Extracted from:

Pragmatic Thinking and Learning

Refactor Your "Wetware"

We can't solve problems by using the same kind of thinking we used when we created them.

► Albert Einstein

Chapter 2

Journey from Novice to Expert

Wouldn't you like to be the expert? To intuitively *know* the right answer? This is the first step of our journey together along that road. In this chapter, we'll look at what it means to be a novice and what it means to be an expert—and all the stages in between. Here's where our story begins.

Once upon a time, two researchers (brothers) wanted to advance the state of the art in artificial intelligence. They wanted to write software that would learn and attain skills in the same manner that humans learn and gain skill (or prove that it couldn't be done). To do that, they first had to study how humans learn.

They developed the *Dreyfus model* of skill acquisition,[1] which outlines five discrete stages through which one must pass on the journey from novice to expert. We'll take a look at this concept in depth; as it turns out, we're not the first ones to use it effectively.

Back in the early 1980s, the nursing profession in the United States used the lessons of the Dreyfus model to correct their approach and help advance their profession. At the time, the problems faced by nurses mirrored many of the same problems programmers and engineers face today. Their profession has made great progress, and in the meantime we still have some work to do with ours.

1. Described in *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer* [DD86].

Event Theories vs. Construct Theories

The Dreyfus model is what's called a *construct* theory. There are two types of theories: *event* theories and *construct* theories.* Both are used to explain some phenomenon that you've observed.

Event theories can be measured; these types of theories can be verified and proven. You can judge the accuracy of an event theory.

Construct theories are intangible abstractions; it makes no sense to speak of "proving them." Instead, construct theories are evaluated in terms of their usefulness. You can't judge a construct theory to be accurate or not. That's mixing apples and existentialism. An apple is a thing; existentialism is an abstraction.

For instance, I can prove all sorts of things about your brain using simple electricity or complex medical imaging devices. But I can't even prove you have a mind. *Mind* is an abstraction; there's really no such thing. It's just an idea, a concept. But it's a very useful one.

The Dreyfus model is a construct theory. It's an abstraction, and as we'll see, it's a very useful one.

*. See *Tools of Critical Thinking: Metathoughts for Psychology* (Lev97).

Here are some observations that ring true for both nurses and programmers, and probably other professions as well:

- Expert staff members working in the trenches aren't always recognized as experts or paid accordingly.
- Not all expert staff want to end up as managers.
- There's a huge variance in staff members' abilities.
- There's a huge variance in managers' abilities.
- Any given team likely has members at widely different skill levels and can't be treated as a homogeneous set of replaceable resources.

There's more to skill levels than just being better, smarter, or faster. The Dreyfus model describes how and why our abilities, attitudes, capabilities, and perspectives change according to skill level.

Figure 2.1: A Unix wizard

It helps explain why many of the past approaches to software development improvement have failed. It suggests a course of action that we can pursue in order to meaningfully improve the software development profession—both as individual practitioners and for the industry as a whole.

Let's take a look.

2.1 Novices vs. Experts

What do you call an expert software developer? A *wizard*. We work with magic numbers, things in hex, zombie processes, and mystical incantations such as `tar -xzvf plugh.tgz` and `sudo gem install --include-dependencies rails`.

We can even change our identity to become someone else or transform into the root user—the epitome of supreme power in the Unix world. Wizards make it look effortless. A dash of eye of newt, a little bat-wing dust, some incantations, and poof! The job is done.

Making It Look Easy

I once was in a position to interview professional organists. For an audition piece, I chose Charles-Marie Widor's "Toccata" (from Symphony No. 5 in F Minor, Op. 42 No. 1, for those who care about such things), a frenetic piece that sounded suitably difficult to my amateur ears.

One candidate really worked it—both feet flying on the pedals, hands running up and down both ranks of the organ in a blur, a stern look of intense concentration across her brow. She was practically sweating. It was a terrific performance, and I was suitably impressed.

But then came along the true expert. She played this difficult piece a little bit better, a little bit faster, but was smiling and *talking* to us while her hands and feet flew in an octopus-like blur.

