



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 #150

February 18, 2009

Uncle Bob Martin: SOLID, this time with feeling.

Uncle Bob Martin responds to the hullabaloo around the SOLID principles from Show 145, his time on the Jeff Atwood and Joel Spolsky StackOverflow podcast, and offers his reasoned response. Is it time for a Software Apprenticeship Program? Other possible titles for this show: "He's back and he's pissed." "Bob's your Uncle." "Joel Who?" "SOLID State" "I got your tests right here!" "Smack Overflow" "Pay Attention This Time: Bob Martin on SOLID" (No, Bob's not pissed. We're just having a laugh.)

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



Our Sponsors



deliver more than expected

<http://www.telerik.com>



<http://www.nsoftware.com>



<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 #150, recorded live Saturday, February 14, 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 with Uncle Bob Martin and gets his response to the fluster over show #145.

Scott Hanselman: Hi, this is Scott Hanselman and this is another episode of Hanselminutes. A little bit of an unusual show this week with Uncle Bob back on the show from just a couple of weeks ago. How are you, sir?

Bob Martin: I'm just fine, thank you.

Scott Hanselman: We had some interesting response from the show on SOLID Principles. I thought it was a pretty straightforward show and I didn't think it was particularly controversial but there were a number of interesting developments since that show.

Bob Martin: Yeah, there has been quite a fuss got raised.

Scott Hanselman: Why was that controversial?

Bob Martin: I guess one of the followers of your show is Joel Spolsky and he was listening to it at some point and I think he heard us talking about the SOLID Principle and it somehow triggered something in his mind that didn't resonate particularly well because he went on StackOverflow podcast #38, for those who have that, and made some rather startling statements, which by the way he has apologized for, he is a gentleman, he was very respectful about it, but he said at that time that, well, my goodness, these principles sound awfully bureaucratic and frankly only somebody who doesn't code a lot could think something like that, and you could imagine that I didn't take that particularly well. That was something that pushed my button. So after that, there was a blog war and Twitter was all the fuss about, all the stuff that was going on and the blogosphere was quite activated, and so I contacted Joel and we did a StackOverflow #41 together and that was a lot of fun, we had a great time with that.

Scott Hanselman: Okay. So it's all genial but I noticed in your comments they were widely varied. There were some people that were saying that you, sir, was out of touch and were not being practical, there were people on the other side that were saying that the StackOverflow website was slapped together with no principles at all. I give a lot of respect to

people who -- basically I like to use success as a metric so I have no idea what the code for StackOverflow looks like, but he did get it live, we'll see if it's maintainable or not. How do you reconcile this different and very polarized opinions on each side?

Bob Martin: Oh, I didn't know. Did you watch the election?

Scott Hanselman: I'm surprised that there's controversy at all. I mean, Principles are not commandments.

Bob Martin: No. Principles of course aren't commandments and I think Joel kind of took it from the point of view that, oh my God, people listening to this might actually think that they have to do this all the time, and that's certainly not the case, I mean these principles aren't logistic; they are guidelines, they are an attempt to categorize things about code that in general are good, but it would be thoughtful of someone to take those principles and just practiced them by rote without giving any thought to them and I really think that was the thought that Joel was following. You know, you can't just follow the principles although it did annoy me, that thought that I had many code, but again he apologized for that so I think we're okay. And you're right, I mean the StackOverflow site, I haven't seen the code for the site and I was very taken aback by that statement that quality just don't matter but he explained that well, and well, he understands that there are just quality in those and so forth. The point is that he recently moved that site, moved all the servers and the exercise went particularly well which implies that there's some high degree of quality involved with the engineering of that site so I don't actually think that he slap the action code together in a kind of hackers paradigm. I think that StackOverflow has probably done very well, I'm sure that FogBugz has done very well. Also, I think these guys really do feel that quality is important, but in their statement they capture the mindset which somehow rubs me the wrong way and that mindset is that the most important thing is to get done. I'm not sure they believe that though, but I think the most important thing is to get done well. I actually think that getting done well is faster than getting done to find any fault. I mean the best way to get that is to make sure that the code that you produce is the best possible code, that the design code is really the best as it can be, that he has put the thought into it, that really matters and in that way you will get done faster and then if you got it done, if you rush, I think that's something that software developers really need to internalize because we are often caught by the need to move quickly and so we will make a mess in the name of moving quickly without realizing that the mess is the thing that makes us late.

