The biggest challenge is professionalism.

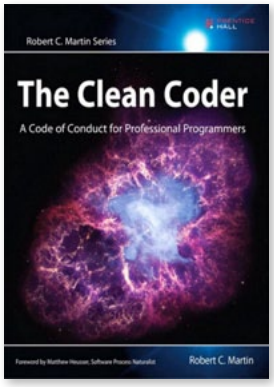
Der US-amerikanische Softwareentwickler Robert C. Martin, auch bekannt als „Uncle Bob“, arbeitet seit den 1970er Jahren in diversen Softwareentwicklungsprojekten, seit 1990 als international anerkannter IT-Berater. 2001 initiierte er die Entwicklung des Agilen Manifests, das Fundament agiler Softwareentwicklung und ist führendes Mitglied der Software Craftsmanship Bewegung.

Im Anschluss an sein zweitägiges Seminar in Karlsruhe »Clean Code: Agile Software Craftsmanship« nahm er sich Zeit für ein Interview mit dem VKSI-Magazin. Die Fragen stellten Marc Philipp (andrena objects) und Susann Mathis (VKSI Magazin).

Craftsmanship and professionalism

Bob, thank you very much for agreeing to do this interview. You just completed a two day course about craftsmanship. How do you define software craftsmanship and what is the difference to professionalism?

Craftsmanship has to do with how well you perform your craft; professionalism is more about the relationship to others. I want to bring the notion of software professionalism and software craftsmanship together. Software craftsmanship is all about turning this job that we have into a real profession. It is about defining a set of disciplines, a minimum standard of behavior for someone who claims to be a professional software developer. And that includes how well they perform their programming, how well their code looks, and also how well they do their job in terms of estimating and communicating to managers and other software developers. Imagine that 300 years ago we were talking about the medical profession.



There would be practitioners around willing to cure your aches and pains and a few people said: no, there's got to be something to raise the level here. That's what software craftsmanship means to me: to raise the level and create a group of software developers that employers and governors can trust.

Would you call someone who writes clean code a Software craftsman? Or is there more to that than just writing clean code?

