



<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 #362
(Transcription services provided by [PWOP Productions](#))



James Kovacs Inverts our Control!
July 24, 2008
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! Put on your *I Heart PWOP Podcast* shirt 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 #362, with guest James Kovacs, recorded live, Wednesday, June 25, 2008. .NET Rocks! is brought to you by Franklins.NET - Training Developers to Work Smarter and now offering SharePoint 2007 video training with Sahil Malik 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 Data Dynamics, makers of ActiveReports.NET, simple, powerful and cost-effective reporting for Windows Forms and ASP.NET web applications, online at www.datadynamics.com. Support is also provided by CoDe Magazine, the leading independent magazine for .NET developers, online at <http://www.code-magazine.com>. And now, the man who wants to open up a can of whoop ass on cute little backwards clichés like that one, Carl Franklin.

Carl Franklin: Hi, this is Carl Franklin. Thanks a lot. You're listening to .NET Rocks! Richard will join us in a minute. Of course, he is flying around the world doing a whole bunch of stuff. I think he went to Bulgaria then he is in Germany and ending up in the Galapagos Islands for a little vacation. So, let's just get right into Better-Know-A-Framework.

[Music]

Carl Franklin: So, today, I want to talk about the System.ObsoleteAttribute and what it's used for is to decorate types and members of types that are obsolete and will cause compiler warnings to be generated if that type has the attribute. You see it all the time in deprecated code. Deprecated is one of those crazy words that means no longer used so when something is deprecated, you put the obsolete tag on it and it's just a nicer way of saying, "Hey, don't use this anymore," rather than just yanking out the method or the member or the property whatever and causing all sorts of problems. There it is, System.ObsoleteAttribute. Know it, love it, learn it. I don't have an email today. Instead I have a guest in

the studio who just happened to stop by, Nick Moldar. Did I pronounce your last name right?

Nick Molnar: Close, Molnar.

Carl Franklin: Molnar and I swear it sounds like a superhero. "I am Molnar." Anyway, Nick, you work for Infusion down at Greg Brill's place.

Nick Molnar: That's correct, down in New York City.

Carl Franklin: So, you were one of the guys at Sleepless in New York?

Nick Molnar: Yeah, you and I stayed up all night, pulled an all-nighter there and that was a great experience.

Carl Franklin: We had fun, didn't we?

Nick Molnar: Oh, yeah, it was a blast.

Carl Franklin: Yeah, the party on the roof in Times Square.

Nick Molnar: Now, that I live in the city, I need to get back up on that rooftop. I need to get Greg to hook me up with that spot.

Carl Franklin: So, they hired you.

Nick Molnar: They hired me, yeah. It took me about six months to pack up my life in Florida and get up to New York but I've been there for a few months now and it's been great. I love it.

Carl Franklin: And you brought your wife along with you?

Nick Molnar: Yeah.

Carl Franklin: Any kids?

Nick Molnar: No, my wife Katie and our dog Jacques, which is he's my son, so...

Carl Franklin: Cool, and you live like in the city?

Nick Molnar: Yeah, we live in the city. We're really new there so we just got a place and...

Carl Franklin: So, what's cool about this is that Nick, you just emailed me, you said, "I'm going to be doing some consulting at a company locally. Can I stop by and see the studio?" I was just about to take you home because it's like 2:00 in the morning here or 1:00 in the morning. I was about to take you back to your apartment and Brandon goes, "Hey man, why



don't you just have him do the intro with you?" So, we thought we'd give you your 15 seconds of fame.

Nick Molnar: Yeah, Brandon's awesome. This is the biggest I'm ever going to be. That's not true, that's not true. I'm coming after you, Carl.

Carl Franklin: All right, cool. So you'd recommend the New York tour to anybody who would -- you've really got a lot out of it is what you're saying.

Nick Molnar: Yeah, I really recommend it. I think Infusion has done a great job of really caring for their people, making sure that they're learning, that they're growing, that they're working on things that they're interested in.

Carl Franklin: What kind of stuff besides SharePoint have you really tackled? Or is that your real focus?

Nick Molnar: So far I've doing a lot of SharePoint in the first four months that I have been here. I've also dabbled with some rich client technologies like we have a lot of Silverlight stuff going on and some other magical stuff, some things I can't really talk about.

Carl Franklin: Okay, you'd have to kill me.

Nick Molnar: Yeah, probably.

Carl Franklin: How about Surface? That intrigues the hell out of me.

Nick Molnar: Yeah, so some things I can't talk about much, but yeah, exactly.

Carl Franklin: But we know you guys are doing some Surface stuff.

Nick Molnar: Oh, do we? Well, then perfect, yeah. So, we have some -- you just tricked me, didn't you?

Carl Franklin: Yeah, yeah.

Nick Molnar: We have some Surface stuff going on. It's quite interesting. We have all the work going on in Dubai so there's interesting travel opportunities.

Carl Franklin: Dude, I want to go to Dubai, man.

Nick Molnar: You should do it. You should do it.

Carl Franklin: Oh, no. There is no way.

Nick Molnar: It's Carl Franklin.

Carl Franklin: Yeah, I would have to bring my kids and everything. It just would be a mess. All right, well, let's roll the interview now which was an interview we did with James Kovacs and Richard and myself of course. Richard, our guest today is James Kovacs. James is an independent architect, developer, trainer, and jack-of-all-trades, specializing in agile development using the .NET Framework. He is passionate about helping developers create flexible software using test-driven development, unit testing, object-relational mapping, dependency injection, refactoring, continuous integration, and related techniques. He is a founding member of the Plumbers @ Work podcast, which is syndicated by MSDN Canada Community Radio. Wow, that's pretty cool. We should talk about that.

Richard Campbell: Yeah, it's John Bristow and co., a bunch of troublemakers up in Canada.

Carl Franklin: Bunch of troublemakers.

James Kovacs: That's right.

