



<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 #397*  
(Transcription services provided by [PWOP Productions](#))



## Michael Feathers talks Legacy Code November 25, 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! Stop converting that AS400 into a refrigerator and listen 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 #397, with guest Michael Feathers, recorded live, Monday, November 24, 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](http://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](http://www.telerik.com), and by CoDe Magazine, the leading independent magazine for .NET developers, online at <http://www.code-magazine.com>. And now, the man who likes a good Danish, and we'll just leave it at that. Carl Franklin.

**Carl Franklin:** Thank you, thank you very much. Welcome back to .NET Rocks! Carl Franklin in New London, Connecticut here and Richard Campbell in Vancouver again.

**Richard Campbell:** Hey sir.

**Carl Franklin:** One more time.

**Richard Campbell:** It's good to be home, huh.

**Carl Franklin:** One more time behind the mike. Yes it is. We had some fun on the road as you heard from some of those shows that we did and we have a lot more coming up especially Or-Dev, we got like five or six shows.

**Richard Campbell:** Yeah, I know, it was a monster week of just recording, recording, and recording.

**Carl Franklin:** Yeah and man, I love Scandinavia, my first time.

**Richard Campbell:** Yeah, it was just a ton of fun, some time, and Copenhagen is some kind of Malmo. Lots of thanks to the Or-Dev folks, they were awesome hosts.

**Carl Franklin:** Beautiful cities, Copenhagen and Malmo.

**Richard Campbell:** You're kind of fun to hang around with there, Mr. Franklin.

**Carl Franklin:** Yeah, sure and you know we also hang out with Glenn Block quite a bit and Tim Heuer.

**Richard Campbell:** Yeah.

**Carl Franklin:** We were like the Four Amigos over there at Or-Dev.

**Richard Campbell:** Off getting into trouble.

**Carl Franklin:** All right, let's get right into Better Know a Framework.

[Music]

**Richard Campbell:** All right, it's been awhile.

**Carl Franklin:** Yeah and Better Know a Framework, of course, is where I shine a little light on a far corner, just a little piece of the .NET framework and the hope is overtime all these things might come together to make you actually a smarter programmer. Today we're going to talk about the System.IConvertible interface. An IConvertible defines methods that convert the value of the implementing reference or value type to a common language runtime type that has an equivalent value. So let's say you have an object that explodes some data and you want to use the convert class on it to get to Boolean, to double, to byte, to char, to date, time, to int 16, all of those. It implements all of these things and then you can either give an equivalent value in that particular CLS type or you can throw an exception. You can throw any valid cast exception. So it's just a nice little interface that you can use in that situation.

**Richard Campbell:** It sounds like it's the right way to be dragging data into CLR types.

**Carl Franklin:** Exactly. So there you go. IConvertible, know it, love it, learn it. Richard, do you have an email for us? It's been a long time since we read an email.

**Richard Campbell:** Well, we've been on the road. We've been getting emails the whole time.

**Carl Franklin:** Right.

**Richard Campbell:** Actually I have a huge pile of email but I just wanted to grab one and it's actually in reference to the show that we did, number [388](#) with Uncle Bob.



**Carl Franklin:** Uncle Bob, Bob Martin. Hey, we did another one with him at Or-Dev.

**Richard Campbell:** Yeah, we did which was a pretty silly show actually.

**Carl Franklin:** It was kind of silly.

**Richard Campbell:** We were all a little punchy by the end of the day on Friday.

**Carl Franklin:** I thought it was a good show.

**Richard Campbell:** We had a lot of fun. Here we go. "Hello. I was listening to your show with Bob Martin and I have to say that I respectfully disagree with the idea that you should decouple only for the functionality you have. I was working on an application that does a lot of socket communications although it uses many different versions of an internal protocol, it only uses sockets for transmitting of this protocol. The socket logic was ingrained into the business logic at the application. One day we got a request from a new customer to communicate using a third party queuing application instead of sockets and the remark of refactoring was incredible. We wound up hacking our way through to get the project done but it was a cold reminder that anticipating potential new functionality in designing for it from the beginning is fundamental to decreasing development time for new features. When Bob talks about seeing and working on code that is hacked together, I think it's more an example of poor initial design and under-engineering rather than over-engineering. Had our communication logic been loosely coupled, it would have been very simple to add another type of communications to a project without affecting any other functionality. We have been finished and paid a lot faster."

**Carl Franklin:** Did he say that software shouldn't be loosely coupled?

**Richard Campbell:** No, he didn't say that but he said that -- well, what Bob was talking about was that we tend to anticipate *too many* needs...

**Carl Franklin:** Right.

**Richard Campbell:** And adds a lot of extra plumbing in that never get used and that sort of gets back to that Agile principle of AGNI, you Ain't Going to Need It.

**Carl Franklin:** Yeah.

**Richard Campbell:** And this is from Bogden Avalomof from North Carolina and obviously Bogden got burned the other way. They didn't do that stuff and then suddenly had a customer that wanted

something that didn't fit easily into that and they ended up doing a ton of work for this.

**Carl Franklin:** Yeah.

**Richard Campbell:** It's tough to bait, not easy to choose either way whether, you know, I think we over-engineer as well as under-engineer.

**Carl Franklin:** Interesting. So that goes into my question of is there such a thing as too much granularity which I believe we talked about on another show.

**Richard Campbell:** Absolutely.

**Carl Franklin:** Yeah, good start.

**Richard Campbell:** Hey Bogden, thanks for your email and for everyone out there, feel free to send us a message, [dotnetrocks@franklins.net](mailto:dotnetrocks@franklins.net).

**Call Franklin:** Hey Richard, did you see in Dubai they had an opening ceremony for some crazy hotel that was like 24 times more fireworks than the Chinese Olympics opening ceremony?

**Richard Campbell:** Yeah, in the middle of an economic downturn, somebody dropped 20 million dollars...

**Carl Franklin:** Million dollars.

**Richard Campbell:** Million dollars on a party...

**Carl Franklin:** Million. That's crazy.

**Richard Campbell:** That I was not invited to.

**Carl Franklin:** Well, let me just tell you. The money is in Dubai so if you want to chase it, send me an email at [carl@franklins.net](mailto:carl@franklins.net) and I will hook you up with the guys in Infusion development who were looking for hot .NET developers to go to Dubai and work on some really cool stuff and from what I hear, the money is pretty good and we will leave it at that.

Our guest today is Michael Feathers and he is a consultant at Object Mentor, and the author of the Prentice Hall book "Working Effectively with Legacy Code." Michael also wrote CppUnit, the initial C++ port of the JUnit testing framework. A programmer and consultant for 20 years, Michael has worked in biomedical, e-commerce, and a variety of other domains. He continues to speak at various conferences and visit teams around the world to help them get their code under test. When he isn't engaged with a team, he spends most of this time investigating ways of altering design over time in codebases. Welcome Michael.