She made it look easy, and she got the job.

Despite the mythological overtones, this vision is fairly common when considering an expert in any particular field (ours is just arcane enough to make it a really compelling image).

Consider the expert chef, for instance. Awash in a haze of flour, spices, and a growing pile of soiled pans left for an apprentice to clean, the expert chef may have trouble articulating just *how* this dish is made. "Well, you take a bit of this and a dash of that—not too much—and cook until done."

Chef Claude is not being deliberately obtuse; he knows what "cook until done" means. He knows the subtle difference between just enough and "too much" depending on the humidity, where the meat was purchased, and how fresh the vegetables are.

It's hard to articulate expertise.

It's often difficult for experts to explain their actions to a fine level of detail; many of their responses are so well practiced that they become preconscious actions. Their vast experience is mined by nonverbal, preconscious areas of the brain, which makes it hard for us to observe and hard for them to articulate.

When experts do their thing, it appears almost magical to the rest of us—strange incantations, insight that seems to appear out of nowhere, and a seemingly uncanny ability to know the right answer when the rest of us aren't even all that sure about the question.

It's not magic, of course, but the way that experts perceive the world, how they problem solve, the mental models they use, and so on, are all markedly different from nonexperts.

A novice cook, on the other hand, coming home after a long day at the office is probably not even interested in the subtle nuances of humidity and parsnips. The novice wants to know *exactly* how much saffron to put in the recipe (not just because saffron is ridiculously expensive).

The novice wants to know *exactly* how long to set the timer on the oven given the weight of the meat, and so on. It's not that the novice is being pedantic or stupid; it's just that novices need clear, context-free rules by which they can operate, just as the expert would be rendered ineffective if he were constrained to operate under those same rules.

Novices and experts are fundamentally different. They see the world in different ways, and they react in different ways. Let's look at the details.

2.2 The Five Dreyfus Model Stages

In the 1970s, the brothers Dreyfus (Hubert and Stuart) began doing their seminal research on how people attain and master skills.

Dreyfus is applicable per skill.

The Dreyfus brothers looked at highly skilled practitioners, including commercial airline pilots and world-renowned chess masters.[2] Their research showed that quite a bit changes as you move from novice to expert. You don't just "know more" or gain skill. Instead, you experience fundamental differences in how you perceive the world, how you approach problem solving, and the mental models you form and use. How you go about acquiring new skills changes. External factors that help your performance—or hinder it—change as well.

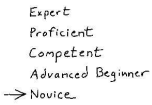
2. Cited in *From Novice to Expert: Excellence and Power in Clinical Nursing Practice* [Ben01].

Unlike other models or assessments that rate the whole person, the Dreyfus model is applicable per skill. In other words, it's a situational model and not a trait or talent model.

You are neither "expert" nor "novice" at all things; rather, you are at one of these stages in some particular skill domain. You might be a novice cook but an expert sky diver, or vice versa. Most nondisabled adults are experts at walking—we do so without planning or thinking. It has become instinct. Most of us are novices at tax preparation. We can get through it given a sufficient number of clear rules to follow, but we really don't know what's going on (and wonder why on Earth those rules are so arcane).

The following are the five stages on the journey from novice to expert.

Stage 1: Novices



Novices, by definition, have little or no previous experience in this skill area. By "experience," I mean specifically that performing this skill results in a change of thinking. As a counterexample, consider the case of the developer who claims ten years of experience, but in reality it was one year of experience repeated nine times. That doesn't count as *experience*.

Novices are very concerned about their ability to succeed; with little experience to guide them, they really don't know whether their actions will all turn out OK. Novices don't particularly want to learn; they just want to accomplish an immediate goal. They do not know how to respond to mistakes and so are fairly vulnerable to confusion when things go awry.

They can, however, be somewhat effective if they are given context-free rules to follow, that is, rules of the form "Whenever X happens, do Y." In other words, they need a recipe.

Novices need recipes.

This is why call centers work. You can hire a large number of folks who don't have a lot of experience in the subject matter at hand and let them navigate a decision tree.