Carl Franklin: He has published articles in MSDN Magazine -- most recently "Loosen Up: Tame Your Software Dependencies for More Flexible Apps" in the March 2008 issue. James is a Microsoft Most Valuable Professional for C# Architecture and card-carrying member of ALT.NET, a group of software professionals continually looking for more effective ways to develop applications. He received his Masters degree from Harvard University.

Carl Franklin: Ah, brainiac.

Richard Campbell: Ah, yes. He's a smart guy and let us make no mistake, James Kovacs is one of the founding members of the NHibernate mafia.

Carl Franklin: That's true.

Richard Campbell: We called them out in 2006.

Carl Franklin: How are you, James? Welcome to the show.

James Kovacs: Thanks for having me guys. Good to be here.

Carl Franklin: Wow, you have the list as it were of technologies that are hip and cool and current that people are using today in software?

James Kovacs: The funny thing is I jumped into those technologies before they were hip and cool and did because I saw value in them and I'm glad to see that more people in the community are becoming



interested in the types of ideas and a lot of it is what is old is new again because a lot of these ideas are filtering over from the Java community and others.

Carl Franklin: Yeah, true. We're going to talk today about a little inversion of control and aspect-oriented programming, right?

James Kovacs: That's right.

Richard Campbell: And you know, part of this show was really -- we did an interview with Hamilton a while back on Castle Windsor and we had technical problems with the recording which was very frustrating because it was a cool show.

Carl Franklin: It was a good show.

Richard Campbell: I realized after we've done that show that maybe we really need to lay down more foundation of core concepts around this just to sort of set the stage for what Windsor is all about. I know you're a Windsor fan too, James, so I want to start with inversion of control but I hope we can get to talk about where Castle plays in the whole equation.

Carl Franklin: Yeah, let's talk about the fundamentals first.

James Kovacs: Really, the fundamentals is looking at your software architecture --traditionally we've had very tightly coupled software architectures where you have one concrete component talking to another so you've got your service component, your customer service that talks directly to some data access component and the problem with that is it's very difficult for those two components to vary independently so if you want to switch to a different database platform or you want to add in logging or auditing or anything else, it's really you have to tear everything apart and put it all back together.

Carl Franklin: Right.

James Kovacs: So, one of the core concepts that came about was the dependency inversion principle which was popularized by Robert C. Martin and he states that high level modules should not depend on the low level ones, but they should both depend on abstractions. So, in more concrete terms, rather than having your customer service depend on your data access component, both of them should depend on some interface or abstract-based class like an ICustomer repository where the repository is your data access component. So, what you've done there is you've essentially decoupled those two components and all the customer service knows about is that it's talking to some IRepository, some IComponent. So, you are actually free at runtime to switch that for

something else and this introduces a lot of flexibility into your software.

Carl Franklin: Yes, and that's really what it's all about. As changes come up, you just want to make sure that you're ultimately flexible. I think a big mistake that a lot of developers make is trying to predict the future in the architecture session, trying to come up with all the different scenarios that could happen and therefore try to architect for them right upfront whereas this approach is let's just architect for flexibility so that when something unknown and unseen comes along, we can adapt quickly.

James Kovacs: You're absolutely right, Carl. That's exactly what we were aiming for. It's all about focusing on highly cohesive architectures and loose coupling between components, just things that we've talked about for years and this is a way to get to it.

Carl Franklin: So, t h a t 's dependency inversion. Tell us what dependency injection is.

James Kovacs: What dependency injection is, now you have a true dependency inversion, you have a high level component depending on an interface. What you want to be able to do is substitute that at runtime. You can use dependency injection to supply the components dependencies through its constructor, typically through its constructor. You can also do center based. What you do is you'd say "new customer service" and one of the first dependencies would be new customer repository.

Carl Franklin: Right and the dependency is just being any component that is accessed during the lifespan of that object.

James Kovacs: That's right, so it's just some other component that is used by the piece of software that you're currently working with, in this case the service.

Carl Franklin: So, is it as simple as just supplying those dependencies on the constructor?

James Kovacs: That's exactly it. The other option that a lot of people use is service locator. What service locator is, in your customer service, in your constructor, you will walk up to a factory or something and say, "Hey, create me this dependency of mine."

Carl Franklin: Right.

James Kovacs: That's definitely another way to do it. The problem with that is you don't get very good testability because all of your dependencies are locked away inside of this service, which makes actually testing the service in isolation of the database

and everything else very difficult. It's possible but quite difficult. By having all of your dependencies supplied into your constructor in your test, you just supply fake diversions of them. You can use the mocking framework or you can write them yourself just have some class that implements that interface and supplies a default behavior but that doesn't actually touch databases. It doesn't do logging and auditing and things like that. What it allows you to do is unit test components in isolation.

Carl Franklin: Yes.

James Kovacs: Now, one of the nice things about this is if software is only used in one place, it typically is fairly inflexible because it's only suited for one purpose. As soon as you have to design it to be used in two different places, it becomes much more flexible. The two different places in this case are tests and production code. So, you all of a sudden are gaining a lot of flexibility in your design by thinking of it, "Okay, how am I going to test this?" So, by supplying all of your dependencies in your constructor, there you have a way of easily testing because you can supply mocks or stubs or your own handcrafted fake objects at test time, but then have real versions in production code.

Carl Franklin: And the other concept, of course, is inversion of control and inversion of control containers and we've talked about this a little bit before on the show but I have got the feeling that it was a little bit unclear. I really appreciate the clarity of your brief and very succinct definitions here. So, let's define inversion of control.

James Kovacs: So, the way you get to inversion of control containers is you look at -- okay, we saw that dependency injection is a good thing. Now, you can imagine you've got a chain of dependencies, you've got some UI component that needs to build a service component that needs to get access to a data access component and each of these things in turn have their own dependencies. So, as you go up the chain, all of a sudden you want to create the repository so you need to supply dependencies. You need to create the service so you need to supply the repository and the other dependencies. So, by the time you hit the UI, the UI has to know all about data access components, service components, logging components, auditing components, and that's the wrong place to be assembling your architecture...

