THIS MARCH, SOFTWARE Engineering Radio had its first 200,000-download month; fewer than half the downloaded shows were from the current or previous month. This means our back catalog contains many shows that listeners continue to download, even years later.

This issue’s column features Peter Hilton, an independent software consultant for Signavio in Berlin and coauthor of Play for Scala (Manning Publications, 2013). He points out that naming isn’t as simple as it seems—most programmers consider themselves good at it, but the costs of naming decisions often aren’t evident until late in the software lifecycle. Naming is one way the code’s creator communicates with other programmers over time. So, names might be clear enough to the author while he or she is working on them but could be less clear to others in the future.

Portions of the interview not included here for reasons of space include the relationship between naming and documentation, renaming and refactoring, naming and pair programming, naming and functional programming, and multilingual teams. To hear the full interview, visit se-radio.net or access our archives via RSS at feeds.feedburner.com/se-radio. —Robert Blumen

Felienne: They say there are only two hard things in computer science: cache invalidation and naming.

Peter: That’s exactly right. It’s a famous quote by Phil Karlton. Cache invalidation is famously hard, but it sounds like it should be hard. We don’t expect naming to be hard. That’s why the joke is so funny—it violates our expectation that hard things should be technical things, and yet naming turns out to be hard.

Felienne: Why is naming hard?

Peter: It’s hard because it’s creative. But that can’t be the whole reason, because programming requires a lot of creative thinking and design thinking. A better explanation is that naming is hard because it’s about communicating with people, which is harder than communicating with computers.

Felienne: Let’s take a step back here. Why do names matter? Why is it so important to communicate with other humans? Programming is ultimately about communicating with a computer, isn’t it?

Peter: It is, but we like our code to be beautiful and clean, and names are part of that. We want people to look at our code and say, “What a nice name,” similarly to when you introduce yourself to another person. But names also matter because code is about communicating with humans as well as with the computer. Names help explain what we mean, not just what the code does.
If you’ve ever been in an argument with someone, you’ll know that trying to explain what you mean can be very difficult.

Felienne: Naming is also creative. You have to really think, “What is the thing I’m programming here?” Is that part of why it’s so hard?

Peter: Absolutely. You have to choose between many options that might seem right but aren’t quite right. Anyone who’s ever named a child knows that this is not an easy thing to do, especially because you have to commit to it. Children don’t get a refactor or rename.

Felienne: Naming is also creative. You have to really think, “What is the thing I’m programming here?” Is that part of why it’s so hard?

Peter: Absolutely. You have to choose between many options that might seem right but aren’t quite right. Anyone who’s ever named a child knows that this is not an easy thing to do, especially because you have to commit to it. Children don’t get a refactor or rename.

Felienne: Talking about renaming and refactoring, this is one of the most used of all the refactoring tools. Do you think people often change their minds about names in a domain?

Peter: They do, and they do use renaming or refactoring in their tools, but I don’t think they use them often enough. This is ironic because renaming is perhaps the safest kind of refactoring there is. But renaming is harder than naming because you have to come up with a new name. Also, if you rename something that people are used to, you’re going to have to tell them about the new name and get them used to the change.

Felienne: The bigger a code base is, the more people work on it, and the more people are used to something, the harder it gets to change a name.

Peter: Exactly. You can’t completely make up names. [But even if you could,] we get used to them. A name can be an arbitrary word, but once you get used to it, you know what it means. If that wasn’t true, we’d have real trouble with product names.

Felienne: Even if names change, people tend to continue using the old name because it’s so hard to cope with the change. Can you give a few examples of names that are really bad?

Peter: Obviously bad names are the placeholder names. When you’re reading code that someone else wrote and you come across names like “data” or “object” or “thing,” you know the coder wasn’t even trying; they just gave up. You have a variable called “thing,” and it could be anything. “Foo” is another one.

Felienne: Can you give us advice on how to name things?

Peter: When a name is good, it’s good in a particular context. You can’t see the endless toil it took to come up with this name. Let’s say you’re working on a logistics system, and you’ve figured out that it’s a shipment and not a consignment. “Shipment” could be a really good name because that’s what people call it. If people in the warehouse call it a shipment, then that’s the correct name. Things sometimes have a good name but you can’t always see that from looking at the code because it’s only a good name if that’s the name it should have in that context.

Felienne: The domain plays a big role in this scenario. Is the advice here that you need to talk more to the domain experts?

Peter: Exactly. There’s a whole chapter on naming in Eric Evans’ book Domain-Driven Design [Addison-Wesley Professional, 2003]. It’s an important part of domain-driven design and of a particular subject matter domain because that’s where names often come from. If you can write code using those names, you’re
probably going to have better, more understandable code. These are the names that should mean something to everyone [who already understands that domain].

Felienne: But not all the developers on your team know the domain through and through.

