The Simplest Thing that Could Possibly Work  
A Conversation with Ward Cunningham, Part V  
by Bill Venners  
January 19, 2004

**Summary**  
Ward Cunningham talks with Bill Venners about complexity that empowers versus complexity that creates difficulty, simplicity as the shortest path to a solution, and coding the simplest thing when you're stuck.

In the software community, Ward Cunningham has a reputation for being a font of ideas. He invented CRC Cards, a technique that facilitates object discovery. He invented the world's first wiki, a web-based collaborative writing tool, to facilitate the discovery and documentation of software patterns. Most recently, Cunningham is credited with being the primary inspiration behind many of the techniques of Extreme Programming.

On September 23, 2003, Bill Venners met with Ward Cunningham at the JAOO conference in Aarhus, Denmark. In this interview, which will be published in multiple installments on Artima.com, Cunningham gives insights into wikis and several aspects of Extreme Programming.

* In [Part I: Exploring with Wiki](http://www.artima.com/intv/wiki.html), Cunningham discusses using wiki for collaborative exploration and the tradeoff between wiki authors and readers.
* In [Part II: Collective Ownership of Code and Text](http://www.artima.com/intv/ownership.html), Cunningham discusses how he designed wiki to be a model for collective code ownership, collective incentives for pride of ownership, and how to deal with disagreements by eliminating the cost of making mistakes.
* In [Part III: Working the Program](http://www.artima.com/intv/clay.html), Cunningham discusses the flattening of the cost of change curve, the problem with predicting the future, and the program as clay in the artist's hand.
* In [Part IV: To Plan or Not to Plan](http://www.artima.com/intv/extreme.html), Cunningham discusses using the programming language, rather than the whiteboard, to design and communicate ideas.
* In this fifth and final installment, Cunningham discusses complexity that empowers versus complexity that creates difficulty, simplicity as the shortest path to a solution, and coding the simplest thing when you're stuck.

Complexity that Empowers

**Bill Venners**: What is simplicity? How do we recognize it when we see it? And why should we strive for it?

**Ward Cunningham**: I actually enjoy complexity that's empowering. If it challenges me, the complexity is very pleasant. But sometimes I must deal with complexity that's disempowering. The effort I invest to understand that complexity is tedious work. It doesn't add anything to my abilities.

A friend of mine once said that there are problems and there are difficulties. A problem is something you savor. You say, "Well that's an interesting problem. Let me think about that problem a while." You enjoy thinking about it, because when you find the solution to the problem, it's enlightening.

And then there are difficulties. Computers are famous for difficulties. A difficulty is just a blockage from progress. You have to try a lot of things. When you finally find what works, it doesn't tell you a thing. It won't be the same tomorrow. Getting the computer to work is so often dealing with difficulties.

The complexity that we despise is the complexity that leads to difficulty. It isn't the complexity that raises problems. There is a lot of complexity in the world. The world is complex. That complexity is beautiful. I love trying to understand how things work. But that's because there's something to be learned from mastering that complexity.

Simplicity: the Shortest Path to a Solution

Now, what is simplicity? Simplicity is the shortest path to a solution. Say somebody does a proof for a mathematical problem in 20 pages. You study those 20 pages, and finally you say, "Oh, I get it." You get a reward as the result of understanding that proof, because the proof was a solution to an interesting problem, not just a difficulty. Later, somebody else comes up with a 10-page proof for the same problem. Maybe the new proof uses a branch of mathematics that you might have to study to master, but once you master that branch of mathematics you can use it. And a 20-page proof becomes a 10-page proof. You'd have to say it's simpler, because it's a shorter path. Maybe it's longer if you have to do a digression to actually learn a new branch of mathematics, but let's assume that over time we realize that this branch is important to know in general, so we all become familiar with it.

What we're really trying to do in software is find a way to make it easy to get value from having solutions to problems. How do we do that? When we work the program, we put in what we think is the shortest path to a solution. When we discover that the problem is different than we thought, we rewrite. And then we rewrite again. We work the program. That process is just like doing the proofs over and over. Sooner or later we discover that instead of doing something in 30 lines of code, we can do it in 15 lines, because now we have another capability that fits in. It really is just the right capability, so the work done there we don't have to do here. We'll just invoke that capability from here. That makes our solution easier to follow. Plus the effort you expend today to understand the code will make you a more powerful programmer tomorrow. So that simplification is very valuable.

If you write a lot of programs, and you're used to squeezing them all the time, you find that it's easy to write a program that's simple. A lot of it is having a clear sense of what you want to say—writing the proof by choosing what to prove, and being clear about that. In programming, a lot of simplicity comes from knowing what matters and what doesn't matter. A lot of times a program is made complicated because it's attending to details that aren't needed, or could have been avoided, or could have been relegated to something else.

Someone says, "You should always check your arguments to see if they're in range." Someone else says, "Half the statements in this program are checking arguments that are intrinsically in range." Have they made the program better or worse? No, I think they've made it worse. I'm not a fan of checking arguments. On the other hand, there ought to be a fail fast. If you make a mistake, the program ought to stop. So there is an art to knowing where things should be checked and making sure that the program fails fast if you make a mistake. That kind of choosing is part of the art of simplification.

Einstein said, "As simple as possible, but no simpler." He was being accused of being complex, and he was saying "Yes, simple is important, but..." He'd taken a body of observable fact that was unaccounted for, and accounted for it. So yes, his theory, his models were more complex than Newton's, but they did more. They were worth studying. He was saying, "Look, I made them as simple as possible, but no simpler."