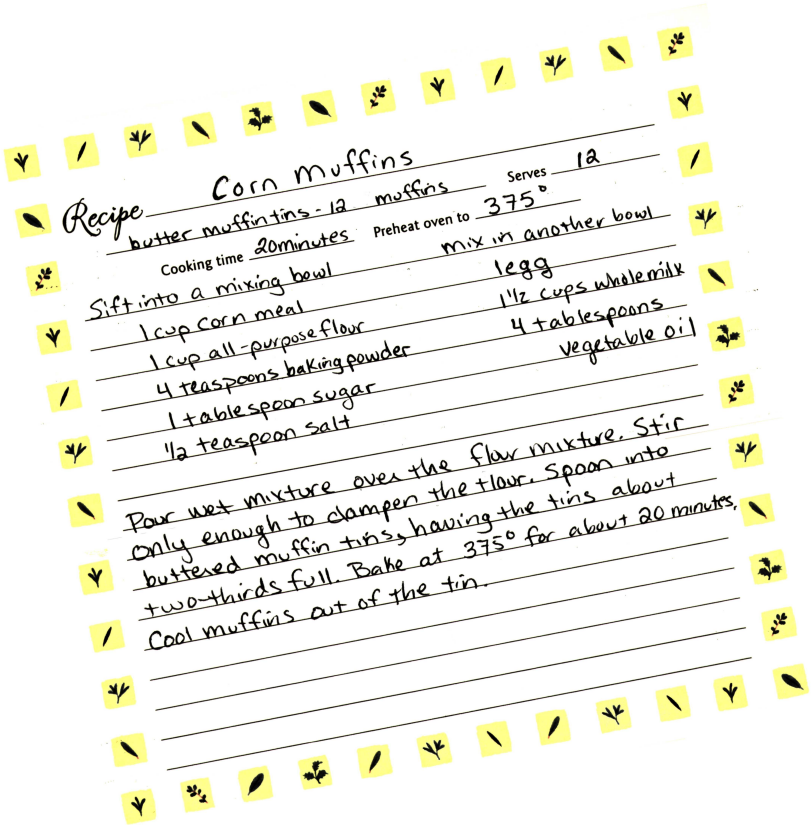


Figure 2.2: Recipe for corn muffins. But how long do you cook it?

A giant computer hardware company might use a script like this:

1. Ask the user whether the computer is plugged in.
2. If yes, ask whether the computer is powered on.
3. If no, ask them to plug it in and wait.
4. *and so on...*

It's tedious, but fixed rules such as these can give novices some measure of capability. Of course, novices face the problem of not knowing *which* rules are most relevant in a given situation. And when something unexpected comes up, they will be completely flummoxed.

As with most people, I am a novice when it comes to doing my taxes. I have little experience; despite having filed taxes for more than twenty-five years, I haven't learned anything or changed my thinking about it. I don't want to learn; I just want to accomplish the goal—to get them filed this year. I don't know how to respond to mistakes; when the IRS sends me a terse and rather arrogant form letter, I usually have no idea what they're on about or what to do to fix it.[3]

There is a solution, of course. A context-free rule to the rescue! Perhaps it's something such as the following:

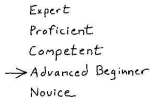
- Enter the amount of money you earned last year.
- Send it in to the government.

That's simple and unambiguous.

The problem with recipes—with context-free rules—is that you can never specify everything fully. For instance, in the corn muffin recipe, it says to cook for "about 20 minutes." When do I cook longer? Or shorter? How do I know when it's done? You can set up more rules to explain, and then more rules to explain those, but there's a practical limit to how much you can effectively specify without running into a Clinton-esque "It depends upon what the meaning of the word *is* is." This phenomenon is known as *infinite regression*. At some point, you have to stop defining explicitly.

Rules can get you started, but they won't carry you further.

Stage 2: Advanced Beginners



Once past the hurdles of the novice, one begins to see the problems from the viewpoint of the *advanced beginner*. Advanced beginners can start to break away from the fixed rule set a little bit. They can try tasks on their own, but they still have difficulty troubleshooting.

