



Hanselminutes

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

Text transcript of show #165

June 18, 2009

Working Effectively with Legacy Code with Michael Feathers

Scott's in Norway this week and he sits down with Michael Feathers. Michael is the author of "Working Effectively with Legacy Code." What is legacy code? Are you writing legacy code right now?

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



Our Sponsors

 **telerik**
deliver more than expected
<http://www.telerik.com>

 **nsoftware**
<http://www.nsoftware.com>

NET 
DEVELOPER'S JOURNAL
<http://dotnet.sys-con.com>





Lawrence Ryan: From hanselminutes.com, it's Hanselminutes, a weekly discussion with web developer and technologist, Scott Hanselman, hosted by Carl Franklin. This is Lawrence Ryan, announcing show #165, recorded live Thursday, June 18, 2009. Support for Hanselminutes is provided by Telerik RadControls, the most comprehensive suite of components for Windows Forms and ASP.NET web applications, online at www.telerik.com, and by .NET Developers Journal, the worlds leading .NET developer magazine, online at www.sys-con.com. In this episode, Scott talks Legacy Code with Michael Feathers.

Scott Hanselman: Hi, this is Scott Hanselman and this is another episode of Hanselminutes. I'm here in Oslo, Norway at the Norwegian Developers Conference and I'm sitting down in a reasonably quite conference room with Michael Feathers, the author of Working Effectively with Legacy Code, works for Object Mentor and he knows all sorts of things that I don't know so I said I'm going to jump at the chance to talk to you so thanks for sitting down.

Michael Feathers: Yeah, thanks a lot. I'm really enjoying the conference here. It's a great venue and lots of interesting speakers.

Scott Hanselman: You know; and as with all conferences, it feels like the real interesting things happen in the hallways and happen between sessions.

Michael Feathers: Yeah.

Scott Hanselman: Uncle Bob was talking about something and the guys were brainstorming that someone was saying that they like to select chunks of code within a long method, something that's maybe legacy code, and they like to go extract a method using their refactoring tool of choice and then that gives them a sense of what the inputs and outputs for that block are.

Michael Feathers: Yeah.

Scott Hanselman: But in a method that maybe has three or four inputs as far as parameters into the method...

Michael Feathers: Yeah.

Scott Hanselman: That might show them that they're really reaching outside the method and looking at global variables, if it is a context, and that all seems very klugey to me that maybe this person should be using a different kind of programming language. I mean, they're basically saying there's a lot of side-effects that could happen.

Michael Feathers: Yeah, yeah.

Scott Hanselman: Should we maybe doing something else?

Michael Feathers: I don't know. I think we have to go and really ask ourselves very seriously in the industry right now. I've spent a lot of time helping people with really radical basis trying to recover, you know, and it's not atypical to have these baked retest and line methods and you have to go and try to figure out how to break them down and do various things. One thing I noticed is that side-effects are really a big deal. If you did have a way to rise through side-effects, you can basically make a lot of change much more effectively and much easier. So it just kind of makes me wonder, it seems like we are making a move in the industry towards functional programming languages and with some of the delving that I've done with functional programming languages has really kind of led me to believe that this could be a bit of an answer.

Scott Hanselman: Uh-hmm.

Michael Feathers: I mean it's not going to be a complete panacea because there are always problems in development, but it seems like if we can get control of the side-effects situation we might be able to go on this more cool stuff.

Scott Hanselman: For those listeners who may not be into this stuff and be really understanding it, we're talking about a way of programming that involves not changing something a variable that then we wouldn't have expected to get changed. Maybe you can explain that, that notion of side effects.

Michael Feathers: Yeah. There are a couple of harms actually in functional programming. One is that functions are like first class citizens. You can pass functions to other functions. The thing that's very important for us, like in the conversation, that really is the immutability aspect. The notion that you really are working with constants all the time. Whenever you have a variable, you don't really reassign to it. You just go ahead and construct a new variable based upon the previous value and do some computation to produce that new value. You know, one thing I noticed over and over again, when you're looking at very long methods, it's the temporaries that get you, okay. When you have temporary variables inside the method and people are going and randomly updating them in various different places, those temporaries are just really kind of like a glue that make it really very hard to go and extract out pieces and you make separate methods out of them. It's just really hard to have risen through all that nonsense. When you have immutable data, then you're constantly constructing new values and you're able to see, like you're saying earlier, exactly what comes in and exactly what goes out and your rising is easier not only for you but also for the tool that needs to figure out whether you can really extract the method or not.