So today, let's write a program simply. But let's also realize that tomorrow, we're going to make it more complex, because tomorrow it's going to do more. So we'll take that simplicity and we'll lose some of it. But tomorrow, hopefully tomorrow's program is as simple as possible for tomorrow's needs. Hopefully we'll preserve simplicity as the program grows.

What's the Simplest Thing that Could Possibly Work?

When Kent Beck and I were playing with Smalltalk, we found it amazing what Smalltalk would do compared to anything either of us had used before. And it really seemed that Smalltalk wanted us to try things. A lot of times, we would just try to see if we knew how to program something. We'd be talking about something, and say, "Gosh. Do you think we could program that?" And we'd just jump in and start programming. And sometimes the programming was almost effortless, as if Smalltalk had been made to write that program. It was amazing. But other times we'd be programming away, and we'd say, "Now, wait a second, what are we working on here?" We'd just get stuck. And if we were stuck more than a minute, I'd stop and say, "Kent, what's the simplest thing that could possibly work?"

It was a question: "Given what we're trying to do now, what is the simplest thing that could possibly work?" In other words, let's focus on the goal. The goal right now is to make this routine do this thing. Let's not worry about what somebody reading the code tomorrow is going to think. Let's not worry about whether it's efficient. Let's not even worry about whether it will work. Let's just write the simplest thing that could *possibly* work.

Once we had written it, we could look at it. And we'd say, "Oh yeah, now we know what's going on," because the mere act of writing it organized our thoughts. Maybe it worked. Maybe it didn't. Maybe we had to code some more. But we had been blocked from making progress, and now we weren't. We had been thinking about too much at once, trying to achieve too complicated a goal, trying to code it too well. Maybe we had been trying to impress our friends with our knowledge of computer science, whatever. But we decided to try whatever is most simple: to write an if statement, return a constant, use a linear search. We would just write it and see it work. We knew that once it worked, we'd be in a better position to think of what we really wanted.

So when I asked, "What's the simplest thing that could possibly work," I wasn't even sure. I wasn't asking, "What do you know would work?" I was asking, "What's possible? What is the simplest thing we could say in code, so that we'll be talking about something that's on the screen, instead of something that's ill-formed in our mind." I was saying, "Once we get something on the screen, we can look at it. If it needs to be more we can make it more. Our problem is we've got nothing."

I think that that's a breakthrough, because you are always taught to do as much as you can. Always put checks in. Always look for exceptions. Always handle the most general case. Always give the user the best advice. Always print a meaningful error message. Always this. Always that. You have so many things in the background that you're supposed to do, there's no room left to think. I say, forget all that and ask yourself, "What's the simplest thing that could possibly work?"

I think the advice got turned into a command: "Do the simplest thing that could possibly work." That's a little more confusing, because there isn't this notion that as soon as you've done it, we'll evaluate it. People ask, "Well, how do you know it's the simplest?" In my case, we didn't know. We were just going to get it on the screen and look at it. But as soon as it becomes a command, then we have to analyze it and ask, "Is that the simplest?" And all of a sudden it becomes complicated. What is or isn't simple?

There's been an awful lot of discussion about what is or isn't simple, and people have gotten a pretty sophisticated notion of simplicity, but I'm not sure it has helped. It might just confuse. Sometimes you think, "Gosh, you know, I'm such a wimp, I can't even understand the discussion of simplicity." It scares people.

Coding up the simplest thing that could possibly work is really about this: If you can't keep five things in your head at one time and make a decision, try keeping three things in your head. Try keeping just one thing in your head, and see if you can make a decision. Then you can think of the next thing. And amazingly, when you write some of this dumb, straight-ahead code, it often turns out that it was all that was required. It works great. When a second programmer comes back later and reads the code she might say, "The people who wrote this are morons. They just wrote a simple linear search here. This thing's ordered, so they could have done a binary search. They could have used a hash table here. Why are they doing a linear search?" Well, because a linear search worked. And when the other programmer looked at the linear search, she understood it in a minute. http://www.artima.com/images/ik.gif

Next Week

Come back Monday, January 26 for the next installment of a conversation with C# creator Anders Hejlsberg. If you'd like to receive a brief weekly email announcing new articles at Artima.com, please subscribe to the [Artima Newsletter](http://www.artima.com/account.jsp).

Talk Back!

Have an opinion about the design principles presented in this article? Discuss this article in the Articles Forum topic, [The Simplest Thing that Could Possibly Work](http://www.artima.com/forums/flat.jsp?forum=226&thread=29371).

About the Author

Bill Venners is president of Artima Software, Inc. and editor-in-chief of Artima.com. He is author of the book, *Inside the Java Virtual Machine*, a programmer-oriented survey of the Java platform's architecture and internals. His popular columns in JavaWorld magazine covered Java internals, object-oriented design, and Jini. Bill has been active in the Jini Community since its inception. He led the Jini Community's ServiceUI project that produced the ServiceUI API. The ServiceUI became the de facto standard way to associate user interfaces to Jini services, and was the first Jini community standard approved via the Jini Decision Process. Bill also serves as an elected member of the Jini Community's initial Technical Oversight Committee (TOC), and in this role helped to define the governance process for the community. He currently devotes most of his energy to building Artima.com into an ever more useful resource for developers.