



[HTTP://www.dotnetrocks.com](http://www.dotnetrocks.com)



Carl Franklin

Carl Franklin and Richard Campbell interview experts to bring you insights into .NET technology and the state of software development. More than just a dry interview show, we have fun! Original Music! Prizes! Check out what you've been missing!



Richard Campbell

Text Transcript of Show #453
(Transcription services provided by [PWOP Productions](#))



Peter Vogel uses Code Generation
June 9, 2009
Our Sponsors





Geoff Maciolek: The opinions and viewpoints expressed in .NET Rocks! are not necessarily those of its sponsors, or of Microsoft Corporation, its partners, or employees. .NET Rocks! is a production of Franklins.NET, which is solely responsible for its content. Franklins.NET - Training Developers to Work Smarter.

[Music]

Lawrence Ryan: Hey, Rock heads! Quit cracking your pretzels and listen up! It's time for another stellar episode of .NET Rocks! the Internet audio talk show for .NET developers, with Carl Franklin and Richard Campbell. This is Lawrence Ryan announcing show #453, with guest Peter Vogel, recorded live, Tuesday, June 2, 2009. .NET Rocks! is brought to you by Franklins.NET - Training Developers to Work Smarter and now offering DotNetNuke video training with Chris Hammond from Engage Software on DVD, dnrTV style, order your copy now at www.franklins.net. Support is also provided by Telerik, combining the best in Windows Forms and ASP.NET controls with first class customer service, online at www.telerik.com, and by CoDe Magazine, the leading independent magazine for .NET developers, online at www.code-magazine.com. And now, the man who says, "I got your Project Natal controller right here," Carl Franklin.

Carl Franklin: Thank you very much. This is Carl Franklin.

Richard Campbell: And this is Richard Campbell.

Carl Franklin: We're at Richards house in Vancouver, British Colombia, actually. I just got here and I looked around at his new house and whoa!

Richard Campbell: Thank you, sir.

Carl Franklin: Whoa.

Richard Campbell: We've worked pretty hard on it. It's far from done as you can see. There's still a lot to be done but...

Carl Franklin: But now I can see why you keep talking about it.

Richard Campbell: It's taking shape, how's that?

Carl Franklin: It's pretty awesome.

Richard Campbell: It's great.

Carl Franklin: And of course he's got a slab of ribs on the bar-b and we're going to have a good time tonight but that's not why we're here.

Richard Campbell: No, it's DevTeach. That's why everybody's in Vancouver.

Carl Franklin: Yeah. So we're here for DevTeach. We're going to just do a short intro today but I thought we'd say hello from Vancouver and we haven't done anything at DevTeach yet so by Thursday we'll have more to report about the show.

Richard Campbell: Absolutely because by Thursday we'll have done our Wednesday night .NET Rocks Live.

Carl Franklin: That's right.

Richard Campbell: So that should be interesting.

Carl Franklin: It should be a lot of fun. Now let's roll the tape of previously recorded interview. Our guest today is Peter Vogel. Peter is a principal in PH&V Information Services. PH&V specializes in the development of n-tier applications using .NET applications. PH&V's clients include Volvo, the Canadian Imperial Bank of Commerce, and Microsoft. He also writes the Practical ASP.NET column for Visual Studio Magazine, along with feature articles on emerging .NET technologies. Peter's most recent book ("Practical Code Generation in .NET" from Addison-Wesley) is aimed at providing developers with the tools to incorporate code generation into their development toolkit. In addition to teaching for Learning Tree International, Peter wrote their ASP.NET and Technical Writing courses. He has written articles for every major magazine devoted to development with Microsoft tools. Peter also has presented at conferences around the world. You can reach him at peter.vogel@phvis.com. Welcome Peter.

Peter Vogel: Oh, thanks. I feel just like Steve Martin out promoting his new banjo album.

Carl Franklin: So, you guys go back a long way, you and Richard?

Peter Vogel: Yeah, the first conference I ever got to present at and the only reason I got to present that is about a week before the conference was supposed to go out, one of the speakers dropped out and recommended me and it was just up the road from where I live, it was in Toronto, Ontario, Richard is also a Canadian and...

Carl Franklin: You're a Canadian?

Richard Campbell: Yeah, who knew?

Peter Vogel: I'm a Canadian, All wonderful people are...



Carl Franklin: No, no, Richard. Richards Canadian?

Richard Campbell: All this time.

Carl Franklin: Oh, my God!

Peter Vogel: I thought you knew.

Carl Franklin: Nobody told -- I didn't get the memo.

Peter Vogel: And I should tell you, Richard you are in fact, I believe married to another Canadian.

Richard Campbell: Oh, yes. She was born in the U.S. but she is a Canadian.

