In the late 1970s, the Smalltalk team at Xerox PARC developed the Model/ View/Controller (MVC) concept. The idea is to separate an application’s data from the presentation of the data. The code to display the data doesn’t mix with the code to compute the data.

I’ve seen this principle in action in my very own home. When my son was in fifth grade, he started writing school reports by using Microsoft Word. I’d watch him type a sentence and then adjust the margins. Then he’d prepare to center the next sentence and fish around for a fancier font. By the time he found the right formatting menus, he’d forgotten what he wanted to say. With this mixing of formatting and content, he couldn’t devote full attention to the ideas he was trying to express. I advised him to separate the content and presentation tasks — that is, to write the words first and then go back to format the paragraphs in the report.

In MVC terms, an application’s data is called the model, and the presentation of the data is called the view. The model belongs in one part of the code, and the view belongs in another. For a very large project, one team of programmers writes code to develop the model, and another team writes code to develop the view. Each team applies its expertise to the task. More importantly, each team worries about its specific problems. The team that computes the millionth digit of pi doesn’t worry about the font used to display that digit. And the team that shapes a neon tube into the digit 1 doesn’t worry about the formula used to calculate that millionth digit of pi. When a company modernizes its equipment (going from an old neon sign to a modern LED display), the calculation team doesn’t want to compute the digits of pi all over again. The millionth digit doesn’t change, no matter how the company chooses to display that digit. So questions about the calculation of the data and the display of the data should be separate from one another.

Models and views are passive things. Each of these things sits around waiting for someone to request its services. So to do anything useful, you need more than a model and a view. You need something that says, “I just got a request to display today’s weather forecast. I’ll get the forecast from the model and tell the view to display the forecast.” The thing that says all this is called a *controller.* A controller is the mover and shaker in the MVC architecture.

The controller waits for a user request (a button click, or something like that). When a request arrives, the controller gets some required data from the model or tells the model to modify some data. Then the controller fires up the view. The view displays the data in a way that keeps the user happy.