Carl Franklin: That is the wrong place.

James Kovacs: In your UI. That's what you're trying to get away from...

Carl Franklin: I see.

James Kovacs: Is having all kinds of logic inside your UI controls.

Carl Franklin: Sure.

James Kovacs: So, the way that you get away from this is you use an inversion of control container and inversion of control container is nothing more than a factory for components.

Carl Franklin: Because you have to supply all of the dependencies, you want to start from another container that can create those dependencies and then pass them in so the UI is actually the last thing in the chain. Is that what you're saying?

James Kovacs: That's exactly what it is. It sounds really complicated but that is what it is. What you do is you walk up to the container and you say, "Hey, I need an ICustomer service," and it looks in its setup and says, "Oh, an ICustomer service is this customer service class. This customer service class needs ICustomer repository," which is the SQL customer repository. It also needs an auditing component and it figures out what all the dependencies are on the chain, constructs them in reverse order, and then supplies you with a fully constructed object at runtime. So, it's just a factory for components.

Carl Franklin: Yeah, that makes total sense to me now. If anybody has ever done a Windows Forms development where you start from a module, from a submain and then you create your form and show it, I mean that's essentially an inversion of control container even though you're really not doing a whole lot with objects. So, you might be, but that's essentially the same thing instead of just starting with a form.

James Kovacs: Yes. You're basically constructing component at runtime. Inversion of control containers provide you with a lot of services over and above object construction but at the fundamental level, that's it, and a lot of people say, "I can't bring in a third party component into my application. I can't bring something as big as Castle Windsor or StructureMap," or something like that. The reality is if you're in a situation where you can't take one of these dependencies, you can build your own inversion of control container in about two dozen lines of code and I actually walked through that in my articles, so if people are curious they can just go read up in the Loosen Up article.

Richard Campbell: That's a great idea, James, because most people push back on this as too complex.

Carl Franklin: Right.

James Kovacs: Honestly, Richard, it's two dozen lines of code. Fundamentally, what do you have? You have a hash table. The hash table has a bunch of interfaces versus concrete types. So, in a very simple container that you construct yourself, I call it the do-it-yourself container, all you do is when your application starts up in your submain or your public static board main method if you're doing C#, you just construct all the types, construct all the dependencies, build everything, throw them into the hash table and then at runtime, you ask for, "Hey, I need an ICustomer service," and it looks it up in the hash table and passes it back to you. Full-pledged containers can do much more than that and will walk dependency chains for you but 90% of what they supply, you can write yourself and it's really quite straightforward.

Richard Campbell: And very quickly.

James Kovacs: And if you understand that, you then can buy into and understand what these other containers are and actually be in a position to evaluate, "Okay, should I go with Spring.Net? Should I go with StructureMap? Should I go with Windsor? Should I go with Unity? Which inversion of control container is right for me?"

Carl Franklin: I just heard some heads spinning out there because you just threw out a whole bunch of products that nobody really knows what they do. Let's start with, I don't know, let's start with whichever one you want to start with but what are we talking about when we're looking at these frameworks?

James Kovacs: They are all inversion of control containers. Spring.Net is a port of the Spring Framework from Java, a very standard Java architecture is Spring for inversion of control NHibernate for object relational mapping. You can do the same thing on the .NET side of things.

Richard Campbell: Right and we had Mark Pollack on last year talking about Spring.Net.

James Kovacs: Spring is one container, Castle Windsor is another one, part of the Castle project and it's my personal favorite. The main advantage of Windsor from what I've seen over Spring is Windsor has a lot more flexibility in configuration. Spring tends to be a lot more verbose, a lot of XML setup. That might have changed in the last year or so since last I looked at it, but it historically has been rife with XML. You just have to dig XML files and configure all your types and how they all hook together. With Windsor, you can do that but you've got a lot more options in terms of configuration such as configuring it through a

scripting language, configuring it in code. Craig Neuwirt created a fluent interface which is now part of the latest build of castle so you can actually string things together. The fluent interface idea actually came from Jeremy Miller and StructureMap so with StructureMap, Jeremy's whole point was rather than going into XML and configuring it in XML, I can actually go ahead and build it up saying "Object factory, for this interface, this is the concrete implementation," and so you end up using generics to define what your component structures are. So, each of them has their own inversion of control containers. They are often part of a larger piece. Windsor is part of Castle project, so Windsor has got very good hooks into Castle Mono Rail and Castle Active Record and the other projects that are part of the Castle umbrella. Spring has its own pieces for building out applications. So, it depends on the style of development you want to do. If you want a really lightweight framework, StructureMap is a good one. Jeremy has got some really good ideas in that and then Unity is Microsoft's play into this space. They put -- people have probably heard of Object Builder. Object Builder is an object construction framework but it doesn't supply inversion control. Unity builds on top of object builder and is a full pledged inversion of control container.

Richard Campbell: Interesting. This is a CodePlex project from the Pattern and Practices team, right?

James Kovacs: Yes. Chris Tavares of the Patterns and Practices group was the lead on it, a very great guy. It's got a bit of a different flavor to it than Windsor does but they all fundamentally do the same thing is figure out your dependency structure at runtime which allows you to be very flexible. The nice thing about it is, let's say I've got some sort of data access component and I want to find out every time the database is touched. Well, I could go in there and start hardwiring in some logging components or the other thing I can do is I can create a brand new component called a logging data access component. It internally has the real data access component, just another IData access. With your inversion of control container, you just wrap it. You use the decorator pattern to when you call into the IData access component, you actually get an auditing data or a logging data access component, which does the logging and then that delegates through to the actual data access component so you can actually implement logging without actually modifying the original data access component which dramatically reduces the chances of introducing a bug into your system because you're not editing the original code. It's the whole idea that the open-closed principle that classes should be open for extension, but closed for modification. You don't want to open up a code file but you want to be able to extend it either by deriving a class or using composition to introduce new

functionality, new behavior into the system while not modifying existing working behavior.