Scott Hanselman: Your book is about Legacy Code but sometimes I feel like Legacy Code is code I wrote last week.

Michael Feathers: Well, yeah, then you're really in tune with the definition of the book and why. It's funny, I can't try to find a decent title for the book when I was writing it and I kept coming back to the notion of Legacy Code. I have a friend named Erik Meade I called him up one time, he was doing a gig with a new client and I said, "So how is the day-to-day?" He says, "Oh man, they're writing Legacy Code."

Scott Hanselman: They're writing Legacy Code but they're creating it every minute.

Michael Feathers: They're writing much stuff. Yeah and I mean that really struck me that there's the dictionary definition of Legacy Code which is code that you got from somebody else, but there's also this definition that we also feel informally which is it's just code you may have written yesterday but it's just intractable. It's really hard to work with.

Scott Hanselman: Uh-hmm. Why is it intractable? I mean, nobody wants to admit that they're the kind of person that would create Legacy Code and they might say, "Well, I inherited the system. I'm only able to write Legacy Code given the constraints of the system."

Michael Feathers: It's really been a fallacy though. You can always, in an old codebase, make fresh starts with the new things that you write and you have to be really -- you know, you always have a choice between going and adding five lines in an existing method than creating a new method that you delegate out to, doing things where you have smaller structure and clear, simpler structure and moving forward with tests and that sort of thing. It's tough because in the existing code you're surrounded by a deadness but it pays to learn design principles and learning good things about code structuring and make fresh starts in the existing codebase.

Scott Hanselman: Am I always going to find myself though, spackling over something? It's like, you know, when you buy an old house and you said, "Well, it just needs a fresh coat of paint," you put a couple of coats of paint or something but you'll always know that there's mold and nastiness in the walls.

Michael Feathers: Well, you know, I think the truth of the matter is that for many large existing systems, you're never going to escape the structure that you've built-up over the years but you can build new abstractions. There's a really cool pattern that's been written up on the net called Strangler Application.

Scott Hanselman: Strangler?

Michael Feathers: Yeah, Strangler Application and...

Scott Hanselman: Like one who chokes someone to death?

Michael Feathers: Yeah, yeah.

Scott Hanselman: Okay.

Michael Feathers: And the idea is so if you have this big hawking mess of Legacy Code, what you can do is you can say, "Look, if we have Events coming into the system, all we can do is we can divert some of those Events to something which is really like a new system," and then the outputs from that end up going in being pushed out with normal outputs. So you're taking certain things that come in and kind of diverting them off into this obfuscating structure which is this new architecture that you're building up slowly with better structuring. That can be good, but sometimes it just really never becomes a complete thing. You're just building a secondary pathway in your application that allows you to do things in a better way.

Scott Hanselman: Why is it called Strangler? I don't understand that.

Michael Feathers: Well, because it's kind of like vine, a Kudzu, or something like that. It's kind of like growing over the application and sapping it's strength by becoming -- you know, the thing itself, the vine becomes the stronger thing. So it's kind of strangling the existing structure...

Scott Hanselman: You're imposing constraints on this.

Michael Feathers: Well, what you hope is that the older pieces really just not end up going and doing things functionally for you.

Scott Hanselman: I see.

Michael Feathers: And that more of the functionality is use to the newer things.

Scott Hanselman: They die away.

Michael Feathers: Yeah. So it's like that.

Scott Hanselman: That they away.

Michael Feathers: Yeah.

Scott Hanselman: Okay. So you build a system around it that's strangle the old system.

Michael Feathers: Yeah.



Scott Hanselman: And forces it to change or to go away.

