In his 1957 book *Parkinson’s Law, and Other Studies in Administration*, the naval historian and author C. Northcote Parkinson writes of a fictional committee meeting during which, after a two-and-a-half-minute nondiscussion on whether to build a nuclear reactor worth US $10 million, the members spend 45 minutes discussing the power plant’s bike shed, worth $2,350. From this he coined Parkinson’s Law of Triviality: “Time spent on any item of the agenda will be in inverse proportion to the sum involved.”

Using Parkinson’s example, the programmer Poul-Henning Kamp popularized the term *bikeshedding*: frequent, detailed discussions on a minor issue conducted while major issues are being ignored or postponed. The functional opposite of bikeshedding is *trystorming*, which refers to rapidly and repeatedly prototyping or implementing new products and processes. In a bikeshedding culture, ideas get only a short discussion before being put off “for further study.” In a trystorming culture, that same idea would be immediately prototyped, modeled, simulated, mocked up, or implemented, and examined to see what works and what doesn’t. The trystorming motto is “Fail early, fail fast, fail often.”

That *fail-fast* mantra also describes the process of Agile software development, a set of principles for making software that emphasizes an iterative, collaborative, and adaptive approach. In one Agile framework called *Scrum*, programmers are assigned *timeboxes*—also known as *iterations* or *sprints*—which are set time periods in which they focus on completing a predefined goal, which then becomes a *WIP*: a work in progress. One of the key elements of Agile is that these goals are almost always usable features of the final product (a process called *continuous integration*). Programmers work in *Scrum teams* under the supervision of a *Scrum master*, who runs daily *standups*, short meetings to review progress in which people physically stand rather than sit.

Another Agile framework is *Kanban*, where remaining tasks are visualized for all to see (*kanban* is Japanese for “billboard”), and coders “pull” tasks from this list rather than having tasks “pushed” to them. What if Kanban teams want to use some aspects of the Scrum methodology, such as standups? Then call this hybrid approach *Scruban*.

Kanban (and indeed all Agile methodologies) are very visual, with team leaders providing lots of big, visible charts depicting project metrics such as the *backlog* (showing the features remaining in the project) and either the *burn-down* (the rate at which the backlog is being cleared) or the *burnup* (the rate at which project tasks are being completed). Some teams even use a *nikoniko calendar* (*nikoniko* is Japanese for “smile”), where at the end of each day team members display a visual indication of their mood (hence the alternative name *mood board*). All such charts and boards are often called *information radiators*.

Agile teams are big into *pair programming*, where two programmers sit at the same workstation, with one operating the keyboard and mouse and the other providing general direction and on-the-fly code reviews. These are known as the *driver* and the *navigator*, respectively, and they typically swap roles frequently. Both programmers are expected to provide running commentary throughout the process, which is known as *programming out loud*. A variation on that theme is *mob programming*, where the role of the navigator is assumed by the entire software team. A similar idea is *swarming*, where multiple team members jump into an in-progress task to help complete the work.

The main aim of Agile is to create happy customers by building great products. But look closer and you see a humane streak running through all of Agile. Teams seek a *sustainable pace*, meaning a work rate—sometimes called a *velocity*—that can be kept up indefinitely.