Hanselminutes is a weekly audio talk show with noted web developer and technologist Scott Hanselman and hosted by Carl Franklin. Scott discusses utilities and tools, gives practical how-to advice, and discusses ASP.NET or Windows issues and workarounds.

Text transcript of show #213

April 29, 2010

Aspect Oriented Programming (AOP) and LinFu with Philip Laureano

Scott talks to AOP expert Philip Laureano about Aspect Oriented Programming. Is it the missing piece of the Object Orient Programming puzzle? It sounds scary but is it? Should I start using IL Rewriting and Dynamic Proxies on my next project, or is it too dangerous? All this and more as Scott and Philip learn about LinFu, an Open Source project that enables these scenarios and more!

(Transcription services provided by PWOP Productions)

Our Sponsors

telerik
deliver more than expected
http://www.telerik.com

/n software
http://www.nsoftware.com

.NET Developer's Journal
http://dotnet.sys-con.com

Lawrence Ryan: From hanselminutes.com, it's Hanselminutes, a weekly discussion with web developer and technologist, Scott Hanselman. This is Lawrence Ryan, announcing show #213, recorded live Thursday, April 29, 2010. Support for Hanselminutes is provided by Telerik RadControls, the most comprehensive suite of components for Windows Forms and ASP.NET web applications, online at www.telerik.com. In this episode, Scott talks aspect-oriented programming with Philip Laureano.

Scott Hanselman: Hi, this is Scott Hanselman and this is another episode of Hanselminutes and I'm sitting down today with Philip Laureano. Philip is actually calling in from Hong Kong. Thanks for taking the time. I guess it's about midnight there?

Philip Laureano: Yeah, it's midnight.

Scott Hanselman: Yeah, well, I appreciate you doing it.

Philip Laureano: Anytime man.

Scott Hanselman: So I wanted to talk to you because you are focused on Aspect Oriented Programming, what is Aspect Oriented Programming and why should I care?

Philip Laureano: Well, Aspect Oriented Programming is that missing piece of Object Oriented Programming that you've been hearing about over the past 10, 15 years but it's kind of hard to explain if you think of it in traditional object oriented design. I mean with object oriented design you have these nice diagrams and stuff but there's one dimension missing there and that's time. There's nothing about object oriented design that actually says, "Okay, we've got to have this one feature there only in certain times and sometimes it shouldn't be there." For example, logging, I mean everybody talks about logging with AOP but one of the problems with traditional Object Oriented Design is the fact that you've got, if you put in a logger, you have to put in the logger in about a thousand different places and you could either have it in there or you could either not have it in there and if you do have it in there it's a maintenance problem because you've got to take all these different instances of that one logger that are scattered all over the place and somehow find a way to take that abstraction and make it nice and pretty and you can't because it's either there or it isn't there. So it's a big problem because I mean this is just a simple logger but imagine there are other features in your application that you might need to have, like for example transactions, thread synchronization, things like that that you won't be able to cleanly separate into different classes simply because they're just scattered all over the place. So with Aspect Oriented Programming it's not, a lot of people say it's interception but it isn't, what it really is, is a way of thinking about when things should be in place in your application and when they should not be. So for example, with the logger itself, you could say if the application crashes, I want the logger to be turned on but it's kind of hard to say that at least with your traditional kind of notation or at least in code without having to resort to some pretty interesting tricks that you would normally rely on AOP to do.

Scott Hanselman: Okay, so a logger is, like you said, the example that people always give and it's a good one because of that notion of what they call cross-cutting concerns, a logger cuts through and typically when you see an Object Oriented Programming described in boxes and lines it involves kind of a stack of things, it kind of looks like Tetris and then when they describe AOP, they draw lines kind of perpendicularly, cross cutting throughout those, what else is there other than logging that I might want to use AOP for?

