The Open Source Way

Episode 08 - Clean ABAP – Best Practices for Effective ABAP Code, Straight from the Expert

Karsten Hohage: Welcome to The Open Source Way. This is SAP's podcast series in which we'll talk about the difference that open source can make. In each episode we'll talk to a different expert and we'll talk to them about why they do it the open source way. I'm your host, Karsten Hohage and today I'm going to talk to Klaus Häuptle and Florian Hoffmann about ABAP Clean Code. Klaus is a software engineer who worked on different products and technologies in governance, risk and compliance and the supply chain management area. During his career he also has been a trainer, a coach and a product owner for agile software engineering. Currently, he mainly drives craftsmanship topics across SAP and is a curator for a large internal community with the intention to improve engineering topics by strengthening collaboration. Florian is a team architect and agile driver for SAP business integrity screening, SAP watchlist screening and SAP governance risk and compliance platform, especially the microservice-automated procedure. He's also a co-founder of ABAP Clean Code. So let's get started. It's about ABAP Clean Code, so, Klaus: What does ABAP Clean Code actually do? What kind of project is that?

Klaus Haeuptle : OK, thanks Karsten for having us. So for other languages like Java, there are many guides and books on how to write maintainable, readable and testable code. The most popular one is: "Clean Code" by Bob Martin. It's written for Java with examples in Java. And those principles can actually be applied to any language. But nothing was existing for ABAP, so our idea was to apply those principles in ABAP to have a discussion with a community on a common understanding of what that means with concrete examples.

Karsten Hohage: OK, so as I understand, this is not a coding project, but it's about producing information?

Klaus Haeuptle : Exactly. So it doesn't contain any source code, it's mainly written in Markdown files. So we use Markdown for documenting the good practices and many approaches which are valid for InnerSource and open source projects are also valid for documentation. So we use the same tooling for managing contributions using GitHub with pull requests. And especially the gated review process, which is also available with GitHub, is extremely helpful to manage this project.

Karsten Hohage: One detailed question to make sure about that, you said it contains no source code, is that, like, completely true or does it contain examples and things that will be source code?

Klaus Haeuptle : It contains examples but those are not executable. It's just documentation.

Karsten Hohage: OK, so documentation, guidelines, best practices, however you want to call it, that's the focus, right? OK, so Florian, before we go further into what kind of project that is, why is clean code itself so important?

Florian Hoffmann: The clean code is about writing maintainable code, as Klaus already mentioned. Maintenance is a huge topic for SAP. I mean, we're not writing software that's being used for half a year so we're writing software that's being used for decades, tens of years to come. The first software that SAP ever wrote is probably still out there running somewhere in some company or in some older system. And so keeping the code maintainable is really a huge day-to-day topic. You find that whenever customers report that something doesn't work, you need to find out quickly what the problem is. Is it just a misunderstanding of what the software is supposed to do? Or is it really a bug that you need to fix then in the end? And where does the bug come from? The easier the software is written, the easier the code itself can be understood by the programmers, the easier it is to identify possible sources of the bug and also to identify possible solutions for it. So clean code for SAP is mostly about making the day-to-day work of us developers easier. And in the end, it's also getting easier to write new code because people understand just better where the new code might fit in, where the plug-in places are. Also, clean code helps structure code better so that even those plug-in places become available which has not always been the case in the past.

Karsten Hohage: This guideline, which is in the end the product of SAP of ABAP Clean Code, that that's not only for SAP, right? That's also for all the partners and, I don't know, customer developers and everyone out there. Right?