**Michael Feathers:** Yeah, thank you. Nice to be here.

**Carl Franklin:** Yeah. Well, good to have you. We've been talking quite a bit on the show about all of these things that you're touching on. Testing of course is a big topic. Just legacy code, we did a show on, I guess you'd call them "brown field" applications.

**Michael Feathers:** Yeah.

**Carl Franklin:** Inheriting -- not the most fun position to be in as a developer, I guess. Do you agree?

**Michael Feathers:** Yeah, it's more and more common these days, it seems.

**Carl Franklin:** Sure is, yeah.

**Michael Feathers:** I remember back when I was in school, I was taking courses in working on neat things like compilers and all these other things. I get on the industry and I'm like ah, you know, it's like what do I do now and you discover you're trying to buy this big talking message you got to have to maintain for a while.

**Richard Campbell:** How legacy are we talking about here? Is this like old dBase app or mainframe app or AS/400?

**Michael Feathers:** I'm kind of lucky I never really worked on the mainframe. I never really did much with Cobol or Fortran. Not that there is anything really wrong with that, but my experience kind of starts with C and I wrote this book about the topic about four years ago and I kind of reached back into that stuff and just talked about the experiences that I had working with teams trying to go and help and get test around existing code just to make change a lot easier than it is currently so I haven't really gone way back to the mainframe.

**Richard Campbell:** But still, I mean you are talking in three object languages?

**Michael Feathers:** No, not necessarily. I have this definition of legacy that I use in the book which is like contentious.

**Richard Campbell:** Okay.

**Michael Feathers:** My point of view is that legacy code is basically code without test.

**Richard Campbell:** Ah, all right.

**Carl Franklin:** Okay, sure.

**Michael Feathers:** Yeah, and it's kind of an odd point to take but you know, this kind of like test driven development made a very big impact on me. I first heard about it in late 1990's and tried it out and I began to notice just how easy it is to change code that has a lot of tests surrounding it and you can definitely see the difference in teams. I mean, some teams that come in the morning and they just look like they want to be -- they place selves other than work because they just know that they got to have to go and deal with this chaos and just have to go and sort of try to find some thread through this large codebase and understand that long enough they're ready to go and make changes. On the other hand, we have teams that have basically done a lot with TDD and they walk in and they basically have a strong sense that they can understand their code and also if there is something that they don't understand, they can run additional test and if they make some mistakes, they've got a safety net...

**Carl Franklin:** Yeah.

**Michael Feathers:** So what my emphasis has been on trying to get that experience back and just in codebases.

**Richard Campbell:** It's almost as if wrapping everything in test means that you have that old Greenfield experience of every bit of new code is safe.

**Michael Feathers:** Yeah, yeah. The thing is it's never really complete though. That's the thing that's kind of odd about this. It's like you can spend years and years trying to get test and place around some large existing app. Codebase had been around for 10, 15 years. There's no way you're going to get tests around everything, but at least to be able to start to piece things together and get test around the areas that you're about to make changes to provides one heck of a strong basis for what you want to be going forward.

**Carl Franklin:** Before you can test, you really need to refactor, don't you? I mean, it's useless...

**Michael Feathers:** Yeah, that's a bit of an odd thing. I remember I started writing my book a bit after Martin Fowler wrote his refactoring back in 1999 and one of the things that he makes a very strong point is that you do need test in order to be able to refactor but once you start to try to get tests in place around an existing code, you discover very quickly that you need to go and do a little bit of refactoring to get tests in place.



**Carl Franklin:** Sure. I mean, especially if you're talking the kind of legacy code that I have to deal with which is great big sub-routines and functions, and the whole idea of granularity is in...

**Richard Campbell:** The do everything routine.

**Carl Franklin:** Yeah.

**Michael Feathers:** Exactly. it's very tough to deal with but the general idea is that there are certain things that you can do that are very conservative that allow you to go and have some confidence as you do those refactoring so you aren't breaking things and just do enough refactoring just to go and basically get testing in place to go and basically do more aggressive refactoring. You'll be able to help teams that way...

**Richard Campbell:** So what are these sorts of basic things? Is this is just like breaking down functions into smaller bites?

**Michael Feathers:** Yeah, there is that. The thing about it though is it really depends on which language you're working when you deal with the large function problem. We're kind of lucky now that in Java and C# we have some decent refactoring tools and quite often you can basically do things like extracting methods without having to do much in the way of testing. The tools are that good. In some older languages like C++ and C, it's harder to do that and have confidence on those distractions and what we tried to do is write test up above at large function in order to go and have enough confidence to go and see that you can refactor bit by bit and see the things that are still working.

**Richard Campbell:** Yeah. This really is a confidence game because often this legacy code is stuff you didn't write in the first place. You don't know how it works.

**Michael Feathers:** Yeah, exactly and for me, it's a consultant coming in. it's double anxiety in a sense because you're looking at stuff and you don't know anything about its history and some changes that people there would feel much more comfortable than you, you just want to consider yourself without having to somebody there to guide you.

**Richard Campbell:** Isn't there a whole business case game here with legacy apps like how much of the stuff is just not applicable anymore? Can't tell how many times when I've talked to people about, "oh, what about this function," "oh yeah, we don't use that feature."

**Michael Feathers:** Yeah, yeah, it's a very strange thing with that because I think people have discovered, you know, it's been kind of a long

realization coming that it's really hard to replace existing applications.

**Carl Franklin:** It really is.

**Michael Feathers:** I mean, when I first starting in the industry, it seems like there was this mania for going and rewriting things and now it's kind of like, wow, we have all these code and we have this considerable investment on it, then the question becomes can we actually replace some of the stuff and quite often teams run into this situation where it loved to replace things but they are not even certain that they understand it well enough it be able to do that with confidence and so then it becomes kind of like this game of incredible replacement and yeah, it's get kind of tough.

**Richard Campbell:** Well, you mention this in your bio too, this idea of gradually changing design which sounds almost impossible. I think most folks look at we have a new design and we jump to it.

**Michael Feathers:** Yeah and it's a real change of mindset also. I think it's very easy for us to try to imagine some pristine castle which is kind of nicer than where we are now. It's easy going through and cast our imagination and say, "you know, it's like I can imagine this much better thing than what I'm doing in now." In general, what is important is be able to try to imagine something which is a bit better than what you have currently and if you can target those things, then overtime you're really able to go and start to make a difference in the codebase.

**Carl Franklin:** So is your book and what you talk about sort of like a lifeline to somebody who is faced with a legacy application and doesn't know where to start? What I'm saying is do you have a framework like a number of steps to scientifically go through a process?

