Foreword by Martin Fowler*

In my early days in the software industry, one of the most awkward and tense moments of a software project was integration. Modules that worked individually were put together and the whole usually failed in ways that were infuriatingly difficult to find. Yet in the last few years, integration has largely vanished as a source of pain for projects, diminishing to a nonevent.

The essence of this transformation is the practice of integrating more frequently. At one point a daily build was considered to be an ambitious target. Most projects I talk to now integrate many times a day. Oddly enough, it seems that when you run into a painful activity, a good tip is to do it more often.

One of the interesting things about Continuous Integration is how often people are surprised by the impact that it has. We often find people dismiss it as a marginal benefit, yet it can bring an entirely different feel to a project. There is a much greater sense of visibility because problems are detected faster. Since there is less time between introducing a fault and discovering you have it, the fault is easier to find because you can easily look at what's changed to help you find the source. Coupled with a determined testing program, this can lead to a drastic reduction in bugs. As a result, developers spend less time debugging and more time adding features, confident they are building on a solid foundation.

Of course, it isn't enough simply to say that you should integrate more frequently. Behind that simple catch phrase are a bunch of principles and practices that can make Continuous Integration a reality. You can find much of this advice scattered in books and on the Internet

*Martin Fowler is series editor and chief scientist at ThoughtWorks.

(and I'm proud to have helped add to this content myself), but you have to do the digging yourself.

So I'm glad to see that Paul has gathered this information together into a cohesive book, a handbook for those who want to put together this best practice. Like any simple practice, there's lots of devil in the details. Over the last few years we've learned a lot about those details and how to deal with them. This book collects these lessons to provide as solid a foundation for Continuous Integration as Continuous Integration does for software development.

Foreword by Paul Julius

I have been hoping someone would get around to writing this book—sooner rather than later. Secretly, I always hoped it would be me. But I'm glad that Paul, Steve, and Andy finally pulled it all together into a cohesive, thoughtful treatise.

I have been knee-deep in Continuous Integration for what seems like forever. In March 2001, I cofounded and began serving as administrator for the CruiseControl open source project. At my day job, I consult at ThoughtWorks, helping clients structure, build, and deploy testing solutions using CI principles and tools.

Activity on the CruiseControl mailing lists really took off in 2003. I had the opportunity to read descriptions of thousands of different CI scenarios. The problems encountered by software developers are varied and complex. The reason developers go to all this work has become clearer and clearer to me. CI advantages—like rapid feedback, rapid deployment, and repeatable automated testing—far outweigh the complication. Yet, it is easy to miss the mark when creating these types of environments. And I never would have guessed when we first released CruiseControl some of the exciting ways that people would use CI to improve their software development processes.

In 2000, I was working on a large J2EE application development project using all the features offered in the specification. The application was amazing in its own right, but a bear to build. By build, I mean compile, test, archive, and conduct functional testing. Ant was still in its infancy and had yet to become the de facto standard for Java applications. We used a finely orchestrated series of shell scripts to compile everything and run unit tests. We used another series of shell scripts to turn everything into deployable archives. Finally, we jumped through some manual hoops to deploy the JARs and run our functional test suite. Needless to say, this process became laborious and tedious, and it was fraught with mistakes.

So started my quest to create a reproducible "build" that required pressing "one button" (one of Martin Fowler's hot topics back then). Ant solved the problem of making a cross-platform build script. The remaining piece I wanted was something that would handle the tedious steps: deployment, functional testing, and reporting of the results. At the time, I investigated the existing solutions, but to no avail. I never quite got everything working the way I wanted on that project. The application made it successfully through development and into production, but I knew that things could be better.

Between the end of that project and the start of the next, I found the answer. Martin Fowler and Matt Foemmel had just published their seminal article on CI. Fortuitously, I paired up with some other ThoughtWorkers who where working on making the Fowler/Foemmel system a reusable solution. I was excited, to say the least! I knew it was the answer to my prayers lingering from the previous project. Within a few weeks, we had everything ready to go and started using it on several existing projects. I even visited a willing Beta test site to install CruiseControl's precursor in a full-scale objective enterprise. Shortly after that, we went open source. For me, there has been no looking back.

As a consultant at ThoughtWorks, I run into some of the most complicated enterprise deployment architectures out there. Our clients are frequently looking for a quick fix based on a high-level understanding of the advantages promised by the industry literature. As with any technology, there exists a fair bit of misinformation about how easy it will be to transform your enterprise. If years of consulting have taught me anything, it is that nothing is as easy as it looks.

I like to talk to clients about practically applying CI principles. I like to stress the importance of shifting the development "cadence" to truly leverage the advantages. If developers only check in once a month, lack focus around automated testing, or have no social imperative to fix broken builds, there are big issues that must be addressed to reap the full benefits of CI.

Does that mean that IT managers should forget about CI until these practices have been shifted? No. In fact, using CI practices can be one of the fastest motivators for change. I find that installing a CI tool like CruiseControl prompts software teams to be proactive instead of reac-

tive. The change does not happen overnight and you have to set your expectations appropriately—including those of the IT managers involved. With persistence and a good understanding of the underlying principles, even the most complicated environments can be made simpler to understand, simpler to test, and simpler to get into production quickly.

The authors have leveled the playing field with this book. I find this book to be both comprehensive and far-reaching. The book's in-depth coverage of the most important aspects of CI will help readers make well-informed decisions. The broad range of topics covers the vast array of approaches that dominate the CI landscape today and helps readers weigh the tradeoffs they will have to make. Finally, I love seeing the work that so many have strived to achieve in the CI community become formalized as the basis for further innovation. Because of this, I highly recommend this book as a vital resource for making sense of complicated geography presented by enterprise applications by using some CI magic.