

--
A blog forum to provide deep dive analysis and community conversations about software development models. For more details click [here](#).

[Interview with Gwyn Fisher – CTO – Klocwork](#)

Tags: [security](#)

Interviewers: [Scott Swigart](#) and [Sean Campbell](#)

Interviewee: [Gwyn Fisher](#)

In this interview we talk with Gwyn. In specific, we talk about:

- [The value of automated code review](#)
- [The advantage of code review at the developer level](#)
- [Judging the security of open source codebases](#)
- [Coordinating bug resolution between distros and upstream projects](#)
- [Educating users about vulnerability in the face of complacency](#)
- [The changing face of computer attacks](#)
- [Why isn't everyone using code-validation tools?](#)

Sean Campbell: Gwyn, why don't you start with your background and tell us a bit about Klocwork?

Gwyn Fisher: Sure. I'm the Chief Technology Officer at Klocwork. I've been with Klocwork for a couple of years now, and as a consultant, actually, working with them for several years before that.

Klocwork was founded in 2001 as a spinoff from Nortel Networks. The Nortel research group that created our basic technology was centered on the head of the CASE department at ISPRAS, part of the Russian Academy of Sciences and affiliated with the Moscow State University, who was very interested in the notion of very large scale software architecture, model discovery, efficiencies and inefficiencies in incredibly large source code bases.

As it happened, Nortel had a specific business problem that they were trying to deal with that required massive re-engineering on one of their switch operating systems, and so as part of the existing partnership between Nortel and ISPRAS, he was funded to help solve this problem. Following the spin-off and creation of our company, he and most of the team relocated to North America, where many of them continue today to be the kernel of the ever-expanding domestic product development group within Klocwork.

We stayed in that space for several years, helping customers in very similar situations, who had requirements for re-engineering very large, very old, very complicated code bases, in order to simplify, componentize, and improve their software for newer devices, reuse, maintainability, and all that good stuff.

And then some of those same customers—and these are all very recognizable brand names—came back to us and asked us to do defect analysis for them. So, in addition to finding things that are wrong with their architecture and the fundamental way they’re building their software, we also started locating things like where they were leaking memory, doing bad things with pointers, generating buffer overflows, and other basic software problems.

After a couple years of research, we brought the first iteration of a product suite to market for defect and security vulnerability location and analysis.

So, since we were founded in 2001, we have sort of morphed over the years, away from a pure architecture and model discovery kind of technology, into a much fuller footprint in terms of what we do around defect analysis. That includes operational defects and security vulnerabilities, as well as continuing to emphasize our strength in architectural and maintainability issues.

There’s also that whole backbone of model discovery, visualization, and re-engineering—all the great stuff our customers need to be able to do in these very large code bases to manage them over time. We help them keep building out across more languages, more platforms, more tool chains, and all the good stuff that goes with working in a production software development environment.

Scott Swigart: What are the things that are really hard to find with standard code reviews and QA that tools like yours can quickly make apparent?

Gwyn: The quick, sort of palm-to-forehead, kind of “Why wouldn’t we have found that? Why didn’t we think of that?” sort of issues tend to be in the area of what is called inter-procedural analysis.

If you have a very large system, it’s made up of a whole bunch of functions, methods, and modules all linked together into large system images. And the problem, whether you’re using code review, runtime testing, or whatever, is finding enough combinations so you can trace through, from source to sink point, exactly what’s going to go on with a pointer, buffer, array, or whatever it happens to be.

We can point out across multiple different function boundaries, that this pointer you’re grabbing from over here, you’re using inappropriately over here, or this memory you’ve allocated here, you’re never releasing on this particular thread. Those sorts of instances would require your average high level architect to really bend their mind around them for days or weeks at a time to try and find manually. They’re the sorts of things that we can point out to someone and they can understand it very easily within 30 seconds or a minute.

Obviously, we all bring our own biases to bear, in terms of how we think about our products and how we think about our tools. And it’s perhaps obvious to think about a tool like this as a QA tool, and certainly the heritage of tools like this has been very much driven by a downstream audit or an adjunct to code review. But, really, that’s not where this technology comes from—this is a developer’s tool. The first one of these was Lint, back in the ’70s.

And as a developer’s tool, well, Lint’s got a pretty bad name. Let’s face it.