Florian Hoffmann: Well, actually, we started out in a single team of 10. That's a scrum team, a team of about 7-10 people, mostly composed of developers that are working together on a single piece of software. And we just started out and found out that two or three people had read that book by Robert Martin and they just found: "Hey, why is nobody using that around here or why are we using it so little here?". We just started collecting things that we found are helpful in our daily work and that fit to the clean code paradigms. And when we started with that, we noticed that: "Hey, why should we limit that to our own teams?". So we expanded it to more teams, we took in the other teams from our organisation, made the code reviews a little more open so more people participated. We talked more about code, had some additional meetings on that and we found that even more people got interested. So we again expanded and Klaus set up that single one meeting that somehow kicked off the rest of this where we said: "Hey, let's do a session SAP-wide where everybody can freely join and let's simply talk about what we gathered there". It was back then and it was a wiki page that was composing some, I don't know, 100 or 200 suggestions on how you can make your code cleaner.

Florian Hoffmann: And - I was just talking about that - there were literally hundreds of people in that call and listening. And they got so much positive feedback that we said: "OK, that's so huge and so good. Why don't we simply share it with everybody?". It's something that every developer should know about especially also customers, partners, who are involved in the ecosystem of ABAP. And this is how we kicked it off in the way of: “Let's talk about open source here. Why don't we simply share it with everybody in the world for free and see what the feedback is?”. And this is where we stand now. We opened that open source repository. Got really a lot of contributions from the outside, lots of suggestions. And the clean code is not something that you keep inside the code as a secret algorithm that makes everything work in a magical way. It's something that needs to be driven by coders and coders need to talk to each other to improve their ways. And yeah, this is this is where we think open source is the exact right thing.

Karsten Hohage: Can you give us an example? What kind of things are communicated in ABAP Clean Code?

Florian Hoffmann: Clean ABAP is basically a large set of recommendations how you can make your code easier. And one of those simple recommendations is actually something that everybody can understand: give things good names. In the past, there were some funny limitations on the hardware side that caused SAP to introduce database tables with a field length restriction of five characters, meaning every field in the database had a name like "PRTYC". At the time, that created a kind of inside language. So everybody knew what was meant if you're talking about PRTYC. I mean, you can't even pronounce it. It's like, I mean, I don't even know how to spell that. And the funny thing is that the coders who were inside of the projects, they understood fairly well what was meant with that and the customers and partners who were involved with the code learned to understand what it meant. So everybody in the end came to that sort of common secret language where they knew what all these funny abbreviations meant.

Florian Hoffmann: But in the end, it's a lot clearer if you simply write it out, PRTYC or whatever it was, could be just a product type. So why don't you just write that in the code: ‘product type’. So let's make a function call that says ‘read product type’ instead of ‘read PRTYC’ and have everybody guess what that abbreviation was about. And this is a huge part of clean code. Just make things explicit in the code and just give them a very clear idea what your meaning is and actually reading things and understanding what they mean is about 70 to 80 percent of understanding what the software in the end does. It's like reading a book where every other word is an abbreviation. That's pure horror. You just don't understand it. That's why reading science literature is often so hard.

Karsten Hohage: So a lot of ‘clean’ actually means standardized, like as in we have a standardized way how we find names for variables or fields or whatever.

Florian Hoffmann: It's more about really revealing things. One huge topic in the book is readability. Readability means simply that you can understand what you are reading and that readability is what makes understanding the software far easier than having a good structure or having performing code or fast code or elegant code, even in some places. Readability is what really brings the developers together and makes them able to talk about a method. In the past, we used to have methods that were huge and abbreviated and cumbersome to understand but now we can sit together in meetings and actually look at the code and talk about it.

Karsten Hohage: I think I get it and I think I also get a picture of why that is obviously driven as a community project. But could you emphasize on that a little bit more? Why is it, especially in a project like that, so important that you involve the community?

