

The Agile Heart: An Interview with Jim Coplien

Jo Stichbury

Published by the Symbian Developer Network

Version: 1.0 – September 2007

Jo Stichbury of Symbian Press recently spoke to Jim Coplien about Agile software development, architectural patterns and books:

Jo Stichbury (JCS): Some of the visitors to the Symbian Developer Network website won't necessarily be very familiar with Agile development. Please could you give us a brief summary of what it means to be Agile?

Jim Coplien (JOC): When people inquire about what "Agile" means, I always return to the source: the [Agile manifesto](#). There are so many branches of Agile, individual practices and individual methodologies that have been equated with the word Agile. Like most popular terms it ceases to have much useful meaning any more, so I like to go back to the source. The source is brilliant and there's some really good stuff there for people to think about and to get their teeth into, but it probably isn't what most people think it is.

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

People go to the Agile Manifesto and expect to be told to do something. The Agile Manifesto doesn't tell you to do anything. It is almost a set of questions or a set of concepts that cause you to think and evaluate where you are at. Agile is not a method; it's above method, and it helps guide your thinking so you are more likely to choose the things that are going to be suitable to you.

My friend Jens Østergaard calls it the Agile Heart — self-organization. That is antithetical to someone coming in and giving you a method and saying, "Follow this". And so in some sense the whole key point of Agile, in terms of being adaptable or dealing with change, is this issue of being able to make decisions for yourself and being able to organise autonomously. Most of what I do these days is to lead, teach, and provoke people into *thinking*. Critical thinking has become a rare skill.

JCS: So it's about finding your own adaptive development methodology?

JOC: Yes. And in fact, as an historical note, it was originally called adaptive development. Jim Highsmith was sitting down talking with some folks, maybe it was Alistair Cockburn, saying, "We have this new idea, we're going to call it Adaptive". But Jim already had a book out called "Adaptive Software Development", so they agreed that they needed another word to distinguish it from the book. That's where Agile came from.

JCS: Has Agile has been over-used in software development recently?

Yes, as with most broad computing concepts of the past forty years, Agile has been reduced to a synonym for "good." Here in Denmark, "Agile" in Danish refers to the concept of dogs running round obstacle courses (something that came out of the UK in the late 1970s). So it's doubly funny!

Agile is now an adjective that can be put in front of anyone's favourite term and it's supposed to make sense. I heard a long talk on Agile databases in Finland back a few months ago, and I was thinking, "How in the heck can I tie this to anything in the Agile Manifesto?" I mean it was really, really a stretch.

JCS: What to you truly defines "Agile-ity"?

JOC: The step in understanding is to appreciate that not all that is Agile is good and not all that is good is Agile. I think the tendency in software should be towards a more Agile direction, but that doesn't mean that every organisation will be comfortable with the same degree of being Agile, nor that they should try to achieve the highest degree of being Agile that they should.

Most people like to define agility as being able to embrace change. That's another favourite term. But part of the responsibility that goes with embracing change is also knowing when to ignore it, because if you blindly react to every change that's coming along, you actually add to the instability of the world around you. There has to be this balance of you taking responsibility for some project direction, along with letting yourself be led by the major business events or major changes in technology that are going to contribute to your project. But there has to be some resolve along with agility. There is a place for stability. There is a place for not dealing with change sometimes. Every project has to find its right point on the spectrum that's going to keep customers happy and keep the business stable.

JCS: At the [Smartphone Show](#) you'll be giving a presentation called "Bootstrapping an Agile Project". Can you summarise what you'll be talking about?

I'll be challenging people to lay the foundations for Agility before trying it. You can't just decide to be Agile and achieve that end with the adoption of some practices, meetings, and product backlogs. To be Agile you must first be strong, be fit, and be disciplined. There are a lot of things you have to get right to do software well, and Agile is no exception.

We have a very poor track record going right now, with about two-thirds of all Scrum projects failing and probably 95% of all XP projects being unable to fulfil all the practices and principles of XP. I'll

start by looking at what some of the common failure modes are. The presentation is about how you start out; what things you look at first. Part of that is assessing yourself before adopting any of the classic Agile practices. Answering basic questions like, “Do you have a customer? Do you have a good idea of what the scope of your enterprise is? Do you know who you are?”. There are also technical issues like, “Do you have automated configuration management? Do you have automated check-in tests?”. So there is also a technical checklist that you can go down.