[laughter]

As sort of a history, we've all sat there and watched the CRTs desperately trying to refresh as acres upon acres of warning messages come out of these things. But when tools like Klocwork bring modern analysis technology and accuracy and complexity into that equation and deliver it back to the developer, all of a sudden you're not just inundating them with all kinds of stuff that they can't deal with. You're actually giving them a few very useful bits of information before they check code in.

That's our approach, instead of trying to kick in downstream and then figure out who the right developer is to blame, instead our goal is to build analysis much more organically into the development process so it becomes an enabling environment rather than a blaming environment.

The uptake within an organization changes overnight, as soon as you can get through that cultural issue. Rather than giving QA a new tool for telling the developer that they're an idiot, we are giving the developer a tool that will help them prevent being called an idiot. [laughs]

Scott: In some of our conversations with Microsoft, they've talked about modeling tools that they use similarly to what you're describing; before code ever gets checked in, the developer does analysis of the code and fixes issues that are flagged.

Talk a little bit about the benefit of having that done at the developer level versus, for instance, in a QA process that happens once or twice a year.

Gwyn: If you do take a milestone approach or do it just before you roll out, the challenge tends to be that you get an overwhelming number of issues to deal with, often all at a critical time in the product life cycle. From the individual developer's point of view, it can be an awful ordeal.

Scott: In other words, the developer would tend to build up a big pile of issues that all come back for resolution at once, when they are periodically unearthed. On the other hand, if the issues are dealt with proactively, you never accumulate that big, ugly pile of bugs.

Gwyn: Right. The other thing to consider is that this ongoing system means that the developer is dealing with mistakes that they made just recently, instead of six months ago, so it's still fresh in their mind.

Sean: You mentioned that you guys do some work with different open source communities. Educate me about your input into their process –are there open source analysis tools that are commonly used?

Do you find that developers are running tools that they have available prior to submitting code to the mailing list, or is it more the case that they really just trust the mailing list and are also interested in the scans that you and other vendors do and what those uncover?

Gwyn: There is no doubt about the wisdom of the thousand eyes approach, particularly for the more mature open source projects, and people do use as many techniques and tools as they can get their hands on.

In the Java space, there's a very nice open source analysis tool—FindBugs—that can really help developers get started with analysis. It doesn't perform inter-procedural analysis, so in the words of the tool's developer, they're really focused on “low hanging fruit”, but it's definitely a great place to start.

In the “C” world, well, not so much. There are a couple of tools that check coding style and things like that, but while they can tell you things like you should declare your copy constructor as private or you might generate memory leaks, their ability to find actual defects (as opposed to poor coding constructs) is very limited, unfortunately.

Sean: And sometimes, it's really hard to spot with the naked eye looking at apps committed to a mailing list.

Gwyn: Exactly.

Sean: This is in kind of a different area, but given your perspective and what you've done in terms of looking at a variety of codebases over the years, what tools do you think organizations should utilize to ascertain whether a given open source project is secure?

That's assuming they're not going to go buy a tool, do a massive code review or utilize a third party; they're just out there having the internal conversation of Apache versus IIS, or MySQL versus SQL Server, or MySQL versus Oracle.

Different people make different claims about which is more secure, but how do we prove it? You end up in this kind of absence of evidence argument, I think it's termed. Scott did a look through one of the mailing lists for one of the major projects, and he didn't see anything about threat modeling. That doesn't mean it's not occurring, but that also doesn't mean it is.

What's your take on that? What should an organization do to really ascertain whether a given project has a robust development lifecycle around security?

Gwyn: To be blunt, they should do everything they possibly can. You just have to assume that the guys on the other side of the fence haven't done it yet. Do you believe Coverity, who says their favorite 11 open source projects are perfect? Or do you believe Fortify, who comes out two weeks later and says the sky is falling and they're all insecure? Do you believe Mozilla, who says they have a thousand eyeballs looking at their code all the time? Or do you believe the users who say, “Yeah, but I filed almost as many bugs against you as I do against Microsoft”?

Nobody has the answer to this question. I've talked to several people over the years who said, “Wouldn't it be nice if we could put together an indemnification agency that would use tools like yours to help some of our customers have more confidence in open source?” The idea would be to say, “If you want to use Apache (for example), here's everything you need to know about it. Here's the support protocol around it, and most importantly here's a security rating for it.” In essence, they'd be attempting to provide a warranty for a particular revision of a particular open source project.

