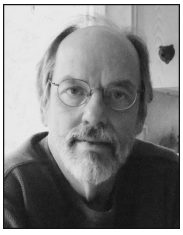
Ward Cunningham

Editor: David Walden



Ward Cunningham has long experience in the computer field. He is particularly known as the inventor of the *wiki*, the concept underlying Wikipedia and other collaborative websites that enable users to share information in a write-publish-review-edit-republish cycle rather than the traditional write-edit-review-publish sequence.[1]

***David Walden*: Please tell me about your youth and education.**

Ward Cunningham: I grew up in the midwest outside of Chicago, in northern Indiana. I consider myself a child of the Sputnik era, when kids who were a little nerdy were looked to as the future. My older brother got into electronics early in his high school career. I remember him kicking around words like cathode ray tube and selenium rectifier. Eventually, I started reading books to figure out what he was talking about. I remember very distinctly when, in the back of a high school English class, I finally figured out how a tube amplified—the idea that you could have a small signal control a big signal. I followed along in my brother's footsteps as he was doing cool projects with his friends. I spent a lot of time in photography, too. I also played with computers a little bit in high school, where there was a linkup via a Model 33 teletype to the Illinois Institute of Technology. I learned how to program. I also learned how to type on that teletype.

***Walden*: What programming language did you use?**

Cunningham: It was called ITTRAN, an interpreter that was designed at the Illinois Institute of Technology. It was a Basic-like language but much more structured.

Around Christmas in my senior year, my friends were talking about what school they were going to, and I sheepishly said, ''How do you get to be going to college?'' My friends, who were smart kids, looked at me and said, ''What, you haven't applied?!'' My best friend opened his folder and pulled out an application and said, ''This is an application to Purdue University. It's a good school. It's a state school. Fill this out tonight and mail it.'' I filled it out, mailed it, and was accepted—just barely, I think. Purdue sent me a catalog, and it said I had to declare at least what school I wanted to go into. I looked at the catalog, and said, ''I don't want to do any of this.'' But flipping through it, electrical engineering looked less worse than anything else, and I said, ''I'll be an electrical engineer.''

School was hard, but it got easier as I went along. Electrical engineering was really good for me. Probably a day doesn't go by that I don't think back to something that I learned in the study of electrical engineering. Professor Leon Chua was at Purdue; I never took a class from him, but he was stirring up the way electrical engineering was taught, to make it pay attention to nonlinear dynamic systems. I realize now that it was a very good electrical engineering program. I've come to realize that it could be just luck that I happened to ask my friend how you get to go to college, he sent me to Purdue, Chua happened to be at Purdue, and my world view was turned to one that thought of forces in time—all that happened from small chance decisions.

***Walden*: Did you also study computer science at Purdue?**

Cunningham: I got a master's degree in computer science. I began the change from electrical engineering in my last undergraduate semester, and I called it interdisciplinary engineering. Then I worked for the computing center and managed to hang around the university for 10 years. After a while, I noticed all my friends were graduate students and that I'd have new friends every two years.

Background of Ward Cunningham

Born: 26 May 1949, Chicago, Illinois

Education: Purdue University, BS (interdisciplinary engineering); Purdue University, MS (computer science).

Professional Experience: Tektronix, 1978–1987; Wyatt Software; Hillside Group; Cunningham & Cunningham, 1992–present; Microsoft, 2003–2005; Eclipse Foundation, 2005–2007; AboutUS, 2007–2011; CitizenGlobal, 2011–present.

Honors and Awards: Pacific Northwest Software Quality Conference Software Excellence Award, 2000; Software Development Productivity Award, 2002; InnoTech/City of Portland Mayor's Technology Award, 2008; Usenix Association Lifetime Achievement Award, 2010; Nike Sustainable Business & Innovation Code for a Better World (Open Data) Fellow, 2011.

 Published by the IEEE Computer Society 1058-6180/11/$26.00

I finally decided I needed to move away from the university and came out to Tektronix in Portland, Oregon, where a lot of my friends had come. I fell in love with Tektronix, partly because it had a university-campus-like atmosphere. I talked my way into their research labs. Mostly, I worked on new product concepts. I worked with electrical engineers, and I would bring computer programming skills to the system design of new instruments. For example, I was doing iPad-like multitouch gestures for oscilloscope control in the 1970s

The VLSI revolution happened during that period, and I learned how to make chips. While I was doing that work, I got to know Carver Mead, who is one of my heroes. He visited Tektronix, and my boss showed him some of the stuff I was doing in Smalltalk for VLSI CAD. That resulted in a trip down to Cal Tech, where I learned that Carver thought of transistors as something that had a natural behavior and that we should all be kind of analog developers in the same way Mother Nature is.