Philip Laureano: Well, a really big one has to be the lazy loading. For example, NHibernate uses Dynamic Proxies whether it's Castle or LinFu to do this kind of lazy loading where even if you pull the object out of the database, it's not really going to make the query until you actually access the object's properties itself. I mean it can be anything from like 5,000 or 10,000 records in one pass but the point there is, you want to make sure that you'll be able to use this, at the same time you won't necessarily have to explicitly say I need to use lazy loading or I need to put that into my application other than through maybe a setting in NHibernate. The point of Aspect Oriented Programming is to have that kind of a nice separation of concerns without having to do some sort of really medieval cut and paste or copy and paste as you would normally see in...

Scott Hanselman: Did you just say medieval cut and paste?

Philip Laureano: Yeah, I did.

[Laughter]

Scott Hanselman: I love that.

Philip Laureano: Basically the problem is it's just, there's no clean way to actually separate this out. It's either it's going to be there or it's not going to be there, I definitely don't recommend cut and paste but from what I've seen in a lot of systems what happens is you've got guys that have all these dependencies all over the place and they have to use those dependencies so you can't really pull it out. Even if you were to refactor it, it's still going to be there. So the problem here is, I mean back to the example of diagrams, I mean in terms of Object Oriented Design we're really looking at it as if this is like the difference between a single frame and a series of frames and animation, I mean that's a good analogy for it because when we have an object model, I mean you're looking at one picture, it never changes but with AOP it really goes to that idea that an object model can change at runtime and there's certain features that can be introduced at runtime that may or may not be there when you actually compile the system.

Scott Hanselman: What did you tell me that was the first rule of AOP?

Philip Laureano: Oh yeah, I call it the Fight Club rule because one of the hardest thing to understand about AOP is the fact that it's supposed to be invisible and if you have to explicitly use it in your application then you're missing the point because if AOP is supposed to separate concerns and then you're explicitly saying, "I need to separate my concerns," then you're defeating the purpose, it has to be invisible.

Scott Hanselman: So what do you use it for other than logging though, I mean like in like a production application?

Philip Laureano: Another one is thread synchronization...

Scott Hanselman: Really?

Philip Laureano: For example, for caching, yeah. I've seen instances where people tag a particular method or a property with a particular attribute that says something like single threaded, so what happens is once the method executes, it wraps it around a particular center for a lock and then you don't even have to worry about synchronizing it because at runtime the AOP part wraps in a lock so that no matter how many threads try to hit that one property or method it basically all goes through with one thread and you don't, and it's completely transparent. Maybe aside from the attribute or maybe the actual code itself, I mean you really don't see it and that's probably why a lot of people shy away from it because it looks like magic and the truth is it's not really magic at all. I mean I've seen the implementations and it's not as complicated as you think. The only reason why I would think that people think it's magic is that you don't see what's going on behind the scenes and it's quite simple actually.

Scott Hanselman: Would you say that this is one of those things that more people should be using and it's like, I mean like you said the first rule is to not talk about it. Is this the greatest thing since sliced bread? I mean should everyone be doing this and just, you have no idea why they're not doing it? Are you that into this?

Philip Laureano: Well, I mean, for me this has been a passion for a long time but I personally, it's no silver bullet, of course. It's just another tool in the tool box, in my own experience it's really made my apps and models really, really clean I mean we're talking like one class, one method in less than 20 lines long, it makes that kind of separation between the concerns and all the implementations that much more simpler. Then you tie it in with something like an IOC container and then it's pretty much golden because of course IOC's handle all the construction but AOP takes all these little bit's and pieces that you put together with IOC and you can separate that even further, so it's a great combination.

Scott Hanselman: Okay, so how do you do this in .NET? How does AOP embedded in .NET and what do you have to do that's, what's built in, like what does Microsoft ship and where does Microsoft drop the ball and leave you to figure it out?