JCS: How can a team get started on an Agile project? Besides giving them all copies of “Organizational Patterns of Agile Software Development” how can they learn more?

JOC: It isn't just a matter of book learning. Of course, people need to have that exposure to the techniques. There are many good books on Scrum, and also many good conference workshops and tutorials that cover most of the Scrum basics, and of course the Scrum certification courses. But a lot of it comes down to trying it out. That's hard because you need to go back home from a conference and tell the boss, “We'd like to try something really, really different here”. And the boss will say, “What's the risk?” and the employee will say, “Well, I don't know”. So it's hard to go home and do these things even incrementally. But that's really where the learning goes on, it's in trying these things out.

If management lets teams take accountability and take control, and if teams take responsibility, then you have the seeds of Agile. You need good soil to support this blossoming: the team has to be connected to the pulse of the market, there has to be a community of trust both within the development organization, between the client and the development team, and so on.

JCS: Would this work for smartphone development – for example, for Symbian and its licensees? One thing that may differentiate Symbian from some other development companies is that there are a number of stakeholders involved and typically the technology roadmaps are very long term.

JOC: In terms of laying out long-term deliverables, within some parameters that is a fact of life. Different Agile methods deal with that differently. Scrum, for example, will start with a 1 year to 18 month vision of what the major deliverables are, and that's the point at which you do your first estimation. You get the team and the product owner and the customer together and you estimate that long-term vision. Now you cut it up into chunks and you do these things in two week to four week increments. Because you are doing this in small increments you have way markers. You have some evaluations to say are we on track to meeting this vision, so you'll know sooner rather than later.

Customers benefit from seeing earlier whether they're going to have the deliverable on time, or if in fact some features will have to be dropped. It's at those points when the customer may have to be more flexible. Now, an important point here, a very important point, is that I don't think we're changing what happens. If something's late, customers still get angry because they feel that promises have been made and then broken. As with all software development, things still do come late sometimes and then you clean up afterwards. It's something the customers have to be willing to recognise as fundamental to the discipline of software development.

JCS: It's sometimes said that Agile methods work best for small projects, where the development team is completely co-located and that they don't necessarily work so well for large projects. Do you agree or can Agile be made to scale?

JOC: I think this question comes from a broad misconception that Agile is about small projects because the focus is on small teams. Actually, we all know that small teams work. Agile didn't invent that concept, and it has been a staple of successful software projects for decades and of human behaviour for millennia.

So, yes, Agile techniques may have a problem scaling, but every technique has a problem scaling. I think this scaling question is largely independent of the Agile values. Individual teams should be small to be effective - whether Agile or not. Scott Ambler reminds people that you scale Agile projects the same way you scale non-Agile projects: divide and conquer, while maintaining effective communication and tight feedback loops.

Confusion also arises in the distinction between projects and teams. Look at Diane Larsen's definition of teams: they are co-located and work within the length of a school bus (Alistair Cockburn's rule of thumb). If you don't sit within the length of one school bus and you're more than about a dozen people you're not a team – you're something else. You could be a department or a group or a something but you are not a team. And if you are distributed you are not a team. Football teams do not play on two fields at once, they are co-located. The concept of multi-site team has little relevance to me. There are, of course, multi-site projects.

The challenge remains: to create closely coupled teams whose members are geographically co-located, while ensuring that the coupling between teams—whether co-located or not—is low. That is hard, and it relates closely to software architecture. You need good architectural foundations that allow individual distributed groups to be able to work independently on different parts of the architecture. This is a very old concept - it's called Conway's Law. Conway's Law says that the structure of any IT artefact will reflect the structure of the organisation that built it. It's very important to have an architectural foundation. Agile architecture is not fragile architecture.

JCS: You said recently that the Agile manifesto leaves usability at the side of the road and is blind to the user's perspective. What can be done to change it? How would you re-state the manifesto, if at all, or is a matter of changing how developers perceive and use it?

JOC: I think we need to do both. With respect to your first question about usability, I think that could be fixed within the manifesto, maybe just by saying useable software instead of working software, or saying something about delighting the customer, rather than just doing the bare minimum that makes their program run according to some specification. That means delighting the customer with the experience of using the product, rather than just being happy that something that meets a specification arrives on their doorstep at the right price at the right time. I think that could be a relatively minor edit to the manifesto.

