# Afterword

This text builds on a host of prior industry work. None of the ideas in this book are inventions or completely original. This book represents a synthesis of ideas from all corners of the software industry across multiple platforms. Coupled with experience, success, and failure, this book provides the best rules of thumb that can be mustered at this time.

The following is a short summary of the prior works and events that were highly influential in the experience that led to this book.

In 2001, a group of software industry leaders met at a ski resort to mull over the problems in their field that had been building throughout the 1990s and the infamous dot-com bubble of the late '90s. This group produced the *Manifesto for Agile Software Development* ([www.agilemanifesto.org](http://www.agilemanifesto.org)). These principles have redefined the way the industry organizes and executes software development work. A fundamental premise of agile development is to organize and perform work in much smaller batches than had previously been used in the late '90s. Now, units of software delivery called "sprints" or "iterations" are commonly discussed. Many organizations run in cadences of iterations that are 1-3 weeks in length. The unit of software changes has likewise shrunk. Now developers target changes that can be accomplished in the current iteration, and many teams experiment with just how small software changes can be while keeping the software stable and releasable at all times.

In 2004, Michael Feathers wrote *Working Effectively with Legacy Code*. The book was released in the years following the *Manifesto for Agile Software Development*, and it addressed a common situation that many organizations were faced with: How can I change my software when every change generates at least two new defects? As teams were attempting to make changes and re-stabilize their software, they realized that previous engineering methods were insufficient when attempting to drive cycle times down to less than a month. Feathers describes methods for breaking apart code bases that were never intended to execute outside of a completely integrated production environment. The author summarizes that without running the software within automated test harnesses, any piece of software, however new, is destined to become labeled as "legacy code;" Unchangeable, brittle, and expensive to maintain. He described techniques illustrating how to insert seams into existing code in order to retrofit tests. Then, armed with tests that protected the functionality, the software could be changed with less fear. Michael Feathers was also the author of some early unit testing framework for C++, namely CppUnit.

In 2006, Paul M. Duvall, Steve Matyas, and Andrew Glover wrote *Continuous Integration: Improving Software Quality and Reducing Risk*. This influential work illustrated a method that some in the industry had been perfecting called *continuous integration* (CI). This method sought to run an automated build of the software application every time any change was committed to the version control system. The book illustrates some specific engineering methods that have to be adopted. A ground-shaking inclusion was the premise that the continuous integration build must include building and testing dependencies that an application owns, including relational databases and other storage mechanisms. The authors dedicated an entire chapter (chapter 5) to *continuous database integration*.

Additionally, in 2006 Martin Fowler penned an article titled *Continuous Integration* ([www.martinfowler.com/articles/continuousIntegration.html](https://www.martinfowler.com/articles/continuousIntegration.html)). In this article, he proposes some standards for a continuous integration build:

* Maintain a Single Source Repository
* Automate the Build
* Make Your Build Self-Testing
* Everyone Commits To the Mainline Every Day
* Every Commit Should Build the Mainline on an Integration Machine
* Fix Broken Builds Immediately
* Keep the Build Fast
* Test in a Clone of the Production Environment
* Make it Easy for Anyone to Get the Latest Executable
* Everyone can see what's happening
* Automate Deployment

While many in the industry were innovating new and better methods for shortening cycle time, 2006 was the year when successful and proven methods were being shared openly and published widely. Continuous integration included the automated deployment of software releases. In the years that followed, continuous *integration* become known as continuous *delivery*, expressly implying the inclusion of software deployments all the way to end users in production environments. Some refer to the method as *CI/CD*, illustrating the confusion that exists to this day about where CI stopped and CD started. Both refer to the full process of integrating source code from multiple developers and providing new working software to end users. However, in the common lexicon, most refer to the "CI build" and then refer to a CD *pipeline* that deploys across successive environments. Historically, however, continuous integration, as a method, in 2006 included automated deployments to production.

2009 saw another seminal work published. Jez Humble and David Farley authored *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. This book cited the 2006 book on continuous integration and pulled the story forward by proposing methods by which to not only build continuously but also to continuously deploy to downstream environments. More proven methods for handling deployment scenarios were included. The stages proposed in this book are:

* Version control
* Commit stage
* Automated acceptance tests
* Manual validations
* Release (production)

This process is much more high level, but the commit stage includes the private build and the integration build, and the automated acceptance tests are prescribed to be run against a fully deployed pre-production environment.

In 2009, the term *DevOps* was coined by Patrick Debois when he organized the first DevOpsDays conference in Ghent, Belgium. The current body of DevOps-focused works and events have grown substantially over the last 10 years.