**Michael Feathers:** Not necessarily. It was really kind of an odd thing in writing the book because I really want to go make it -- I had a lot of theory I wanted to go and kind of get across, things I kind of abstracted away from the experience I had with teams but I ended up adapting kind of like an FAQ format. it's really structured in a way that you say look, you know I have this particular problem, and you basically find a chapter that goes and describes that problem until you go and get past it and for the most part it's about doing the dependency, breaking what's necessary to go and get test in place.

**Carl Franklin:** Yeah.

**Michael Feathers:** Not quiet so much about the bigger design issues.

**Carl Franklin:** Yeah.

**Michael Feathers:** But yeah, it's just really I guess the way to give people confidence to get start on some of these.

**Carl Franklin:** You know, if you can get past day one, you're making progress.

**Michael Feathers:** Yeah, definitely.

**Carl Franklin:** And a lot of the apps that I have to -- well, I'm thinking of one in particular that I was faced with which was an app that was written in access and was all flat -- used flat data model and they wanted to put on the web and I was like, yeah, good luck with that.

**Michael Feathers:** Right.

**Carl Franklin:** You know; the whole application is...

**Michael Feathers:** Mission impossible, right.

**Carl Franklin:** Yeah, the whole application is predicated on the fact that this data is flat and he extended the whole SQL Query model to the end user which is a bad idea in general.

**Michael Feathers:** Yeah.

**Carl Franklin:** And then of course it was all about having direct access to the table so if you just did a port for that to the web, just a port you know, you're asking for trouble.

**Michael Feathers:** Yeah and I guess a lot of it really comes down to basically making decision on when do you actually have to rewrite pieces of code.

**Carl Franklin:** Right.

**Michael Feathers:** You know, actually testing can help you quite a bit and just really hoping to discover what you really got because that's quite often hard to figure that out.

**Richard Campbell:** Especially when we're dealing with legacy app like this, how do we apply a test harness to it, and I'm thinking about an app that maybe isn't living in Visual Studio. We don't have that codebase option. I'm just wondering how we actually get test attached to it.

**Michael Feathers:** It really kind of start to figuring out what you need to change and the moment you figure out what you need to change, you have to figure out some place above that change where you can be able to sense the effects of your change and

at that point, you have to go and start breaking dependencies and then you just try to write test at that level.

**Carl Franklin:** Yeah.

**Michael Feathers:** In general, I recommend using various unit testing frameworks to go and do this sort of thing.

**Carl Franklin:** Right sure, wire up your test around what's already there. You have your stack and you can do test first because it's already written. So you said that you have to do a certain amount of conservative refactoring first. I'm wondering how much -- I mean, there's a real, I don't know, would you kind of say most people would feel an urge to just refactor the hell out of it the first time because there is so much of it that's going to change, but you're a fan of just getting as much refactoring done as needs to happen so that you can run a test framework around it, wire up the test and then change things.

**Michael Feathers:** It doesn't sound very satisfying, does it?

**Carl Franklin:** It doesn't, yeah I know. That's what I'm saying.

**Michael Feathers:** Here is what you do. If you have some big existing codebase, one thing you can do is just safely take it, put it out of version control and just refactor the hell out of it.

**Carl Franklin:** Yeah.

**Michael Feathers:** But just don't check it in again.

**Carl Franklin:** Sure.

**Michael Feathers:** And I call that scratch refactoring. It's really a great way of building and starting to discover what you really got and try to figure out what direction you might want to go and take things to. So for a team, if they got this big horrible application, go and just take an afternoon or a couple of hours and just try to copy and paste, move things around, rename them and not be terribly diligent about making sure they aren't introducing errors but just do enough to go and get a sense of, oh wow, this is what I see behind what we've got here, and then you will at least know it got some kind of direction you can move towards and it will take a long time to get there and maybe getting there isn't really the right metaphor.

**Carl Franklin:** Sure.

**Michael Feathers:** But you know you can make things better in a particular direction.

**Carl Franklin:** Yeah and you know, it makes sense now that you're saying it. You just have to make sure you fight that urge to...

**Richard Campbell:** But then it's definitely a throw away in the end likely.

**Michael Feathers:** Definitely it is.

**Carl Franklin:** Surely.

**Michael Feathers:** You'd never want to check that back in again.

**Richard Campbell:** You know; this is really about mitigating fear. I can go smash this thing up for awhile and at least get a sense for what is going to happen and then I can go back and try it again.

**Michael Feathers:** Yeah, definitely it is and there's one I think that just doesn't really occur to people all that often to go and just sort of do all sorts of crazy one time refactoring and just try things out. There are people who are just so used to the idea that they're going to check back in again and this scratch refactoring is a great way of going in and just getting that idea, getting those ideas.

**Richard Campbell:** But this is also the same since we get it from testing. Great test harnesses in general is this fearlessness. I can go and bang around on the code and it's going to point out to me where I made mistakes.

**Michael Feathers:** Yeah, definitely. That is one thing I just really wish that we can get to the industry. It's great now. TDD is getting higher penetration across the industry and even places that aren't doing TDD; they're talking a lot more about testing. They're doing a lot more with regard to testing and it's nice, integrate improvement are what we always have in the past.

**Richard Campbell:** So of course once again this whole legacy code, a question mark comes in which is what's really legacy and even if we do have a modern app, modern language anyway working in our existing environment, it's legacy just because it doesn't have a test harness around it so it sounds to me like you deal legacy once you put that test harness on it.

**Michael Feathers:** Yeah, we're pretty much making the commitment to go and move forward having test and it's really a hard proposition for many people that haven't that experience of working in a codebase that has that safety net. Yeah, I basically chose the definition of legacy to be that provocative. I

think it's a distinction which is so important right now that people need to really start to move towards it.

**Carl Franklin:** So as your going through this legacy code and updating it making changes, are there tools, languages, frameworks that you recommend people look into as possible solutions to help you along the way?

**Michael Feathers:** Well, there's the test harnesses themselves. I have much more to say about now.

**Carl Franklin:** Sure.

**Michael Feathers:** I like lint space tools and things like PMD and...

**Carl Franklin:** You said lint?

**Michael Feathers:** Lint, yeah, just a whole terminology from C and C++ tools that you go basically find – it's basically like jacking you warning level. Very high.

**Carl Franklin:** Is that an acronym, LINT?

**Michael Feathers:** I don't think so. I think it's just named after like laundry lint.

**Carl Franklin:** Oh okay.

**Michael Feathers:** Find the little puffs that are kind of odd on your code and do something about it.

**Carl Franklin:** I never heard that.

**Michael Feathers:** Yeah, yeah. Covered tools are nice and yeah, I think I would just go and give you a bit more information about your code. The thing is that for most part of this, there is so much that you learn by just seeing all the testing harness and trying to write test for things.