Peter Vogel: Yes, so...

Carl Franklin: Oh!

Peter Vogel: So you know, the secrets are coming out, I'm afraid.

Carl Franklin: Peter, you're rocking my world.

Peter Vogel: Anyways, we both ended up at this conference in Toronto and that's where I met Richard and I'm not ashamed to say that he mentored me.

Richard Campbell: Ah, sweet. That was Val Madison's show, wasn't it?

Peter Vogel: That was the one at the Prince Hotel and we did one day at the Royal Ontario Science Center and the other day, back in a conference back at the hotel.

Richard Campbell: Right, right.

Carl Franklin: Richard is a mentoring kind of guy, he really is.

Richard Campbell: Yeah, that was like 1995. That's a long time ago.

Peter Vogel: You have to remember that in Canada, if it's consenting adults and it's done in private practically anything is okay.

Carl Franklin: That's true. So, Code Generation...

Peter Vogel: Yeah.

Carl Franklin: This is not a new topic, in fact Code Generation seems to have taken off quite a lot

recently and people are using Code Generation, people are using ORM's and a lot of tools to help them be more productive. I guess I'm just going to throw that out there, what has been your experiences with this?

Peter Vogel: Well the whole .NET framework is absolutely dependent on Code Generation. Just any number of things that we do wherever you go into a designer, like dragging controls onto an ASP.NET page or Windows Forms page or dragging, setting tables and stuff up in the Entity framework, almost anytime that you're with a designer behind the scene, Visual Studio and .NET are conspiring together to generate code for you. Anytime you do declarative coding, like putting an attribute.

Carl Franklin: Yeah.

Peter Vogel: Behind the scenes, somebody is either interpreting and writing a general purpose program, that was the bad old way of doing things, the general purpose programs which are evil or more likely, for instance, with attributes I bet you could look up at the compiler when it sees an attribute, it actually generates the code of that attribute represents. So, .NET couldn't exist and certainly couldn't run as fast as it does without Code Generation but the problem is developers don't use it. It's something that, I don't think developers use it or not that anyway, and it's something somebody else does, it's not. I'm going on and on but the neat thing about .NET is everything you can do in .NET you generally see doing, do use some .NET object, same with Code Generation, all the things that Microsoft and Visual Studio and .NET and all of those various components entities you're using to generate code are built into the .NET framework and built into Visual Studio and just lying around waiting for you to use them and developers don't.

Carl Franklin: I guess the key to Code Generation in doing effectively is, first of all, it's always been a rule of mine to never use a code generator to generate code that you don't understand. In other words, what the thing is generating, you need to know how to do, you at least need to know what it's doing, otherwise, you get surprised you have a bug or something and now you're looking through hordes of codes that you don't understand the flow of and all that kind of stuff. So, some of the most successful code generators have been those that use templates that you can create yourself so that you are a part of the process of what gets generated.

Peter Vogel: Oh, I think you guys had Kathleen Dollard sometime in the past, didn't you? She wrote a book on...

Carl Franklin: She did.

Peter Vogel: Code generation for .NET.

Carl Franklin: Yeah.

Peter Vogel: And she had five or six rules for code generation and that was like number one or number 2 and I think that's really true.

Carl Franklin: Yeah.

Peter Vogel: And she really takes that template to perfection. She uses XML and XSLT which the T in XSLT, of course, stands for template to generate all of the code and she says this whole, she built this, well developed, did you get her book? She's got this well-developed framework for executing those templates and code. The thing I don't like about it, but A) it's just XSLT and of course, all sorts of people can't program XSLT, though, for anyone looking for an XSLT consultant, I can't but, and then the other thing is it happens outside of Visual Studio and I think one of the reasons that code generation has been successful in making .NET possible is that it's fully integrated with Visual Studio but we don't have access to those templates and we don't have access to. We can certainly look at the code especially in Visual Studio 2005 and 2008 if you're debugging, you frequently find yourself dropped into this code you've never seen before in your life with this naming convention...

Carl Franklin: Right.

Peter Vogel: You don't use and that's all been written by the great minds at Microsoft.

Carl Franklin: Yeah.

Peter Vogel: So yeah, I think if you're going to write it for yourself, if you're going to write it for your organization, then you do need to use some sort of template, some sort of template-based process, so that you can manage it and you know, the other thing, 75% of our time is spent enhancing existing applications. So you've got to assume that any code generation solution that you build, you're going to spend 75% of the time you ever spend on it is going to be making it better or adapting it to changes in the world or fixing bugs or whatever.

Carl Franklin: Yeah. The other option and maybe you can use both of them together is to use sort of a ORM thing where the Entity Framework, your or any of these other ORM's where you can NHibernate where this sort of the code gets generated on the fly based on what you, what exactly you're trying to do. It's a little bit different.