Scott Hanselman: Well, there's always that sense of I'll get back to that, I'll get back to that and even 20



years in I'm still --that's such an attractive concept that I'll just, you know, to do and then I'll be right back. I never seem to come back though.

Bob Martin: And you know, this is an ancient problem, people have been rushing through things probably since people work so we have all the attitudes that say slow and steady wins the race, instead of just dashing as fast as you possibly can. So ancient history will tell us that there's something to be said for taking your time and doing things well. Another rule is that anything that's worth doing is worth being well. There's a reason for those old things. People got burned by taking the shortcut and then come back and say, well, you know, it probably would have been better if you go light instead of rushing out things.

Scott Hanselman: There was an interesting blog post by one of my team members named Rob Conery in response to this whole thing, and before Rob was a geek, he was a geophysicist and he kind of talk about how he went from being an apprentice to being a journey man and kind of move his way up, and there was a really excellent comment within his larger post about another apprentice who went through the process and he spent eight years as an electrician and had some very interesting anecdotes about how he avoided basically getting killed by listening to his mentor in the formality of this apprentice journeyman process. Well, pretty much anybody can jump out of school and whip out code. Is there a direct analogy between that? I mean, I know that there's an excellent book called The Software Architect's Profession that draws in absolutely direct allegory between software architecture and building architecture and people find using building codes and things like that a very attractive parallel story to tell. Is it that simple?

Bob Martin: Well, I think simple is a very loaded word in this case. I do think, however, that we have a big problem which is that people do come out of school and they turned into a room and expected to be able to write really great code and there's very little of the mentorship or apprenticeship or guidance given. Where it is given has proven to work extremely well. If you have a group of young developers or students and you put them under the care of a senior mentor, a senior programmer and they program together and they work together, those students will learn tremendously and become much better software developer and software engineers because of it. We see this all through various kinds of industries, a doctor practices, a lawyer practices, a martial artist practices. We have not yet internalize it as part of our profession but I think it's something that will have to happen. Did you read Pete McBreen's book from, oh jeez, seven or eight years ago called Software Craftsmanship where he proposed the whole apprenticeship journeyman model?

Scott Hanselman: Yes and that model is something I'm having trouble getting my mind around because if you kind of draw a parallel, again let's just say somebody, and again I'm do not mean to impugn people who didn't go to college, but let's just take this to it's kind of ridiculous conclusion. Guy drops out of high school, decides to become an electrician has use the apprenticeship program, doesn't talk to the union, decides to just start putting wires in houses. Eventually, he'll get caught up with it. Same things apply with lawyers and doctors. I can't just decide that I'm going to be a doctor and then when another doctor with 20 years of experience says, hey, you didn't go through a residency, you didn't get your PhD, the young doctor doesn't get to go and say, hey, says you, I'm awesome, I have a natural talent, school is not for me, man. That same story works in the software industry except it's allowed.

Bob Martin: Well, it is allowed. We haven't really internalize the idea that software development skills are acquired over a long, long period of time and part of the reasons of that I think is that writing your first program is just so easy. For some of us, some people are just born with the skill where they can string logical statements together and can write programs that do things, but they don't recognize that it takes a good long time to figure out what a good structure for software is and how to subdivide systems into module and how to manage the dependencies between those logic. There's a lot of skill involve there and there is a lot of skill involve with the negotiation with customers and the backing of requirements and the writing of test, there's a huge body of knowledge and a set of skills that needs to be acquired. College doesn't even begin doing such stuff and in fact much of the graduates are just barely able to survive in a corporate environment and certainly should not be thrown into the fire of building team systems without appropriate guidance.