There are various organizations around the country and around the world who do little bits of this, and some of them are quite successful, but no one has ever gotten to the point of taking responsibility for saying “Tick or cross: you should use this.” It all comes down to providing a lot of data for the customer to evaluate, because no one is willing to take on the legal liability of saying what people should consider safe to use.

Because of all the sources out there and because all the tools and techniques are out there, you’d assume somebody would at some point do it. But there are enough litigious arguments against it that I don’t think it will ever happen, unfortunately.

I think it would be a great thing to do, and I think that most of the senior committers on those mature projects would want to see that happen as well. But they’re giving their time for free; it’s not something they’ll ever want to do themselves. And I think any commercial entity is just going to be scared witless of some bank somewhere going, “You know what, you told me to use X.Y.Z of Apache, and it turned out it had this vulnerability in it. We’d like you to compensate us for Grandma’s password finding its way to the Ukraine.”

Scott: We’ve talked a little bit about the value of static analysis to define things that aren’t necessarily easy to see with the human eye, and the value of doing it up front. One of the interesting things I find in open source is that with a proprietary company like Microsoft or Oracle, for the most part they own the code from soup to nuts. If there’s a bug, there’s a place to file the bug. They fix it, or they don’t fix it, but it’s their codebase.

When you’re dealing with something like a Linux distro, on one hand you’ve got this downstream distro that’s packaging a lot of upstream projects. And so, you’ve got the distro provider who has hired people to do the packaging and things like that. But, they’re also paying people to work on these upstream projects, whether it’s the shell, the web server, the kernel, or whatever. Then, when a vulnerability happens, there’s a relationship necessary for a fix posted to the distro to get posted to the upstream project.

Distros also talk about things to make these upstream projects enterprise ready. In your experience, do you find at the distro level that people are using tools like yours so that if an audio player for example didn’t go through the same kind of analysis upstream, they maybe are doing it downstream and working with the upstream project to improve the quality of the code?

Gwyn: Wouldn’t that be a good way of organizing life? Unfortunately, it doesn’t happen so much. Some of the distributions today are backed by serious individuals or serious companies. It’s not just two guys off in Europe trying to bang a CD together, and in some ways, they’re doing phenomenal things. Look what Ubuntu has done around the user experience. They’re doing amazing stuff for the operating system as a whole.

I would say the distro manufacturers as a whole certainly validate in an industrial strength way and do the needed extra development, but they don’t really take responsibility. In terms of indemnification, guarantees of quality, or anything like that, they tend to fall back on a very dinosaurish kind of mentality. Got a problem? Hey, it’s only a patch, suck it up.

In just the same way as indemnification players don’t really want to step up and say it’s secure, distro vendors really don’t want to take this step of guaranteeing not just observed quality but absolute ground-up code quality, because of the exposure that would entail with all kinds

of people over whom they have no control. Who do they go to if they have to build a fix on some kind of SLA? Of those thousand eyes, half of them are asleep at any given time. And the other half work for somebody else.

I'm certainly not saying you should go and buy commercial software all the time because you can always go and get hold of somebody. At the same time, though, the community just doesn't have the level of leadership yet that lets it take that particular kind of responsibility for the quality and security of the software they're producing. Perhaps the fact is that there's just not that level of requirement being surfaced by the enterprise customers.

At the end of the day, nothing's ever free. I think that open source has absolutely proved that software can be as free as you like, but that's just one small component of the total cost of ownership of these things.

Scott: I imagine that you run into the idea of a security IQ that starts out fairly low, where people say "Well, we haven't had a problem, so we must be doing things right."

Gwyn: [laughs] And my five users are all very happy with it.

Scott: Right. So, how do you approach educating somebody about the fact that just because nothing has blown up and it hasn't been the end of the world, they might not want to assume that everything's hunky dory.

Gwyn: It is actually fascinating how you can never feel anybody else's pain. I think that's a rule of life, whether it's physical or intellectual. You can't actually empathize with all the people who are going through it until you've done it yourself.

At one point early in my career, we got this very polite email from the DOD saying that they had taken 65 servers offline because they found a vulnerability in one of the products I was responsible for. And this was back ages before anyone was really worried about such things.

It was a tremendous wakeup call to realize that we had just taken away a significant capability from a very important agency just because of an error on our part.