Philip Laureano: Okay, well a couple of years ago what happened with, when it comes with Microsoft is something called the context down object and again I call it medieval because it's just really, really crude and it had these certain events that you could hook into, I think it was before or after a particular method call and then you can add custom code to it but where they really dropped the ball was it forced you to inherit from one base class and sometimes you can't really do that with your model, if you want a clean model that has nothing to do with Microsoft's libraries other than maybe using some of the system libraries itself. That's Microsoft's implementation, there are other implementations like mine which is LinFu for dynamic proxies there's also Castle, these are dynamic proxies and basically what these things do is it takes any existing class and inherits from it at runtime and then it takes all the calls that you make to that inherited class instance and redirects it to some sort of interceptor that allows you to inject your own custom code. There are other types of writers, IL rewriters that people use for, Aspect Oriented Programming and one of them is LinFu the other one is PostSharp, which is pretty popular. Basically, what these things do is they take your compiled assembly, disassemble it to some degree and then reassemble it with their own custom code so now it has all those nice aspects that you'd want to put into your application like lazy loading or transactions or what not and the last and probably the one I probably would never ever do is the Unmanaged Profiler API. Basically, you're working with the CLR's, unmanaged C++ API for dealing with unmanaged code. You can pretty much hook into the JIT and actually change the instructions right before it actually compiles into native code. For me I only stick with dynamic proxies and IL rewriters because most of the time, they get most of the job done. If you're really masochistic I would probably recommend the Unmanaged Profiler API.

Scott Hanselman: IL rewriting sounds really scary. I mean it sounds like, it's just like initially, my initial reaction is like, "Wow, really? I've just compiled this and now I'm going to go and mess around it again?"

Philip Laureano: Well, actually that's one of the things that IL, misconceptions about IL that people afraid of because if you look at the IL, the IL language itself, it's a really simple language I mean it still has all the same good constructs that you would normally associate with your CLR 1.0 language. For example you've got your IF statements which is a little different, for example, you've got branches and there is no such thing as loops but the nice thing about using these libraries for IL rewriting is that you never actually have to do it yourself and these libraries whether, it's LinFu or PostSharp or Castle or whatever, they're all well-tested so you can't, there's no way for whatever IL rewriter you're using to screw up, I mean unless you do something really crazy with it. In most cases the uses are pretty benign. I mean I've seen couple of people put in Pervasive Method Interception throughout their entire app but I mean even in cases where they want to intercept everything and the kitchen sink, I've never seen any case where it actually caused something to break simply because we had our own test.

Scott Hanselman: So what were the some reasons why someone would be afraid of this, though, I mean can you understand my initial aversion of like I've just assembled this and I'm going to break it apart and then weave to it?"

Philip Laureano: I think it's more psychological than anything I mean I think, as programmers we are pretty much control freaks, we like that control over everything and one of the things, the trade off that you do have to have is that you have to surrender a part of your application and say, "Okay, yeah I need to add a little bit more code to this but when it actually changes the IL, it's going to be a little different.

Scott Hanselman: Right.

Philip Laureano: I mean functionally it's still going to work the same way like it's still going to do the same thing for your business app but when you look at Reflector, it looks completely different.

Scott Hanselman: How do you debug that? I mean if I'm going to go rewrite a bunch of IL, what happens when I decide to go on F11 through my code?

Philip Laureano: It depends on the particular IL rewriter, if you go dynamic proxies you'll be able to step right through it and you can just, you'll see where it stops. In other cases where you have IL rewriters that do post-build time compiling where they modify the assembly right after a build, it really depends on whether or not it changes the PDB file and updates the program debug information so that when you run through it, you'll be able to see what's going on there. So like I said, it really depends but there's so many options that you could pretty much go with whatever you decide to have, whatever fits your needs.

Scott Hanselman: Which do you prefer?

Philip Laureano: Well for me I'm kind of biased, I mean for a simple scenario I'd probably use LinFu. For more complex scenarios where it's just really off-the-wall I'd go with AOP which would be LinFu.AOP, you could also go with PostSharp or some other library...

Scott Hanselman: So just to make sure that for folks that are listening who may not be speak English as their initial language, he' saying LinFu, L-I-N-F-U and that's a project that's up at Google Code and then he' saying PostSharp, P-O-S-T-sharp.