In another VLSI experience, I went out to MIT and spent two weeks working with the people in the AI lab who were doing scheme processor designs. I remember Howie Shrobe, and John Batali was there as a graduate-student teaching assistant. I talked to him about their design procedure language (DPL), which is what they were teaching us to use to make parts. It was a large Lisp program. I was impressed that they were confident that if they could think it, they could program it.

Back at Tektronix, the fellow across the hall from me was developing a Smalltalk interpreter. He showed me about four inches of computer printout for the Smalltalk interpreter. The DPL, written in Lisp, also was about four inches of printout. I started flipping through these systems, and they both had the unusual property that wherever I opened it I could start reading and make sense of it. The thing organizing both of these large programs, and giving them the property that I could read them anywhere, was object oriented programming. I thought, ''I need to master object oriented programming so that I can have the confidence to program anything.''

***Walden*: Were you working with Smalltalk all along or also with other computer systems?**

Cunningham: The Smalltalk work didn't surface until maybe the last three or four years I was at Tektronix.

When I started, I did a lot of work with APL. In APL I could type expressions that would describe a signal and make it more real by adding some reflections, noise, and distortion and pretend to capture that on digital oscilloscope and look at the picture.

By the time that I was doing CAD work on integrated circuits, we were using the UNIX system and a bunch of little UNIX tools. I wrote a lot of what we now call domain-specific languages.

The Smalltalk stuff surfaced at the very end. I had been sort of a ''language of the month'' kind of guy in the 1970s where if you had an idea, you invented a language that embodied the idea. The big change in the 1980s was that when you had an idea, you created an object that embodied it. You didn't have to make all the objects; you just made the new ones that you cared about. That was profound. I believed for a long time that Smalltalk had the keys to the kingdom. I worked in that area for 15 years. In my last few years at Tektronix, Smalltalk became popular elsewhere in the world and not so popular in Tektronix. I kind of followed Smalltalk out of Tektronix.[2]

It was good work at Tektronix. There was the feeling if it was cool, then you could do it. I could reach out to MIT, Cal Tech, and Xerox PARC and understand what was going on throughout the 1980s and 1990s. I thank my lucky stars.

I'm kind of an accidental success—from asking my brother, to finding my way to the computer in high school, to getting through college and into Tektronix, to getting to know brilliant people. I think accomplished people like to know that somebody understands at least a glimmer of what they are doing. I take what they tell me and try to figure out why everybody doesn't love it. I grind on that, and I discover why some people don't get objects, for example. They don't have a way of thinking about objects that makes it simple. For the people who invented it, it was a simplifying thing. But for most people, it's taught as a lot of rules, so the beautiful simplicity of it doesn't shine through. I try to communicate that beautiful simplicity, which people bothered to explain to me. And then people do get it. It's been a formula that works for me.

After Tektronix, I worked for a small consulting firm. Then I joined a company that was building fixed-income security trading software. Smalltalk and its objects had this power to capture complexity. In finance, the complexity came from the free market.

I look for the natural power of the computer's ability to create something that has never been thought of before—that's what I find exciting.

You'd be selling some sort of security, and someone else would fiddle with that security and make something similar but a little better, a little different, a little more profitable. That was a good place to apply Smalltalk's strengths. A natural hierarchy in the nature of securities could be leveraged using the hierarchy of object decomposition.[3] That turned out to be important. We kept our program flexible. When we expected something to be easy to program and it got hard, we flexed the program until it was easy again. The object hierarchies that organized our work were empowering to us. We could write software faster. By keeping the code clean; we were able to discover something late in the process, making it just as easy to insert a new idea into the software in year four as it was in year one. (We now call that process refactoring.) I understand that piece of finance software is still in use.

I feel my whole life is connected by a realization about the power of computers and then trying to promote that realization to the point where other people benefit from it. That opens a door to another realization. I look for the natural power of the machine's ability to create something that has never been thought of before—that's what I find exciting.