They were very polite about it and didn't intend to cause any significant commercial ramification, but as professionals, how were we to get to the root cause of it? How would we stop it from happening again in the future? Until that moment, I had never felt that pain, and I didn't really understand it.

Increasingly, we get people coming to us from the device space to ask specifically about security. And they're not coming to say "OK, we know we need to worry about this;" it's "why would I have to worry about this? Everyone's talking about this and I don't get it."

So, we were talking with this one medical device manufacturer, and they make truly scary stuff. If this stuff goes wrong, it's not going to be happy days for anybody. Their senior architect was sitting in front of me and he said, "Marketing tells me I have to put an IP cable on the back of this box because he wants the paramedics to be able to send stuff off to the hospital before the guy gets there."

And I asked him, "OK, and what are you doing about making sure that your device is doing what you think it's doing and hasn't been hijacked or something?" And he responded, "Why would anyone ever want to do that?"

Explaining to people that just because they would never think to go and hijack somebody's super-fridge or their pacemaker or whatever it happens to be, doesn't mean somebody else out there doesn't have too much time on their hands and would just think it's a huge joke.

In many ways, it's that first moment of enlightenment that's the hardest for people. You've got to get across that notion that just because they would never do it, and just because they can't understand why anybody would, doesn't mean it's not going to happen to their system or device.

Once they're through that, then it becomes a very pragmatic process, but that very first thing you've got to have is for that senior architect, VP, or whoever to feel the pain.

Scott: It seems to me that in a way, Microsoft got a little bit lucky (though I'm sure they didn't feel that way at the time) back in early 2000, when Code Red, NIMDA, Blaster, and Slammer came out. These really high profile exploits were very embarrassing, and they were the kind of exploits that brought a whole company network down. And so, they decided to just pour dollars and resources into solving the problem.

Today, it seems like the nature of exploits is very different. They are far less likely to absolutely saturate your network and let you know that you've been hit. It's more about setting up a botnet and what some people call crime as a service.

It's a lot more valuable to have Sally not know her computer's been compromised, or to have some Linux server in a data center doing command and control for a botnet, without its admins knowing about it.

Like I said, Microsoft got lucky in the timing and the nature of the exploits. In some of these projects, there's a false sense of security because they say "Well, where's our NIMDA? We've never had one." Well, they're never going to have one, because that's not the kind of exploits people are writing.

Gwyn: Yeah, exactly. Over the last ten years, we have moved from script kiddies doing dumb things to these really very, very smart attack vectors. It is a whole new day.

We can't go a whole interview without me poking at Microsoft. [laughs] Look at the animated cursor type of thing that hit the public eye a year ago. This was vulnerable code that had been in Windows since day one, but all of a sudden, out of nowhere, you don't have to see the person, you never have to see their machine, you can do everything remotely, and all of the sudden that thing is yours.

And what's more, if you're smart enough, the person who's getting compromised doesn't know what's going on. It sort of became that perfect storm of an attack vector. And at the end of the day, what is it? It's a simple exploit of a buffer overflow vulnerability, albeit a phenomenally complicated one. Those guys should get a PhD just for creating those things. [laughter]

You've got to give them kudos, right? But, that happened at the height of Microsoft's secure development life cycle, and that bug had been reported. It was in their queue, and they knew all about it, but it had been de-prioritized and was buried in a pile of sixty eight thousand other things they had to worry about.

So, they are at one end of the spectrum because they have so many products and so much going on, but if you take the whole open source domain as a thing, there's probably more code there. And you know that same level or magnitude of vulnerability is out there.

Scott: That's one of the benefits of tools like yours. On one hand, there's putting it in the hands of every developer so that they can have a 'get clean, stay clean' sort of mentality. But, the other value I see is that the most dangerous code in the world is probably the old code that nobody is looking at anymore, because it was written before people knew about integer overflows and buffer overruns.

I remember, way back when, we bought these Unix systems from a vendor. We were writing some low level network code. We did something that crashed the server. We had a bug in our code, and it was sending a packet where the frame was the wrong size or something. The response from the Unix vendor was basically, "Hey, there's an RFC, and if you're operating outside that and bad stuff happens, that's not something we're going to worry about."

[laughter]

Gwyn: There you go. Good rationale. On that same tack, IBM's X Force does all these amazing things all the time, like showing how forcing an invalid pointer usage can compromise Flash applications, etc. — seriously smart people.

