



---

# Foreword

---

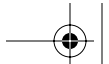
In my first professional programming gig, I was hired to add features to a bug database. This was for the Plant Pathology Department of the University of Minnesota farm campus, so by “bug” I mean actual bugs, for example, aphids, grasshoppers, and caterpillars. The code had been written by an entomologist who’d learned just enough dBase to write his first form and then duplicated it throughout the rest of the application. As I added features, I consolidated as much of the functionality as possible so that bug fixes (code bug fixes) could be applied in a single place, enhancements could be applied in a single place, and so on. It took me all summer, but by the end, I’d doubled the functionality while halving the size of the code.

Many, many years later, a friend of mine and I were hanging out with nothing pressing to do, so we decided to program something together (it was either an implementation of IDispatch or IMoniker, both of which weighed heavily on our minds at the time). I’d type for a while with him watching over my shoulder, telling me where I got it wrong. Then he’d take over the keyboard while I kibitzed until he relinquished control back to me. It went on for hours and was one of the most satisfying coding experiences I’ve ever had.

Not long after that, my friend hired me as the chief architect for the newly formed software division of his company. On many occasions, as part of my architecture work, I’d write the client code for objects that I wished would exist, which I’d pass along to the engineers, who would keep implementing until the client worked.

Like many kids who learned applied techniques in the back seat of a ’57 Chevy before sex education became a normal part of the curriculum, I’m guessing that my experiences experimenting with various aspects of agile development methodologies is not unique. In general, my experimenting with agile methods, like refactoring, pair programming, and test-driven development were successful, even though I didn’t quite know what I was doing. Of course, there have been agile materials available to me before this, but just as I’m unwilling to learn how to ask Suzy to the sock-hop from back issues of National Geographic, I’d like my agile technologies served up as appropriate for my peer-group, that is, .NET. By using .NET (even though he’s clear to say that .NET is no better than Java in many cases), Robert is speaking my language, just like those high school teachers that bothered to learn your slang, knowing that the message was more important than the medium.





But not just .NET; I'd like my first time to be gentle, to start slowly without scaring me, but to also make sure I get a grounding in all of the good stuff. And that's just what Robert "Uncle Bob" Martin has done with this book. His introductory chapters lay out the basics of the agile movement without pushing the reader towards SCRUM or Extreme Programming or any of the other agile methodologies, allowing the reader to join the atoms into the molecules that pleases them. Even better (and easily my favorite part of Robert's style) is when he shows these techniques in action, starting with a problem as it would be presented in a real-world environment and walks through it, showing the mistakes and missteps and how applying the techniques he advocates leads him back to safe ground.

I don't know if the world that Robert describes in this book really exists; I've only seen glimpses of it in my own life. However, it's clear that all of the "cool" kids are doing it. Consider "Uncle Bob" your own personal Dr. Ruth of the agile world whose only goal is that if you're going to do it, you do it well and make sure that everyone enjoys themselves.

Chris Sells

