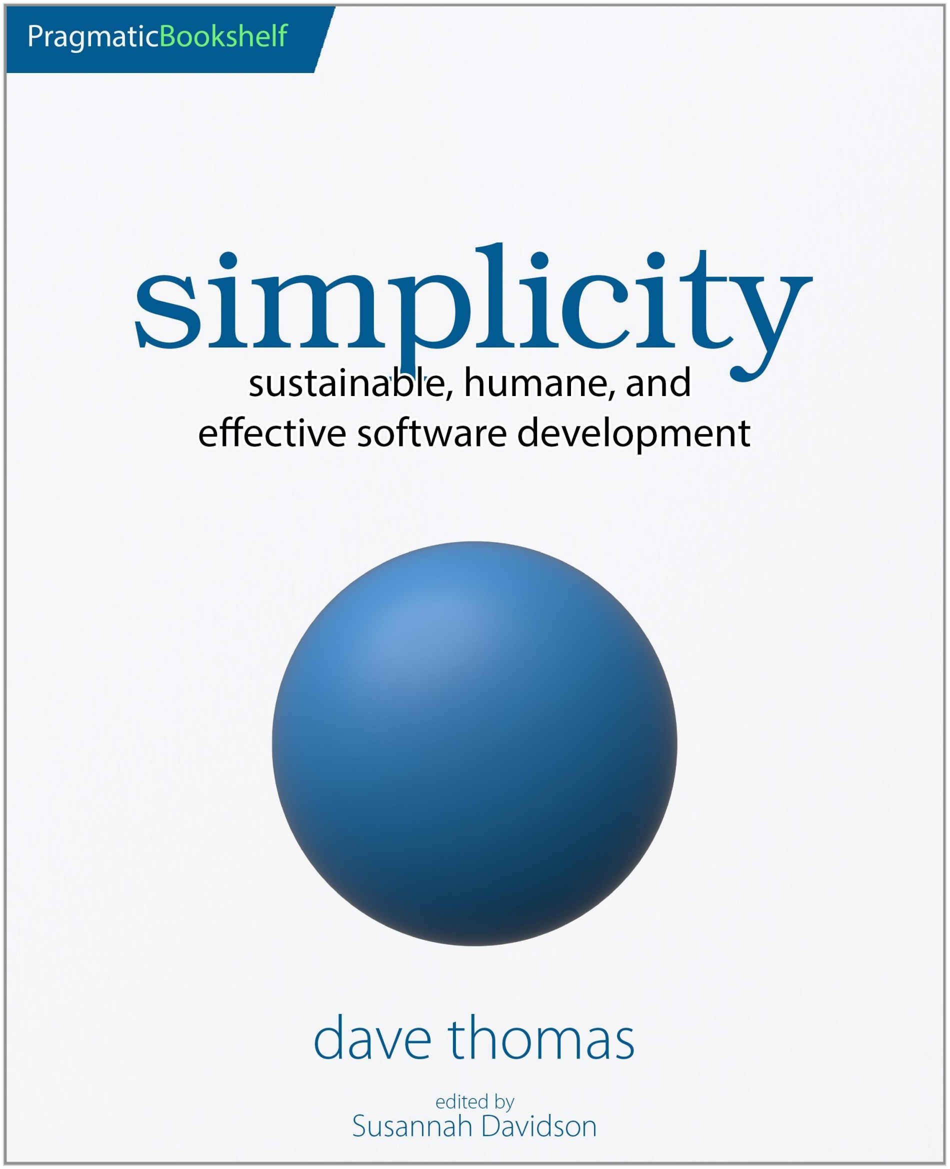

PragmaticBookshelf

simplicity

sustainable, humane, and effective software development

dave thomas

edited by
Susannah Davidson

Agility On An Index Card

Here's how I try to develop software.

- I work in the smallest increments I can, using feedback at all levels to assess the value being produced.
- I constantly adapt what I do, but only after I've decided how to tell if the impact of the change was good or bad.
- I deliver value incrementally, as soon as I have something new to show my users. That way I can use their feedback to guide my next steps.

I'm not always successful. In 2023/4 I wasted nine months because I went heads down on a complex project. I got caught up in the technicalities and forgot to deliver incrementally. I'm still learning. But the vast majority of the time, I stick to these principles, and it works well.