Now, the question is of course, what are the practices and techniques to get there? What do we give people to back that up? It is known, but it's not very much taught in universities, and those kinds of skills come from communities that are largely divorced from contemporary programming and contemporary computer science. So for example, there is a usability community. Usability people tend to be psychologists, anthropologists and people like that. They just don't talk to computer people and computer people don't talk to them. When I bring up this issue with some clients and say, "You really should have a usability person", they'll say, "Well there aren't any". Then I'll give them a list of the five or six who I could immediately recommend that they hire. There

are a lot of them around, it's just that computer people don't know who they are. And the people who have these skills don't know where to plug into our community to make that work.

It's one thing to change the manifesto, and it would be another thing to get some awareness out there, get some names and concepts and organisations in front of people to make them aware that this is a problem and that there are resources available. That's something I've been trying to do – both in my writing and in getting in front of audiences and using every opportunity just to draw people's attention to this.

Jef Raskin, who was the founder of the Macintosh project, repeatedly said, "The interface is the program". The rest is just code that makes the interface work, yet this is the one aspect of software product development that Agile misses. It is largely an internally focused movement. For something that has its roots down at the coding level, Agile has done a remarkably good job to raise awareness about customer engagement. It's just that the most important product attributes—dutiful attentiveness to usability, customer experience, and domain knowledge—have been lost.

JCS: Have you found that any of the patterns you explore in "Organization Patterns of Agile Software Development" have evolved? Have you noticed the emergence of anti-patterns?

JOC: Actually it's a little bit surprising, but I can't think of any instance where I've seen the application of the patterns go awry. I almost wish that I could think of something where I've seen some challenges, because I lose credibility if I say all my stuff works! But I honestly can't think of a place where they've gone awry.

I think the reason might be is that the main focus in the pattern work isn't in the patterns themselves but it's in the process of working your way through the patterns and the set thought process. One of the things that's explicit in the book, and that I always ask clients when I work with them, is this: If you are going through a pattern that seems to match your problem and in the middle of the pattern you come up with a solution different from the one that the pattern says you should use, which one do you use? The one you came up with or the one in the book? Of course you use the one you came up with because you understand it. Patterns are a tool for critical thinking, and not rules or methods to tell you what to do. I think when you get people thinking, and getting feedback from each other – their ideas, their wonderful brainstorming and their fantasies about things that could go wrong – great things happen.

All that said, my understanding of some the patterns in the book has evolved, so that if I were to write the book again I'd probably make changes. One of the things that is buried in the book, in *SIZE THE SCHEDULE*, is that we recommend that people keep two sets of books. You keep an internal schedule that you publish for developers that's shorter than the external schedule. That's just lying. If you set up those kinds of structures in any kind of communication environment things eventually go wrong. So this double book keeping idea in *SIZE THE SCHEDULE* doesn't hold water and I'd like to remove that one.

JCS: Are you planning to write another edition of the book?

JOC: I'm in no hurry because in some sense the changes are small and the book is still having a lot of power out there. The reviews that it's getting are very powerful; the feedback is stellar.

The book itself took about seven or eight years to do. I don't know if people know, but it started as a Wiki web with a lot of people getting their fingers into it. I think the Wiki was a major factor in putting the right communication in place to allow the book to go all the way to completion.

It was mainly Neil Harrison and I, but also Alistair Cockburn and Bruce Whitenack and a lot of other people spent time on that Wiki. I wrote a program that sucked up the contents of the Wiki, reformatted it into FrameMaker© graphics format and spat it out as a publication-ready document. The tools are all in place, it's a matter of getting the new content there on a Wiki, and again doing that in community to get to a point where we're ready for a second edition. But I'm in no hurry because it's not the kind of thing where you go to the library and quickly do some research and write a book in six months. I want to do this right and do it in community and do it with engagement and make it a fun exercise that involves a lot of people.

JCS: Your 1991 book “Advanced C++ Programming Styles and Idioms” is a classic, found on the bookshelves of anyone serious about writing elegant C++. Why do you think it has continued to be relevant?

JOC: I really think it's because of the book's focus on fundamental questions of design instead of just details of syntax and semantics and, in particular, of how language can help express design. You can see that in the title of the book. Instead of just being about language, syntax and semantics, it's about idioms. Idioms are the things that come out of experience and maybe an informed and powerful use of the language, rather than just putting things together end-to-end.