Philip Laureano: Actually it's on GitHub...

Scott Hanselman: Oh, LinFu is not on Google code anymore?

Philip Laureano: It's still there for historical references but...

Scott Hanselman: Oh, but you moved it over to GitHub, I see there's a link right from there.

Philip Laureano: Yeah, there's a link straight away. Yeah LinFu is just one of those things where I just sat down and played around with IL and then I realized that there's so many other language which I could probably implement with just writing the IL yourself.

Scott Hanselman: Hey, it's alternate universe Scott Hanselman. We're going to thank our sponsors for making this free show possible. We've been doing Hanselminutes for almost four years now. We really couldn't do it without Telerik so I'm going to tell you about some of the stuff they're working on. If you're already started developing with Silverlight, then now you probably need a solid testing tool to do your Silverlight UIs. Unfortunately though, a lot of the tools available today only help you with unit testing. There's not really a good way to simulate the actual behavior of those end users unless you spend days and weeks doing a lot of manual testing, but the guys at Telerik have got a new point and click UI testing tool for Silverlight called the WebUI Test Studio. The beauty is that you can quickly record your tests with a cross browser recorder. You can enrich them with code if you've got really complex scenarios and on top of that, it supports all the standard controls as well as the Telerik Enhance Controls. You can verify not only Silverlight but even complex AJAX applications. Best part, WebUI Test Studio lives in Visual Studio so you don't have to leave your favorite development environment. You can check it out at telerik.com/web-testing-tools and be sure to thank

Telerik for supporting the show on their Facebook fan page at facebook.com/telerik.

Okay, before we start digging into what, how someone can get started with LinFu, talk a little bit about dynamic proxies, what is a dynamic proxy and how does that work?

Philip Laureano: A dynamic proxy is basically a special, when you look at it it's basically, if I were to write it manually and you were just to write it in say C# or VB, what you're really doing is you're inheriting from a class that has a bunch of virtual methods and for every single one of those virtual methods you redirect it to some interceptor instance that has all the information, like the type arguments, the method arguments, the method name, and based on that input you can actually change or make it act like a real object. It's a dynamic implementation of the proxy pattern, I think it was around 2004 or 2003, somebody figured out that you can take reflection.init and you can take any given type provided that it's properties were virtual and then redirect those calls back to something that look exactly like the object. I mean it was pretty brilliant I can't claim that I invented it because I didn't but that's basically how proxies work. On the Java side I believe they have it built in but for .Net side, we basically had to create our own.

Scott Hanselman: So dynamic proxies, and that's something I can get within LinFu. So you're talking about, you're saying that there are multiple flavors of AOP, there's dynamic proxies, there's IL rewriting and actually digging into the compiled stuff that you just made and emitting more IL and the LinFu framework has all those different things available. So I can use different tools, it's a whole tool box of stuff.

Philip Laureano: Yeah and the best part is you don't even have to see the IL, you never see it. It might sound scary because I'm saying that yeah you have to...

Scott Hanselman: You're giving up control?

Philip Laureano: Yeah, you do have to give up control but I mean it's pretty, for me, I think it's a worthy sacrifice compared to seeing all this stuff look like, at least on an architectural level like it's copied and pasted everywhere. Things tend up, I hate to generalize but for me, at least, it comes out much, much cleaner when you don't have to see like six dependencies in a hundred different places. Instead you probably just see it in one place and with Aspect Oriented Programming, you get to say where these particular dependencies should be at runtime and that's the difference with traditional Object Oriented Design, you just say, it's either there or it isn't there. You can't really say when it should be there or when it shouldn't be there and that one extra dimension makes a huge difference in making sure that everything is clean.

Scott Hanselman: Okay, so I want to get started with LinFu, I've got my, maybe I'll take NerdDinner right, I've got an existing ASP.net application and I want to add, let's just start with logging, what do I have to do to get started with LinFu?