They want information fast. For instance, you may feel like this when you're learning a new language or API and you find yourself scanning the documentation quickly looking for that one method signature or set of arguments. You don't want to be bogged down with lengthy theory at this point or spoon-fed the basics yet again.

3. I forward it with my compliments and a large check to my accountant, who is expert in these matters. I hope.

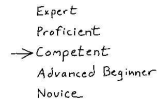
Advanced beginners can start using advice in the correct context, based on similar situations they've experienced in the recent past but just barely. And although they can start formulating some overall principles, there is no "big picture." They have no holistic understanding and really don't want it yet. If you tried to force the larger context on an advanced beginner, they would probably dismiss it as irrelevant.

Advanced beginners don't want the big picture.

You might see this sort of reaction when the CEO calls an all-hands meeting and presents charts and figures showing sales projections and such. Many of the less experienced staff will tend to dismiss it as not being relevant to their individual job.

Of course, it is very relevant and can help determine whether you'll still *have* a job with this company next year. But you won't see the connection while you're at the lower skill levels.

Stage 3: Competent



At the third stage, practitioners can now develop conceptual models of the problem domain and work with those models effectively. They can troubleshoot problems on their own and begin to figure out how to solve novel problems—ones they haven't faced before. They can begin to seek out and apply advice from experts and use it effectively.

Competents can troubleshoot.

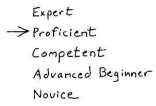
Instead of following the sort of knee-jerk response of the previous levels, the *competent* practitioner will seek out and solve problems; their work is based more on deliberate planning and past experience. Without more experience, they'll still have trouble trying to determine which details to focus on when problem solving.

You might see folks at this level typically described as "having initiative" and being "resourceful." They tend to be in a leadership role in the team (whether it's formal or not).[4] These are great folks to have on your team. They can mentor the novices and don't annoy the experts overly much.

4. See *Teaching and Learning Generic Skills for the Workplace* [SMLR90].

In the field of software development, we're getting there, but even at this level, practitioners can't apply agile methods the way we would like—there isn't yet enough ability for reflection and self-correction. For that, we need to make a breakthrough to the next level: proficient.

Stage 4: Proficient



Proficient practitioners need the big picture. They will seek out and want to understand the larger conceptual framework around this skill. They will be very frustrated by oversimplified information.

For instance, someone at the proficient stage will not react well when they call the tech support hotline and are asked whether it's plugged in. (Personally, I want to reach through the phone and remove the first vital organ that presents itself in these situations.)

Proficient practitioners can self-correct.

Proficient practitioners make a major breakthrough on the Dreyfus model: they can correct previous poor task performance. They can reflect on how they've done and revise their approach to perform better the next time. Up until this stage, that sort of self-improvement is simply not available.

Also, they can learn from the experience of others. As a proficient practitioner, you can read case studies, listen to water-cooler gossip of failed projects, see what others have done, and learn effectively from the story, even though you didn't participate in it firsthand.

Along with the capacity to learn from others comes the ability to understand and apply *maxims*, which are proverbial, fundamental truths that can be applied to the situation at hand.[5] Maxims are not recipes; they have to be applied within a certain context.

For instance, a well-known maxim from the extreme programming methodology tells you to "test everything that can possibly break."

5. See *Personal Knowledge* [Pol58].

Pragmatic Tips

When Dave Thomas and I wrote the original *The Pragmatic Programmer,* we were trying to convey some of the advice we thought was most relevant to our profession.

These tips—these maxims—were a reflection of our collective years of expertise. From the mind-expanding practice of learning a new language every year to the hard-won principles of Don't Repeat Yourself (DRY) and No Broken Windows, maxims such as these are key to transferring expertise.

To the novice, this is a recipe. What do I test? All the setter and getter methods? Simple print statements? They'll end up testing irrelevant things.

But the proficient practitioner knows what can possibly break—or more correctly, what is likely to break. They have the experience and the judgment to understand what this maxim means *in context*. And context, as it turns out, is key to becoming an expert.