Carl Franklin: This is what AOP, aspect-oriented programming, is really all about, isn't it, is that you want to keep sort of the goo where you can hook these things that happen in a separate place in your code staying out of your business logic code.

James Kovacs: That's right. Aspect-oriented programming is a little bit of a different flavor. It's a bit of orthogonality, you have orthogonal concerns. Logging is different from data access.

Carl Franklin: Let's define that big word, shall we?

James Kovacs: Orthogonality or something being orthogonal just means at right angle. So, they're completely separate concerns...

Carl Franklin: At right angles?

James Kovacs: Right-angled, so doing something on one access doesn't affect the other access. If you got an XY coordinates, you can move up and down your Y axis without affecting what your X coordinate is.

Carl Franklin: So you're just talking totally decoupled. That's what that means?

James Kovacs: You're just decoupling your code. So, the idea of aspect-oriented programming is to be able to implement an auditing component and rather than having to implement a customer auditing component and an orders auditing component and a foo auditing and a bar auditing component, all these different components, you implement one auditing component that receives all calls and it can make decisions. It's a very different style of programming and they are aspect-oriented programming frameworks. Inversion of control containers are often an ingredient in getting there, you don't need to use AOP when you're doing inversion of control though. They are two separate but related concepts.

Richard Campbell: One of the frustrations I have with this technology, James, is that every example I get cited is about logging. Can we come up with a story that's more complicated like how I could fix something really challenging inside of an app using this technique?

James Kovacs: Actually, I used this in one of my customer's applications. They needed to have validations rules. So, my service component needed a bunch of validation rules for what a valid purchase order was.

Richard Campbell: Okay.

James Kovacs: Traditionally, you have this big, ugly if block, if-then-else switch block, and every time you need to add in a new rule, you add something else into this big gory validate method.

Richard Campbell: Which is also a very test resistant chunk of code like it takes a lot of tests to deal with that huge switch block.

James Kovacs: Exactly, and the interactions between the different branches of the ifs and switches, it just gets unwieldy very quickly.

Richard Campbell: Right.

James Kovacs: What I did instead is my service component depends on an array of validation components. The validation components get injected at runtime. So, in my configuration, I just have a list of all my validation components and they are very simple. They are does the purchase order have a date? Is the purchase order over a certain amount? Is the purchase order to a particular customer? It has very simple rules about what a valid purchase order looks like. Those can be tested independently and very easily. The service then just iterates through whatever collection is given.

Richard Campbell: And it's probably using some kind of data structure to figure out what rules I need to apply to what data structures.

James Kovacs: That's right. So, if I need to add in a new rule, I just create a new IValidation rule, configure it in my inversion of control container, and next time the application comes up, it's gone. I don't need to go through a lot of contortions to test it and to make sure all the various branches of logic are working. It's a very simple matter of just adding this new validation rule. I'm not going to break the existing ones. If for some reason I do discover a problem within an existing validation rule or I could use it for promotions, so I've got a promotion that's only good for a certain amount of time, I need to disable it, turn it off. It's a matter of just commenting out a line of code. You can do this in an XML configuration file. You can do it in a separate assembly that can figure the application; there's a variety of ways of doing it, but it's very easy to dynamically add and remove pieces to your application without affecting the overall structure.

Richard Campbell: I was about to come at you and say, "Should I just be using a rules engine for this?" and I've answered my own question just thinking about what you're saying here which is that the challenge I have with the rules engine is while I avoid writing code, I'm limited to the creativity of the rules

engine author and you're giving me a model here where I can still get to write code but without the punishment of the big switch where it's a bear to test and it's likely I can break things. Here, I still get to write the code the way I want to write it, but I'm injecting it into the system in a way that it doesn't endanger the existing system.

James Kovacs: That's exactly right, Richard, and sometimes a rules engine is the right answer but sometimes it is not. It's a matter of looking at the application you're trying to create, the intended end-users. Oftentimes, it's programmers who are the ones writing the rules and testing it. There, you're probably going to be much more comfortable working in C#, writing unit tests around it. If you do need to hand it off to an end business user to configure and write the rules, in that case, then commercial rules engines do have front-end editors and all the other pieces that go along with it. You have to look at the application you're writing and the end-user base and figure out what is going to be best for your client, but the point is that you don't need to whip out a big rules engine just because you've got a few validation rules or you need to make your system configurable.

Richard Campbell: Yeah, I like the fact that I don't have to take things apart in this respect. One of the other concerns I have about this is this must make code reading more challenging like bringing somebody new into a project. They've got to really -- like understand the flow of your program. It's not easy now.

James Kovacs: Yes and no. It's not as easy because it's not as linear and people might not have seen it before. As soon as you have worked with this out of system though, it becomes second nature. You see a block of code like in your service component, it will say, "For each validation rule in the validation rules collection that was injected, rule out validate." So, that's easy enough to follow along. Your next step is just to find out everybody who implements the IValidation rule interface and if you're working with a tool like most people who are doing test-driven development, use ReSharper. It's a single keystroke to say find me all implementers of this interface and there are other tools that will do this. That's one of the reasons that people don't kind of understand the value of a tool like ReSharper until they start working in these inflexible systems and the ability to navigate through your code base easily and find all implementing types, go to derived types, navigate, inherent hierarchies very quickly, you don't kind of see the value in it.

Richard Campbell: You know what? You've switched tools up. When we first started this conversation, you said, "Okay, I want to insert this capability in all these classes." My first thought was,

"Okay, I'm going to make friends with the Find and I'm searching code." Now, you're talking more about doing that same task but doing it from an object browsing point of view.