Peter Vogel: It is but I mean, fundamentally, what you do, if you look at those designers, what happens is, I'll use Entity framework because everybody can start on Visual Studio 2008 and play with that. You end up with some tables and start sitting on your design surface and then you can go and look at the source and see that what happens is you're moving these things around graphically, it's generating XML tags because that's what everybody does, is XML tags but then the code generation process kicks in and reads those XML tags and generates real .NET code in one of the two official languages C# or Visual Basic, Canadians understand two official languages, and that language, those XML tags that are being generated from the designer are really a kind of declarative programming language. I mean, that leaves us inevitably to saying, "Oh yes, we have a domain specific language." And developers can do that too, I mean there's nothing stopping you from using existing code generation solutions like Entity Framework, like NHibernate or whatever and then turning right around and saying, "Well, you know what, I want to generate code, here's a whole bunch of code that I write over and over again and I'm sick and tired of writing it over and over again, it's a waste of my time, the next time I do it, the only enhancement I can make to it is I can screw it up.

Carl Franklin: Yeah.

Peter Vogel: Right? I can get it wrong because it's already perfect. So what you want is some way that you can put in a specification and you'll probably do it in XML, but you don't have to, put in a specification and let Visual Studio and some other stuff I've written right out all the dull, boring, repetitious, I'll-only-get-it-wrong, .NET code for you in the language of your choice.

Carl Franklin: Do you use code generation in concert with other tools such as Entity Framework or maybe CSLA.NET Frameworks and I mean, you're adding complexity as you add more tools in the mix but I imagine that you can end up with a really powerful solution where you focus on UI and business objects, for the most part.

Peter Vogel: Sure, let me give you my favorite example because it's so simple and it's something that everybody goes, "Oh, that would be nice." Let's say you put your connection strings in your config file, either app.config or web.config, so there's, I'm picking tools that we all know, any .NET application has an app.config or web.config on them. You put your configuration in there you put your connection strings for your database but if you go to retrieve them you have some whacking long namespace you use and you get down to configuration manager and then you correctly type in the name of your connection string, you guys haven't

seen me type but the odds of me correctly typing in the connection string, vanishingly small, I screw up all the time and you don't find out until you execute that line of code because it is a string. So I've got a little code generation tool, the first one I wrote, simplest one I ever wrote and every time I save the web.config file, it looks in the web.config file and it sees if I have any connection strings and it generates a class, that class has a static method for every connection string, I'm sorry static properties, for every connection string and that property has the same name as the connection string. So every time I close my web.config file, I get regenerated a new class called configuration manager, I hit the period, I get an IntelliSense drop down list of all of the names of all of my connection strings, I pick the one I want, even I cannot screw this up any longer and if I type in a connection string that doesn't exist, I get warned of design time, with a little red wavy line which I don't get if I just pass the name and then from there you can go on and that's easy it's together, it's something every developer could use, something you can build in the morning.

Richard Campbell: Isn't that more of a designer rather than really a code gen tool?

Carl Franklin: Designers are code generators.

Richard Campbell: I guess that's true, the end result is a little generated strip of code and this gives us a string but it feels like a designer to me.

Peter Vogel: Well, yeah, if you want to say web.config, your web.config file, your config file is your designer, yeah it is. Boy, in this case I bypassed the designer because I am actually putting in the XML tags myself.

Richard Campbell: Right.

Peter Vogel: Because that's what you have to do anyway. I mean that's the case that I want to make for code generation is that, that's a simple little point solution and it's like a drug because once you start doing it you go, "Wow, what else could I get this thing to generate for me?" and from there on you just start building until you're generating things that complement Entity Framework. You have a designer that generates your user interface objects for you or your workflow objects for you. I mean, the first one is free and for me that was the one that was free.

Carl Franklin: What are some of the, what are some of the dark, dark downsides of code generation? Obviously, it can't be, you obviously have to walk a line between control and usability and productivity and control but what's the pitfall?

Peter Vogel: Well, I think the major pitfall that I've ever faced is you pick the wrong project to use code generation for. Fundamentally, code generation is a niche solution, it's a big niche but it's a niche solution and if those situations where all the parameters are known at design time and so now you've just got to thresh it out with the necessary code and you don't want to do, at least I don't want to do code generation at runtime. I want to supply all the necessary information and then it generates the code for you. Then you run into the problems, like you say complexity and productivity and that is that a lot of the tools for generating code in the .NET environment can result in you writing very complicated code in the sense that this is code that you do not want to have to maintain.

Carl Franklin: Yeah.