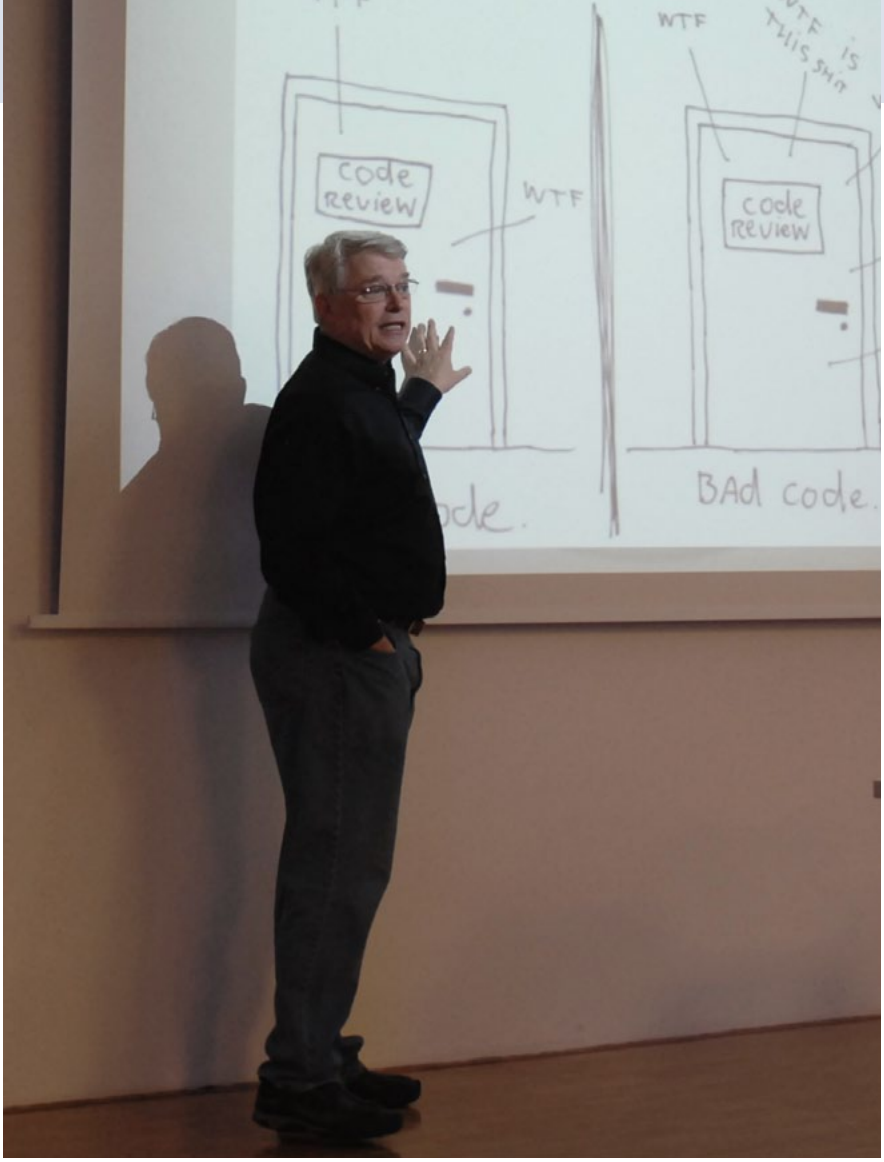
There is much more of it because our civilization is heavily depending on software. How many times per day do you put your life in the hands of some 22 year old programmer who wrote some code at three in the morning? The answer to that is: too often! If, at some point in the near future, there is going to be an accident where thousands of people will be killed, I would like that our profession could say: this is not because of some piece of software written by somebody who got a little bit careless, sloppy, messy or unprofessional — we had been doing our job well. But today I couldn't say that.

What's the most important thing a software craftsman should do?

Passion and Love

First of all: make sure that you are in the right job. There has always been a subgroup of programmers for whom software is a passion. They knew what they wanted to do maybe by the age of 16. They touched one of their father's computers or they found some interesting device, they began programming and they fell in love.

And there is a whole group of programmers who got into the profession because it was the smart thing to do: it was the profession of the future. But they never had the passion for it. Some of them, I think, can develop that passion but the others are not helpful to the profession. I think the people we need in our



Bob Martin: »It's all about turning this job into a real profession«

industry are people who find this passion early and nurture it and want to stay programmers. When you go to a doctor, you want a doctor who loves what he does. When your freedom is going to depend on, you want a lawyer who loves his profession. If you're going to commit a massive project to software developers, you want those software developers to be in love with their profession. If you are not in love with your profession, find one you are in love with. Don't be in this one.

Second thing to do is to learn like crazy. Read as much as you can. Most professions have centuries – we have just five decades. But in those five decades there have been notable professionals who have written about their job. You can read the works of people like DeMarco, Dijkstra, and a range of numerous others and understand what they thought professionalism is. Learn the languages of the past. If you are a Java programmer, learn some other languages because Java is not the end of the story and it is not the beginning of the story. It is just a little, tiny, brief segment and it's going to move on to something else relatively soon. Find other languages, learn those other languages and communicate with other programmers. That's very difficult for programmers to do, but it's important for us to be a community of practice and a community of professionals.

Career and Chicken Farmer

In the United States we face a lot of pressure to move into management. If you are a software developer for three or four years, you leverage up and you start running a group. I find that very unfortunate. We need good managers, but software developers typically don't have those skills and they don't want those skills. When they feel the pressure to move in that direction, many of them fail. Maybe they pursue some career in marketing and lose their origin passion for what they were doing. I have certainly faced that several times. I have been a programmer, then I have run a group, then I went back to be a programmer, then I ran a division, then I went back to be a programmer and I ran a company. And now I'm programmer that does a lot of speaking and talking and I am very happy with that.