Proficient practitioners have enough experience that they know—from experience—what's likely to happen next; and when it doesn't work out that way, they know what needs to change. It becomes apparent to them which plans need to be discarded and what needs to be done instead.

Similarly, software Patterns (as espoused in *Design Patterns: Elements of Reusable Object-Oriented Software* [GHJV95], also known as the Gang of Four book) can be effectively applied by proficient-level practitioners (but not necessarily at lower skill levels; see the sidebar on the next page).

Now we're getting somewhere. Proficient practitioners can take full advantage of the reflection and feedback that is core to agile methods. This is a big leap from the earlier stages; someone at the proficient stage is much more like a junior expert than a really advanced competent.

Misapplied Patterns and Fragile Methods

As you may realize by now, some of the most exciting new movements in the software development community are targeted at proficient and expert developers.

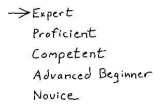
Agile development relies on feedback; in fact, my definition of *agile development* from *Practices of an Agile Developer: Working in the Real World* (SH06) says this: "Agile development uses feedback to make constant adjustments in a highly collaborative environment." But being able to self-correct based on previous performance is possible only at the higher skill levels.

Advanced beginners and competent practitioners often confuse software design patterns with recipes, sometimes with disastrous results. For instance, I once knew a developer on a project who had just been exposed to the Gang of Four (GoF) book. In his enthusiasm, he wanted to start using design patterns. All of them. At once. In a small piece of report-writing code.

He managed to jam in about seventeen of the twenty-three GoF patterns into this hapless piece of code before someone noticed.

Stage 5: Expert

Finally, at the fifth stage, we come to the end of the line: the expert.



Experts are the primary sources of knowledge and information in any field. They are the ones who continually look for better methods and better ways of doing things. They have a vast body of experience that they can tap into and apply in just the right context. These are the folks who write the books, write the articles, and do the lecture circuit. These are the modern wizards.

Statistically, there aren't very many experts—probably something on the order of 1 to 5 percent of the population.[6]

6. See *Standards for Online Communication* [HS97].

Experts work from intuition.

Experts work from *intuition*, not from reason. This has some very interesting ramifications and raises some key questions—what is intuition, anyway? (We'll delve more into the details of intuition throughout the book.)

Although experts can be amazingly intuitive—to the point that it looks like magic to the rest of us—they may be completely inarticulate as to how they arrived at a conclusion. They genuinely don't know; it just "felt right."

For instance, suppose a physician looks in at a patient. At a glance, the doctor says, "I think this patient has Blosen-Platt syndrome; better run these tests." The staff runs the tests, and indeed, the doctor is correct. How did she know? Well, you could ask, but the doctor may well reply with "He didn't look right."

Indeed, the patient just didn't look "right." Somehow, in the vast array of experiences, distilled judgment, memories, and all the rest of the mental effluvia in the doctor's brain, a particular combination of subtle clues in the patient came together and suggested a diagnosis. Maybe it was the skin pallor or the way the patient was slumped over—who knows?

The expert does. The expert knows the difference between irrelevant details and the very important details, perhaps not on a conscious level, but the expert knows which details to focus on and which details can be safely ignored. The expert is very good at targeted, focused pattern matching.

2.3 Dreyfus at Work: Herding Racehorses and Racing Sheep

Now that we've looked at the Dreyfus model in detail, let's see how to apply the Dreyfus lessons at work. In software development at least, it turns out that we tend to apply them pretty poorly.

Experts aren't perfect. They can make mistakes just like anyone else, they are subject to the same cognitive and other biases that we'll look at later (in Chapter 5, *Debug Your Mind*, on page 124), and they will also likely disagree with one another on topics within their field.

But worse than that, by misunderstanding the Dreyfus model, we can rob them of their expertise. It's actually easy to derail an expert

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Pragmatic Thinking and Learning's Home Page
http://pragprog.com/titles/ahptl
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/ahptl.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragprog.com/catalog
Customer Service:	orders@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com