Philip Laureano: Well with LinFu, depending on what you want to do with your ASP.net application, if it's MVC you probably want to hook it in to as one of the controller factory and it's like one of the very few controller factories that could pretty much take an infinite number of arguments and you could pretty much parse it out like that. So the first thing that you do is you hook it into your controller factory after that there are different, there's a gazillion different ways that you can tell the IOC container itself, "Okay, once you create this I want you to hook into this particular aspect or introduce this particular feature." Like for example I've got a friend in Norway and he was able to introduce WCF to an object model, that was just a plain old CLR object and you don't even see the WCF at all, it's just like one line and all he did was he just put in a hook into the container that says, "Okay, once you create something and once it comes out of there, I want WCF support," and it was pretty amazing considering it was just like magic, one minute it's there and the next minute it wasn't there but when you look at the object model it's just a bunch of properties.

Scott Hanselman: Someone had complained to me recently at a presentation about some of the complexities of WPF around event notification and how basic properties when used in WPF, have to get really complicated, especially with the notifications and stuff. Are there examples of using LinFu to clean up WPF objects, objects that are used within the context of WPF or Silverlight?

Philip Laureano: Oh no, there aren't any examples with LinFu and WPF but there a couple of examples with Bernhard Richter, I'll post his blog and what he did was he actually put up, he's got a project up called, I can't remember the project but we'll post it later but he has a project that actually puts an INotify property change...

Scott Hanselman: Right.

Philip Laureano: Implementation as well as ID error validation and it's pretty much transparent. For example, you can have this one particular interface that validates this one particular class type and you just drop the DLL into some directory and LinFu scans it and then injects it right into the container and the container is smart enough to say, "Okay, this particular instance needs to be intercepted and I need to put these features onto that one particular class or object." It's pretty smart and a lot of containers can do that but at least for LinFu, it's one of the interesting contradictions, it does a lot of complicated things but for most of the time you get most of the job done in about one or two lines of code and no fuss, I mean you never really see IL at all unless you really want to get into it.

Scott Hanselman: Gosh, what if I have to debug these things? What if I've changed something? You keep saying that it, I guess we both agreed that it can be scary but it's psychological and then you're saying that I don't necessarily have to see the IL, what if something goes wrong, how do I debug my logging if I can't even see where it got called?

Philip Laureano: Well you still to get actually see the logging because once the IL calls back to your code it tells you where it's actually being called from and in layman's terms it's like it's dialing home, it's pretty much telling you where it was intercepted, it gives you all the information you need to say, "Okay this was intercepted here, this was intercepted here," as well as it's, yeah you do give up some control but the control that it gives you back is you get to decide where and when these particular points of interception execute? For example in dynamic proxies, you will often get the method info and it will tell you everything you need to know about the particular method that is executing and it has everything there including the arguments whether it's generic or not. So basically, on one hand, yeah you lose control about how it's actually compiled but when it actually runs you get to decide where the things execute, how they should be executed and whether or not you should just replace the code altogether. I mean it sounds scary but it's actually quite simple.

Scott Hanselman: Okay.

Philip Laureano: And a lot of it is abstracted away for you.

Scott Hanselman: So I put this into NerdDinner then I plug it into my central controller factory and then what are some of the different kinds of interception I can do? I mean I can intercept at the class level, so I could say like I want all Dinners to be intercepted and then could I intercept more granularly than that?