**Carl Franklin:** Right.

**Michael Feathers:** And beyond that also, if you have tools which can go and basically do a bit of diagramming of the codebase that you have, that's great, but often, I just kind of fall back on paper and pencil, just go ahead and punch a piece of code and figure out what's connected to what and understand how things are pieced together and knowing where will I want to go and take them overtime.

**Carl Franklin:** Right. What about functional languages? Are you a fan?

**Michael Feathers:** Yeah, I am to me there's an awful lot of interest in them now. I saw the conference down in Australia last year, or early this

year, and run into Eric Meyer and sort of asked him all these questions about Haskell that I'd kind of saved up in a sense. I think I'm playing with it on and off in the past six months or so and it really struck me that if you look at these things from the point of view of testing, there's an awful lot that functional programming has that tends to kind of get us away from some of the issues that becomes so problematic in testing.

**Carl Franklin:** Such as?

**Michael Feathers:** One of the hardest things to deal within testing is just they have some basically rather complex piece of code and it has all sorts of crazy side-effects. You don't even know where they are and if you're working functional programming language which has some level of purity which basically tries to reduce the amount of side-effects, then there's probably less to check out. Another thing that's nice about Haskell as well is the notion that basically IO is kind of sequestered off into some place with the MONAD.

**Carl Franklin:** Yeah.

**Michael Feathers:** That's really kind of nice because many applications I visit that I have really hard applications, you have no idea where access external system is happening. It's just scattered all throughout the application and you had to dig through in order to go and understand how some piece works. You don't know what it's really connected to so some of these accesses to external bits of the world can be sequestered in some area of your program. I think it's a nice step forward not only for engineering but also for testability.

**Richard Campbell:** I'm just thinking that those are points of failure where you're going to have trouble in your app.

**Michael Feathers:** Yeah, yeah.

**Richard Campbell:** Those are external points that...

**Michael Feathers:** And it's just nice to know that this is all kind of relatively in one place.

**Richard Campbell:** Absolutely. I mean, how into the functional language space are you? You thinking about the F# like this is really going to be the general purpose language to use?

**Michael Feathers:** I'm not sure. I know Microsoft is using F# internally from what I've heard. I'm frequently in contact with people, they're playing around with it and doing some things. It's interesting to see what's happening in .NET space that so many

of these features that are kind of concern with main stand functional programming are kind of migrating their way back into C# which is nice in everything. The only thing that kind of gets me a bit is that I feel sometimes like C# is getting to be a bit too big in a way like every time you turn around, something new is being added to this language and it's okay, I mean it's nice to have new features that people can use and everything but it's still -- I would hate getting into space with C++ is where basically there's just so much there to have somebody walk in and try to maintain the piece of code that was thrown out 5 or 6 years ago cause a tremendous amount of need for learning.

**Carl Franklin:** Do you think C# and F# could live happily side-by-side in a project, different assemblies?

**Michael Feathers:** I'm not really sure. I suppose they could. It's nice that .NET is oriented the way that it is that it allows interoperability nicely.

**Carl Franklin:** Yeah, that's for sure.

**Michael Feathers:** But I haven't really seen, I'm not sure.

**Richard Campbell:** I just don't get the sense that developers actually want to do that but they'd like to have their app more or less in one language, at least the code they need to maintain.

**Michael Feathers:** Yeah, yeah. It's a strange thing -- I really we're really in this odd place in the industry right now, it's almost like the Tower of Babylon in a sense -- like we have this really monoculture in the industry for awhile with Java and with Visual Basic and now we're at a point where basically everybody, you know, brothers designing a new language and putting it out there.

**Richard Campbell:** Yeah, for better or worse but isn't that really -- the sense we got and I think when we were talking with Ted Neward about this is that object-oriented languages hit the wall. So that's as far as they were going to go and now we're looking around for other solutions.

**Michael Feathers:** I think there's something to that. It is interesting though because I think that some of that really has to do with the type of systems also. There's this very strange thing that happens with object-oriented languages where if you have subtype polymorphism, you know it's good. If you have parametric polymorphism like generics and you have this strange cruft that's kind of a curse to the inner section of those two pieces.

**Carl Franklin:** Maybe we should take a step back and define those for our listeners again in a little more detailed...

**Michael Feathers:** Sure. Object-oriented languages have inheritance and that's just one form of polymorphism.

**Carl Franklin:** Sure

**Michael Feathers:** The way it's allowing you to go and substitute one thing for another and there's been this tendency within language psyche community that's going to basically add features to make it easier to go and have type save collections so things like the generics that we have with C# and C++, we have templates, and in Java, we have generics also and it seems like those two forms of inheritance, they mix but they mix in an odd ways and kind of force us to deal with more complexity than you'd like to sometimes as programmed, and I don't know, people haven't really talked about this all that much but I see this in C#, I see this in Java and I see it in C++.

**Carl Franklin:** They also make it much easier to do lots of things in generics...

**Michael Feathers:** Sure.

**Carl Franklin:** Although it's a little more complex to rock but I find it makes so much things so much easier. What was the term that you use for the type of polymorphism the generics represents?

**Michael Feathers:** Parametric polymorphism.

**Carl Franklin:** Parametric polymorphism.

**Michael Feathers:** Yeah.

**Carl Franklin:** And what's the idea behind that term parametric?

**Michael Feathers:** Well, it's based on the word parameter, just parameterized...

**Carl Franklin:** Ah okay.

**Michael Feathers:** Those are just that you parameterized types in order that you go and generate new types.

**Carl Franklin:** You know why that threw me? It's because the only other time I think of something that's parametric is in an equalizer. I have a selectable range of frequencies.

**Michael Feathers:** Yeah, a little bit off the side of that.

**Carl Franklin:** Sorry.

**Michael Feathers:** No problem but it's interesting with dynamically typed languages. You know, you haven't had the typed system going and guiding your hand in two different directions in a way so basically programming in a functional style and dynamically typed language and mixing with straight OL is odd in a way.

**Carl Franklin:** This portion of .NET Rocks is brought to you by our good friends at Telerik who bring you this special message. What's more important for your web applications? High performance on the server or on the client? How about footprint, the number of server request? There are so many potential bottlenecks that can drag your application performance and of course there is no universal solution for them. The good news is the guys from Telerik understand the complexity of that problem. When building their UI components, they isolate every probable source of performance loss, then they apply respective solution for different products, different scenarios, and even different browsers, the techniques, very dramatically. As a result, you the developer receives out of the box, highly reliable components that are optimized in every aspect of their behavior. I'm sure you'll be interested to learn more about the various performance boosting techniques for web applications. Just go to [telerik.com/topperformance](http://telerik.com/topperformance) for details and live demos.