Break It Down

Here's how I apply these principles. There are just three activities.

1: Orient

- I find out where I am.

2: Step

- I take the smallest possible step towards where I want to be, doing the least amount of work possible to generate something that gives me some feedback.

3: Learn

- I assess whether what I did produced the expected value.
- I adjust my understanding based on the feedback.
- And I go back to the first activity, and reorient myself.

Honestly, there's nothing special about this; it's really just common sense. But it only works if you consciously follow the three steps.

Idea 4 **All Of agility Comes Down to Orient, Step, Learn**

Here's an example. This afternoon I had a call with Susannah, my wonderful editor. She mentioned that she could no longer build my book; she got a "file not found" error. Every other book built just fine for her. She sent me the console log. (I think it's so cool that my editor uses the command line…)

I went through the three activities in a cycle. (And, yes, I'm laboring the point a little. Bear with me.)

Orient #1
: I tried building my book on my machine and had no problems, so the issue was likely a difference between her environment and mine. One thing she mentioned was that she stored all her book projects in a directory with a space in the path.

Step #1
: We've had problems with paths like this in the past, but we fixed them and haven't seen any issues for years. But, in lieu of any better idea, the simplest thing would be to check the book out into a path that contained spaces on my machine and see if it worked. It failed just as hers had done.

Learn #1
: Despite my strong belief going in to this, it *was* the space in the path causing the problem. Time to reorient my thinking. Back to step 1.

Orient #2
: I knew the book built successfully on her machine last week, but not this week, so I was probably looking for something I'd messed up over the weekend. But I wasn't aware of changing anything to do with paths.

Step #2
: The simplest thing was to scan the change logs. I saw no modifications to any of the actual tools, but I did see that I'd bumped the version of JRuby that we use.

Learn #2
: Given that this had to be related to code, and the only code that changed was the third party JRuby compiler, I had a suspect.

Orient #3
: Over the years I've taught myself that when something breaks, 99.999% of the time it's my fault, and not a bug in some third party code. So the orientation here was to convince myself that a quick test was needed.

Step #3
: I tried running `jruby -v` and it also failed.

and so on…

In the end, it was a bug in the `jruby` script in the newly updated package. I rolled back to the previous version and reported it to the JRuby team on GitHub. Charles Nutter acknowledged it within a couple of hours, and a patch was available a few hours later (which is seriously agile).

But Dave…

I know what you're thinking: *"That's nothing special. That how everyone would attack an issue like this."*

Maybe—but not in my experience. I've given many classes to all levels of developer, and most people do not take a methodical approach to this kind of problem. I have sympathy, because neither did I.

Also, remember that these three steps apply to everything I do, from naming functions though picking a technology for our new website: *Orient, Step, Learn, repeat.*

A Larger Example

The publishing industry is mired in the Victorian times. All information is exchanged in spreadsheets, with people at one end manually creating them and people at the other end transcribing information into some 1990s mainframe system. We receive monthly distributor statements this way, but we process them in code.

A few years back, our international distributor started creating a different monthly spreadsheet. We adjusted our ingestion code to match. A couple of months later, it failed: they'd renamed some columns, and added a few new ones. This triggered an Orient, Step, learn cycle.

During orientation, we determined that this was likely to be an ongoing problem, and that we'd likely have to adjust our import now and then going forward. We also knew that we had to retain the ability to reimport all historical data, so our code would have to add new CSV formats while still accepting the old. This mean that we'd need to decouple to reading of the raw data from the processing of that data. A pipeline was born.

Step one was to make that split in the code explicit and run regression tests to make sure nothing had changed. Not much to learn, so we went back to the orientation step.

We had to deal with two types of change: the order (and number) of columns could be different, and the column names would often change (sometimes with typos, so it was clearly created by hand every month). We needed to decouple the finding of the data we needed from the layout of the spreadsheet.

Step two was to replace the code that analyzed the spreadsheet with a table that mapped column names to internal variables so that the internal value `total_sales` could be extracted from a column called `Sales` one month and `Sale total` the next. Again, we ran regressions to verify the step.