Klaus Haeuptle : I would like to add another aspect. I think also it's not only about readability, it's also a lot about testability. So I think it's important that you get an agreement in the community in order to write effective test automation. We also need to to structure the code and architecture in a way that you can write effective, fast and stable tests. And the Clean Code Guide also gives a lot of guidance on that. Again, it's not about standardization. It's more about the agreement and the common understanding when it comes to testability. And then we make a lot of proposals on how the code can be done in a testable way but also how to write your tests that they are readable again. When it comes to involvement of the community, I think it's important that you start small. So I think originally it started with a few teams in governance, risk and compliance. So Florian did a great job in summarizing all the stuff on the wiki page and based on code reviews with colleagues in governance, risk and compliance, there were a lot of intense discussions and sometimes also conflicts but the result was very good. And then we decided this next step: We do it InnerSource as a GitHub project. I also have an internal community which I manage with colleagues from all over SAP who are interested in craftmanship topics and then we organized the presentation in this community and lots of colleagues that gave a lot of amazing feedback and also contributed and opened up issues and so the guide evolved and got much better. At a certain point in time, we also decided, yeah, let's make an open source project out of that. And again, there was a huge community of colleagues, of developers across the whole world, who wanted to adopt that and also gave feedback and sparked additional ideas on top of that. So on the one hand, to start small, involve a lot of feedback from the community, get a lot of contributions. So you get better over time. But on the other side, it's also a great chance to involve the community, get a common understanding, get a common commitment on the direction, and get early adopters who then adopt this guide. That is a first step then also to even get a broader community of people now who adopt it. So it's a natural curve of any idea. And using the community approach helps a lot on the way. An additional idea we had was that we want to get more ABAP colleagues familiar with clean code, with abapGit, with GitHub and InnerSource and open source approaches so that they maybe also start their own projects.

Karsten Hohage: You mentioned the adopters. So this is where we go from: "ABAP Clean Code itself does not contain any source code", but with the adopters of course is then when source code is being generated on the basis of... this leads me to the question: What would you say, between an open source project that produces guidelines, best practices like ABAP Clean Code, and an actual source code open source project, where do you see the main differences?

Florian Hoffmann: The main difference between a real source code repository and the clean ABAP repository that we have is basically, yeah, there's a bit less code and it does not execute that well. I mean, if you run the command on the clean ABAP repository, it will do nothing. In the end that means a couple of things. For example, we don't have unit tests in the clean ABAP repositories, simply because it's not that easy to write unit tests for clear English text that is supposed to, I don't know, check the grammar or something. Maybe we could check the formatting or something like that. But there are some other differences which are very important. The first is - and this is actually very outstanding I think - it's actually easier to participate in clean ABAP than to contribute to a real code project because in the end, it's just text. It's just written language that everybody can understand who is speaking English and, by the way, we also have some translations now. We got it in German and French; I think we have a Japanese translation - even that was contributed by an outside non-SAP person, a really excellent contribution that already shows where the community comes in here. We also got the Chinese version and this already shows that it's simply a text and everybody can understand it and immediately relate to it and make suggestions where the text does fit or does not fit so well. So actually, getting contributions is easier than getting real code contributions. And the second outstanding thing that we noticed: there is a lot more dialogue. When we write software, it's like two or three or a handful of people who are really in that code so deep that they can talk to each other and understand what each other means and work out what the best way of doing this or that is in that piece of code. When it comes to clean ABAP, you will notice that there are hundreds or thousands of people who understand what's meant with this.

Karsten Hohage: It's more like someone submits something and then you discuss it. Is that right? Or do I just imagine that?

Florian Hoffmann: With code, we're actually discussing more in the beginning, like design, architecture. You just want to make some important decisions before you lay down the wrong software components and end up with a mess. But with a plain text thing, like a recommendation and so on, it's a lot easier. And also, by the way, we can really benefit here from the basic techniques that something like GitHub is offering us.

So people can just open the pull request with their suggestions and we can simply talk about that. It's not already in the project but you can see the project, what it would look like if that were merged now. And that makes it a lot easier for people to discuss what the outcome is or what it should look like in the end.

Karsten Hohage: Right. But speaking of outcomes and seeing the results, now we have ABAP Clean Code out there. Do you also kind of see the results in ABAP-based open source coding projects?

