Editors: Andy Hunt and Dave Thomas ■ The Pragmatic Programmers
andy@pragmaticprogrammer.com ■ dave@pragmaticprogrammer.com

Preparing the Raw Material

Andy Hunt and Dave Thomas

Rotten wood cannot be carved, nor are dung walls plastered. —Confucius

Any construction project begins with raw material, and as Confucius suggests, the nature of the raw material is critical to success—so much so that you shouldn't even begin if the "wood" is poor. Even if you have sharp, finely honed tools, your project will still fail if the raw material isn't sound.

And what, might you ask, is the raw material of software development?

Us.

People. We are the only raw material of consequence in software development. Oh sure, the process will involve a few keyboards and mice, some compilers, database products, and myriad office supplies, but they're all completely secondary. Contrary to popular myth, we don't write software on computers. We don't write software in programming languages, integrated development environments or case tools, whiteboards, or 3 × 5 cards.

We write software in our heads.

It's all in your head

Ideas, plans, requirements, domain expertise, and all those other pesky little memes somehow get into our heads and ferment into a vision that we jointly and individually express as working software. Here, the construction analogy breaks down a bit, because we are not only the raw material but most of the manufacturing process as well. Technology offers some help, but make no mistake: the playing field is in our own heads. We are the raw material—our attitudes, abilities, capabilities, and even emotional states.

So how do we prepare this material? Obviously, we don't want it to be "rotten," but how can we tell? How can we tell if hidden voids lurk beneath the surface, just waiting to ruin the project once we start carving?

When you lack the right material, you'll keenly feel its absence. For example, warning signs might include

- The developer who only uses one favorite solution for every problem
- Folks who don't learn from mistakes—or worse, are too afraid to make any
- The developer who can't be bothered to tell anyone what he's doing or why

Fortunately, unlike real wood, we can repair our personal wood if the spirit is willing.

Dave and I think you'll always find certain qualities in the best raw material. Moreover, you can learn, expand, and improve all of them in yourself and help your teammates do the same.

Good developers ...

Have a pragmatic outlook

The best solution is the one that works well for the particular problem at hand, in a timely and cost-effective manner. No two software projects are ever alike: just because a technique or product worked well once, you can't guarantee it will work as well again in a different context. The pragmatic razor is simple: it's only working if it's working.

The best solution doesn't come from the "reading edge" of a textbook, course, or magazine. Those offer a good start but no feedback. Applying feedback is critical to pragmatism. It's the only way to know if you're on the right track—the only way to know whether "it's working" or not.

Feedback comes from hands-on experience: from getting your hands dirty, trying your idea out under actual conditions, and filing away the results. It also helps to be a pack rat for little facts, ranging from how to implement symbolic links to knowing the best time of day to ask an end user a question.

See multiple perspectives

Always look beyond the immediate problem and try to place it in its larger context; try to be aware of the bigger picture. After all, without the big picture, how can you be pragmatic? How can you make intelligent compromises and informed decisions?

The real world is not a rational, elegantly modeled, context-free place. It's messy, filled with ambiguity, uncertainty, and of course, context—a critical element we often taken for granted.

For example, consider the word *bachelor*. Merriam-Webster's marriage-related definition for this says "an unmarried man." But to most people, it means a lot more than that—the Pope is an unmarried man, but few would call him a bachelor. Nor do we use the term for someone involved in a long-term unmarried relationship, even though it's technically appropriate. The word's social context gives it its real meaning, which is closer to "an unmarried man of eligible age and circumstances who is not yet married but should be." Boy, there's a lot of context there. Put this same word in a different language or cultural context and you'll open a whole 'nother can of worms.

Part of seeing multiple perspectives is fully appreciating the business context in which programmers operate. Cool technology has its own rewards, but delivering business value is what keeps you employed or keeps clients coming back. Delivering software that works and works well in a business context is key to a sustainable career.

Take responsibility

Painful as it is to admit it, for most people, most of the time, the problems that come up are our own fault—not the compiler's, not the OS's, not the database vendor's, not our bosses', and not our coworkers'. Yet many people and teams, when facing a disastrous problem, first embark on a search (aka witch-hunt) to fix the blame.

It goes somewhat against human nature, but you should always try to fix the problem, not the blame. Remember, we all write software in our heads, so it makes sense to go ahead and take responsibility for it. "The cat ate my source code" just doesn't cut it anymore—when problems come up, we need to invent options, not excuses.

Communicate well

Programmers are information hubs. We communicate with everyone and everything, from customers, end users, and teammates to the machine itself. John Donne's pithy "No man is an island" holds especially true for software developers. We must communicate well for effective results.

The isolated, lone developer huddled over a terminal with a can of highly caffeinated cola offers not only an inaccurate and misleading stereotype but a dangerous one as well. Isolated developers can inadvertently duplicate other teammates' work, use outdated or inappropriate methods, and even build the wrong product—at least, not the product the sponsor wanted.

Instead of taking more Java and UML courses, we should work on our technical writing, public speaking, and group facilitation.

A CPU with no I/O isn't very useful. Neither is a developer who can't communicate.

Learn continuously

On a par with communication is continuous learning, which means learning about more than just the technology involved. The technology alone poses a huge challenge; it's like taking a sip from the proverbial fire hose. But the need for learning extends to the problem domain, about how the team works together (or how it doesn't), and about ourselves.

Learning continuously about ourselves is harder than it sounds; at first, most people aren't comfortable constantly critiquing and appraising their own work. But of course, you need to do exactly that to improve or even to just keep up. It doesn't happen overnight or by accident. You have to deliberately, consciously plan to learn, apply feedback, and improve to remain competitive.

So, what does this have to do with software construction? Everything! An investment in preparing the raw material is always repaid. Because just as in real construction, any problems in the raw material will become problems in the project. If you unknowingly build using rotten wood or on unprepared ground, the cost of going back and making repairs can be astronomically high.

You might find workarounds for the smaller problems—paint or patch the rotten spot, or use it where it's not structurally critical. But you have to know a problem exists before you can work around it.

Unfortunately many of these qualities are hard to measure directly. We don't have certification programs in pragmatism or responsibility, for instance. But that's okay, because degrees, certifications, titles, and the lot aren't reliable indicators of developer performance—of the raw material's quality. Only one thing is, as Confucius reminds us (emphasis added):

> *In my first dealings with men, I hearkened to their words, and took their deeds on trust. Now, in dealing with men, I hearken to their words, and* watch their deeds.

Andy Hunt and **Dave Thomas**, The Pragmatic Programmers, develop complex software systems for clients and offer pragmatic resources for developers and managers. Contact them via www.pragmaticprogrammer.com.