**Richard Campbell:** Now, aren't these two separate things really, that we're seeing dynamic languages with their very loose typing very separate from functional languages or are there strongly typed and loosely typed functional languages as well?

**Michael Feathers:** Yeah; it's funny, that once they gain more press they're really the strongly typed functional programming languages but Schema's more loosely typed and functional and I believe that Erlang is not exactly certain about that.

**Richard Campbell:** It's like Scala, OCamel, Erlang, like there are so many functional languages all of a sudden.

**Michael Feathers:** Yeah, there are.

**Richard Campbell:** They are all trying to accept they're slightly different.

**Michael Feathers:** Yes, it seems. There are a couple of dominant themes. One is type inference which we're seeing in C# now and the other is Laziness, the ability to go basically create odd data structures and basically go ahead and choose to kind of like put limitations on them after the fact.

**Richard Campbell:** Right and really there's a performance opportunity there. Rather than populate everything, just let it run for a while and limit itself by execution.

**Michael Feathers:** Yeah.

**Richard Campbell:** It's just an interesting way of thinking about the problem. It's so different from what we've been taught.

**Michael Feathers:** Yeah, I definitely recommend that people, if they have a chance, try to learn Haskell. It's a very -- it really enriched your mind in some very strange direction.

**Carl Franklin:** We did a couple of dnrTV or at least one with Venkat Subramaniam on Haskell just to get our -- it's a good language to get your mind around functional programming.

**Michael Feathers:** Definitely.

**Richard Campbell:** This comes back to that whole legacy question of are object-oriented apps headed to being legacy apps at some point just because they're not stuff people are going to work on anymore.

**Michael Feathers:** It seems to me that's the state of everything in a way. I mean, progress happens and so the things were hot, in the technology sector at one point becomes less hot...

**Richard Campbell:** And I also buy into the notion that C# is becoming this grab bag that every interesting feature that comes along in any language, C# has some implementation of it.

**Michael Feathers:** Yeah, yeah and it's something that I think that you just have to deal with it at some point and time. I mean, there's a lot to be said for very minimal languages.

**Richard Campbell:** Right. Yeah, I think C# is definitely the opposite of a minimal language.

[Laughter]

**Carl Franklin:** So you really think that C# is getting too *Swiss Army Knife* on us.

**Michael Feathers:** I think so, I think so. I think it's kind of hard to argue against that.

**Carl Franklin:** Oh, it's hard to argue against that. I'm just wondering if that's necessarily a bad thing.

**Michael Feathers:** It may not be. I know it's just one of those things. I mean, the language will continue to grow in market share and people find the things that they want to.

**Carl Franklin:** Yeah.

**Michael Feathers:** But you just go back to that same issue at a certain point. People that are new to the programming industry, they have to basically start from ground zero and learn the language and that just gets pretty tough after all.

**Richard Campbell:** It's feeling like the barrier to entry for developers is getting higher.

**Michael Feathers:** It seems like that too to me. It's a strange thing in the industry now. It seems like we're kind of revisiting a lot of old stuff in a way which is kind of neat.

**Carl Franklin:** It is cool.

**Richard Campbell:** I don't see that as a bad thing. I think that there is that sort of inflection point of okay, we did sort of hit the point with objects with these managed frameworks, we're like, well, okay, that is as good as it's going to get and then; yet I'm still not happy.

**Michael Feathers:** You know, that's great and I think the nice thing is that many of these things were really settled a long time ago. Now we're settled that basically people came across a very interesting idea a long time ago in the functional programming community and just kind of resurrecting these things.

**Richard Campbell:** Right, they were living in the periphery for years and years and years and now they have suddenly gotten new interest.

**Michael Feathers:** Their history in the next 30 years; we'll be mining the last 30 years or so.

**Richard Campbell:** Yeah and I just wonder if the last, the previous 30 years was very much finding the 30 before that.

**Michael Feathers:** Yeah, I don't know.

**Richard Campbell:** I don't remember, I'm not that old.

**Michael Feathers:** Yeah, I know. Me neither.

**Richard Campbell:** I don't know how if you are in the functional programming, we can go away from this if you want but where do you feel that aspect of functional programming and parallel programming tied together?

**Michael Feathers:** No, there's never anything interesting about that and you know, I think other people have mentioned this enough time often but the whole notion that if you're trying to minimize state, I mean my state transition, it becomes much easier to share data structures across different...

**Richard Campbell:** Right.

**Michael Feathers:** Because it's much easier to go take a look at the problem and say, okay, well, I throw these bits to this process and I throw these bits to this process and move on in a good way. The thing that is just so awkward about this is that this is a very different way of thinking that programming function and it took me a while to kind of grasp certain things and we were telling friends when I was learning a bit of Haskell I said this is great. The thing is just you have to learn how to think like a Martian and as a Martian I'll think of that. So much about what we're used to in programming, it's basically maintaining state and so we move away from that as bit of a chore.

**Richard Campbell:** Yeah, it's a very different way. Being very respectful a state, I would argue that as web guys haven't been doing that for a long time because state has been such a pain for us and so in some ways, we're well equipped for that that we think about this very transitory things in great independence between execution cycles because we were so gentle with our state. The size of it matters, how often you touch it matters, that whole matters so much more in the web world.

**Michael Feathers:** Yeah, I think so. The thing is the developed proper intuitions for it.

**Richard Campbell:** It's a beginning. I'm just wondering if that's a species that developers are going to do well in this new immutable world.

**Michael Feathers:** Yeah, it might be, might be the case.

**Richard Campbell:** I was poking thru your blog a while back and run across -- I know you're a testing guy and a big fan about testing and then there was that discussion about the flaw in unit testing.

**Michael Feathers:** Yeah, yeah.

**Richard Campbell:** I think the listeners would really probably appreciate this sort of insight into how testing sort of got here.

**Michael Feathers:** Okay, yeah sure. The thing I'm trying to get across in that blog is, well, it's plenty of things. I basically go ahead and have my Google

blog search set to go and give me all sorts of hits about unit testing and so I get to see a lot of conversations about that topic. One thing that kind of strikes me however how small is that people talk about unit testing versus integration test, what's really important. I mean isn't it better to go and have test at a very high level in the application to see that all pieces are tied together in a good way, and it took me awhile to kind of see this but I think that that really kind of rest on a bit of flawed reasoning. A friend of mine, Steve Freeman who was one of the guys who is kind of instrumental on coming up with the idea of mock objects spec in like 2000 or so was talking to me about a style of TDD that he'd done with some other friends and they were basically going and creating classes and mocking up all the collaborators and that was all the testing that they were doing. They were basically going ahead and creating all these units and testing them to death and then just looking at them all together and things just running and it worked out fine and I said, "Well, that's really kind of interesting that you can do that but it's like what about the integration level, don't you have lots of integrationist?" And he said, "Well, not really. We really didn't have all that much in that area. We had very, very high quality numbers." And when you think about that, there's something very strange about it. Why would testing the unit level basically go and make a big difference at the high level of quality and program?