Florian Hoffmann: As there are not so many ABAP open source projects, it's kind of hard to really get a representative cut of the... what the portion of software is that is getting better. What we did see in the very beginning even was that the teams, who started to adopt, even only a handful of those recommendations were suddenly producing code that was a lot better. And everybody or nearly everybody agreed on the code getting better. So it's not something that is theoretical or so, we're not pushing out some recommendations that nobody ever tried out, it's just the distillated findings of years and years of experience that people are putting together. And we actually do have code bases which are pretty much sticking to most of the recommendations in Clean ABAP. And they look pretty neat in the end. It's really readable code, understandable. You can easily talk to others about that code, even if you have absolutely no clue what the actual business application is supposed to do. There are always technical things that are, in the end, very easy to understand.

Karsten Hohage: OK, but isn't that also right that the Clean Code project basically triggered some ABAP tooling projects also written in ABAP that kind of have evolved around the Clean Code project?

Florian Hoffmann: Yeah, there has been a rather small section of ABAP open source projects for some years now. For example, I think we already talked about abapGit which is basically a Git client that can be imported to any ABAP system to just download and upload code from the GitHub or other Git repositories. And the idea there is simply to make it easier to share code. This project was there before clean ABAP and it is one of the most excellent projects that we have out there regarding ABAP. But there were also others, for example, there's one very common example that most people or many people know who are actually writing business applications. It's called "abap2xlsx". It's actually an ABAP to Microsoft Excel sheet converter. So it's actually generating Excel sheets for you or reading them in and so on. A lot of people have that as a task to: "Hey, I want to generate that data table as a Excel sheet". And that also has been around for a while. This is also a very good project. It's very cleanly styled and so on. It's not really related to the Clean ABAP project, of course, but we think the structure itself, that it's using, the object-oriented syntaxes and so on, that already shows that the people who wrote that had a lot of the clean code recommendations were already ingrained in the code and in their mind when they wrote down the structure. But in the end, Clean ABAP, we think, also gave rise, or at least boosted some new projects, that are also available. Now, for example, we have "Code Pal", which is a rather recent project and a very great one. That is actually a thing that helps you realize the Clean ABAP recommendations. It's just scanning your code and highlighting things where things are not really clean regarding that recommendation or so on. It's like, if you're familiar with it, the Code Inspector that helps you identify portions of the code where you, I don't know, harm certain guidelines and you should write it this another other way, so it's better for performance or something like that. So it's actually that for Clean ABAP helping you getting the style of the software better. And there's also another project which is more recent, which I think is also worth mentioning. It's called "abaplint" which is then a kind of tool that's supposed to help you make code reviews more feasible in the ABAP world. In the ABAP world, we don't really have those really cool code review tools that some of the other languages have directly incorporated into development toolage itself. So we had the feeling we needed to add something there and some people came up with the idea of "abaplint". And this is also an excellent example. And we do hope that the yeah, the large community that Clean ABAP has collected around it also helps these projects now to gain some traction and also helps other people to come up with new ideas and just start something new and cool.

Karsten Hohage: Thank you, Florian. I was going to take this a little bit into another direction and back to Klaus maybe here. Because another thing I learned, that is now based on the Clean ABAP project in its online representation, basically is, there is going to be a Clean ABAP book as far as I know. So the wheel has come full circle, like basically from the initial proprietary project via the open source and it's now a book.

Klaus Haeuptle : Yeah exactly, but maybe I would like to add another interesting project which is currently mainly InnerSource. But some colleagues developed some additions to make code review with GitHub simpler. So some teams already use GitHub intensively for reviewing ABAP code. So hopefully, that will also be available this year or next year externally to the partners and customers. Well, when it comes to the book, yeah. So we talked already a lot about the evolution from a few teams to the InnerSource repository to the open source repository. And then again, we asked the community who wants to write a book with us. So Florian and me, we don't have so much time. Writing a book is time-consuming and we asked the community and four colleagues applied to work on the book with us. Yeah, and half a year later and after a lot of work, we published a book on Clean ABAP.