Peter Vogel: You do not want to---like if you're using, a lot of people go into code generation go, "Oh, well Microsoft got this tool called CodeDOM and that's a code generation tool." It's not. CodeDOM is the tool you need if you want to generate language neutral code, that you say, okay, runtime will decide or we'll decide later on if we're going to generate C# or VB or J# or whatever and so they get involved in CodeDOM and it's like an eight to one ratio, if you want to generate 10 lines of your own code, you're going to have to write a hundred lines of CodeDOM codes and you can imagine trying to maintain a hundred lines of code. So now you've got to think about the things that we always think about which is good program design, good architecture so that as you are building your code generation solution, you're building something that you can, in fact, maintain and enhance and adapt to the world around you.

Carl Franklin: What about database design? Can you design your database in such a strange way that it will throw off code generators?

Peter Vogel: Oh, wow, funny you should ask. I've got a client right now, we're building this system for the client and the idea is that as data is flowing into the system, we're going to have this monitor that watches the data going past and look for particular conditions and when these conditions happen go out and perform some activity like for instance notifying the system manager that this condition has occurred. We wanted to be completely user configurable, we want the user to setup what condition to look for, we want the user to be able to pick from a menu of actions or send us a dll with their own code in it to be activated when these conditions are met and all of that stuff. Now in the bad old days, what we would have done is we would have had the user enter, "Okay, check this field, compare it to this field and by the way, or this value and by the way, this is a numeric comparison and in my code, I would

have had a whacking big select statement that says, "Okay if it's this field, if it's stringed, do these type conversion, if it's a numeric value, do these types of conversions, if it's a greater than operator do this, if it's an equal to operator, do that, blah, blah, blah, we don't do that. Instead, what happens when the user saves their changes, we look at the set of tasks and we generate a class with one method for every rule that they've put in the table and we say, "Oh, okay, this is this field so we generate code that uses that field. You say it's this operator so we put the equivalent operator in that programming language into the code and we say, okay it's this data type so we write all of the variables out, we declare them as that data type and everything." So instead of ending up with this whacking big select statement check and see what operators they picked, we end up with typically like a method with two lines of code in it and that are the representation in the language of our choice of what the user entered in the database. So there's just one example where the user puts in stuff and based on what the user typed in, we generate the code to support that activity.

Carl Franklin: Now, you also, I mean putting my Agile hat on here, you have to take into consideration that your relationships in your databases, in your tables and all of that might not be as granular as you want your classes to be, you may end up with classes that have too much in them, procs that are attached to entities and things like that, so you end up refactoring and do you refactor your code generation templates or do you refactor after generating the code and what if you need to do that to some entities but not others?

LAST EDIT 20:33

Peter Vogel: Again, I think Kathleen Dollard brings out in her book, I certainly bring it out in mine, is that sometimes you want to provide the option for the user to step in or the developer to step in and add customizations to the generated code and this happens all the time in the .NET framework, if you're working with, I'll pick ASP.NET pages because that's what they use most.

Carl Franklin: Yeah.

Peter Vogel: You can overwrite methods or you can insert code into events and effectively what you're doing is you're customizing the processing that the underlying page object when you go through. So you build that into your code generation, you can say, "All right I'm going to generate this code but at these points my code is going to call out to some other events, some other methods or we, or alternatively, the developer can take my generated code and override, allow them to override some methods or just

inherit from my object and build on top of it." The thing you want to avoid in most situations is you don't want to generate the code and then say now the developer is typically or even often going to have to go and rewrite that code.

Carl Franklin: Yeah.

Peter Vogel: So I'm going to have to plead the point of code generation, it'd be better off just...

Carl Franklin: Exactly.

Peter Vogel: Let the developer write their own code but you do want to provide and what I normally, when I'm generating code I would generate a file of generated code in one partial class and I'll generate another file with another partial class in it, that's all the customization stuff so the developer can go in if they need to and the connection manager I talked about earlier does that because sometimes you need to fiddle with the connection strings, sometimes the connection strings has parameters and stuff in it that you want to substitute in at runtime so that it connects in the second version of my connection manager generated customization file and there was a method that was called before the connected string was returned where the developer could put their own code to substitute in perimeters in you just fiddle with the connection string before it goes back to the application.

Richard Campbell: Where does the T4 technology fit into the code generation equation?

Peter Vogel: Well that's very cool stuff because that really goes back to that complexity issue that Carl's raising earlier and that is if you write, you could write code to write code and that code because of it's sort of meta code, I guess, sort of has an inherent level of complexity. Well, when Microsoft is rolling out their domain specific language package, they rolled out this T4 templating and if you used to use, in the bad old days we're working with ASP, you had an inter-mixture in your ASP file of your own code, and ASP tags and stuff and that...

Richard Campbell: Yeah the write once read never code.