For big systems, a team of people have to work together. You can use the power of the computer to do that. The computer is the work, the thing that has to respond to you and your colleague's hands and become something. If it's not going right, you look at the computer; you don't look at your colleague. You say what is it about this computer that isn't what I expected. As soon as you can articulate that, then your colleague can say, ''Well if that's what you're trying to do, why don't we try changing this?'' Here I'm sort of describing pair programming and incremental design, which is the jargon that has evolved around the idea of having a three-way conversation between you and me and the computer as the third person.

***Walden*: A number things are associated with you: object oriented programming—which you've touched on, class responsibility and collaboration cards, pattern languages, the wiki, Extreme Programming, and Framework for Integrated Test. Is there some logical or chronological progression of your involvement with these?**

Cunningham: There is a sequence there, and the theme was always that the computer is an amazingly powerful thing—let's not turn our backs on it. Then I would identify something that was missing. With object oriented programming, we learned to think about the previously missing idea of class.

Class, responsibility, and collaboration cards (CRCC, http://c2.com/cgi/wiki?CrcCard) involved another missing idea. Historically, we were taught that if you make a mistake, don't blame it on the computer—'fess up to it. However, I realized that when figuring out how things are going to get done, it's actually convenient to think about the computer as if it's alive and it wants to contribute to the computation. I put the name of something on an index card and put that card on the table, saying, ''This card is going to take care of one thing, and then another card's going to go over and talk to that card and then they're going to exchange information.'' I was treating the cards as if they were people and telling a story in a way that it's natural for humans to imagine. Talking about what each card was responsible for, we could tell a story that could be programmed, and we could accomplish bigger things.

***Walden*: What was your designated role?**

Cunningham: I was a consultant; people would hire me because they had heard I was good at design. I would drop in and spend a week, and we'd design enough to find the missing thing that would allow people to have enough success that they could find on their own the other things that had to be done.

My approach was to say, ''Let's get to the nub. What is the thing that we're missing that's preventing us from making this a success?'' I would then develop that on dozens,

maybe even 100, index cards on a big table with a dozen people sitting around it. I would start telling stories. As I handled the cards, I would pay attention to when the decision maker started nodding his head, with everybody else paying attention. Then we could forget all those other cards and just make a smaller set of cards work.

In the 15 years I spent as a consultant, I dropped into a lot of projects and gained a lot of experience. The pace was much faster in commercial software than at Tektronix. I had to have a good idea "every 15 minutes." It took creativity to figure out how I could sustain that pace. Mostly, it was making sure that when I came up with bad ideas, I could protect myself from foolishness. That got us into this whole notion of emerging design because you'd keep the good ideas and throw away the bad ones.

The structure applied from objects gave us a common vocabulary to talk to a bunch of people, and the index cards were the first step. Kent Beck, my colleague from Tektronix, and I thought, "This is really important. If we can write responsibilities on index cards, why don't we write a catalog of all the responsibilities that you would ever give an object, and that catalog would kind of teach people object oriented programming."[4] So we set out to write that catalog, but we needed some way to structure it. That led us to the work of Christopher Alexander, who had coined the phrase of "pattern language"—where you recognize a problem that has a kind of stereotypic solution, and you just apply that solution. We studied his book *A Pattern Language*[5] and said, "This is it."

We put a lot of energy into the pattern catalog—had a conference and got a lot of like-minded people together, all trying to write patterns. We had our first workshop at the University of Illinois in 1994, a public workshop on crafting the complete pattern language of programs. The graduate students there in Ralph Johnson's organization helped put it on. After the workshop, one of them said, "This pattern language catalog sounds really important, and we think it's going to be a hypertext. Let me show you how the World Wide Web works." Brian Foote sat down with me for a day, explained what a URL was, and taught me HTML. Brian then said, "We think you should build a repository of patterns using HTML." I agreed. It didn't occur to me that he should; he was the guy at the University of Illinois that knew all this stuff.

I came back here to Portland, opened the Oregonian newspaper, and there on the cover of the business section was Robert Chew, my old boss from Tektronix. He was working for a company that was helping people get on the Internet—another freak chance. So I called him up, and one of his people brought over a computer to be my Internet server, hub, and gateway—all that stuff. He plugged it in, I heard it go deet-deet-deet-deet, and it called his data center. I was on the Internet.