Karsten Hohage: Oh, it's already published?

Klaus Haeuptle : Yeah, so it's already published in the U.S. It's already available to buy and soon it's also, I think next week, you can also buy it in Europe, in Germany.

Karsten Hohage: Cool. And that was also a community effort, great. What difference does the book make when you compare it to just, I don't know, reading through the GitHub repositories?

Klaus Haeuptle : The demand from the community for a book was mainly also, I think, that the guide is very good and it describes a lot of the principals with a lot of examples. But it's not a great experience to read. You can't read it on the beach, on the couch. And I think the book is much better. It also explains more context and if you're a beginner in those topics, it makes your journey easier. Another thing, we also added some additional chapters. For example, we have an additional chapter on how to implement it as an individual, as a team or as an organization. So there are a lot of practices and techniques you can try out with your team to get better on those things which are described in the clean code. But there are so many other books which build on top. For example, there's a book on clean architecture by Bob Martin. A lot of books are on test automation, evolutionary architecture and all these techniques which I described can help a lot in learning and implementing them also in your teams.

Karsten Hohage: OK. Which leads me to the question, if you were in a development team that is just starting to think about clean code or starting to clean up their ABAP code, where would you start on GitHub or with a book or how would you go about it?

Klaus Haeuptle : Actually, I would use both. So start reading the book so everyone could read the book and also use the clean code repository, especially also helpful if, for example, if you would like to do your code reviews with GitHub and then you have a discussion, you can also reference to the chapter in the GitHub repository which makes the discussion simpler. And what I would also do is not take the whole book and apply it at once, that's too much. You should digest it into smaller chunks. So there is also another approach available, independent of clean ABAP, that is called Clean Code Developer Initiative. It breaks down clean code topics into smaller chunks. So you could choose a first chunk which is also described in the Clean Code Developer Initiative and then read through the material. Take the clean code guide and work through it as a team, not only as an individual, but also as a team to get a common understanding and an agreement and also a commitment to go that way. Because otherwise, if you do it only via code reviews the discussion can be quite tedious and sometimes also conflicted.

Karsten Hohage: Yeah, and I guess my question was rather general and I guess it also makes a difference if you're the architect of something larger or developer or whatever. Right?

Klaus Haeuptle : For architects which are in teams, they are developers and all the principles also apply to them. If you're a chief architect and responsible for a whole product, I think it's still valid to know all the principles. Maybe it's not so relevant for your daily job but there are some topics which also touch the overall system architecture, especially when it comes to testability or other general principles around interfaces or solid principles. I think that's very valid also for a chief architect. And he should know about that.

Karsten Hohage: Now, maybe switching back to Florian, as a before-last-question, do you have any more details on how would you design the very first steps of the project that has its ultimate goal to have clean ABAP?