Peter Vogel: Yeah, exactly, that's what T4 does but in a better way, it's still not perfect but it's a better way. What you do in a T4 template is you have a combination of code that you want to be dropped in to your generated code file and in most generation solutions, a significant portion of code anywhere from like 50% to 80% is the same for every generation. Going back to the example that I have of generating a little method for every rule that the user created, I mean what do we change from one generation to

another, well, we change the operator and we change the name of the field that's being checked, but that about it, the rest of the code was boilerplate, so the template you just type in the code that you want to end up in your code generation file just like you would in an XSLT template where code and not an XSLT tag just ends up in your output file and then you can surround that with whatever and only code that you need, so your code that generates code becomes a small part of the template file and then the code that, the generated code that doesn't change from one template to another just, it's just sitting there, easy for you to read, easy for you to modify, easy for you to change and invoking a template is relatively, it's a very simple method to, three or four lines of code you can load a template into memory. Actually anybody listening to this can do a template right now, open any project in Visual Studio 2008, add a text file, give it the file type, TT, type anything you want into that template, into that text file and close it and you'll find that if you show all files, sitting under that TT file, you now have a C# file with whatever you typed in your template sitting in that C# file it won't, of course, compile unless you typed in C# code into that text file and that's how easy it is to use the T4, I think it stands for some thing like Text Templating Tool Kit, I've lost a T in there, but that's how easy it is to use.

Richard Campbell: T e x t T e m p l a t i n g Transformation Tool Kit.

Peter Vogel: That's the word I forgot, Text Templating Transformation, your wife transformed from an American to a Canadian, Text Templating Transformation Tool Kit. Any file with the TT extension will automatically invoke the T4 process or when it's saved, there's Visual Studio conspiring with .NET to generate code and the resulting file is automatically tucked as code behind file underneath, in the solutions explorer, underneath your TT file and anything you just typed in is just dropped into your generating file. Now you can go and start adding your own code to modify and in most code generation solutions you don't want to want the same code every time, right? So you're going to have to add some code to play with it and alter it a bit.

Carl Franklin: This portion of .NET Rocks is brought to you by our good friends at Telerik who bring you this message. One of the drawbacks of using third party tools is that you have to deal with numerous vendors so say goodbye to consisting quality in service level. Fortunately, that's not always the case. Our friends at Telerik, for example, are true one-stop shop for .NET. They recently rolled out their Q1 release which is just packed with good stuff. Start with Silverlight, an incredible grid, chart, editor, and everything else, a whole suite. A 3D chart, yes, 3D in Silverlight is coming soon as well. The traditionally strong ASP.NET AJAX got even cooler. New

controls, Visual Studio extensions for quick project kick-starts, new examples and scans, you name it, and how about web testing. Yup, Telerik is now offering a powerful solution for automated testing of modern AJAX applications. It's called WebUI Test Studio and is developed in partnership with ArtOfTest. Then comes reporting, WPF, Win Forms, but I'm running out of time so just go to www.telerik.com, and be amazed. And hey, don't forget to thank them for supporting .NET Rocks!

Richard Campbell: I do feel like T4 is sort of a side show, they've not really taken it seriously, there's not much in the way of samples, Microsoft put it in there and then it's almost like they didn't want to talk about it, so they hid it.

Peter Vogel: And I think the emphasis, I mean it was something they did as part of rolling out the domain's specific language package and I think it hasn't got the attention it deserves because all the emphasis was on these other stuff to make DSL work and so T4 which is just a fabulous entry point for doing code generation because it's so easy to get started and the nice thing about that old ASP model is, write once read never code, was it was easy to start typing in it and start playing with it. Behind the scenes, it's actually a very complicated process because what happens, it looks at your T4 template generally a program and then that program generates your code, unfortunately you don't have to be aware of any of that, it's a much simpler process if you're just, if you're just looking at it but actually it generates a program to write your program. So, we're not even writing code to write code, we writing code to write code to write code.

Richard Campbell: Yeah.

Peter Vogel: But it works and it works very reliably and very quickly.

Carl Franklin: What about dynamically data driven subroutines or classes, in other words, instead of using an external program to write code, why not write code that's data driven?

Peter Vogel: And you know what is, if you don't know the parameters at design time, if you've got to pick up the parameters, the input parameters at runtime then that is exactly the right way to go, you have no choice, you've got to do that because you don't want to be generating code at runtime. For one thing, it's very difficult to read and debug but if you know all the stuff at design time like these rules that you're entering, users entering them into a database, so soon as they enter the database we know what they are, the rules won't be executed until some time later when the actual data starts flowing through the system so we can generate code but the problem with

general purpose code and that's the term I use to describe those things which in you know, data driven code, that general purpose code is that it's 1) necessarily inefficient, it's got to read the data out of the table, 2) as soon as you get any sort of complex problem at all, you find yourself doing all sorts of typecast, begin...