James Kovacs: Yes. So, there is the on the one hand, you need to have things happen at runtime, you need to have actual code constructed and object supplied. On the other hand, as a programmer, you need to be able to efficiently navigate your code base and figure out what the heck is going on. What are all the possible rules that could be injected here? What are the implementers of the interfaces? Who overrides these methods? In that case, a tool like ReSharper is invaluable because it can answer those questions very quickly and very efficiently with just a few keystrokes.

Carl Franklin: You're listening to .NET Rocks! from dotnetrocks.com. This is Carl. I have a message from our sponsor Telerik who wants you to know about the best way to learn using new dev tools and technologies. Well, is it reading manuals? Watching videos? Playing with sample code? How about all the above? So, Telerik recently launched their new interactive trainer tool to help you effectively learn all the Telerik products in your own pace. The Telerik trainer is a slick WPF app that combines a video player with synchronized highlights, a table of contents for topical navigation, and a context sensitive code launcher. While playing the narrative videos, you'll see a code button light up at a relevant section, click the button and you'll open the respective file from the provided project directly into Visual Studio. No more searching for code while watching a training video. This is indeed innovation in training. They are always releasing new tutorials for all the Telerik products, so don't waste any more time and download this amazing new training tool now at telerik.com, T-E-L-E-R-I-K, and as you know, when it comes to developer tools, it's not just about great products, but also about reliable support and effective training materials and that's exactly what our friends at Telerik have done. Check it out. Now, let's get back to the show.

Can we talk a little bit about some of the particulars of Windsor and particularly the Windsor and the microkernel or maybe we could start with Windsor? What is the experience of using this tool? Is this where you start your development at Windsor? Walk us through a little bit of the process of using it.

James Kovacs: Okay, starting right off the bat with the full-pledged inversion of control container can be very daunting and all the services that it offers can be quite confusing. I would recommend and the way I started was to just create a very simple inversion of control container, the sort of do-it-yourself use hash table. You get the core concepts and it's much easier

to follow the flow of what's going on. You can then later on -- the nice thing about it is later on you can just swap it out for a full-pledged inversion of control container like Windsor after the fact because everything is dependent on interfaces.

Carl Franklin: So, you would start by writing your own IoC container?

James Kovacs: Well, I would take one like edit my article or off of Orin Eini's blog. There are a few people who have published very simple like two dozen lines inversion of control containers and just understand what they are doing before trying to tackle everything that Castle or Spring or some other framework does offer.

Carl Franklin: Is it because you can see the code and you can step through it?

James Kovacs: That's right and there's not that much code like honestly, Carl, it's two dozen lines. It's not hard to follow. You've got hash table of interfaces versus concrete types or versus object.

Carl Franklin: But it's really the flow that is new.

James Kovacs: It's the flow; it's the decoupling that is the biggest bang for the buck. The other features of inversion of control container are nice bells and whistles but you're biggest bang for the buck is the decoupling that you get the flexibility in your architecture which you can get out of a simple inversion of control container. The other confusing part of an inversion of control container, they are just big. They offer a lot of different services and so I just found it very difficult to kind of wrap my head around everything that it was doing at first. If I was going to start off a project now, I've used Windsor enough that I just drop it in right off the bat.

Carl Franklin: Now, after you've written your own IoC or used an existing one, would you recommend starting with the micro-kernel which is, by the way, a smaller version of the full blown container in Windsor.

James Kovacs: Well, the idea of Windsor in micro-kernel is they go hand in hand. Micro-kernel is your core inversion of control piece. It's the engine. It's the gut of the system. What Windsor provides on top of that is all of the configuration.

Carl Franklin: Ah, I see. So, it isn't like a stripped down version. It's really the core of it.

James Kovacs: No, it's not a stripped down version. It's really -- you actually would want to start with Windsor and then if you really need to dive down

into micro-kernel. Micro-kernel provides a lot more services than Windsor does. Windsor provides the facade of it which worries about reading XML config files. That's also where you get Oren implemented, something called Binsor, which uses a language called Boo to configure Windsor. There's a variety of different ways. Another fellow I know, Donald Belcham, implemented Cindsor, which is a C# way to do it. Craig Neuwirt implemented fluent interface which is part of the Windsor. When you're dealing with micro-kernel, you're dealing with the really low level gut of an inversion of control container and configuring dependencies that are much lower than most developers ever need to go.

Richard Campbell: Right.

James Kovacs: Windsor provides that higher level abstraction to it where you're most productive. So, you'd start off with Windsor. A good example was where when I was doing the validation. So, what I wanted to do is I wanted to say, "Hey, get me all components implementing the IValidation rule interface." Windsor at the time didn't have that but it was in micro-kernel so what I did is I drilled down into micro-kernel, called the appropriate method and then jumped back up into Windsor. Since that time, Windsor has actually added the ability to, "Hey, get me all components," rather than just "Get me the one that implements the interface." So, really, going into micro-kernel is much more of a specialized...

Richard Campbell: It's an exception.

James Kovacs: Got to be something really low level.

Richard Campbell: Right. It sounds like an exception thing I would do.

James Kovacs: That's right.

James Kovacs: Traditionally, the way you've gotten into Windsor is you would hammer at this big, long XML file which specifies "for this interface, create this type. This type has these dependencies. Here are the default values." You'd supply properties and values.

Richard Campbell: This sounds very Spring-like now.

James Kovacs: It's very Spring-like. They use a similar mechanism. Windsor's XML configuration tended to be a bit more terse than Spring from what I saw.

Richard Campbell: Okay.

James Kovacs: But, still, it's fundamentally XML and it tends to be very verbose.

Richard Campbell: Right, which in theory is a feature. This is about legibility, right?