Florian Hoffmann: Yeah, that depends a lot on your product and no, not so much on your product, it's actually more your team and the kind of project you're working in. We actually found that one of the most important things to get clean code to work is, have fun. It's not done with serious discussions about whether certain names should be this one or that one. It's actually sometimes you need to throw in a really, really stupid, silly name to just make the idea clear behind something. And that's getting everybody to laugh. Actually, we did have a lot of fun when we wrote Clean ABAP and clean code recommendations. And I think it's really important to keep this and carry it on in the dialogues. Dialogues between developers tend to become a bit over-serious. If people can't agree on a common opinion, if one person has a different opinion than another, then we find that coders are often not the best to argue out a good agreement between those two opinions. We are not really politicians here, so those discussions can actually become a bit... turn into small conflicts that are getting people's nerves up and so on. So it's actually easier to keep a fun attitude to the thing. And let's see it this way: You've got the code. It's already there. It's working, you've got the unit tests and now you can do it a bit better. And the next small thing that is very important is start with small things, like Klaus already mentioned. It's really important that you do not apply the huge chunk of those thousands of recommendations in one piece. That's simply impossible. Start out with one or two rules and try to implement it in your daily work, try to convince the people in your team to agree on trying it like this and that. And you will already see a lot of improvements throughout the code base with even those simple steps. And another important thing is, even though it's now a book and it's somehow a hard copy that you got in your hand and that doesn't change anymore, Clean ABAP stays something that is still evolving. The language itself is changing, the requirements are changing. For example, we are now making that huge leap from the on-premise world to the cloud world where we have to basically move whole ABAP systems into a cloud world that they have not really been designed to. And they need to be rewritten in pieces. They need to be repurposed. And that also has a huge impact on the way how we write code. It needs to evolve together with the requirements that it has around it. So it's really important that you get the team, the people that you're working together with to have a sort of basic consensus on what the rules are that you want to focus on and what are rules, that you can maybe try out and what are rules that are maybe even too hard for beginning with. I mean, there are some rules that are really, really hard to tackle. We can see that day by day, there are some rules that are just taking so much practice and they can only be applied after two or three other rules have been successfully implemented that you should really not start with them. They are pointed out in the Clean ABAP repository. They are also highlighted in a lot more detail in the book itself. What are good places to start and what are things like what you shouldn't better start with or rather move to the end of a list of your To-Do list with Clean ABAP. And if you adhere to those simple facts, like have fun, evolve it and start small, you're already in a huge step into a successful implementation of Clean ABAP.

Karsten Hohage: OK, great. Thanks, Florian. And parts of what you said, you just reminded me of one of my favorite jokes because when you said, like sometimes it's hard to get people to agree on things and they can get into fights almost between people. You know what the difference between a terrorist and a software architect is?

Florian Hoffmann: I have no idea.

Karsten Hohage: You can negotiate with a terrorist.

Klaus Haeuptle : Haha.

Karsten Hohage: Anyway, let's come to our usual closing question. If you could have people take two or three main messages away from this podcast, which two to three main messages would that be?

Klaus Haeuptle : I think the most important one is: A clean code is important, so most of your time is spent reading, reading code and the code exists, especially in the SAP world, for many decades, usually, if the product is successful. So it also affects many developers who have to maintain your code in the future.

Karsten Hohage: OK, so clean code is important, which comes from Klaus, then maybe the second one from Florian.

Florian Hoffmann: I would say clean code or Clean ABAP especially is something that was made by coders for coders. So it's not some architect like me trying to tell people what to do and have fun because I know best and I'm the best programmer in the world and you need to do what I do. It's actually just the other way round that people are actually convincing each other. So it's on the grassroots level what's happening there. And people should really take that into their minds that this is not something that somebody high, high, high up in the hierarchy decided. It's actually something that somebody next to you on the same desktop maybe even decided that it's maybe a bit better to write that code in that way.

Karsten Hohage: Mm hmm. That's one. Then it's Klaus's turn again if you have another point you want to add, Klaus.

Klaus Haeuptle : So if you start with the Clean ABAP using the book or the guides, I think it's very important to involve your team because in the end you usually have shared code ownership. And if you do something in a certain way and another one has another opinion, as you said, there can be a lot of conflicts. Also, if you want to do code reviews it's good to do a joint learning approach. It's not necessary, you can also learn it on your own and there is immense benefit for yourself. But to get really better, it's good to do it together with the team.

Karsten Hohage: All right. So involve the team. Let's leave it at these points then. I thank you Klaus. And I thank you, Florian, for being our guests today. It was great to have you here and thanks all for listening to "The Open Source Way". If you enjoyed this episode, please share it and don't miss our next episode. It's published every other Wednesday around noon. You'll find us on openSAP and also in all the regular podcast distributions like Apple podcast, Spotify and the likes. Thanks for listening in again and thanks Klaus and thanks Florian again.

Klaus Haeuptle : Thanks a lot for having us.

Karsten Hohage: You're very welcome. Bye bye, everyone.