Carl Franklin: Branching and forking.

Peter Vogel: Branching, I mean you look at most of the design patterns that I'll adopt and most of the design patterns are designed to remove branching, they're designed to replace branching with just use the right object.

Carl Franklin: Yeah.

Peter Vogel: And SELECT K statements and stuff like that rule thing, I said we would have had a SELECT K statement what each or alternately, you're going to have to invoke a runtime interpreter and that's got it's own performance problems and complexity problems related to it and generally speaking, if you're building high performance applications, you want to do it and furthermore I think when you do a self-generated purpose code, testing that code because of all the branching...

Carl Franklin: Yeah.

Peter Vogel: Becomes tremendously onerous.

Carl Franklin: It's true.

Peter Vogel: So on the other hand, the generated code is typically very simple. So you build a system that generates the code you want instead of having to execute 50 lines to general purpose code, what you do is you basically call that one line of fully compiled code that's sitting there waiting for you.

Carl Franklin: Here's another thing, trying to put logic in an external file like an XML file that you can declaratively, that you can declare. Basically it's a declarative model for business logic and then your generated code reads that logic in as directives.

Peter Vogel: Yeah and there is, I think what you suggesting there is there's a hybrid approach.

Carl Franklin: Right.

Peter Vogel: I mean if you go in a fully data driven thing, it would read in an XML file and parse it out and have all those K statements and those typecast because everything XML files string that we're talking about.

Carl Franklin: Yeah.

Peter Vogel: If you go fully the code generated solution, right, you put the, you put your code in an XML file you put, you describe, essentially, what you're doing is you're providing the specifications for the resulting program.

Carl Franklin: Right.

Peter Vogel: So you come up with some XML syntax that let's you do that and you put that code in an XML file and then the fully generated solution would look at that and generate the code for you.

Carl Franklin: Right.

Peter Vogel: But what if some of the parameters, some of the input aren't known until runtime or you want to be able to customize...

Carl Franklin: Yeah. You want to be able to customize without regenerating. I mean that's usually what bites you right, because the business logic is always the stuff that has to be handcrafted because it's always so particular and different from one thing to the next. So it's nice to be able to put that stuff in a place where it can automatically be read in and implemented without having to, "Okay now that I've generated my code, fork off these classes that sort of get injected or inserted into my generated code."

Peter Vogel: Well, I mean, you're exactly right. It's part of a big application, what's going to happen and so, oh, all right we want to change the way this works and you go in and change the code generation solution or change the specifications in your input file and now it's generated the code, we're now into version control, we're into getting waiting for another release to get this out of there, why don't I just generate, why don't I just drop this XML file on my, in the appropriate folder and it will be read and controlled processing when my general purpose program starts up but you know, with reflection especially with MEF now, you could generate a new DLL and then drop that DLL onto your new, into the appropriate folder as easily as you could drop that XML document and get all the performance benefits and all the cleanliness benefits and all the, oh we don't have to keep reading this file to memory, I mean the slowest thing you could do in the world is to read something off of your hard disc, the DLLs is more likely to be cached than XML files are. So if the concern is, "Oh my gosh I have to rerelease the whole application, then don't regenerate that DLL and drop that DLL in, maybe do some MEF coding to see what DLLs are lying around and waiting for you to learn.

Carl Franklin: Right.

Richard Campbell: I'm always concerned with any of these technologies that as soon as you get past sort of the demo case, the sample case, the complexity makes it unmanageable, just sort of gets away from you.

Peter Vogel: And I think that's again, a terrifically valid concern and we are in fact, when you're writing code to write code, you're in fact creating meta code, the code that generates your code is meta code and I think the tools that Microsoft was providing before and the tools that I've got in my book all very good but if you don't think about it in advance, if you don't architect it, you're going to end up with something that if you're doing anything interesting is unmaintainable, unmodifiable, unadaptable and unextendable, that's what drove Kathleen to using XSLT templates because they are more modifiable, more extensible than pure code generated, pure code generating code solutions, template generating code. That's one of the reasons I'm so excited about T4 is that it does go to a template pattern where right off the bat, any code that is boilerplate from one implementation to another is just sitting there in a text file, you can just see it and look at it, heck, you can copy and paste it into the class file and copy and paste into a class file and make sure it even compiles and you only have to write code for the typically small percentage of your solution that does vary from one code generation to another code generation. You only have to write code that fondles the lines that actually change, the boilerplate code just sits there, it's a very clean situation. I've done a couple of code generation solutions where, literally, we just copied in text files and assembled the text files in the appropriate order and then modified two or three lines, we had parameters in there, use the string.format replaceable parameters and that was how we generated code. So if we needed to change the solution, we didn't really had to change our code, we just changed these text files that had all the components that were being assembled. So if you do think about it and say, "Okay, what does change?" And put the complex, what does change from one code generation to another and put the complexity there and saying, "Okay this stuff doesn't ever change from one solution to another." Then put that in a database or in a text file or something and that can, that, all by itself can dramatically reduce the complexity in the solution. So, you do have to think about it, you can't, somebody once told me about threading applications, you can't evolve a threading application and I don't think you can evolve a code generation...