Scott Hanselman: Yeah, I agree with that. I taught, I was a professor in a State school for a while and I taught the 400 level C# class and I was pretty surprised, regardless of the age of the students, I was pretty surprised that they had really no background in much of anything other than the syntax of the language and then a couple of theory classes on operating system design and, you know, they practice MINIX but when it got to the time to start doing TDD and thinking about software holistically, it was not within their make-up. The entire scholastic system had failed them, and they sit to stay low.

Bob Martin: You know, you can say that it has failed and I'm thinking in many cases scholastic system does fail in software development in particular, but on the other hand how is a college suppose to give the requisite experience to a set of students. This experience, you need to be a really good software developer, it's not something you can acquire over a curriculum, a four-year curriculum,



especially if you're taking other courses at the same time. The knowledge to become a good software developer involves holding in your hand the fate of the project or the fate of the company, dealing with decisions that are going to be very consequential to the project, to the company. We can't even get that kind of experiences though. Words get in a large team coordinating huge software efforts or feeling the problems of deploying a system and then watching as it crashes and burns and all the field service people go crazy and they're coming to you asking what the hell just happened, and you've got to solve it fast. Those kinds of things exclude that approach.

Scott Hanselman: Hi, this is Scott coming at you from another place and time. Are you looking for an Object Relational Mapping Tool for mission critical projects using LINQ and .NET? I want to share with you GNOME, it's specifically designed for developing .NET Enterprise Applications. GNOME is a mature LINQ integrated ORM tool. It has been employed in numerous large scale projects in the last six years. GNOME was created for the .NET platform as oppose to being a port from Java and its derived from platform innovations since .NET 1.0. GNOME has LINQ since its CTP release in May of 2006. It offers several unique features such as encapsulation and reuse of LINQ Query and Expressions. You can really and fully harness the power of LINQ while benefiting from your database platform's unique features, compose complex LINQ Query, decompose the Query logic in your domain model. LINQ supports all the major database platforms you find in enterprise environments like SQL Server and also Oracle and IBM DB2. You can find out more about how GNOME integrates tightly with Visual Studio and what tools GNOME offers to reduce irrelevant time at <http://tinyurl.com/trygnome> where you can also download a free and fully functional trial version. I hope you enjoy it.

Are we missing a sense of stewardship, of respect for the power that software engineering kind of provides us?

Bob Martin: Oh certainly I think we are, and so the analogy I draw very often is the analogy of doctors. You would not take someone out of medical school and have them operate on appendix right off the bat. They better watch somebody do it, they better observe someone do it, they should probably do that kind of assistance several times they should see what could go wrong, they should see the demeanor of the profession and how you respond to issues and crises, they should see the profits in the way the people work in the OR and eventually they should pick up the skill but not right off the bat. So for the software developer, it should be the same thing. You come into a team where the team is supporting a large and significant project and no way should the person be given a module and responsibility for a module that's critical for the operation of that project.

Every line of code that the new person writes should be looked at by the more mature members of the team and guidance should be provided. In my world, fair programming should be done step by step, you should walk through with the new people, how to deal with system, and how to debug it and how to run the test, and where to go to get this information and who to talk to to get that information so that they are gradually walking and getting the expertise to actually work in the team environment project.

Scott Hanselman: Well, I have two questions then. My first one is that when dealt with talented young people, and when I say young I don't necessarily mean age, I mean young in the career, they might be 55 and have three years into the business, so when I deal with talented programmers I get into what I call says you architecture discussions where one who believes that they know, myself or someone who is a fairly senior guy says something and the other guy is like nah, says you, I don't think so. It seems like the discussion between you and Joel and Jeff, all people who have been doing this for a nontrivial amount of time, they kind of fell into ah, says you kind of a situation. Well, with more concrete things like medicine and being an electrician, it seems like there's a little bit less of that. Are we just being elite as old dudes? Are we just trying to hold the kids down?

Bob Martin: There is something to be said for young kids going off and burning themselves in a lot of critical environment, but I don't think we're just elite as old dude kind of holding kids like that.

Scott Hanselman: Well, I mean I worked with a company and a young man wanted to know why he wasn't a senior developer, and I asked him how long he'd worked there, he'd worked there three years.