One of the things they do is to issue a "top ten" offenders list for attacks during a year and something that struck me was, I think, in last year's report. No prizes for guessing who is the topmost vulnerable vendor, because Microsoft wins every year. But, of all the reports that they had coming to their NOC over the year, Microsoft products were responsible for three percent. Three percent of all the reports had something to do with Microsoft software, somewhere along the way.

The fascinating thing I saw was at that number five or number six—I forget exactly where they sat—was Mozilla. How many products do they make? They have one fantastic browser I use every day, and some other random stuff that is much less popular, but they are responsible for 1.4 percent, 1.3 percent ... don't quote me on the figures, but some relatively close number of available vulnerabilities, compared to all the things that Microsoft puts out.

Now, in reality, how many of the things that Microsoft puts out are actually getting attacked at the same level that Firefox is getting attacked? But still, it makes you think... Oh, and that top 10 was only responsible for about 15 percent of all reported attacks, so over 80 percent were against software and vendors you've never heard of.

Scott: Well, I want to be sensitive to the time. We've covered a lot of great stuff, but what have we left out?

Gwyn: If I were in your place doing the interview, I would ask what's wrong with these tools? At face value, they are no brainers. Everybody should be using them, so why not? What's the barrier to acceptance? Is it just because they are very new? What's really going on here?

We try to answer this question for ourselves all the time. Why is nobody in this market a Microsoft yet? Why are we still a very nascent segment? Obviously, there are some elements of market maturity there. Most of the companies in this space have been around for a very short time.

It doesn't matter which community you are working with, whether open or commercial. There is an overwhelming thought process involved around analysis. People tend to look at this process the same way that we started this conversation about it today—thinking of this as an audit environment.

People tend to regard it as a milestone activity and as something that is an onerous, annoying part of the software development process akin to code review. I can't imagine anything more mind numbing than having to revisit the days in which I did code reviews all the time.

If this kind of technology is equated to that area of things, that's a problem for this industry, and for the service technology type. It behooves companies like us—and open source projects who are trying to do the same sort of thing—to get over that, and to engage with the developer in a meaningful discussion around what you actually need out of one of these tools.

Is it that you need the most accurate tool with the lowest noise ratio? Or do you need something that finds every bug imaginable, regardless of the noise ratio? Do you need something that's really fast, or that integrates with every environment, or something you never see and just pops up when you do something dumb?

For various vendors in this space, we have markedly different models as to how we engage with development organizations. As for ourselves, we are very upstream. Everybody we ever talk to is in development. Other people in our space are very downstream. They talk to auditors and CFOs—guys who in the morning are buying firewalls, and in the afternoon, they are buying software validation tools. There is a completely different feel to how you engage with a development organization, when you start getting that viral distribution down at the volume end of the buying pyramid.

I think it's an interesting challenge in this space—how to get that massive distribution. When did IDEs suddenly hit the tipping point from being a bizarre luxury that a few companies used, to being something that every developer on the planet sees as a right and a prerogative?

It's the same with runtime profilers. It's the same with memory profilers. It's the same with all these tools, which are just part of today's normal, everyday SDLC.

When did they get there? How long had the market been around? Is it just a volume-based thing? Is it a market approach thing? I think that's the fascinating question in full scale analysis now. Obviously, it has value and is having great effect in both commercial and open source, so when does it go big time?

Scott: That's a great closing thought. Thanks for taking the time to talk with us.

Gwyn: You bet. Thank you.

Bookmark this:



[Comments \(2\)](#) Posted by campsean on Monday, February 2nd, 2009

[« Interview with Chris Messina – Vidoop](#)

[Interview with Alexey Rusakov – Project Manager – Alt Linux »](#)

You can follow any responses to this entry through the magic of "[RSS 2.0](#)" and leave a [trackback](#) from your own site.

2 Responses to “Interview with Gwyn Fisher – CTO – Klocwork”

- [Microsoft Press](#) Says:
[February 11th, 2009 at 7:21 pm](#)

Even more on Microsoft's SDL...

“The Security Development Lifecycle,” by Michael Howard and Steve Lipner, helps prompt a new series of articles on Microsoft's SDL....

- [jean-marc seillon](#) Says:
[February 23rd, 2009 at 3:47 pm](#)

Hi,

I am using klocwork for C code.
Unfortunately klocwork raise too many false error.
It seems klocwork doesn't analyze enough deeply the complex code.