Carl Franklin: You mean a multi-threaded application, yeah.

Peter Vogel: Yeah, a multi-thread, you can't evolve that, you've got to think about it in advance and make sure you're doing the right thing.

Richard Campbell: Right and getting back to that whole write once code, it's easier for you to redo the template than it is to try and fix the template.

Peter Vogel: Exactly right and hopefully there's only small parts of it that you have to redo but ideally, good program design doesn't go away. If you apply the same principles of good program, good program design, good architecture to your code generation solution, you don't have to end up with read only code and you can in fact get code that is perfectly maintainable and extensible but going back to the earlier point that is data driven, that the data in this case are, is the boilerplate code, that sits on a text file or sits on a database, very easy to look that stuff up and change it and extend it.

Carl Franklin: So it sounds like you're saying that there isn't any one way to do this right. I mean a lot of it depends on your project and on your data and on the tools that you plan to use together. What we've talked about is just a myriad of options to, imagine that there's the audience is thinking, "Hmmm, which one, which tools should I use? Which--is there any way that I can get started? Give me some good advice to get started, when should I use what tools?"

Peter Vogel: Yeah you are my best friend in the whole world because that was the reason I wrote The Practical Generation Code In .NET book because there are an enormous number of tools, there's the codeDOM which most developers don't need because they don't need to generate language neutral code, their shop uses C# but that's the one they heard about so they're stuck with it. There's a file code model which will generate enormous amount of code for you and yes, to a certain extent, language neutral. You want to build it in to Visual Studio so that code is automatically generated, you don't want to have to start up some other application or constantly be shelling out of Visual Studio, you want to be able to create visual studio add ins that will do this for you. So the--and a documentation seems to be sort of spread evenly over the whole internet, it's not gathered into one place. Richard's comment about T4 not getting any attention is another great, there's not many examples and there's not much there. So what I wanted to do was, in the book, was basically devote a chapter to each one of the tools, not the whole tool but just the part that was relevant to code generation and then provide two case studies to show how this stuff would work together. Now, unfortunately because of the publishing thing, the book came out before T4 and I also wanted a book that would apply to Visual Studio 2003 right tool to 2010 when it comes out, so I left out two things, I left

out T4 and I left out VS packages because they're really only available in Visual Studio 2008 but they're going to be baked in to 2010. So, I'm going to be self-publishing an expansion pack that covers T4 and then the VS packages but if you wanted to have a one-volume reference to all the stuff that you needed to know in order to do practical things, actually solving problems developers have in Visual Studio 2003 to 2008 get that book. If you don't want to buy the book and how many does buy books anymore, whenever we want anything we just go on the internet and search on it, you want to look at the file code model. File code model, very easy object model to use, it's language independent by and large and it will allow you to add files to your solution in visual studio, it will allow you to generate class modules, most controlled structures, if then properties, methods and then after that it's just string insertion to say, "All right, I want this line of code written in there." Another great tool, little bit awkward to set up Code Tools, if you've ever created the data set. If you're going to look at the data set in solution explorer and look at its property window called Code Tool and that Code Tool is a little DOS program that every time you save your data sets designer, it looks at the XML file that's generated when you're working with a designer and looks that XML file and writes out all your data set code for you and I did an article for Visual Studio magazine, am I allowed to plug them?

Carl Franklin: Sure.

Peter Vogel: All right, there you go, I did an article for Visual Studio magazine on how to create your very own custom code tool and again the information for how to do that is all in the net, it was just spread out evenly over the whole net and I'd love to claim I did original research but mostly I just dragged it all together into one place because my goal was that if you're creating a code tool, you go in, you configure your Visual Studio project and then every time you want to regenerate your code tool you just click the build menu, you shouldn't have to shell out anything, you shouldn't have to manually install into the gaps or anything like that and now you can just create an XML file put the name of your code tool and it will write out the code for you.

Carl Franklin: You mentioned, get that book, you were talking about your book, Practical Code Generation...

Peter Vogel: That's my book.

Carl Franklin: So that's not on Amazon yet, is it?

Peter Vogel: No, no. Addison-Wesley should have it out about a month or so.

