and what the architecture and networking will need to be. The main contention is that existing IP-based networking is too big, cumbersome, and complicated for the myriad small devices that will be the principal end devices of the IoT.

Instead, a very simple "chirp" protocol will more likely be used by often minimal, low-powered, and only intermittently connected devices. Consolidating this data and routing it to its target (called integrator function in the book) will require a network of propagator nodes that will perform some filtering, aggregation, and batching of chirps — leading eventually to packaging them in standard TCP/IP packets to send them onto the existing Internet.

This scenario, which keys off current business uses of IoT-like technologies, seems far more likely than the scenarios that dominate the consumer narrative of extensive data packets being routinely streamed from all sorts of devices in the home to keep the otherwise uninformed individual up-to-date on every aspect of his household's state.

The author is a thinker and practitioner who has a lot of real-world experience in building massive distributed systems, embedded software, etc. While this is more of a technology treatment, rather than a programming volume, I found myself nodding in agreement as I read many of the concepts and ideas. Nonetheless, I have some doubts about a few of the dynamic properties of the IoT presented here, such as an emergent self-organizing network of

propagator nodes and a combination of public and private markers used for generic filtering and routing.

Overall, Rethinking the Internet of Things is a very deep and detailed book that explains clearly an emerging field, which will become a major part of computing — and if IoT is of interest, it's a thoughtful explanation that removes the hype, presents what's likely to emerge first, and sketches out how it will function.

— Gigi Sayfan

by Adam Shostack

It has become all too usual to read news about specific software being vulnerable to serious security threats. Considering that most modern software usually runs on Internet-connected devices, we must become more aware of modern security threats and design our software to protect it against specific potential threats.

In this book, the author focuses on modeling software projects to address or mitigate potential threats. You don't need much security expertise to read the book and the first two chapters provide easy-to-understand, real-life examples to introduce the threat modeling approach. By this means, you begin to find security bugs early and understand your security requirements. The author uses diagrams, tables, and easy-to-understand examples to explain modern threats that you should be able to identify, then describes the different possible ways to either mitigate or eliminate them.

The book also discusses the different ways of modeling software to address threats, as well as techniques and tools to find those threats. Once you've read the first two chapters, you can focus on the threats that are most important for your security needs, and read the techniques and tools for those in particular.

The author also discusses how to manage and address threats, with an interesting focus on evaluating and making risk tradeoffs. Illustrative experiences in threat modeling in specific technologies are also provided, with great coverage of threat modeling in modern Web, cloud, and mobile applications and a cookbook approach that you can use as a baseline for your security requirements analysis. The last part of the book presents interesting ideas to introduce threat modeling as part of your software development projects. Unluckily, the author has chosen to focus on modeling and didn't include code samples in the book. Code samples would have been very useful to make the subject clearer for developers who must imagine in their own lines of code how some of the attacks are performed.

In the U.S., modeling is still viewed with a certain amount of resistance, although it is widely accepted in Europe and elsewhere. Security vulnerabilities might well be the medium by which modeling demonstrates its value to U.S. business developers and hobbyists. If so, Threat Modeling is likely to be a key part of the dialog, illuminating both the technique and the way it seal off holes into which crackers can place crowbars. Overall, this is an excellent volume that should be examined by most developers concerned with issues of security.

— Gastón Hillar

by Michael Mikowski and Josh Powell

"In the time it takes to read this page, 35 million person minutes will be spent waiting for traditional website pages to load." That's what the authors say and that's why they wrote this book — to show you how you can dramatically reduce the wait time for your website.

Single Page Web Applications (SPAs) are a Web architecture that organizes all the business logic and page rendering to occur entirely within the browser. Generally, the server side does only authentication or database access. Except for possibly the initial page load, there are "no spinners" and no waiting because subsequent page loads do not take place for remaining user interactions. Site response is thus always immediate and seamless. The technology behind this, both as to putting all the actions in one page load and, at times, creating the perception that more than one page is being shown, can be fairly complicated. In the samples in this book, for example, the authors make extensive use of HTML5, JavaScript, and jQuery on the client, and node.js and MongoDB on the server. They also use testing tools to verify the code.

In the core of Single Page Web Applications: JavaScript End-to-End, the authors show how to implement SPAs using a highly disciplined JavaScript-based design. They provide a sample project copiously annotated with design rationale and explanations. Recognizing that running the entire business application logic in the browser can result in a JavaScript code chunk sized in the tens of thousands of lines, the authors use their years of experience to also present the reader a well thought-out architecture, insightful best practices, and an unusually comprehensive JavaScript coding standards document to help control the complexity implied by this size. JavaScript is used end-to-end to aid in the project management of large SPAs by using a single language all the way through from server to database.