Bob Martin: Ah okay, he is senior.

Scott Hanselman: And that's a very common story. I mean, think about in software engineering, because the speed of the iteration of the technology, if you've got five years in any technology, you're very good or you've at least been slapping the keyboard for awhile. People don't have the patience to wait through an apprenticeship.

Bob Martin: Somebody told me once and this was a very long time ago, that the breakpoint of being experienced where you can use the word experience with authority is five years. I think that's probably true. If you have been in an industrial or a professional environment for five years, you can then say that you're experienced. Prior to that, not really very experienced. You may have some experiences but that you probably have not faced enough issues to really be effectively on your own efficiently. Now certain people get through it faster than that and some people never get through it, but I think that's



probably a good average. Our industry does a disservice to people because there's no consistent dual ladder. In a lot of cases, the career path was a couple of years swinging code, a couple of years leading a team and after that I'm a manager and once you're a manager you never write code again. I think this is dead wrong, I think it's the wrong approach for our industry altogether. If you are really serious about your profession, you're going to be explaining codes for 30, 40 years. Surgeons stay surgeons, lawyers stay lawyers. Some of them go up in the management track, but for most of them, they keep their craft.

Scott Hanselman: But we burn out.

Bob Martin: I don't.

Scott Hanselman: I mean, I don't know about you but I mean this is year 17 for me, my hands hurt.

Bob Martin: Yeah. You mean carpal tunnel?

Scott Hanselman: Well, I don't know if it's carpal tunnel but I'm just saying that surgeons have the same problem. It's just that slapping the keyboard for 20 years is painful and there's a certain amount of kind of self-objectivity. One has to develop otherwise they can become the old dude with the beard who thinks that everything needs to be done the way it was done in the 1970s. Well, maybe there's a new technique or some new revolution that's happening that I'm just not seeing because of I'm letting my years of experience blind me.

Bob Martin: Well, then that's clearly initiative and certainly that does happen. Of course a professional doesn't do that.

Scott Hanselman: It's a very interesting point. You're pulling out the word professional and kind of capitalizing it verbally. The difference between a software professional and maybe just a code slinger might be what we're talking about here.

Bob Martin: Well, sure. You know, someone who is dedicated to his craft is going to stay up with all the latest technology to the best of his ability. So you know, guys my age are still learning Ruby and still learning Scala and F# and all these stuff that are coming out now and try to get their arms around all these new technologies because it's important, you have to stay with it, and you know you can imagine a doctor, a doctor better be keeping up with the latest technology, better be learning the newest technique otherwise their patients are going to go somewhere else.

Scott Hanselman: Well, ultimately they have to care and I think that's the number one thing when I want to pick who I'm going to work with, I want

someone to care and that's the difference between a craftsman and someone who is just not, I mean I think the same thing applies with woodworking.

Bob Martin: Yes.

Scott Hanselman: If someone just wakes up excited about what they're going to do that day, what project they're working on, I think they're more likely to be a craftsman than not.

Bob Martin: Well, and that's certainly in my case. I've been writing codes for over 40 years but I wake up in the morning and think okay, I got to get this feature in the kit now and I'm just dying to get my hands on the keyboard so that I can add more functionality and more code and see that system grow into something better than it is now.

Scott Hanselman: Okay. So let's take what we've talked about and try to apply it back to the situation with Joel and with Jeff. I mean, Joel has a successful and non-trivially large company so we can definitely point at him and say there is a person who has built a successful software company. We may not agree with some of the decisions he has made technically. Jeff appears to be in the direction of a successful software start-up. He's put together something and is definitely a game changer of some kind. Again, in both cases we don't know what techniques and what code Joel used nor do we really know what Jeff use but if they have used some of the principles that you're kind of espousing but they have measured business success, who are we to criticize.

Bob Martin: I wouldn't criticize them at all as long as they're maintaining that success. I was concern by just statement, the quality that will matter all that much. I'm still actually concern about that on a mark. I'm quite sure I can get my arms around what is just an occasion for a quip. I honestly don't think he believes it that way because he is developing a system that seems to be very high quality. I've looked around StackOverflow, it's pretty cool.