Carl Franklin: Oh, okay, I was going to ask for a link to it but I couldn't find it.

Peter Vogel: Ah...and I can't tell you how sorry I am about that.

Carl Franklin: Oh, well but we'll keep looking for it.

Peter Vogel: I mean you don't make money from books but I am sorry you cannot get a link to it but I'll send it to you when it comes out.

Carl Franklin: Okay and if you just search for practical code generation in .NET in a month or so?

Peter Vogel: It will be there.

Carl Franklin: It will be there, make it a Google watch word.

Peter Vogel: Yeah. Practical is the thing and the tool code solutions, the two case studies that I used, one is the connection string manager which is something that I think everybody needs and then the other one was, I'm an ASP.NET guy, so it was an ASP.NET solution, I wanted, I looked up validators that I could say, "All right check this value against this field against this table in this database." And it will write out the code so to do that look, that's the thing with code validation, there's no validator that will check the value against the field in a table in a database so I created, that's the other case study, you drag the look up and a great example of integrating in Visual Studio, the lookup validator control appears in your tool box as with the other validators, if you wanted to, you drag it onto an ASP.NET page and boom, it generates all the code necessary to implement the look up for you. You just have to tell the name of the table, name of the field and the name of the connection string to use, oh and the data type. So those are the two case studies I used, an expansion pack again, I'll go through the tools then I'll also make sure I put in a couple of case studies and say, "Here's something you'd want to do." I think I'd probably use the rule generator for the expansion pack. So, yeah look, Practical Code Generation, you'll find the book.

Carl Franklin: Okay.

Richard Campbell: I've been looking at Framework 4.0 and Studio 2010, are we seeing some evolution in this area?

Peter Vogel: Not that I've seen, I think T4 was the real add on and that came out with the domain specific languages, I mean generally speaking in the, one of the things that you have to look at I think, when you're working with code

generation, if you wanted to have an automatically, want to happen on the background so you've got to build it into, build it as a Visual Studio add in and tie it to events in the Visual Studio model so that when you close the web.config file or switch away from it or when you build your application, your code is generated for you, you don't have to click any extra buttons or anything. The visual studio object model is constantly evolving with 2010 with them taking advantage of some of the foundation classes in Visual Studio 2010, we're seeing some changes there but in terms on the kinds of things that you used for code generation, I haven't seen anything yet that I went, "Huh!" like that. I mentioned the file code model earlier on and the file code model allows you to analyze existing classes like it turns your source code into a set of objects, so you can find all the classes in a file or find all the methods in the file at design time unlike reflection which works at runtime, it will also and this is what I use it for, is generating code but it doesn't always generate code. The file code model not all of it is implemented for every language, if you want to generate a method or a property in C# that will always work but if you're trying to generate for a Visual Basic file, a case where you'll get back the message that says, "That's not supported." So, I'm assuming in 2010, if I'm running tests, I'm assuming in 2010 we'll see more of the file code model implemented for visual basic. So I think we're seeing extensions but I think the really new thing is that T4 which came out with domain specific languages geared to....

Carl Franklin: Okay.

Peter Vogel: So that's the nice thing, it means everything in this book also applies in Visual Studio 2010 and I don't have to do a massive rewrite, I hate rewrites.

Richard Campbell: Right.

Carl Franklin: Well, we're coming down to the wire here, is there anything that we missed that you want to talk about?

Peter Vogel: No. The big message I want to get out is that everything that the typical developer, well first off, the big mess you want to get out of is if you ever find yourself writing the same code over and over again, stop doing it. You should be using code generation, take half an hour or three hours or a day, if you're building something really complicated out of your life and let Visual Studio and .NET generate your code for you. It's all there, you're using exactly the same stuff that the guys in Microsoft are using to generate the Entity Framework and all their good stuff, it's there waiting for you to use it. It's a big important, useful tool and people aren't using it enough. So, that's the big and the reason I wrote the

book was so if you didn't know where to find all this stuff, you can buy this, if you like buying books, you can buy this book and you might just get new shoes.

Richard Campbell: Nice.

Carl Franklin: Excellent Peter. Thanks very much.

Peter Vogel: My pleasure. Thank you guys, this is a real thrill for me to be in this show, thank you.

Carl Franklin: You're welcome and thank you. And we'll see you next time on .NET Rocks!

[Music]

Carl Franklin: .NET Rocks! is recorded and produced by PWOP Productions, providing professional audio, audio mastering, video, post production, and podcasting services, online at www.pwop.com. .NET Rocks! is a production of Franklins.NET, training developers to work smarter and offering custom onsite classes in Microsoft development technology with expert developers, online at www.franklins.net. For more .NET Rocks! episodes and to subscribe to the podcast feeds, go to our website at www.dotnetrocks.com.