In a world in which Web applications are once again assuming a dominant role, SPAs represent an interesting and useful niche for developers. For them, this volume is the master handbook.

— Roland Racko

by Bjarne Stroustrup

Bjarne Stroustrup, the inventor of C++, is a prolific author. His definitive handbook to C++ weighs in at more than 1300 pages. This volume, which is an explanation of programming in a native language, also tips the page count past 1300. Fortunately, Stroustrup's style is neither excessively academic or dry, and his explanations are clear and well thought through.

Those traits notwithstanding, who will read so much material? The primary audience for this book is, I think, undergraduate engineering students — especially those who have grown up in a world where computers are ubiquitous, powerful, and abstracted from their underlying hardware and who have never known a world without Java, protected memory, and graphical user interfaces. Stroustrup's approach, which he describes as "depth-first, concrete-first, and concept-based," may not be the right approach for people whose ambitions end with scripting and task automation or for those few students who actually want to study computer *science* (for whom I think an argument can be made for a mathematical approach), but it is a good choice for students who want to become programmers or who want to work in technical fields.

The book is primarily a programming volume using C++ as the example language, rather than a book on the language itself. *Dr. Dobb's* readers don't, of course, need to be coached through "Hello, World!" and C or C++ are likely to be, if not their day-to-day language, part of their arsenal.

But for many of us, the C++ with which we are most familiar is, if not obsolete, at least antiquated. C++11 and C++14 have essentially rebooted the language and I, for one, found useful material in even the earlier chapters that explained the new features in basic, straightforward, and easy-to-assimilate language. The later chapters and appendices also presented entire swaths of the standard library with which I was unfamiliar.

A book that is primarily an undergraduate text is an unlikely candidate for a Jolt Award, but the depth and clarity of Programming: Principles and Practice Using C++, 2nd Edition and the explanation of the newest features of the language make this a very appealing volume.

— Larry O'Brien

by Mark Summerfield

Mark Summerfield's Python in Practice is a fascinating book intended for intermediate and advanced Python developers. Rather than being a primer, it attacks advanced issues in Python — the ones that go beyond bread-and-butter programming. It imparts the skills that distinguish the expert from the journeyman. In other words, if you're a decent, but not yet great Python programmer, this is book is for you.

The first three chapters are a Python perspective on the Gang of Four (GoF) design patterns. In a dynamic language like Python, many of the problems the original design patterns aimed to solve look different or have a different subtext. Unlike the familiar, long-winded academic presentation of patterns, The author uses practical illustrations to explain the patterns' implementations in Python. The explanations are "what you need to know" and the code examples are concise and sufficient to illustrate the specific point Summerfield is making without spilling into tangentially related side arguments. This combination makes the patterns both more approachable and more likely to be used.

There follows an excellent chapter on concurrency, which discusses I/O versus CPU-bound concurrency and multithreading versus multiprocessing, including explanation of the famous Global Interpreter Lock (GIL).

A chapter on extending Python deals with one of Python's ostensible weaknesses — its slow performance. The book presents multiple solutions for compiling Python to native code and calling C and C++ libraries directly. It drills deeply into the built-in ctypes module and Cython.

My one disappointment is the chapter on high-level networking. It focuses on remote procedure calls (RPCs) — which represent an approach that has lost much of its appeal in recent years. Instead, I would have liked to see a chapter on low-level, socket-based programming and high-level REST/HTTP-based programming with proper Web API design.

The chapter on GUI programming is very good. The author presents the various options, discusses cross-platform concerns, and explains which projects/tools are active. This is important because choosing a GUI toolkit for a UI-intensive application is a decision you will typically have to live with for a long time. The author's experience in this area (he wrote the definitive book on Qt programming) serves him well. He goes on to demonstrate basic techniques, dialogs, and even a full game using Tkinter, the built-in cross-platform GUI toolkit that comes with Python.

The final chapter of the book further extends UI coverage by exploring 3D programming with OpenGL. Two different libraries are used: PyOpenGL and Pyglet.

Overall, the book has the right mix of high-level concepts, low-level details, and code samples. Its pragmatic treatment of useful topics, lucid explanations, succinct code, and careful attention to presentation make it an excellent book for intermediate to advanced Python developers and the Jolt Award winner for 2014.

— Gigi Sayfan