The ability of a language to express design intent is called “intentionality.” It's an important foundation of the most crucial aspects of software engineering: program evolution and repair. You can understand and evolve a program that you can understand; you can understand it if it speaks in terms of the problems you are trying to solve. The breadth of C++ expression makes it a natural language to this end. Bjarne Stroustrup — a fellow Dane, and the inventor of C++ — has never called C++ just an object-oriented language, but has always made it clear that it is something more. The book brings out these powerful elements of language, expressing them as idiomatic extensions of the “object-oriented core” that programmers learn and expect to find in the language. That makes this book not only unique (and therefore timeless) but extremely useful as well.

JCS: You've said that it is really about making the distinction between accomplishment and excellence.

JOC: Yes. Of course, you can put words together to make a program run, and that's accomplishing something. You have shown that you can use the language. So for example, right now I am learning Danish. I've passed my first government Danish test, and I'm working up to taking my second one at the end of next month. I'm still at the stage where I can put words together and order a hot dog or say good morning and good evening and things like that. But in terms of getting into Danish culture and using the language expressively, where I'm using its idioms to understand the subtle, cynical, Danish sense of humour - that requires an appreciation of a language that goes beyond syntax and semantics.

There is kind of a culture of understanding. In the book that culture comes largely from a long legacy of software engineering, but more so from the perspective of object oriented design. So you have this design culture perspective that's above the level of the language syntax and semantics. And that's where the excellence comes. If you can express things at that level you have a much more compact expression, you're more likely to be able to meet the needs that are

imposed by whatever domain you're working in. And are more likely to be able to develop a vocabulary and culture where you can effectively discuss the kind of problems you need to solve for customers. You just can't do that down at the bare bones and muscles and sinews of C++, you need higher level constructs.

JCS: In the UK, we have a radio programme called “[Desert Island Discs](#)” where the participant is invited to list the eight records they would take with them if stranded on a desert island, plus a book (excluding the Bible or other religious work, and the complete works of Shakespeare) and luxury. What single book would you recommend for a software developer stranded on a desert island? Let's assume that he or she will be rescued at some point and return to work!

JOC: To me the choice is clear - it is a book called “Design after Modernism” which is a collected work edited by John Thackara. It has a very engaging picture on the cover. We'll let the readers' imaginations run wild and see if they can track that down. There's not too many of the original books left. So that will be a bit of a treasure hunt for them.

It is broadly a book on theory of design. It speaks to a fundamental shift in thinking and philosophy that took place generally in the arts and literature, and specifically in the field of design even as it pertains to software, in the middle of the latter half of the 20th Century.

When I say “design” here I mean design of houses, design of statues, design of cities, although by coincidence, it actually contains chapters from a few people knowledgeable about software design. The contributors are people who are trying to understand what design is and whether there could be some notion of formalising design or making some kind of method so that you could either automate design or at least have something that would guide design in each of these disciplines.

On one level it's practical, on the other level challenging and thought provoking, and on some level it's downright profound. I mean it goes to the very foundations of existence and some of the great questions of philosophy that have been rattling around in humanity for centuries or millennia. So, for someone on a desert island there's a lot in there to think about, not only in terms of, “What am I going to do better when I get back home?” but in terms of, “Who am I and what's it all about?” It's an amazingly challenging and well put together book, with great writing and very thought-provoking ideas.

JCS: Thank you very much for the recommendation – and for taking the time to talk to me.

Biography

Jim Coplien is a Senior Agile Coach, software developer, and Systems Architect at [Nordija](#), where his role allows him to combine mentoring and teaching while still designing and writing software. He is a well-known industry speaker, author, innovator and consultant in areas of organizational development and software architecture, with more than 30 years of experience. He was author of the influential books "Advanced C++ Programming Styles and Idioms" (1991) and "Multi-Paradigm Design for C++" (1998). He co-authored "Organization Patterns of Agile Software Development", published in 2004.

Jim holds a BS in Electrical and Computer Engineering, an MS in Computer Science, both from the University of Wisconsin at Madison, and a Ph.D. in Computer Science from Vrije Universiteit Brussel. His past work has included architecture, design, and implementation in the EDA industry, as well as leading positions in academia, Bell Labs Research, in telecoms R&D, and in international consultancy.

He is a Visiting Professor of Computer Science at University of Manchester Institute of Science and Technology and was the 2003-2004 Vloebergh Professor of Computer Science at Vrije Universiteit Brussel.

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.