Scott Hanselman: Well, maybe that's just talk show banter between two guys like kind of shooting the poop.

Bob Martin: Yeah. Well, I think that may certainly do beat the StackOverflow podcast that way, and it's two guys in a bar just sitting and you happen to have a camera on them and says they don't even know there are microphones. There's a risk in there because bar and banter can get a little far field which happens in...

Scott Hanselman: Well, certainly making a small kind of political parallel, almost all of the political talk shows get into trouble when it goes from being analytical to being just kind of let's just chat and the banter kind of builds up and it builds up and builds up



and then somebody says something stupid and they apologize for it the next day.

Bob Martin: Yeah. Although, you know, at those cost, interesting things happen so we won't be going to be doing this, right.

Scott Hanselman: That is true, although my favorite Quentin Tarantino quote is that he who is most likely to make declarative statements is mostly likely to be called a fool in retrospect, so I don't know.

Bob Martin: Yeah, well...

Scott Hanselman: One does in this show equivocate too much but at the same time it seems like the SOLID Principles, while not right for every single moment of my life, seem to be a fairly straightforward kind of thing that I want to laminate and put on the wall along with the series of other principles that I've learned over the years.

Bob Martin: Did you save the motivational posters?

Scott Hanselman: Well, but at the same time Jeff made a kind of -- I think it was kind of a link-bait inflammatory post about Farengi programmers...

Bob Martin: Oh yeah.

Scott Hanselman: Making a reference to the Farengis of Star Trek who used to have hundreds and hundreds of rules that would guide their behavior. I'm not quite sure how he took the five rules of SOLID and a couple of sub-rules and turn it into the 285 rules of acquisition. I thought that was kind of a yelling fire in a crowded theatre.

Bob Martin: Yes, it was an interesting post. I read that one in particular and at least he made the point that to be a painter, you don't read the rules on the paint can, those rules on the paint can, they'll teach you how to be a good painter which I think is true. The rules on the paint can don't give you any talent, on the other hand, I think all good painters know the rules on the paint can.

Scott Hanselman: Exactly, I mean my mother was a house painter and the very first time I helped to paint the house we read the rules on the paint can. It doesn't mean you need to be obsessed about them in 40 years but it does mean that you need to know them. Day 1, step 1, read the rules in the paint can.

Bob Martin: Did you see the -- I don't know if this happened because of the podcast but someone had done motivational posters for the known SOLID Principles. Oh yeah, they're pretty good. All of the SOLID Principles are there and they're in the form of, you know, those funny pictures hanging on the wall of people climbing rocks and eagles flying.

Scott Hanselman: Right, right, right, or like despair.com has those.

Bob Martin: It motivates people to do a good job.

Scott Hanselman: Here it is. It's Derick Bailey at Los Techies, lostechies.com. Nice.

Bob Martin: That's a good one, yeah.

Scott Hanselman: Nice. "SOLID Software development is not a Jenga game," that's a fatal Jenga here. Wow. The Single Responsibility Principle with a pretty intense looking Swiss army knife that is wider than it is long. Then there's the Open Closed Principle with "open chest surgery is not needed when putting on a coat." These are really good. I need to make T-shirts of these. "Would you solder a lamp directly to the electrical wiring in the wall?" Dependency Inversion Principle. That is absolute brilliance. I'm going to link to that from the show's site. Absolutely brilliant.

Bob Martin: I thought this was pretty good stuff.

Scott Hanselman: Yeah, yeah.

Bob Martin: I'm going to hang these on my wall some place.

Scott Hanselman: Oh yeah, he says he's going to do high res versions. I think that it would be a crime if he did not. You should put this on your next book or make licensed, what are they called, postcards.

Bob Martin: Oh yes, yes.

Scott Hanselman: Brilliant, brilliant, brilliant.

Bob Martin: Oh, wow.

Scott Hanselman: Well, maybe it was just a tempest in a teacup then.

Bob Martin: I think so, yes, and I doubt if it would have really gotten to the level that it did if Joel hadn't said some guy who hasn't written a lot of code. That was first my book.