Philip Laureano: Oh, absolutely, you could intercept at an instance level. So if you have one particular instance of that one class, you could just go ahead and intercept pretty much anything on that particular class instance of types of interception we could go over, at least for LinFu, is there's method body interception, that's basically you take any method body whether it's sealed, static, private or public and you can pretty much swap out the method body implementation. There are a few restrictions for example, you still have to return the same type but for the most part you could pretty much do whatever you want with it. The other one is method call site interception which is, would scare a lot of people, basically I can take any system or any third party method call and replace it with your own. For example, if you want, if you had a Hello World program and you wanted to replace it so that it would instead send Hello World over a network socket, you could easily do that with just and you just swap it out. The other one is the new operator interception which is the ability to take your standard C# new operator or in IL it would be the new instruction and you basically wrap an, in IL what it's actually doing is it's wrapping an IF statement around it and saying, "Okay, should I just let this thing create itself or should I provide something else in it's place?" And it's pretty handy in some certain situations once you want to trace object construction. The other one is field getter and setter interception. So for example, I could take any field and intercept the point where it's actually being accessed or changed. So if I were to set a Fu field I could actually prevent it from being set or I could actually just replace it with another value with whatever I want. The last type is what I call throw an exception catching which is the ability to dynamically catch any exception thrown from any given method. There is a small note with at least with LunFu's way of doing it which is it's kind of opt in. So you have to explicitly say, "I want to catch exceptions out of this method and if I do catch it I either want to swallow it or I can just let it throw itself back out. Those are the kind of interceptions that LinFu provides, of course your mileage may vary and I really leave it up to the responsible developer to actually figure out what they want to do with it but it gives you a lot of flexibility, at the same time you never ever see any line of IL, ever.

Scott Hanselman: Do you find yourself doing static IL rewriting, static IL rewriting makes more sense to me, the idea that I'm going to take some code and I'm going to compile it into IL and to DLL's and I think sometimes, some newer .Net developers forget that there is IL, I mean they think that they compile and this binary pops out, that there's IL in there and it makes sense that post-build steps I can deal with, like I'm emotionally I get that, but dynamic IL rewriting that seems scary like I'm over the, "Hey, I'll rewrite my IL," part of things but I'm not over the "I'll do it dynamically."

Philip Laureano: Well it's actually two steps, when you're dealing with static IL rewriting, it's basically that post-build step that changes your assembly. For example, I know PostSharp is, last time I checked, they do static IL rewriting so you could mark a particular attribute on a particular method and all of a sudden they would just change that one particular method. With LinFu, it goes dynamic by taking the same static step but instead of putting in a specific type of aspect or a piece of code, it wraps up a small little framework around your method so what happens is it will ask you, once that framework is in place and you run the program, it will ask you, "Okay I'm inside this method, should I intercept it or should I not?" If you don't want to intercept it, it acts like its not even there. So when we're talking about dynamic

IL rewriting what is actually, it's kind of it's static plus runtime interception, that's a more accurate term for it. The benefit, of course, for dynamic IL rewriting is you can pretty much decide where you want it to be. There's also, in terms of build time, it's pretty much negligible, it's not like static where you'd have to basically filter through everything and decide what you want to put in and what you don't because there's some decisions about what you need in the application that cannot be decided at compile time.

Scott Hanselman: Ah, okay and there is where it gets interesting and that's where, and I think of things like authorization.

Philip Laureano: Yeah, I mean there's no way you could actually put in authorization at compile time, it just sounds silly but at runtime it makes perfect sense because you need to be able to get this information when the program is running and of course you're not going to be able to get that if you have some that are statically compiled and you don't what machine it's going to run on and all these different settings that you might be worried about.

Scott Hanselman: So what do you think the future is for this, I mean is this going to be, I mean how many people, let me ask the hard question, I was trying to be nice but let me just ask this, who's using this other than Philip?

Philip Laureano: Well, right now this is the two framework geeks, I mean it sounds scary but I honestly think, at least on the .Net side, it's one of those rare things that you won't hear about. I mean on the Java side they've been using this a lot of AspectJ and they take advantage of this a lot and this is again, you did bring up a valid point, this is pretty exotic, I mean I don't recommend it for everybody but it's definitely another tool that you put into your tool box as far as people that use it, I mean pretty much anybody who uses NHibernate uses some tool of dynamic proxy, it's unavoidable, it's either Castle or Linfu and even in your practice I mean you probably used NHibernate more than a few dozen times but the thing that's nice is you never really know that there's a proxy there, it never really, I mean aside from maybe that setting that they say, "Okay, you pick one proxy," but for the most part, they never really, you don't even realize it's there and that's the beauty of AOP, I mean they're doing all this stuff for you but it's all in the background. So the future, at least for AOP, is that it's going to be invisible but there's still a lot more interesting things that I would love to dive into. I mean maybe I could talk about more on multi threading or use AOP to do transparent IoC container injection, for example one of the big things that they're trying to do with some of the containers right now with, even with MEF, the Managed Extensibility Framework, is the big question is how do you make it invisible? How you actually make extensibility invisible or at the same time allowing users to be able to take advantage of it? With AOP it allows you to add those features into place without necessarily telling the end user dev that you've got to put this in yourself.