Michael Feathers: Yeah and that's one strategy. Another strategy is to essentially just keep creating new classes and building their structure in existing application without trying to go and create a parallel pathway. The third way of approaching things is to take a big area of code and try to go and write enough test for it so that you feel that you can go and do some serious refactoring in that area. That's kind of what I specialize in a bit with the book, it's trying to go and break dependencies in ways. They're very conservative so that you feel like you aren't really breaking the applications' behavior.

Scott Hanselman: Right.

Michael Feathers: And then you're able to go and write tests against that and with those tests get a lot more confidence that you can start to refactor and make some really nice structural changes in the code to make your maintenance easier.

Scott Hanselman: Hi, it's Scott here from another place and time. I hope you're enjoying the show so far. I apologize for interrupting it but I want to let you know that assembling a podcast like this every week isn't free. Certainly the bandwidth bill crushes us every month so I want to let you know that this show is sponsored by Telerik. They make the show possible and they make some pretty cool products as well. For example, if you're trying to build a Web 2.0 AJAX C application trying to use the Web 1.0 components, it's kind of difficult. You got to get to the next gen stuff to kind of build the next gen websites, and that's exactly what the folks at Telerik have got in their new upcoming product which is codenamed RadControls Prometheus. It's a big pack of web controls built entirely on top of the Microsoft ASP.NET AJAX stuff that you already understand. It's going to give a lot of performance interactivity in your next project. They mirror the ASP.NET AJAX API so that development is really straightforward. Client scripture is shared, loading time is pretty fast, it sets a couple of properties, it even bind the web services for a really efficient operation. The new RadEditor for ASP.NET AJAX loads up to four times faster than before, and the RadGrid will do thousands of records in milliseconds. But of course it's better to try these things for yourself so you can visit www.telerik.com/aspnetajax and download a trial. Thanks a lot for listening and we'll get right back to the show.

So maybe I could ask you a question about -- I was on site recently at a large company.

Michael Feathers: Yeah.

Scott Hanselman: They had a Legacy system and the Legacy system was in C++ and it was probably, gosh, almost maybe 15 years old.

Michael Feathers: Yeah.

Scott Hanselman: But it worked.

Michael Feathers: Uh-hmm.

Scott Hanselman: And it worked great, but it was Legacy because it wasn't on the new shiny interfaces and what-not.

Michael Feathers: Yeah.

Scott Hanselman: It worked and they had people actively adding to the system but it was what I would call an, I don't know, I think I invented this, maybe I invented it in the case of I pull it out of my butt, I have this caste system in my mind for APIs.

Michael Feathers: Yeah.

Scott Hanselman: Third class APIs like the ones that were written in the early '90s were always .execute, or .do it, or .master, whatever and you pass in a magic string and maybe it's tag-value pairs or SQL but it's something that the language that you're writing can't see but it's some custom thing.

Michael Feathers: Right.

Scott Hanselman: These are often in the large systems that were multi-tiered. The only thing that we could easily marshal across the strings.

Michael Feathers: Yeah, uh-hmm.

Scott Hanselman: So we pass tag-value pairs around. That was a third class interface, and then like a second class interface might be something like a dataset which is a structure that organizes these, or a hash table, or a dictionary.

Michael Feathers: Right.

Scott Hanselman: Still not first class, and then a first class is like a proper interface. We ended up building over the course of a week a prototype that was kind of a series of layers on top of this onion the hope being that we would create data transfer objects that look closer and closer to the way we wish they'd look as they got out towards the outside to the service layer.

Michael Feathers: Uh-hmm.

Scott Hanselman: And then in the future, we would remove the core of this onion and replace it with something better.



Michael Feathers: Yeah.

Scott Hanselman: Is that a reasonable way to do that?

Michael Feathers: Yeah, it is. It's reasonable because you're keeping the whole thing running while going and building a parallel structure that you're going to use to eventually to replace something there. It's not quite the same as the Strangler Application thing, but it is a very common strategy for large refactorings to do that sort of thing.

Scott Hanselman: Okay. Well, that's good.

Michael Feathers: Yeah.

Scott Hanselman: I was only going partially through your book at the time...

Michael Feathers: Yeah.

Scott Hanselman: So I hadn't completed it.

Michael Feathers: Yeah, I know. But yeah, that's a decent stragedy for this sort of thing.

