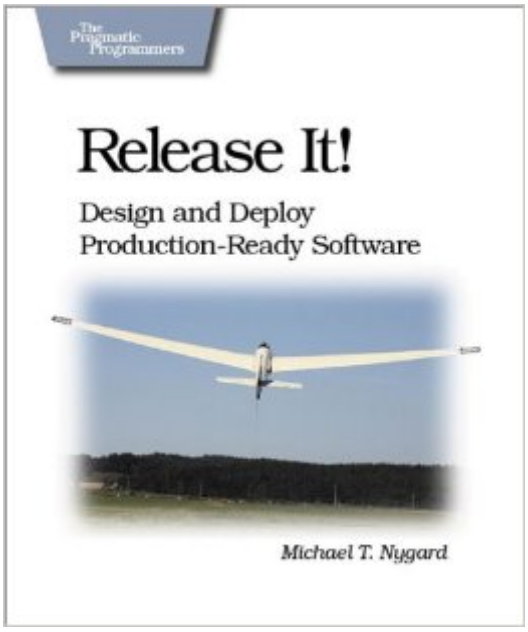
Release It!: Design And Deploy Production-Ready Software (Pragmatic Programmers)



Synopsis

Whether it’s in Java, .NET, or Ruby on Rails, getting your application ready to ship is only half the battle. Did you design your system to survivef a sudden rush of visitors from Digg or Slashdot? Or an influx of real world customers from 100 different countries? Are you ready for a world filled with flakey networks, tangled databases, and impatient users?If you’re a developer and don’t want to be on call for 3AM for the rest of your life, this book will help.In Release It!, Michael T. Nygard shows you how to design and architect your application for the harsh realities it will face. You’ll learn how to design your application for maximum uptime, performance, and return on investment.Mike explains that many problems with systems today start with the design.

Book Information

File Size: 3954 KB

Print Length: 326 pages

Simultaneous Device Usage: Unlimited

Publisher: Pragmatic Bookshelf; 1 edition (March 30, 2007)

Publication Date: November 5, 2012

Sold by:Â Digital Services LLC

Language: English

ASIN: B00A32NXZO

Text-to-Speech: Enabled

X-Ray: Not Enabled

Word Wise: Not Enabled

Lending: Not Enabled

Enhanced Typesetting: Not Enabled

Best Sellers Rank: #163,761 Paid in Kindle Store (See Top 100 Paid in Kindle Store) #19 inÂ Kindle Store > Kindle eBooks > Computers & Technology > Networking > Client-Server Systems #63 inÂ Books > Computers & Technology > Networking & Cloud Computing > Data in the Enterprise > Client-Server Systems #513 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Software Development

Customer Reviews

The subtitle of this book might as well be Architecture and Design for the Paranoiac. The book lays out some critical aspects to creating and rolling out stable software systems. It’s directed to those working in the enterprise arena and need the utmost from stability, capacity, and overall design.

Nygard’s definition of "enterprise" is somewhat broad in that he considers "enterprise" to be any system providing mission-critical support to a business. Regardless of how you define your particular software, I’m sure you’ll find something useful in this book.Nygard presents the book from an anti-pattern/pattern approach: he uses case studies to illustrate how critical errors in design or implementation (anti-patterns) have caused disasterous outages. He then moves on to show how application of solid design patterns could have avoided the problems. He also spends some time going in to detail on how some of the outages have happened, including brief discussions on network packet captures and decompiling third party drivers.There are a lot of solid fundamentals in the book: dealing with exceptions at system integration points, thread synchronization, avoid rolling your own primative feature libraries such as connection pools, and make sure to test third-party libraries which play critical roles. The general approach of discussing anti-patterns followed by patterns is also a nice way of putting forth the material.There are a lot of more complex bits covered as well, such as thinking ahead on how you’ll deal with bots and crawlers, avoiding AJAX overkill, designing ahead for and using session. I also liked that Nygard talks about the importance of involving the customer in decisions on thresholds and other critical boundaries.

Once in a year, I tag a book as "book of the year", the best book I read during the year. 2007 is not over, but this my "2007 book of the year", I know that.Frankly, I just bough this book because it’s published by the "pragmatic programmers" and I trust these guys. The title is not even appealing. I knew quickly that I will discover many things.For a long time, I wonder what to do to build up a system which is fine in production, but I didn’t understand quite right what was needed (I know now that I really misunderstood the problem). The first thing that came to my mind was to make the software strong (a good thing to do by the way) ; the second thing that came to my mind was to make it really, really strong (which starts to be stupid).Michael helps us to understand that systems fail anyway. But it should fail fast (and can often fail only partially), it must facilitate diagnosis and quick restart. And design must deal with that. But the author doesn’t stay in general considerations, he points out specific patterns and antipatterns for the systems design, by means of stability and capacity. The vast majority of article tend to exposes how new technologies make the life so easy. The author revisit technologies and technical choices throught the production glasses: why AJAX should be considered with care, why we must think about pre-computed pages instead of ynamic composition in some cases, why caches is not a one-size-fits-all answer and so on. Another important point well illustrated: a system is software + hardware and the architecture must be though with physical deployment and hardware architecture in mind. Promotion of full independance

of the architecture over the deployment is plain wrong. There are so many subjects tackled her, I can’t speak about them all, sorry.

This book is intended for architects, designers, and developers of software on which a business depends and whose failure costs money. The tone is informal and the style is easily read. Some architects may wish for more rigor and consider it too easily read but they might still benefit because it contains quite a bit of wisdom earned by experience.The book discusses issues of uptime, failure, and maintainability with examples drawn from the author’s experience and from other industries. Making the point from more than one point of view serves to drive it home.This is not a programming book but the illumination of a problem is often improved by a snippet of code. The code is Java and is easily read by anyone familiar with programming. Having some familiarity with multi-threaded programming in following the explanations and their examples will make them a little easier to read but is not necessary to get the point. (Even if you truly have no knowledge of Java, looking up JDBC, JVM, EJB, JSP, J2EE, log4j, and servelet will not be much effort because not much knowledge of them is required.) The examples emphasize web applications because, I suppose, that’s the environment most vulnerable to huge capacity requirements, more complex environments, more numerous causes of failure, and failures that are more visible.The author’s analysis of the problem space has two dimensions --- stability and capacity --- in which a given enterprise system can be located. The analysis also has two categories: general design and operations.Stability and CapacityA given coordinate, on the stability axis, for example, implies the presence and absence of features that improve and diminish stability.

2013: Design, Deploy and Deliver an Enterprise Messaging Solution Controller-Based Wireless LAN Fundamentals: An end-to-end reference guide to design, deploy, manage, and secure 802.11 wireless networks Create Your Own Operating System: Build, deploy, and test your very own operating systems for the Internet of Things and other devices Full-Stack JavaScript Development: Develop, Test and Deploy with MongoDB, Express, Angular and Node on AWS Danger Ready: Prepare to Survive Any Threat and Live to Tell the Tale: (Terrorist Attacks, Mass-Shootings, Earthquakes, Civil Unrest - Be Ready to Protect Your Family Whatever the Danger) Prepper Paracord: Quick Deploy Sinnets Algorithms: C++: Data Structures, Automation & Problem Solving, w/ Programming & Design (app design, app development, web development, web design, jquery, ... software engineering, r programming) Formulas and Calculations for Drilling, Production, and Workover, Fourth Edition: All the Formulas You Need to Solve Drilling and Production Problems Formulas and Calculations for Drilling, Production, and Workover, Third Edition: All the Formulas You Need to Solve Drilling and Production Problems