Some people don't have that love because they got into software development because of the wrong reasons. Some of them develop the love because they find out it's very good to be a programmer and some of them lose the love because they find themselves in an environment that crushes that love out of them. It's an environment where they are driven by deadlines and driven by pressure and they have to work overtime and after a while they ask themselves: why did I get into this horrible situation? Then they leave and become a chicken farmer.

Do you think high-quality SW is possible without such professionalism?

Policing and high-quality

Yes, there can be software in high-quality without this attitude. But the vast majority of code is not high-quality. Unfortunately our customers don't have the ability to look under the hood. They have to trust us. And we tell them: the code is working, it's fine. If our customers ever develop the skill to look under the hood they'd →

→ be terrified. And they'd realize they'd spend a lot of money for rickety, very fragile software that has a great capacity to harm them. That needs to change and that's where professionalism comes in. I want the software developers to police the software developers. It should be the software developers who set their own standards and then keep to them. And right now that's what we are doing.

What would you respond to someone who says that tests are a waste of time and that clean code is too expensive?

Schedules and distrust

(Bob laughs very hard). As long as the code doesn't have to work I can do it as cheap as you'd like and at any schedule you want. Actually it's not the managers who say that. If you ask the CFO if he wants high-quality software that doesn't make any mistake, he will answer: Damn right I do. You don't have enough time *not* to write tests. How much debugging do you want to do? How much chasing down a horrible invoice you want to do or how much correcting corrupted databases you want to do? Of course you have time to write tests.

The people who complain most about not having time are the programmers themselves and they do that because they are afraid about that their managers will say "we don't have time to write tests, we don't have time to clean code". And they have cause to be afraid because they have been put under a lot of pressure to deliver. And they have been put under that pressure to deliver because they have not delivered the previous time because of an estimate mistake. Software developers bow to schedules where they should have said: No, I can't meet that schedule. Developers need to learn that one of the marks of professionalism is the ability to say no. There is a huge amount of distrust right now between the software management and companies and the developers themselves and that is directly caused by the lack of professionalism.

When it comes to quality many people turn to metrics to measure the quality. Do you think it is worthwhile doing? Does that help?

Metrics and QA Groups

There are some useful metrics like cyclometric complexity, lines per method and so on to measure the internal quality of software. But these metrics are not conceived for the management. These are metrics that we as developers should use to police ourselves.

Where does quality really come from? Quality comes from developers writing good software, using good requirements, writing good tests, making sure those tests pass and developing a discipline to never rush in order to meet a deadline. Your rush to meet a deadline makes that you're going to go slower, you're going to create lots of bugs; you're going to make a mess..., so you should develop a discipline never to rush. QA groups actually do have a good function but we use them in the wrong way. We take the QA group at the end of the project to test the code that the developers wrote. Why do not the developers test that code? I want QA group writing these acceptance tests but I want the developers execute those tests. So they can push a button in order to have done what they are supposed to do.

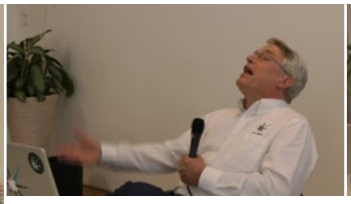
Is there a place for manual testing?

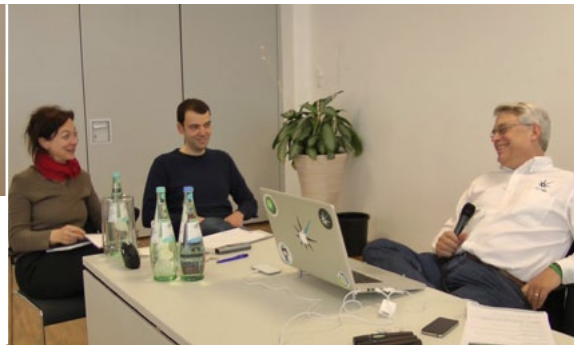
Break the system

Manual testing is very important! I want someone with their eyes on the screen and their fingers on the keyboard but I also want their brain engaged. I want them thinking about interesting and clever ways to break the system, I want them going to bed at night strategizing how they are going to destroy the system, what interesting sequence of inputs will confound and break it. I want the QA people to be applying a massive amount of creative energy to find out ways to break the system that software developers never thought of. The developers and the QA folk should be in a kind of competition: Development is absolutely certain that QA is not going to find a bug and QA is determined to find a bug. This would be a very profitable situation.