Scott Hanselman: Okay. Well, that's comforting.

Michael Feathers: It is a weird thing though if you've noticed that for very old applications, you know, I feel like for some of them you often have multiple ways of doing things, they just end up co-existing.

Scott Hanselman: Yeah.

Michael Feathers: It's kind of like you look around and you can say, oh, man invented fire here, and it's like, oh, man discovered wheel here and you're finding tire history of 15 years of software development stuck in one codebase...

Scott Hanselman: Could you enumerate the different techniques for working with Legacy Code that you're like friendly mnemonics for something like that?

Michael Feathers: Not really. I mean, one of them is just like a modification of the Test Driven Development cycle. I mean, a typical thing with Test Driven Development is just to say -- you know, first thing you do is you write a filling test case and then you make it compile then you make it pass and refactor. I add another step to this which is basically first thing you do is you break dependencies. You want to break dependencies in order for you to go and get the test in place, in order to go and write the test and refactor underneath them.

Scott Hanselman: Yeah, that's really interesting because, it's funny, one of the things that we found in

this Legacy system we're working on was that I had a dependency, in this case, on IIS.

Michael Feathers: Yeah.

Scott Hanselman: On t h e Internet Information Server, the Windows Web Server. It was profound.

Michael Feathers: Yeah.

Scott Hanselman: So we decided to spend the first day mocking away IIS basically making a test harness to lie to this old system, and once we did that we realize that now we can start writing tests.

Michael Feathers: Yeah and it's great. The thing that you realized after doing this a lot is kind of like most applications are really glued together, it's like they have hard coded dependencies upon singletons and upon third party APIs and all these other stuff that you can't really mock out and you really need to be able to mock those these things out in order to go and test some things in isolation.

Scott Hanselman: Yeah.

Michael Feathers: You know, it's just really painful to have to go and have IIS running and the database running and all these other stuff and build your own little world just to run some simple tests of things you want to do and you never really get the benefit of quick feedback from the testing that you really need.

Scott Hanselman: Right, right. Is it a good idea to write lots of tests around the Legacy system so that you can make sure that it works the way you think it works?

Michael Feathers: Yeah. I mean, this is a first step, is to try to write tests around stuff that describe, and I called this characterization testing, you want to characterize behaviors so that you can refactor underneath, you can add new functionality and find out whether it's compatible with your functionality. The thing I would like to go and underscore though is that you shouldn't really approach this and say why, why do I test for everything, because for any large application you could spend your entire career going and writing tests for all these stuff.

Scott Hanselman: True.

Michael Feathers: What you want to be able to do is develop the skill to go and do this on a spot basis. You want to be able to go and say, "Okay, look, I want to change code here so where do I have to write my test in order to go and get a bit of coverage just to go and kind of understand as I'm working on this code, as I'm trying to make it better, as I'm trying to add features to it, that I'm not breaking anything."



Scott Hanselman: Interesting. We ended up calling this fear-based testing.

Michael Feathers: That's good.

Scott Hanselman: We write tests around those sections that we're most afraid of.

Michael Feathers: Yeah.

Scott Hanselman: It's like, okay, I'm going in. All right, well, let's hang on. If I'm going to get your back, I'm going to need tests in order to feel I'm not afraid.

Michael Feathers: Yeah. It's good and I think there's a level beyond that too which is kind of cool. It's like once you're not scared anymore and you have the support to go to get test in place, you've broken dependencies. The step beyond that is you're going to have curiosity-based testing and people are quite often, you know, they often ask the questions like how many tests do I need and I really don't like to rely on coverage numbers all that much. I think coverage is a decent metric but what I really want is the situation where people can say, "Look, I don't quite understand how this whole piece of code works," and instead of going and trying to hunt through and read all through the code and try to figure out a mental model about how it works, what they can do is just write a test and they can write that test to go and say, "Oh, okay. Now I've seen it. I apply these inputs" and he does this for me and the cool thing with the tests though as those tests remain in the codebase. You can look at these tests and use them to go and really understand how pieces work intimately.

Scott Hanselman: Right.