Scott Hanselman: And I think that nails it, that's the way to do it. There are things that developers shouldn't have to worry about.

Philip Laureano: Exactly.

Scott Hanselman: I mean think about separation of concerns, this allow you to do separation of concerns organizationally.

Philip Laureano: Basically, and the other thing is I mean there's so many things that the developer has to worry about nowadays that I mean if you're talking about like the second or third tier framework guys like the architects, these guys, I mean for me, it's my job just to make sure that when it gets down to the grunt level or your average dev, they don't see this, I mean they don't have to worry about lazy loading, all they've got to worry about is the business domain and at the end of the day, that's really the focus here. We want to take away all the plumbing so that once you strip everything away the only thing that you see is the business domain and I think that's what's missing right now because once you look at these different frameworks they have everything from security to logging, to transaction management and it gets confusing once you go through the code. AOP offers that kind of separation, of course, but it's going to take some time before people wrap their heads around it. I mean it took a while for people to switch from imperative programming to event driven programming and I think that's the shift that we'll see within five or 10 years. in the same way everybody takes Object Oriented Programming for granted nowadays, I mean that's I think where were going to be within five to 10 years with AOP.

Scott Hanselman: In closing, give me something that people can take action on. So someone has listened to this, they're sitting in the driveway of their house or their place of business and they're like, "All right, I might go download LinFu." What's an exercise that they can do with an existing application that will make them, that'll light them up?

Philip Laureano: Well, one of the things that you can do with LinFu and even AOP is XCOPY deployment, I mean we're talking about instant pluggability I mean we've been talking about this for awhile, I was doing this back in 2007. Basically, the idea is with AOP you could just plug things right in without even changing a lot of code. For example, there was one project where a guy took LinFu and it took his 5 year old web app and they couldn't change the production code so what he did was he actually plugged LinFu into it and he was able to change it without much changing a single line of code, everything worked, all the tests pass, it just works.

Scott Hanselman: So I could change an app I don't even have the source for?

Philip Laureano: Correct.

Scott Hanselman: That would be interesting, to add logging to an existing app but you're either afraid or unable to recompile it?

Philip Laureano: Oh yeah and it happens a lot, there are other things I've played around with is you could actually add, wrap unit tests or observe existing code so you could see what the input and the output of each method is. So I could actually write your Michael Feathers' Working Effectively with Legacy Code, type tests where you put in a legacy characterization test and you don't even have to touch the code, you just have to observe what comes in and what comes out and then you write the assertions but that kind of thing is it's only possible with AOP because you don't want to mess with production code especially if it works.

Scott Hanselman: Yeah, definitely.

Philip Laureano: If you've got unit tests it's even better because even if you plug the AOP right in, you'll still be able to verify that it works and 99.999% of the time it works and you could easily verify it with the test if it doesn't.

Scott Hanselman: Very cool. Well, Philip Laureano thank you very much for taking the time to call in and I don't know you're up at one in the morning in Hong Kong.

Philip Laureano: Not a problem dude.

Scott Hanselman: I going to put links up on the show notes for your blog, for you on Twitter, twitter.com/philiplaureano...

Philip Laureano: Great.

Scott Hanselman: And also LinFu which is now hosted at GitHub.

Philip Laureano: Correct.

Scott Hanselman: Thank you so much for the time.

Philip Laureano: Thank you.

Scott Hanselman: This has been another episode of Hanselminutes and I'll see to you again next week.