James Kovacs: That's right. You tend to very quickly get lost in mounds and mounds of XML. It is one of the challenges with it. So, that's traditionally the way you configure Windsor. The recommended way to do it now is much more through Craig Neuwirt's fluent interface approach which he borrowed from Jeremy Miller's StructureMap where you basically in your C# code say, "For this interface, here is the implementing component and here are any of its default properties." For instance, if you have a GST calculator, one of its constructor properties might be a decimal amount indicating current percentage of GST or you might have data access component that would look it up inside of a table inside of your database, something like that. So, there's the XML route. Now, you much more do it through the fluent interface.

Richard Campbell: I think it's got to be really useful to have a good visualization component around seeing those relationships.

James Kovacs: Well, the way things are going, you actually don't need that visualization component. One of the things that's been happening is from the Ruby on Rails side of the world, Rails is very big on conventional reconfiguration so unless you're doing something really exceptional, do the thing that most people do. Don't make me have to specify my defaults. This is very much baked both into StructureMap and into Windsor's fluent interface where you can say, "Just go ahead and register all the controllers." So, you're working with ASP.NET MVC, so find all things that implement IController and register them. Often, when you're dealing with inversion of controller, you're just trying to decouple things so you will have an administrator controller, an IAdministration controller interface. You will have a home controller and an IHome controller. You will have a product controller and an IProduct controller. So, whenever you have those pairs, you can just use convention over configuration, use convention that your foo controller implements an IFoo interface.

Carl Franklin: Right, that's a given.

James Kovacs: You just tell Windsor to go ahead and configure all these things. It walks through all your types, figure the mappings, you don't have to manually specify them.

Carl Franklin: That's cool.

Richard Campbell: I totally agree with you. I still like the visualization angle on this in finding out where it's wrong. I'm just getting a sense that when something goes wrong with this model, because it is so abstract and it's relatively tough to read through, it's got to be really challenging to debug.

Carl Franklin: Getting back to the AOP, the AOP part of it, I think that not only is it better for testing, but also for debugging I would think. I mean the whole idea of an interceptor is to get a pre and post process call on any method and this is a new concept, is it? This is something we were doing in remoting, isn't it?

James Kovacs: In terms of debug ability, I've got an amusing story for you. So, one of the things that StructureMap traditionally has done very well is what Jeremy called environmental testing. What you could do is you can throw some attributes on certain methods and when your inversion of control container started up, you can give it a command that would run through any of these tests method and so what it would do is rather than when you try to call a component, when you try to create a component, you realize that the connection Spring, database connection Spring was wrong. It would go ahead and test, can I go, get into the database, can I load up these customer records or these name value tables, that sort of environmental testing. It's like can I get into the SMTP server and run through a bunch of code to do this. So, Jeremy was very proud of this because that makes IoC a lot easier because you can basically say, "Hey, IoC start up. First thing you do is run through these smoke tests and make sure I got a good connection spring." I got to contact the SMTP server, exchange this up. I can get to my message cues, anything, any infrastructure pieces. Jeremy was very proud of this and he presented it at DevTeach Vancouver just this past November and Oren was in the audience who is a big Windsor fan and Jeremy says pointing to Oren, "Here's something environmental testing that StructureMap can do that Windsor can't." Oren whips out his laptop and madly started hammering away. In the half-hour remaining of Jeremy's presentation, Oren taps me on the shoulder and he says, "It's implemented. I'm just going to commit it now."

Carl Franklin: That's so crazy. Those guys are out of control.

Richard Campbell: We are talking about Oren Eini though. He is crazy.

James Kovacs: He's a great guy, but oh my gosh, he can code like there is no tomorrow.

Richard Campbell: Yeah, the intensity level is unbelievable.



James Kovacs: Yes, in half an hour, he managed to implement one of Jeremy's favorite features.

Carl Franklin: Do not challenge me. I will kill you.

Richard Campbell: I will bury you in code.

Carl Franklin: Will bury you in code.

James Kovacs: That's something that I've seen him ask on Unity is some sort of visualization configuration component. In my own experience of using Windsor, I haven't really felt a great need for it. When you do get a configuration wrong, it will just say, "Can't find implementation for IFoo." You kind of look and say, "Oh, I didn't supply anything there." You can also get around it with the conventional configuration because most of the time, you just follow your naming conventions and everything works. Oren was telling me about a time on a project that he implemented IoC and Windsor where he just had everything set up. He had naming conventions. Of the developers, of the kind of dozen developers, he is the one that worried about all those infrastructure pieces and he left the project. About a year later, he came back to it. He got a call from the project lead who had taken over and they asked, "Oh, what's this configuration file that was doing the convention over configuration?" They hadn't had to. They've added new classes, they've added new interfaces, and they hadn't had to really worry and understand what was going on because it was just doing the right thing.

Richard Campbell: I just worry about the just doing the right thing like that's all well and fine until it isn't and then how tough it is to diagnose.

James Kovacs: Absolutely, Richard. I fully agree. I think that you really need to understand why it's doing the right thing and how it's doing the right thing, but the fact that the development team was able to productive for a long time and yes, that knowledge should have been transferred but it's a testament to the power of convention over configuration, the things that just worked it to the box. Honestly, as I said, when something does goes wrong, the container is out pretty good about saying, "Hey, I'm trying to create component X" It has these dependencies which have these other dependencies and I don't have an implementation for this logging component and to implicitly add to what it is trying to construct and where the things are going wrong and I have found it a huge concern in terms of trying to needing some visualization component as well. The other side of the coin is if your interaction of component is still getting so complex that you need to have a visualization component, you probably want to

rethink your interaction of components. Overall, it is about keeping things simple, not having really tightly coupled measures of objects and if you do start having to think about how you can slice and dice your architecture to reduce that coupling. The other thing is because you're dependent on interfaces or abstract based classes now, it's fairly straightforward to move the code around and to break dependencies. You're not tightly coupled any one component anymore. You can refactor your overall architecture much more easily and hopefully you got a good screen of tests around it so you can ensure that the changes that you are making are not impacting the actual functioning of the application.