***Walden*: So you set up the pattern language repository on this machine?**

Cunningham: I made a repository,[6] and I said, "Okay you guys, send me your patterns and I'll put 'em up on the Web." To make things easy, I provided a few simple rules—like put a blank line everywhere you want to start a new paragraph and I'll figure out how to turn that into HTML. Nobody followed the instructions, though. Each did it in different way, and I was stuck editing their files. So I decided to write a little translator to do the editing. Then it occurred to me to put that translator in the loop and have them submit directly to the translator. It would translate the submission and show them the result. If they didn't like it, they could fix it. This could get me out of the loop, especially if I made hyperlinking easy.

When it worked well and was fast, I decided to give it a name that would last. I almost called it Quick Web, but then I remembered the word "wiki" from a visit to Hawaii. Wiki means quick, so I called it Wiki Wiki Web, because in Hawaii you double a word for emphasis. It would be the Wiki Wiki Web—the very quick web. The alliteration reminded me of the World Wide Web.

Having invented a new way to write, I had to get people to write that way, so I created about 100 wiki pages. They were pages about computer programming, using this pattern style of describing problems and solutions. I didn't put my name on any of the pages; I just made it look like there'd been a whole community writing these pages. I invited the people like those who helped put on this conference to try the wiki, and many said, "Oh yeah, this is pretty neat." Back in 1995, no one expected a website to look that way.

I was worried that people wouldn't want to write because the idea of writing on somebody else's website was new. So I told them, "Make a page with your own name on it

My wiki showed that a simple Web interface was enough to form trust and community.

and edit your own page first.'' That got them used to writing, and then they'd get caught up in the creative cycle.

Something important was happening that I had stumbled onto. At the same time, a lot of people were making content management systems (CMSs), where one of the assumptions was that there is a workflow that leads to publication. Someone does some writing, somebody else does some editing, somebody does some reviewing, and if you get enough signoffs, boom, it's published. I did the reverse. I said you publish something, and then there'll be a little bit of reviewing and maybe a little bit of editing. By making the publishing the first step, that shortened the feedback loop. People could get on the system, and it was almost conversational. It empowered people to be creative. All the checks that we put into a CMS, to raise the quality, were actually lowering the quality. I got more quality out of the wiki system by letting people respond quickly to other people.

I had decided to do something simple, and if it only lasted for six months, so be it. But it lasted and still is running 16 years later. Eventually, Wikipedia had caught on to the idea, and my community had sort of reached its goal of changing computer programming. I was happy to see my site become kind of an historical artifact, which is how I think of it now.

***Walden*: Is your fatherhood claim pretty much accepted, or are there other people saying they did it first?**

Cunningham: I defended it a little bit. Bo Leuf approached me when he was thinking of writing a book about wikis. I was thinking about it too and had a good relationship with publishers. I set the direction and drafted the table of contents, he wrote a book, and we both put our names on the cover. It was a good book.[7] We thought about how to explain a wiki. By that time, there were already five or six clones (it was cloned practically in the first week because it was such a simple idea), so we wrote a book about how to clone a wiki and how to invent variations on it. It was the right book to write. Somehow, along the line, I managed to get my name in the Oxford English Dictionary under the word wiki, so I think my name is safe.

Although if you ask people about wiki they'll say, ''Oh, you mean Wikipedia?'' Of course, Jimmy Wales has done a tremendous amount of work. I corresponded with him briefly when he first tried wiki, and I met him for the first time at OOPSLA in San Diego. He invited me to keynote the first Wikimania in Frankfurt. I've also interacted with him on occasion as an advisor to the Wikimedia Foundation, and I did a joint interview with him at OSCON in 2007.[8]

My wiki showed that a simple Web interface was enough to form trust and community. Jimmy Wales adopted that idea and figured out how to get the licensing right. He also believed that it had to be global—in every language—not just English. A wiki embodies quick turnaround on publishing and forming a community that owns, in some real legal sense, their work. That community is global. It's a fabulous thing now.

The essence is that you're using the computer as a medium, and through that medium, you're getting to know other people using it with nothing more than a text area and a simple markup language. You can learn to trust other people and together you can create something bigger than either of you could imagine yourselves.