Michael Feathers: It makes everything feedback-driven. It makes everything just exploratory and really much more pleasant.

Scott Hanselman: It's funny there's so many different words that you could put in front of hyphen-based testing.

Michael Feathers: Yeah.

Scott Hanselman: You know, comfort-based testing. From my point of view, anything that you can do that makes it easier for you to sleep at night is a good thing.

Michael Feathers: Yes, definitely. Yeah, and I think at the end of the day with all these, this is all about making developers' lives easier and the cool thing is you make developers' lives easier and it's good for the business also because you can end up having faster-feature development and more maintainable systems.

Scott Hanselman: Exactly. So lots of people have legacy issues, but at the same time you've got these systems that you want to maintain but you need or you want to get involve in the new stuff, and I've got this archetype that I've used a number of times on the show and I think I have a buddy who lives in this area of the country who always laughs when I use this. I've got this guy, I call him the chief architect of the Nebraska Department of Forestry, and it was a silly archetype that I threw out a while back but I'm realizing it's more interesting than I thought because here's a guy who's got an important job if a small job, but he's got Legacy systems that people are counting on. He matters, his system matters but he is an alpha geek, he wants to get involved in the new stuff, maybe he is running access databases from 20 years ago but he is smart enough to understand what he is missing. How can he use some of these techniques to make sure that he is meeting his jobs' needs and his constituencies' needs while still getting involve in the new technology?

Michael Feathers: Yeah, it's a great question and I think it comes back to something which is kind of interesting that I asked every once in a while when I visit teams. It's like, "When was the last time you wrote a new class?" It's funny because in some places it's like people are not really writing new classes all that often. They can just go and they add codes into existing methods, they add methods on existing classes because they kind of feel like they're maintenance and the design is over, but I think it's one of the worst mistakes you can make in software development, it's to feel that the design is over. It's really a continual process and even in the older systems you should be creating new functions or creating new classes.

Scott Hanselman: Is that a rule of thumb, don't stop?

Michael Feathers: Yeah. Don't stop designing. I mean, design is a continual process of examination and reexamination and evaluation and change.

Scott Hanselman: What if I've got some systems in VB 6.0, shall I still be developing in VB 6.0 and...?

Michael Feathers: Well, you've got to make a change, you know, make a decision. You may want to go and move to VB.NET or move to C# or something along those lines, but even if you stay in VB 6.0 you've got to recognize that all the choices that you're making are really design decisions and that you should try to go and use good principles and make good evaluations and do good things.

Scott Hanselman: It's interesting though. I'm thinking about, again with fear, you know, I don't want to mess it up. I don't want to break it.

Michael Feathers: Yeah.



Scott Hanselman: It's one of those things where if you haven't got into the build/deploy cycle...

Michael Feathers: Yeah.

Scott Hanselman: if you've fallen out of it even for six months, even a three-month old system, I may be afraid to deploy it for the sense that I might, I don't know, make the gods angry.

Michael Feathers: Yeah. Well, you know, one thing to point out though is just like if you're scared in making changes in a big long method, you know, it's really is a very scary thing.

Scott Hanselman: Uh-hmm.

Michael Feathers: Some changes that you make can be articulated as new methods. You can go ahead and say, "Look, let me create a new method. Let me go and delegate from the big, old method to this new method," and what's cool about that is you have a new context. This is a completely different new context. You can pass variables in and write code that just deals with those particular parameters and do it using Test Driven Development which is a great thing for your architect to do. I mean, you want to get into something which is new. Well, TDD has been around for quite a while but for many people it's new.

Scott Hanselman: Right.

Michael Feathers: Practicing Test Driven Development is a wonderful practice and you can build up new things in the context of old systems and really if you can delegate out to something else and build it up as a new structure, that's a very valuable thing to do in the context of the old code.

Scott Hanselman: Now I'd be interested if there are people who are listening who are maybe using old systems, maybe the older version of FoxPro, older version of VB, whoever written test harnesses, like is there a VB 6.0 test system, the sixth unit?

Michael Feathers: Yeah, I hope it's still there but there's a XP programming website by Ron Jeffries that contains a list of all of the variations of the xUnit testing framework, what we called xUnit, there's actually a brand xUnit now...