Orienting again, we needed to make sure that we could read all historical imports while adding new layouts.

The next step was to use the tables we'd defined in the previous cycle to pattern match against the incoming spreadsheet. If the column names and locations matched a table, we'd use that for the extraction. If no table matched, we'd abandon the import and someone would have to create a new mapping table.

Again, we ran regression tests, this time against the accumulated months of data. At this point we'd learned all we could until new data arrived.

Over the following months, the distributor managed to throw a few more surprises at us: columns with the same names as before but with different values, different formats for currency values, and so on. We used each of these to trigger a new Orient, Step, Learn cycle. Often we'd find that we'd spend more time on the Orient and Learn steps with these changes, as we tried to find patterns in the changes, and tried to build in more safeguards to ensure we weren't misinterpreting the data.

Over the last 36 months, there have been 15 different formats sent to us. Sometimes we'll get the same format for three months in a row, and sometimes a format from the past will suddenly reappear.

I draw two lessons from this. First, the initial orientation was critical; working out that we needed backwards compatibility informed all the decisions we subsequently made. In particular, splitting the code into separate parsing and processing steps turned out to be the right step.

The second lesson was the benefits of the small steps. I am absolutely paranoid about royalty calculations: getting authors and editors paid accurately for their hard work is critical to me. When dealing with something as uncertain as this feed, taking lots of small steps, and running full regression tests at each, gave me the confidence to deploy these changes.

The Benefits

It's a simple plan, but *Orient-Step-Learn* is the fundamental essence of agility. It makes me conscious of the changes I make and the feedback I need when making them.

I find there are so many benefits to following this ritual:

- *Pausing to orient myself* before attacking my keyboard saves my butt many times a day. I realize I was about to solve the wrong problem, or that I'd already solved the same problem elsewhere, or that I didn't actually understand what I was about to do.
- *Being deliberate about feedback* focusses what I do in each step; I only do work that's contributing to generating a successful result.
- I give myself *permission to experiment and fail*, knowing that the cost is minimal because I've taken just a small step.

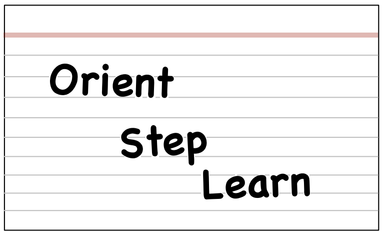
Idea 5 **Pause, Then Act**

It can also be a difficult and sometimes painful plan to implement. It means slowing down and giving up the illusion of control. (Hint: we never really had control in the first place.)

It means constantly experimenting, and constantly using those experiments to improve what you do and how you do it. It means getting used to the idea that failing is an integral part of succeeding.

Your Turn

Please try organizing your work into these three practices. It's a purely personal thing: you don't need permission to do it. It's simply a way of organizing how you think about what you do.



In practice, it's probably less work that you're doing now, and it's more productive. But it also requires that you build a set of habits, so that you naturally think about feedback when you both start and finish anything. And, as with any habit, this means that initially you'll need to do it consciously and deliberately. Maybe write on an index card and tape it to your screen.

And, obviously, use your daybook to record all of this. (And if you aren't using a daybook, perhaps have a look at this article[1] before continuing.) At first, record every decision you make, and go back when you finished that particular step and record the feedback; a tick if it worked and an angry X if not.

And, just to keep you on the straight and narrow, never start something until you've established what feedback you'll use to decide whether it was successful.

The rest of this chapter is a deeper dive into Orient, Step, and Learn.

Try this

Write "Orient, Step, Learn" on something and stick it in your line of sight while working. Or, stick it in a file and make that the default buffer your editor shows. Or decorate your desktop. Maybe tattoo it on the inside of your eyelids.

Do whatever it takes to remind yourself to be deliberate as you work, consciously taking the three steps as you move forward.

(This is going to sound silly, but you might try saying "Orient" aloud as you start an activity, then "Step" when you've decided what to to, and then "Learn" when it's done. I do this all the time when I'm trying to adjust my habits; I find that vocalizing seems to engage a different part of my brain, and doing that speeds my adoption.)

Think about

Do you suppose this idea might apply to things other than software development?

1. https://pragdave.substack.com/p/why-you-need-a-daybook