**Richard Campbell:** Yeah.

**Michael Feathers:** And I think that what it comes down to is this, that when you are choosing the right test for something, you're thinking about it very deeply. You have to understand exactly how it operates and you're learning how it operates and chances are that as you go through that process, you'll avoid certain things and you'll think things through well enough that you arrive at higher quality without necessarily pinning the bugs in the first place. So it seems like there's this way that test driven development and unit testing in general kind of give us quality but not through test failures. They give us quality by going and it's kind of forcing us to go and basically think through our code a bit deeper than we would otherwise and it's funny about this because when you say this to people, the first thing they think is "Oh, that's great. All I have to do is basically go ahead and think about my code a bit more and I got all the high quality that everybody would ever want to have." But I don't think it's quite that easy. I think that we need a better approach in a way to go and basically help us think through things and to kind of force us into different modes of thought and beyond that when you think about it, it's great that at the end of the process, you have all these tests you can just run progression that you send to go and kind of pin down the current behavior of the system and know

that when you make changes you can find out quickly what's changed and what hasn't. So I think it's funny, you'll see people quite often getting into this conversation about what level of testing do I need in my application. When I say level, I'm talking about that one level in the architecture...

**Richard Campbell:** Right.

**Michael Feathers:** What is this thing that I need to test the high level or the low level? What's going to get me to quality numbers and to me it's really the process itself.

**Richard Campbell:** The process of writing the test seems to be the point at which you have that thought. It's not running the thought.

**Michael Feathers:** It's pretty much like the old design by contract stuff.

**Richard Campbell:** Right.

**Michael Feathers:** So you are going to write a class and you come up with the pre-conditions and the post-conditions for every method that you write and you know that process forces you to think about things in a very different way and so I think you need unit testing as in the sense nice trick for us that way.

**Richard Campbell:** That it really has that -- to me the thing is forcing the thought but it's interesting to thing that just because you have the test, doesn't mean you have the thought. It was the creation of it that did the thinking for you.

**Michael Feathers:** Yeah and in fact one of the things that I really hate, I once saw or visit a team and I hear them say that, "Well, you know, people that we work for are basically saying that they want to go and raise the coverage numbers." They say that by the end of two springs, we've going to have our coverage up to 40%. it's only 30% now. You can do that sort of thing but the thing is I think at the end, developers have to feel the unit testing is something that's there for them and having an artificial goal to raise coverage to a particular number, quite often even without trying to gain things, developers get into this situation where they write tests which are not quite as informative as they would be otherwise.

**Richard Campbell:** They're just trying to reach the number. They're not trying to achieve the goal which is really -- I mean, isn't the goal really confidence?

**Michael Feathers:** Yeah, the goal is confidence and really understanding what it is that you're doing and so it's nice to -- you know whenever I'm working with teams, the first thing I try to get across is that this

is for you, this is supposed to make you like future developers.

**Carl Franklin:** Yeah.

**Michael Feathers:** And if you don't look at it that way, you're not going to get it.

**Richard Campbell:** And I like this description you've given of this whole I'm looking forward to going to work today because I feel like I can write code successfully today.

**Michael Feathers:** Yeah

**Carl Franklin:** Right.

**Richard Campbell:** So many applications, the more code we have, the less able you're to do that.

**Carl Franklin:** That's right, yeah.

**Michael Feathers:** Yeah.

**Carl Franklin:** It certainly is a way to make your abilities scale.

**Michael Feathers:** Definitely.

**Carl Franklin:** Agile development, has this worked out the way that we thought it would?

**Michael Feathers:** I think as with everything else, you will have very interesting ideas. They were tried and they work. With very high performers, they basically try to go and take the ideas and spread them out across a team, a company in the industry. I think that the uptake of Agile has been very high and it has actually went to some nice successes and there is a sense that basically a lot of Agile has become a little bit dilute in a way. There are some teams that I've seen that are just kind of like Agile in name only.

**Richard Campbell:** Right.

**Michael Feathers:** But I have to kind of admit that it's better than we've seen previously. I think the only thing that kind of really gets me right now is that it seems that there has been a really moratorium on the discussion of design in the mystery of the past, I guess 10 years now. There was a period of time when people are talking about the intricacies of object-oriented design and various things along those lines...

**Richard Campbell:** Well, and this idea of I don't disagree with you and all, Michael here, that because I fail like people stop talking about this but...

**Carl Franklin:** That's true.



**Richard Campbell:** Microsoft had their whole DNA model where you had the presentation layers separate from the -- business object layers separate from the data layers, those kinds of discussion I just don't see them happening anymore.

**Carl Franklin:** Well, people are still gung-ho on the MVC idea.

**Michael Feathers:** They are, they are and that's nice and it's also great like Eric Evans' book, Domain Driven Design, has been very, very good and helpful and kind of raising the profile to these discussions again. I think it's like anything else. Once you basically go ahead and say, "Well, you know, design is going to emerge," and people sort of say, "Okay, well, we have to talk about that."

**Carl Franklin:** I agree. I think that most of the opponents are talking about separation of concerns and that's your mantra and then everything else is left up to you.

**Michael Feathers:** Yeah, yeah, I think we can have better discussions.

**Richard Campbell:** I think we're falling over at times. People are building these apps with separation of concern and then not getting performance or not getting the distribution they want because they didn't really think through a design. I almost wonder, are we missing the architects? Is architecture sort of been forgotten about?

**Michael Feathers:** You know, it's funny. I think to a degree, yeah. It's certainly kind of hard because there has been a lot that's happening with the nominal title of architect that's been not all that nice in the past 20 years or so, but also the thing is it architectural expertise or is it definitely something which is necessary in many application domains?

**Richard Campbell:** And is it different skill from development?

**Michael Feathers:** It can be but we've also seen in the past where basically if you have architects that haven't -- they don't basically get their hands dirty in the cold, then they can start to offer advice which is a little bit off to the side of what is really necessary like their view into the code has to be real in order to go and basically make a decision sometimes also, so it's funny. I think there's a lot of blurring in the lines that has happened over the past 10 years or so which I think is good and necessary but the thing that we have to just remember is that expertise is still needed.

**Richard Campbell:** Well, yeah and at the same time you think if this industry is really maturing,

specific roles would tend to fall out. We'd un-blur the line. The generalist becomes a less effective way. You need a specialist but I don't know that we're actually getting there.

**Michael Feathers:** Yeah, well, I think it's like when you look at that it's kind of there is that one extreme where basically you have someone who is an architect and they didn't know anything about that code at all, where their last experience with code was 10 years ago.