Scott Hanselman: Yeah. All right, well, I feel like we've un-pushed them a little bit today then.

Bob Martin: Oh yeah, yeah, yeah, yeah and the whole episode I think turned out very well and quite pleased that Joel and Jeff and I were able to -- I had a very respectful and fun conversation.



Scott Hanselman: Well, turning dramatically to a different direction, you said there's a name about deployment you just want to talk about.

Bob Martin: Well, one of the issues that came up in effect over Club 41, Joel was asking me why would you do a Single Responsibility Principle. What is so great about that? And I said to him, well, you don't want to have modules which are attractors for many dependencies. So Single Responsibility Principle says that modules changed for only one reason. Push it away, you push out of the thing to change it differently. You gather into a change for the same reason. He says why would you want to do that? Well, if you don't do that, you have all of these dependencies impinging upon this one module and if you change that module then the other module have to change, and he asked me this question what do you mean change, change at runtime? Then I said no, change at compile time. If the source code changes the feature change, then if you have a module that attracts many different dependencies, then many different kinds of feature change will force you to modify that module and that's a problem because that module will crash. It will have to get rebuilt all the time, you will have to redeploy all the time. And he said, well, what's the matter with that? I said, well, if you're redeploying it all the time, then you just got to re-release and retest it and then if it's in its own separate JAR file; that means you've got to redeploy the JAR file. I probably should have said DLL in this case.

Scott Hanselman: Sure, sure.

Bob Martin: And he said, well, when I ship a project I didn't bundle all the stuff together and ship as one big package. I said, well, yeah, that's true. In many cases, we will do that. In fact, I do that, I just take the whole thing instead of putting it in one package but that doesn't mean I don't want to be able to independently deploy the individual pieces while I'm developing it. So for example if I had a team or an organization of say five teams, I would like those teams to be able to work independently of each other so I would like them to have the dependencies between them, the dependencies between the modules that they work on very tightly controlled so that each team can deploy its own stuff very independently, and everybody else, and then incidentally at the end I can package the whole thing off and then ship it, but the whole development environment has been made much, much easier because these guys can independently deploy their own particular components. Even when we can bundle it all together into a single unit and deploy it, it's much nicer if we can deploy the individual bits within our environment instead of having to rebuild the whole thing and recompile the whole thing and reset the whole thing every single time. It also provides for a good plugin structure so you can create a base module with a whole bunch of plugins and those

plugins can be independently deployed from the base structure. They were seeing a lot of plugin applications that enjoy this kind of Single Responsibility Principle benefit.

Scott Hanselman: Well, when that gets larger, when it does all of the things that you described, you're looking at using all of the principles in tandem, together.

Bob Martin: Yes. Well, if you're following those principles, what you're really doing is you're managing the kind of -- all of these principles are about the management of dependencies between...

Scott Hanselman: And did you all find this controversial or did this help answer his question?

Bob Martin: No, now he is trying to roll off his back and he went on to the next point right away and I want to bring it up here because I thought it really hadn't been concluded.

Scott Hanselman: Okay, interesting. Someone on twitter, earlier I mentioned we were doing the show and I've heard this from a loud minority but if we spent as much time discussing these principles as we did coding... It seems people think that folks are jawing too much about these principles, when they need to be, as I've said before, slapping the keyboard. That makes me feel like the stuff we're talking about at the beginning where I see Jeff is the expert about this. Jeff would have actually asked the question is it more fun to talk about software than it is to write software.

Bob Martin: Oh no. Not for me anyway.

Scott Hanselman: Not for you, not in my life. I better let you get back to writing software then. All right, well, Uncle Bob Martin, thanks so much for taking the time to sit down and chat with us here again on Hanselminutes.

Bob Martin: Oh, it's my pleasure.

Scott Hanselman: And you can follow Uncle Bob on Twitter and I think the twitter name is Uncle Bob Martin, is that correct?

Bob Martin: That is correct, yeah.

Scott Hanselman: All right, this has been another episode of Hanselminutes and I'll see you again next week.