Scott Hanselman: Right.

Michael Feathers: But like everything which is basically adapting the same architecture that JUnit did and it was one of the very first early testing frameworks. There's CVP Unit, there's NUnit, there's FUnit for Fortran, every language you can imagine.

Scott Hanselman: Really?

Michael Feathers: Yes.

Scott Hanselman: Rock on.

Michael Feathers: They are really are so it's just a matter of doing a little bit of digging...

Scott Hanselman: Yeah.

Michael Feathers: But chances are the language you're working in, there's a HUNit for Haskell.

Scott Hanselman: It's funny, I date myself by coming back to VB 6.0. I started in C and C++...

Michael Feathers: Yeah.

Scott Hanselman: But I didn't think about unit testing back then.

Michael Feathers: Many people didn't.

Scott Hanselman: This is like, you know, '89, '90, in that area, and then VB 3.0 came along but for me the fat part of the bell curve was VB 6.0.

Michael Feathers: Yeah.

Scott Hanselman: That's the bell curve of my career so I think about it and that's where I really got a lot done.

Michael Feathers: Uh-hmm.

Scott Hanselman: So a lot of really crappy stuff and God help the people who have those things that I wrote.

Michael Feathers: Yeah.

Scott Hanselman: It's interesting to go back and instrument that code and get the same feeling that I get with unit testing today...

Michael Feathers: Yeah.

Scott Hanselman: And the feeling of productivity I had at that time and combine those things.

Michael Feathers: Ah, it's very possible. I know teams who are working Delphi and they're basically doing unit testing, I mean the early Delphi, as opposed to the later .NET stuff.

Scott Hanselman: I was so productive in early days.

Michael Feathers: Yeah.

Scott Hanselman: Lots of great stuff.



Michael Feathers: Yeah. It's a cool language. It's bit different, a bit off the mainstream because it's a Pascal derivative, but yeah.

Scott Hanselman: Where do you find yourself writing code, in all systems? I mean, you consult, right?

Michael Feathers: I am a consultant, then I go around to many different teams. The thing that I find rather interesting is I'm called in quite often for C and C++ teams and the main reason, I think, is because the problems are harder and particularly in C++, it's like the work that it takes you on break dependencies to get tests in place is really kind of tough. The language wasn't really engineered for that sort of thing.

Scott Hanselman: Yeah, it's really interesting. The system I was working on last week was a C++ system.

Michael Feathers: Yeah.

Scott Hanselman: And we had situations where they had a specific compiler and a specific version of a compiler and the exports were in a certain way, there were memory mono things, they've replaced the memory manager. You can't just go and say, "Compile it."

Michael Feathers: Yeah.

Scott Hanselman: You had to set up the build environment and that in itself was a task.

Michael Feathers: Yeah, it's really tough and I think the thing that I really hope for most teams that are out there now that are working with current technologies, it's what they do is try to maintain the upgrade chain, try to upgrade your language periodically, and try to upgrade your libraries, it's terrible when you visit teams that are like three or four versions behind because the more versions you fall behind, the harder it is to go and catch up again and you really just pull yourself out of the path for a lot of goodness.

Scott Hanselman: Right and you're not necessarily upgrading just because it's new and shiny.

Michael Feathers: No, but just basically to keep yourself in the path of new tools and new technologies and stuff like that. It's not the worse thing in the world if you go and fall completely behind on some of these things. You can still do some cool stuff, but it's just so simple. I mean, it's a great discipline to go and keep upgrading because you deal with different change cases and basically increases robust systems for your code.

Scott Hanselman: Yeah. Well, it looks like we've got to get going because we've got to go to another session right now here at Norwegian Developers Conference.

Michael Feathers: Yeah.

Scott Hanselman: I really appreciate your sitting down and chatting with me.

Michael Feathers: Well, thanks a lot. It's been great. Very good conversation.

Scott Hanselman: Check out Michael Feathers' book, Working Effectively with Legacy Code.

Michael Feathers: Uh-hmm.

Scott Hanselman: And this has been another episode of Hanselminutes. I'll see you again next week.