**Richard Campbell:** Right.

**Michael Feathers:** And we know that that doesn't work and now we have people there working in teams and some of them have very good architectural experience and that can work out. I think the key thing is that we just have to make sure that people do have that knowledge and that knowledge is able to spread through teams to the places where it needs to be. We'd hate to see us go back to the original extreme and that is very *cartoonis*; I guess.

**Richard Campbell:** Right.

**Michael Feathers:** But I have seen some strange things.

**Richard Campbell:** Yeah, people do these strange things...

**Michael Feathers:** Its just one thing to have the goal, but the thing is it has to be a very communicative role in a sense.

**Richard Campbell:** This sort of drives us back again to this whole legacy thing which is -- isn't there a problem here as to lack of documentation that we lose track of how this app was built and why we did things?

**Michael Feathers:** I visit a lot of teams that has some documentation. I have a lot of documentation. This similarly pertains to the structure of what they've got. I think the main thing is really continuity of knowledge in the team and when you have that, everything works out okay. When you don't, you basically have this fishing expedition that you need to go through everything you need to make changes. Tests can help a great deal if the tests are written well and they communicate the functionality of the system, but yeah, documentation, I mean, there is a place for some of it that you definitely want to get in that state where you're using explicitly detailed documentation that has been modified every time you modify the code, by that time it gets crazy.

**Richard Campbell:** Right. Well, it always ends up being wrong.

**Michael Feathers:** Yeah.

**Richard Campbell:** And that's the bigger thing. In reality is that even when we had the documentation, the devs wouldn't read it; and when they did read it, the thing that they were looking for is how it was wrong from what was actually in the code.

**Carl Franklin:** Yeah.

**Michael Feathers:** Yeah.

**Richard Campbell:** So Michael, swinging back to your book a little bit more, you talked about specific techniques for breaking dependencies and I guess that's primarily to be able to get the test in there?

**Michael Feathers:** Yeah, yeah.

**Richard Campbell:** So what kind of techniques are we talking about?

**Michael Feathers:** So yeah, this is one technique I use with Java code quite often. it's called Introduce InstanceDelegator and the idea behind this is that, you know, you've got some code you want to test and it goes in there and then making some calls to some to some static methods on some other class and that can be okay and everything, but if those static methods kind of hide some functionality that you don't want to have happen inside of your test like maybe calling through a database or something along those lines, you're kind of stuck in a sense because you have static methods and you can't really override them, you can't really go and step pass and override them in a sense. So Introduce Instance Delegator is really all about going to that class that has static methods on it creating virtual methods on it and having that delegator over to the static and then you have to find some way to go and basically get an instance of that object into the code that you're going to use they call the static methods and you ended making a call to the virtual methods rather than the static method.

**Richard Campbell:** So you're slipping a shim in essentially.

**Michael Feathers:** Exactly. Yeah and in fact one of the concepts I discuss in the book is what I call a seam, it's kind of like the shirt, the seam between the sleeve and the body of the shirt.

**Richard Campbell:** Right.

**Michael Feathers:** In programming languages, quite often we have this places where it's easy to replace one thing with another.

**Carl Franklin:** Right.

**Michael Feathers:** Not really editing the code in that place and I don't think the language designers are really conscious of this, that the seams are there and that they can be very useful and fantastic but one thing that's pretty clear, the static is not a seam really. When you have static methods quite often, it's very hard to replace them in a testing situation.

**Richard Campbell:** So you're just creating that point where you have the virtual method to the static method gives you that ability to sneak in there.

**Michael Feathers:** Yeah, yeah.

**Richard Campbell:** That's a good one.

**Michael Feathers:** I tend to find overtime that I use static method periodically but not just really around, I think, in one I want to replace and it's actually kind of nice to just basically keep things virtual also.

**Richard Campbell:** Right.

**Michael Feathers:** It gives you more flexibility overtime.

**Carl Franklin:** Everything can be solved with one additional layer of indirection.

**Michael Feathers:** Yeah, an old classic saying.

**Richard Campbell:** And certainly forgetting test is one of the things we're going to do. That's only one. Do you want to throw out a couple of others?

**Michael Feathers:** Yeah. Well, there is a couple there, very simple. One is extracting interfaces. Going to any particular class and choosing extract interface. One thing I find very powerful though is when you have some very large class, I mean a class that has more than five methods -- I'm waiting for laughter.

[Laughter]

**Richard Campbell:** Yeah. I'm thinking that's large huh, five.

**Michael Feathers:** Yeah. There are classes that have 50, 70, 80 methods.

**Richard Campbell:** Yeah.

**Michael Feathers:** It's really kind of hard to get handle on code that has like that kind of bulk. One thing that can help sometimes is to go and just create an interface and make the class implement that

interface and just keep the interface empty, then you go some place near your code where you instantiate that class and you basically go have them create a reference to the interface and you assign the instance of your object to that interface and once you do, your idea is going to tell you all these -- give you all these error messages saying, oh, you have these reference to such and such and it doesn't have this method and it doesn't have this method and it doesn't have this method and if you basically hunt that through all those errors, you're basically given a list of all the things that are used in that context.

**Richard Campbell:** Right.

**Michael Feathers:** Starting on pull the signatures over into that interface.

**Richard Campbell:** So you're using error messages to generate an inventory of utilization.

**Michael Feathers:** Yeah, yeah and when you think about it, the one thing that people don't really appreciate sometimes is the notion that really your compiler is a test also.

**Richard Campbell:** Yeah, a good one.

**Michael Feathers:** It gives you this feedback and the cool thing in using this technique is that if you have one of these very big bulky classes, you can do this and start to get an inventory of what methods are being use in some particular context and of course once you get that down, you can go and give that interface a good name and start to go and increase the net understandability of the code you're working at and often that is a great first step we're going...

**Carl Franklin:** Sure is.

**Michael Feathers:** In recovering architecture to where you want to go next.

**Carl Franklin:** Very cool.

**Richard Campbell:** Definitely a clever technique and I've heard that line before that the compiler is one of the best test tools that you can possibly have.

**Michael Feathers:** Yeah.

**Carl Franklin:** Yeah.

**Richard Campbell:** We don't tend to think of it that way, it's almost like we're embarrass when it doesn't - - so much energy has been put around breaking the bill...

**Carl Franklin:** That's right.

**Richard Campbell:** Feeling compilation.

**Michael Feathers:** Yeah and not only that, just the notion of kind of introducing errors to figure out where to make a change in your code. You're also a bit hooky in a way.

**Carl Franklin:** Debug.Assert.

**Michael Feathers:** Yeah.

**Carl Franklin:** I have fallen into this trap too which is; I'm writing some code where I want to bury an exception like I'm closing a socket for example.