Peter: That’s true. At least if you have the right domain name, you don’t have to explain it just because you wrote the code. A domain name appears in the dictionary somewhere, and crucially, it appears in a dictionary that you didn’t have to write yourself. Anyone who’s familiar with the domain can explain to you the difference between “shipment” and “consignment,” or perhaps there isn’t a difference because “they’re synonyms, but in this company we use this one.”

Felienne: Coming up with good names can improve not only your source code but also communication between the developer’s team and the domain experts.

Peter: Absolutely. As developers, we have to talk to other people about our code. Even if we don’t show them the code, we’re having a conversation with noncoders about the software we’re writing. This works a lot better if we use the right language and if we use consistent language in all of our conversations—with the compiler, but also with humans. This is another central idea in domain-driven design—it’s not only about using the right names; it’s about using the right names all the time.

Felienne: We’ve talked already a little about bad names, but there’s a thing you call “naming smells”: code smells that relate specifically to naming. Can you give us a few examples?

Peter: Bad naming is one particular kind of code smell. I’ve mentioned a couple already, like “foo” and “data.” “Foo” is clearly meaningless. “Data” is not meaningless, but it’s too abstract to be remotely helpful. As your names get progressively more useful, you’ll still have bad names, but they’ll be less bad. You have vague language, vague verbs—most famously, “get.” If you have a method or function name and the first word is “get,” you’re not being very imaginative or specific about what this thing is doing.

Felienne: But it’s getting stuff. What’s wrong with that?

Peter: It could only be worse if you said it was doing stuff. Doing stuff and getting stuff is what all functions do. That’s like calling a variable “data.” Sure, it’s true, but it’s not specific. There are much more interesting words you could use instead, like “fetch” or “calculate” or “derive,” that are more specific. “Get” might work for every function that returns something, but it doesn’t tell you exactly what it’s doing. It’s worth mentioning that “get” is popularized in Java—particularly by the JavaBean standard—which compensates for the lack of properties in the language but has trained a whole generation of programmers to call their methods “get,” unless they’re called “set.” This is not helpful, but it’s no different than using vague nouns in class names. If you have a class that’s some kind of manager, that’s not helpful or specific. Classes manage data. Calling a class a manager is not as helpful as calling it a builder. What kind of management is it doing? Is it a builder; is it a calculator; is it a data access class? There are more specific ways to describe responsibilities. It should be obvious that “manager” is not very specific—if you’re introduced to a colleague as a manager, do you have any idea what they do? Probably not.

Felienne: Tell me a few more—I’m sure I’m guilty of all of them.

Peter: Single-letter names. One letter is not long enough to communicate anything. Most words are more than one letter, so it’s a special kind of abbreviation. If it’s the letter “A,” you don’t know whether it’s a placeholder, the first letter, an abbreviation for a word that begins with “A,” or some standard accepted meaning. The intent of the code is not clear.
Abbreviations work when we know exactly what they mean. We’re quite used to “char” meaning character. I remember a project years ago where I was coding an elaborate bug based on my assumption that “char” meant character. The software was a database attribute or something like that, so “char” meant characteristic, which stood for something specific in that domain. This caused me to make some wildly wrong assumptions and code a significant bug. Abbreviations will always get you.

Names can also be too long. If you use overly simplistic language in names, you can end up with convoluted names like “AppointmentList.” That might make sense technically because it might be a list object that contains appointment objects. It’s not as bad as a vague name, but it’s not a great name. A much better name for “AppointmentList” is “calendar,” because we normally call that a calendar. Or if you have a “Company-Person,” we normally call those “employees,” unless it’s the owner of the company or a shareholder.

These average names, in addition to being unwieldy, are wrong because they’re not what you would use in the domain. They aren’t as specific as they could be. Names should be as specific as possible because that gives them more meaning.

Going back to the discussion about domain language names, you can have the wrong name. I worked on a logistics project where “shipment” and “consignment” meant the same thing. “Shipment” was the right name because it’s the one people used, but an order is not the same thing as a shipment. If you say “order” when you mean “shipment,” you have the wrong name. That’s going to sound reasonable in a sentence or look reasonable in code, but it’s going to confuse people.

Felienne: If you think about good names, you’re writing in English. Naming is a literary type of activity, the way you describe it.

Peter: There’s a definite link here. If your name isn’t very good, the only way to make the code understandable is to add a comment to explain what it means. Often, that’s how you can tell whether you’ve come up with a good name. It’s like coming up with a newspaper headline or a tweet. If the headline is good enough, people know what the article is about. If you get it wrong, they’re misled. This can happen too, sometimes deliberately but hopefully not, in code. If your name isn’t good enough, you’ll have to continue the explanation.

Felienne Hermans is an assistant professor at the Delft University of Technology. Contact her at felienne@gmail.com.