What do you think is the biggest challenge the SW is facing today?
The biggest challenge is this issue of professionalism.

The second challenge is a technical one, and it's no less daunting. Around 2000/2001 Moore's Law began to fail. The speed of computer stopped doubling and they have been stuck now at 2 ½ GHz since five or six years. To compensate for that, hardware developers have begun to change the game: They've added multiple cores. The laptop I'm working on has four cores inside it, four processors each running at 2 ½ GHz.



You should mean that there are 10 GHz in here. But that's not the way it works. The software to use multiple cores is not written yet. So we'll wind up with an inability to spread the execution of the software across those growing number of cores. Most software developers think that they live in a single machine with a single thread; they have no idea of the complexity that can run in a system of 128 cores. So the technical challenge is for the software community to learn that much more mathematical, much more scientific and much more technical skill of building software that can run in a heavily Multiprocessing environment. That will become essential. Everybody has to learn to be a functional programmer so that they can live in a multicore world. That will be a very interesting time.

Have you started writing another book already?

Clean Architecture

The next book is called Clean Architecture and Design and I am in the midst of writing it right now. It talks about the next level up from clean coding. Once you know how to write clean functions and algorithms, you want to understand how to create clean architecture. And we have a huge problem with that right now. We've got massive amounts of systems being written and the people who are writing them have no idea how to structure the architecture. They have been subverted by the marketing literature out there, those massive amounts of stuff being done with service-oriented architecture. This is no real architecture. Service-oriented architecture is an interesting technique, useful in some cases, but it's not the massive big architecture that's going to solve the world's problems.

There is a tremendous amount of architectural experimentation going on. The problem with that is that a lot of the architectural science was done 20 – 30 years ago. We know how to build these systems. Yet most of the developers out there don't know what system architecture ought to look like or what the rules of system architecture are. So this book attempts to address that — as prescriptive as possible. I am making clear that it is my opinion. That puts a different slant on this.

How does this fit with emergent architecture and this ideal of extreme programming that architecture will evolve...

Not without human brains

That's an interesting question. Architecture and design emergence exist but not without human brains. It is not the design that emerges out of nothing. It is the design that emerges out of the activity of a programmer evolving their program forward and while doing so, applying design principles that they have learned. All the design principles from the last 40 years still apply, they are all still valid. It is just the reordering, , that has changed: Instead of putting them all up front and planning some massive software thing, we apply those principles sideways, we write a little code, we write some tests and then some code and we look at the structure and understand: There is a principle validated there, let's shift it a little bit and refactor it, and then we write a little more code and we refactor for the principle and we begin to build a system that starts to look like it needs an architecture. And then we start to build up the pieces of that architecture. We evolve it. It emerges. But not without human thought. Not without all the principles that had always existed. People get into this religious war that agile stuff is about not planning. It is not. It is about planning all the time. People say: design emerges you are not supposed to use your brain. No, of course you are supposed to think! It is just that you think at a different time and in a slightly different way.

Bob, thank you very much for this interview, is there something you would like to add?

Message for all the software engineers in the region of Karlsruhe

There is a discipline you need to learn that is called Test Driven Development. It has been controversial for a number of years but the jury is in: this is something you need to do, something you need to learn and something you need to practice. Probably 10% of the developers are beginning to practice test driven development. It will become one of the disciplines that will help to define professional behavior over the next 10 years. Learn it, understand it, you will go to need it; it will probably be the most important thing to learn within the next 10 years.

http://www.objectmentor.com
twitter.com/unclebobmartin