**Michael Feathers:** Yeah.

**Carl Franklin:** I don't care if it throws an exception because if it's already close, fine, that okay, I just want to close it.

**Michael Feathers:** Yeah.

**Carl Franklin:** You know, I'm not getting an exception because it was open and I couldn't close it. I've fallen into the trap of just burying it right there and then before I test anything and of course, you want to leave all those things in place while you're testing so you know that you've written some good code. So I go back. So now when I'm writing code, I'll always leave the try/catches in there and do a Debug.Assert or something like that until I'm satisfied that I have a case that works as expected, then I will go back and handle those however I want to.

**Michael Feathers:** Yeah. I think a lot of times developers have this -- they don't like to write temporarily.

**Carl Franklin:** Yeah.

**Michael Feathers:** Working on something else. It's very powerful. If you just want to just put something in temporarily just go and...

**Richard Campbell:** This whole idea of -- there's a couple of things we talked about over the course of this hour. We've really got into this idea of messing with code that ultimately you know you'll throw away.

**Michael Feathers:** Yeah.

**Carl Franklin:** Yeah.

**Richard Campbell:** It's interesting how we have this culture that says if it comes from my fingers, it must be gospel. We must keep it.

**Carl Franklin:** Yeah.



**Michael Feathers:** Well, you know, I think a lot this as just really just to run on, in fact, the fear. If you basically have this experience where every time you change code, something terrible happen; every time your co-worker change code, something terrible happens. You're going to be extremely deliberate in about any change that you make.

**Carl Franklin:** Yeah.

**Michael Feathers:** And for a good reason but if you're able to back off and say, wait, I have to check this in right away and see what happens...

**Richard Campbell:** Or ever.

**Michael Feathers:** Or ever, right and try this out and see what happens. Now you start to -- I get passed the stasis.

**Carl Franklin:** Also you've had the situation where there's an environment in the office of non-cooperation so that you've got people vying for other peoples' jobs and so everybody is trying to look like they're the smartest developer in the room and oh, my God if you have code that breaks, well, you know...

**Michael Feathers:** I really feel that you know, if you've heard, like Conway's Law. This gives the notion that every system in architecture is doomed to duplicate the structure of the team that produced it.

**Carl Franklin:** Ooh, interesting.

**Richard Campbell:** Nice.

**Michael Feathers:** Yeah, yeah, a guy named Conway. He basically came up with this in the 1970's and...

**Carl Franklin:** Conway, what a genius you are!

**Michael Feathers:** Yeah, there's just so much that kind of follows out of that, I mean it's truly funny when you see a lot of code that's really crazy and odd. I think it says a bit about the team that produced it and I mean, you can look at these things as being like an individual sport in a sense. It does seem that if you want to produce a code and it's a really multi-person effort, these people need to get along with each other in order actually produce good code.

**Carl Franklin:** Yeah.

**Michael Feathers:** Otherwise you have people sabotaging or just by not lending help in certain situations

**Richard Campbell:** Right.

**Carl Franklin:** There is so much truth to Conway's Law here that I'm just up and spending the last couple of minutes distracted thinking about that. There is so much deeper stuff there.

**Michael Feathers:** There really is.

**Carl Franklin:** It is really is.

**Richard Campbell:** Yeah, it has some possibilities. You know, we find that over and over again, this is how that whole rising to your own level of incompetence.

**Michael Feathers:** Yeah.

**Richard Campbell:** Oh here he is, Melvin Conway in 1968.

**Michael Feathers:** Wow, earlier than I thought.

**Richard Campbell:** Wow, that is old.

**Michael Feathers:** Yeah. There is a lot that this kind of falls us back into.

**Richard Campbell:** An organization which design systems are constraint to produce designs which are copies of the communication section of these organizations.

**Carl Franklin:** Wow.

**Michael Feathers:** It's a funny thing about this now that I'm thinking about and really it has to do with this notion of emergent design and everything, and the scene of emergent design works very well but one thing that's troubling me sometimes is in very large organizations, sometimes you have extremely large codebase and you have teams that are working on a little pieces of it then moving on, working on little pieces of it and moving on and the original idea was naturally to have collective code ownership which is good but the thing is, it's like when you think about that and you kind of start to run it through the veil of Conway's law, truly your arriving at that point as to something which is more like no ownership in a way and you end up with this really odd tragedy of the commons.

**Richard Campbell:** Yes.

**Michael Feathers:** So I think that's something we really have to go and deal with in marginal organizations. The notion that you do really need to have some ownership and if you don't, you shouldn't be surprised at what you get.



**Richard Campbell:** Yeah, no kidding. There is a variation on Conway's law that says in every organization, there is one person who knows exactly what is going on at all times, this person must be fired.

**Carl Franklin:** Hey, you know Richard, in most cases that person is you.

**Richard Campbell:** Yeah, thanks.

**Carl Franklin:** Not with us. I mean, in your history of projects that you've dealt with, you're the guy who knows where everything is going on and who's messing up.

**Richard Campbell:** Yeah and I make sure I got to go.

**Carl Franklin:** Eventually it's your turn.

**Richard Campbell:** Yeah, that's very funny. Michael, I think we're getting close to the end of the show here. Any sort of final thoughts?

**Michael Feathers:** I guess the main thing is I know you had Bob Martin on a bit earlier and he talked about clean code and really that's the answer with all this.

**Carl Franklin:** Yeah.

**Michael Feathers:** I spent a lot of time going around and visiting teams, they've already gotten themselves in quite a bit of trouble and they're escaping at, there are things that you can do to make things a bit better but it's best not to get to any of that situation in the first place. It's not a nice place to be.

**Richard Campbell:** The overlay on all of this is testing angle. Are you willing to just go as far as to say that with great testing infrastructure, we don't have legacy code anymore?

**Michael Feathers:** No, because the thing is there's always a way to screw things up.

[Laughter]

**Carl Franklin:** Yeah, that's true.

**Michael Feathers:** This is something which is really a great pre-condition in a way. It's a great tool to go and basically arrive at better code and yeah, I know of all the people I meet developing, I have the most respect to people who have been kind of through that, they kind of noticed that they've thawed up things in various different ways and just discovers like, you know, gee, if I just write a test to what I'm working on, and start to notice how that resonates

with their work, it's really a powerful thing and I notice that people who get that epiphany tend to do much better overall.

**Richard Campbell:** Awesome.

**Carl Franklin:** All right. Michael Feathers, thank you very much for joining us.

**Michael Feathers:** Okay, well, thank you very much.

**Carl Franklin:** It's a pleasure talking to you.

**Michael Feathers:** Thanks.

**Carl Franklin:** 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](http://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](http://www.franklins.net). For more .NET Rocks! episodes and to subscribe to the podcast feeds, go to our website at [www.dotnetrocks.com](http://www.dotnetrocks.com).