Robert C. Martin: Clean Agile. Back to Basics

Patrick Bucher

Contents

1 Introduction to Agile

The *Agile Manifesto* originated from the gathering of 17 software experts in early 2001 as a reaction to heavy-weight processes like *Waterfall*. Since then, Agile was adopted widely and has been extended in various ways—unfortunately, not always in the spirit of the original idea.

1.1 History of Agile

The basic idea of Agile—working with small, intermediate goals and measuring the process—might be as old as civilization. Agile practices might also have been used in the early days of software development. However, the idea of *Scientific Management*, which is based on Taylorism, with its top-down approach and heavy planning, was prevalent in many industries at that time, conflicting with the *Pre-Agile* practice then prevalent in software development.

Scientific Management was suitable for projects with a high cost of change, a well-defined problem definition, and extremely specific goals. Pre-Agile practices, on the other hand, were a good fit for projects with a low cost of change, an only partially defined problem, and goals only being informally specified.

Unfortunately, there was no discussion at that time on which approach was the better fit for software projects. Instead, the Waterfall model—initially used as a straw man to be proven unsuitable by Winston Royce in his 1970 paper *Managing the Development of Large Software Systems*—was widely adopted in the industry. And Waterfall, with its emphasize on analysis, planning, and closely sticking to a plan, was a descendant of Scientific Management, not of Pre-Agile practices.

Waterfall dominated the industry since the 1970s for almost 30 years. Its subsequent phases of analysis, design, and implementation looked promising to developers working in endless "code and fix" cycles, lacking even the discipline of Pre-Agile.

What looked good on paper—and produced promising looking results in the analysis and design phases—often terribly failed in the implementation phase. Those problems, however, have been attributed to bad execution, and the Waterfall approach itself wasn't criticized. Instead, it became so dominant that new developments in the software industry like structured or object-oriented *programming* were soon followed by the disciplines of structured and object-oriented *analysis* and *design*—perfectly fitting the Waterfall mindset.

Some proponents of those ideas, however, started to challenge the Waterfall model in the mid-1990s, for example Grady Booch with his method of object-oriented design (OOD), the *Design Pattern* movement, and the authors of the *Scrum* paper. Kent Beck's *Extreme Programming* (XP) and *Test-Driven Development* (TDD) approaches of the late 1990s clearly moved away from Waterfall towards an Agile approach. Martin Fowler's take on *Refactoring* emphasizing continuous improvement certainly is a bad fit for Waterfall.

1.2 The Agile Manifesto

17 proponents of various agile ideas—Kent Beck, Robert C. Martin, Ward Cunningham (XP), Ken Schwaber, Mike Beedle, Jeff Sutherland (Scrum), Andrew Hunt, David Thomas ("Pragmatic Programmers"), among others—met in Snowbird, Utah in early 2001 to come up with a manifesto capturing the common essence of all those lightweight ideas. After two days, broad consensus was reached:

> We are uncovering better ways of developing software by doing it and helping others do it.
>
> - **Individuals and interactions** over processes and tools
> - **Working software** over comprehensive documentation
> - **Customer collaboration** over contract negotiation
> - **Responding to change** over following a plan

This *Agile Manifesto* was published after the gathering on agilemanifesto.org, where it still can be signed. The 12 Principles were written as a collaborative effort within the two weeks that followed the conference. This document explains and directs the four values stated in the manifesto; it shows, that those values have actual consequences.

1.3 Agile Overview

Many software projects are managed using approaches based on faith and motivational techniques. As a result, such projects are chronically late, despite

developers working overtime.

All projects are constrained by a trade-off called the *Iron Cross*: good, fast, cheap, done—pick three! Good project managers understand this trade-off and strive for results that are done good enough within an acceptable time frame and budget, which provide the crucial features.

Agile produces data that helps managers taking good decisions. The *velocity* shows the amount of points a development team finishes within an iteration. A *burn-down chart* shows the points remaining until the next milestone. The latter not necessarily shrinks at the rate of the velocity, because requirements and their estimations can change. Still, the burn-down chart's slope can be used to predict a likely date when the milestone is going to be reached. Agile is a feedback-driven approach. Even though the Agile manifesto doesn't mention velocity or burn-down charts, collecting such data and taking decisions based on it is crucial. Make that data public, transparent, and obvious.

A project's end date is usually given and not negotiable, often for good business reasons. The requirements, however, often change, because customers only have a rough goal, but don't know the detailed steps how to reach it.

1.4 A Waterfall Project

In the Waterfall days, a project was often split up into three pases of equal length: analysis, design, and implementation. In the analysis phase, requirements are gathered and planning is done. In the design phase, a solution is sketched and the planning is refined. Neither phase has hard and tangible goals; they are done when the end date of the phase is reached.

The implementation phase, however, needs to produce working software—a goal that is hard and tangible, and whose attainment is easy to judge. Schedule slips are only detected in this phase, and stakeholder only become aware of such issues when the project should be done almost finished.

Such projects often end in a *Death March*: a hardly working solution is produced after a lot of overtime work, despite deadlines being moved forward repeatedly. The "solution" for the next project usually is to do even more analysis and design—more of what didn't work in the first place (*Runaway Process Inflation*).