Richard Campbell: James, how easy is it to retrofit this sort of technology into an existing application?

Carl Franklin: Ooh. Ouch.

James Kovacs: That is more challenging.

Carl Franklin: I was going to say that's fundamental, right?

Richard Campbell: Because on one hand I keep thinking this should be easy but on the other hand I'm thinking, "I can see where there could be some really tough places to loosely decouple."

Carl Franklin: That's a refactor hell right there, I would think.

James Kovacs: The very first thing you want to do before you attempt this is go read Michael Feathers book, Working Effectively with Legacy Code.

Richard Campbell: Michael Feather?

James Kovacs: Michael Feathers, Working Effectively with Legacy Code. If you can get him on the show, he's really a great guy. He has some fantastic ideas. I haven't actually the pleasure of meeting him myself, but that book is phenomenal because it gets you to rethink your architectures and how you assemble applications. Michael's entire point is that legacy code is any code that doesn't have tests around it.

Richard Campbell: Interesting.

James Kovacs: It doesn't matter whether it was written 20 years ago or a week ago. If you don't have tests, you don't have the flexibility to change it. You don't have the confidence to change because you don't know your test cycle so long so the first thing that you want to do if you have a tightly coupled application, the first thing that you want to do is get some tests around it. Michael's book helps you understand how to get tasks into essentially an

untestable code base. Once you've got test in there, then you can start breaking your code dependencies, you can start introducing inversion of control and dependency injection and definitely separating out your architecture loosening the coupling to make it more flexible in the long run, but it is a multi-step process. I talked to a fellow who had attempted this once before and it didn't succeed for a variety of reasons. We talked about some of the issues. He read Michael's book, read some other books that I recommended. We hashed through some of the how you would approach this and with about two months of hard work, I think he said it was about 150,000 line code base and a about a dozen developers. With about two months of work, they dramatically reduced coupling. One of the biggest challenges is they had singletons throughout their application which makes it very difficult to test. Anytime you have a singleton, it's a point of high coupling because typically singletons talk to other singletons and if you want to test one component, you ended up bringing along the entire application along with it. So, we talked about how to remove this coupling and it was about two months of hard work to decouple it but he said they went from an application that they dreaded to work on to one that was a delight.

Richard Campbell: Well, there is an interesting point in this which is every code gets tested when you put it out on the field, it is just a question whether you can control the testing and make sure the testing is comprehensive upfront. The interesting reality here is once I have these suite of tests, I immediately know what the consequences of my change were, rather the lack of consequence of my change.

James Kovacs: That's exactly it, Richard. You need to -- in order to refactor your code. Refactoring is about improving the quality of your code, improving the code's design without changing its functionality. In order to do that effectively, you need tests around it, you need unit test, you need integration test, to ensure that when you do make a modification, when you do say, "Oh, this switch block is really ugly, can I put in the command pattern?" Or, "This command pattern is really not doing much. A switch block would be much simpler." Doing that back and forth, how am I going to improve this code? You need those tests to ensure that you're not making any mistakes and it's the reality that if you got a good suite of tests, you basically make a change, you run your unit and integration tests, two minutes later, you've got an A whether the change has broke anything. That is a lot better than traditionally of "Okay, now I have to do a day with the changes and tomorrow can be devoted to testing." Everybody dreads it. You walk through all the test cases and realize you broke something. It's a much longer feedback cycle. If you want nice type feedback cycles, which is what the techniques do and they all kind of dovetail into each other.

Carl Franklin: To those people who say this is too much code to write, I just need to get my business applications done and out the door and I can do that and it has been working for me, why should I totally invert my brain in the way that I write code, why should I do this?

James Kovacs: Don't. Honestly. If what you're doing is working, if you're not having problems with coupling, if you're writing a lot of kind of fire and forget type applications with one-offs, they're never really touched again, all these techniques are going to buy you a whole lot. I still do it this way but I had a lot of success in it. If you're having problems where as code accretes in your code base, things are more and more difficult to change, it's worthwhile to start looking at these techniques. The ramp up time is a bit slower but one of the nice things is you maintain steady velocity because your code is very flexible and open to change, it's pretty much as easy to add a feature six months into a project as on day one.

Carl Franklin: Did you ever find other products that people use sort of get in the way and interfere with, let's say, any inversion of control container? I'm thinking of like CSLA.NET for example. I mean it's just an application framework. That shouldn't really get in the way of anything that you're doing with that, should it?

James Kovacs: It depends on the nature of the framework. When you're dealing with inversion of control, you're hiding away your dependencies so if you've got the type of framework that kind of enter its tentacles throughout your application, that can make things a lot more challenging.

Carl Franklin: Lots of factories and things like that.

James Kovacs: That's right. Another classic example that makes endorsement of control a bit more difficult and testing more difficult is ASP.NET, the classic ASP.NET, because as part of your page, you've got request response and server variables hanging off of it. You need a full ASP.NET pipeline spun up and running in order to exercise these things. Often, you have to do different things. JP Boodhoo has talked a lot about the model-view presenter pattern in ASP.NET and that's one way to use inversion of control inside of ASP.NET applications.

Carl Franklin: The MVC Framework obviously is going to help too, right?

James Kovacs: That's right. Yeah, the MVC framework from Microsoft bakes in testability, bakes in inversion of control. One of the watershed moments was when Scott Guthrie presented it at the



ALT.NET Austin conference last October and he said, "Insert whatever inversion of control container you want. We're open for business."

Carl Franklin: Cool.

James Kovacs: He demonstrated -- I've seen examples with Windsor, with Spring.Net, I've seen it with Unity, bring whatever you want and come play. The ASP.NET MVC team is really getting it in terms of what developers need. The ability to select the framework that can be most effective for them and also work with the frameworks that are already existing in their organizations.

Carl Franklin: Well, there you go. If you are looking for a reason to get into the MVC Framework and thought it was just a kind of a waste of time, now you know.