Of course, Tim Berners-Lee saw all this from the beginning. In fact, he built a better system than mine. However, somewhere along the line, as it was getting deployed, the editable part got left behind. His original paper described documents just as alive as a wiki. However, Web servers of the era didn't implement Tim's POST command, and I figured out a way around that omission. My contribution was taking a great idea and fixing that little flaw.

***Walden*: You haven't mentioned Extreme Programming and yet your name is associated with it on Wikipedia.**

Cunningham: A bundle of principles has surfaced, one after the other, in this area.[9] (Our industry ignored what it has previously known to be right.) Kent Beck is doing a lot of the work here,[10] I've done work (for instance ideas on evolving design and the CRC cards), as have others. There has been a realization that these various practices supported each other. If you could adopt them

all at once, you could program in a way that appears extreme from outside but that is actually very calm, relaxing and confidence inducing—like an expert skier looks extreme to an intermediate skier but the expert just practiced until the skiing became effortless). What makes this possible is the power of the modern computer plus a dozen different practices such as programming together and talking about what you're programming. Also, we insist that people write clean code and suggest ways to make sure it's clean, so we don't have to write another document that explains how to read the code; the code is readable as is. Lots of people got the idea that extreme coding is just any kind of programming without documentation and that looks daring from the outside.

***Walden*: Where does the Framework for Integrated Test[11] fit into things?**

Cunningham: A set of tests that you trust works like a ratchet that keeps you moving forward. The modularity of objects allows for a fresh approach to testing. I coded this up for various projects. Then I thought that with FIT I could make a universal representation. It has been a step forward that has worked for lots of people but has fallen short of becoming universal.

***Walden*: On your website (http://c2.com), you have all kinds of other stuff.**

Cunningham: If the URL is c2.com without "~ward," then that's my more public space and in that environment I host the wiki. If the URL includes the "~ward" extension (http://c2.com/~ward), it's sort of just me talking to my friends. I try things and experiment with the Web. When I first got a digital camera, for example, I took some pictures and put them up there. I did the same when I first got a scanner. I was always trying things. It's kind of an embarrassment now because what I've done there is, you know, so 1990s.

***Walden*: You've suggested that you have accidentally gotten interested in things and one thing has led to another. Is there a particular mission you are now on, or are you sort of taking life as it comes and continuing to go where your work takes you?**

Cunningham: I think I've always had a mission—though an elusive one. I want everybody who cares to to find programming a computer to be as empowering and as delightful as it has been for me. As the computers get more powerful, they become more empowering and delightful every year, and I just want people to see that and apply it. At the same time, I want them to use that power to make life easy for people who don't care to program. (In fact, I want to apologize now for causing all those authors to write an encyclopedia in a text area. The only good thing is that they got to do it with others.) The computer experience should be fabulous. In a few cases it is, but in most cases, it's tedious. I want people to love programming and love delighting others with it, and that remains a challenge.

***Walden*: Thank you very much for taking the time to do this interview. It's been fascinating.**

References and Notes

1. This interview was conducted in Portland, Oregon, on 14 Mar. 2011. The recording of the interview was transcribed and edited down from 13,500 words with the approval of the interviewee.
2. K. Beck and W. Cunningham, "A Laboratory For Teaching Object-Oriented Thinking," *SIGPLAN Notices,* vol. 24, no. 10, 1989; http://c2.com/doc/oopsla89/paper.html.
3. "The WyCash Portfolio Management System," addendum to the Proc. of OOPSLA 1992, http://dl.acm.org/citation.cfm?id=157715.
4. See http://c2.com/cgi/wiki?WardAndKent for a description of their relationship.
5. C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language,* Oxford Univ. Press, 1977.
6. The Portland Pattern Repository is available at http://c2.com/ppr.
7. B. Leuf and W. Cunningham, *The Wiki Way: Quick Collaboration on the Web,* Addison-Wesley, 2001.
8. OOPSLA is the ACM Annual Conference on Object-Oriented Programming, Systems, Languages and Applications, and OSCON is the annual O'Reilly Open Source Convention.
9. For more details on Extreme Programming, see http://www.extremeprogramming.org and http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap.
10. See, for example, K. Beck and C. Andres, *Extreme Programming Explained* 2nd ed., Addison-Wesley, 1999.
11. R. Mugridge and W. Cunningham, *Fit for Developing Software: Framework for Integrated Tests,* Prentice-Hall, 2005; http://c2.com/cgi/wiki?FrameworkForIntegratedTest.