James Kovacs: Yup, if you're going to be doing ASP.NET MVC, you should be thinking about inversion of control because these are...

Carl Franklin: They go hand in hand.

James Kovacs: Good software techniques that help you build better software that is highly cohesive, that's loosely coupled and helps you ensure a single responsibility and separation concerns. So, lots of good solid software engineering ideas in there, inversion of control is one way to get there.

Carl Franklin: We're just about out of time. Is there anything left that you want to call out or shout out? We've got a bunch of links we'll put up on the show, links to your blog, to Psake. You want to talk about that for a minute?

James Kovacs: Oh sure. Yeah, pronounced sake, P-S-A-K-E.

Carl Franklin: Silent P?

James Kovacs: It's a make tool in the flavor of Rake Ruby. So Rake is make using Ruby, Bake is make using Boo. The whole idea is creating build scripts and rather than using NET or MSBuild which are heavily XML based, you can use PowerShell it creates this little framework called Psake that I just released last week and it orchestrates your entire build for you.

Carl Franklin: Nice.

James Kovacs: You write PowerShell script, you divide things into tasks, so you say, "Okay, this represents my build. This represents my deployment steps. This represents where I zip up everything into an archive package," and then you specify

dependencies just as you would in NET or MSBuild, but it's all PowerShell, so you don't have any of the XML angle bracket tasks, so it's very concise. One of the things I like about it over something like Rake or Bake is because Rake and Bake are higher level languages, you always have to implement this adapters or bridges. So, you want to call them as build, you have to write a little function that knows how to spin up a command line and call MSBuild with the appropriate command line parameters. Same thing with .NET, you have to write the .NET object that will interpret the XML angle brackets and create the appropriate command line and run it and receive the response back and do all the error handling for you. Because Psake is written in PowerShell, it's a scripting language. It all comes along for free. It is an idea that's been floating in my head for a few months. I decided, "Okay, I'm going to just try to write this," and it took me about half a day to throw it all together. Because PowerShell gives you so much out of the box, it ended up being 182 lines of code including comments.

Richard Campbell: That's all?

James Kovacs: That's all.

Carl Franklin: Wow.

James Kovacs: If you strip out all the comments, it's a hundred lines of code to write a build in PowerShell, which I thought were pretty neat. There are some interesting hacks in there in terms of making things, maybe what I was doing is I was using PowerShell to write a domain specific language around building. I did some interesting things in terms of making the syntax nice and clean so I encourage people. We'll put the link in the show notes. Have a look. See if it tickles your fancy, if it's going to solve some of your problems. One of the nice things about it is you don't have to drop .NET or MSBuild if that's what you're using, if you just want to use Psake to orchestrate, "Okay, run the .NET build, build through the solution. Now, here's my compiler stuff. Here's is my zip stuff. Here's how I test things. Here's how I deploy to a server." The other great thing about PowerShell is it's got this whole PowerShell provider model. There are providers for Active Directory, for Exchange. There are scripts being written around Exchange Administration. IIS 7 has PowerShell scripts and PowerShell commandlets for it. Microsoft is investing heavily in PowerShell so my thought is that you can not only build your application, test your application, but you'll also be able to deploy your application, punch keys into the registry, manipulate IIS also within a common scripting environment using scripting commands that you would use on the command line anyways.

Carl Franklin: Just to call out those URLs, they are shrinksterized, www.shrinkster.com/zur is your announcement for Psake and the project is at shrinkster.com/zuq. I also noticed this ultimate developer rig Kovacs edition at shrinkster.com/zv4. Is that just like the biggest, baddest module machine you can possibly think of? What is that all about?

James Kovacs: Well, what I did is I unfortunately had my main rig die. The motherboard just gave up the ghost thing and stopped working.

Richard Campbell: I hate it when that happens.

Carl Franklin: I hate that.

James Kovacs: I hate that and the problem with it was it was an AMD 939 platform. I don't know if you guys are familiar with it, but...

Richard Campbell: I have one here.

Carl Franklin: I have that.

James Kovacs: AMD has moved over completely to AM2 where you cannot get 939 motherboards anymore.

Richard Campbell: Yeah, they're essentially dead.

Carl Franklin: Great, that's good to know.

James Kovacs: Yup. So, the processor I had wouldn't work on AM2. The RAM I had wouldn't work on AM2. So, pretty much all the components I had, it was a new rig.

Carl Franklin: Pretty much done.

James Kovacs: Yeah, I was done. I took a look at Jeff Atwood's build for Scott Hanselman...

Carl Franklin: Oh right, yeah.

James Kovacs: The ultimate developer rig he built and looked at the components that were available at the time, did some research, did some investigation, updated some things and built out my own system and I've been very happy with it. It's running really well. I was just on stock components. It gets a Windows Experience Index of 5.9.

Richard Campbell: Nice.

James Kovacs: Which I'm very happy with. I'm doing a lot more podcasting and screen casting these days as well so the additional speed, it would typically -- Camtasia keeps all four processors humming along quite smoothly and I can produce a Camtasia video in about the same time as it takes to play it.

Carl Franklin: Nice.

James Kovacs: I can actually encode, say, at real-time speed which is pretty cool.

Carl Franklin: That is cool. All right, James, thanks. I can't tell you how much I appreciate your passion and more importantly you're a very good explainer. You're a very good teacher. You're very clear and concise and I know the listeners appreciate that as much as we do.

Richard Campbell: Sounds like we've got some dnrTVs in our future.

Carl Franklin: Absolutely.

James Kovacs: We can definitely talk about it. Well, thank you guys for having me on and happy Canada Day to you both.

Carl Franklin: Excellent. Watch out for moose.

Richard Campbell: Nice.

James Kovacs: Or out in Sweden which have the drunken moose. If you've never read about that, Google it.

Carl Franklin: Yeah, okay. 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.