Johner Institute

Code Generation for Medical Devices

With Matthias van der Staay, Prof. Dr. Christian Johner

Transcript

00:00:05 Speaker 1

Medical Device Insights, a podcast by the Ione Institute for medical device manufacturers, authorities and notified bodies.

00:00:18 Speaker 2

Software is playing an increasingly important role in the development of medical devices, as you know.

00:00:23 Speaker 2

So not only is software part of the medical device, often it is

00:00:27 Speaker 2

the software itself of the medical device.

00:00:30 Speaker 2

Especially in view of the fact that it is becoming increasingly difficult to find experts who can write software, good software, the question arises as to whether we will not find a possible way out of this dilemma with software generation.

00:00:46 Speaker 2

And that's why in today's podcast we want to look at what code generation means, when does it make sense, how do you give it, do you go about it

00:00:57 Speaker 2

And what should you pay attention to, especially when developing medical devices?

00:01:01 Speaker 2

And for this I invited an expert, namely Matthias van der Stey from the company I.M.T.

00:01:07 Speaker 2

This is a Swiss company, also a development service provider, yes, with whom we have been working for years.

00:01:14 Speaker 2

Yes, Mr. van der Stey, if you briefly introduce yourself, perhaps also about the training and what you do

in the context of code generation, then there will now be a good introduction to the topic.

00:01:25 Speaker 3

Yes, my name is Matthias van der Stey and I have been working at I.M.T. for over 10 years now.

00:01:32 Speaker 3

I started there in the field of signal processing and control engineering and was able to work on various projects.

00:01:42 Speaker 3

I have also worked on projects in medical technology, for example on a ventilator.

00:01:51 Speaker 3

A little later I was allowed to take over the team leadership for the signal processing team and we mainly deal with control technology and signal processing in the areas of controller design and signal processing, i.e. all the filter designs and adaptive processes.

00:02:13 Speaker 3

But we also solve our problems in the field of image processing and machine learning.

00:02:20 Speaker 2

And if I understand it correctly, you also apply the code generation methods directly, don't you?

00:02:28 Speaker 3

Exactly, that's the case, especially in the context of control engineering, now less so in the field of machine learning.

00:02:34 Speaker 3

But in the field of control engineering, there are a lot of people doing it, a lot of industries are using code generation, because it fits very well into the whole development process.

00:02:47 Speaker 2

Yes, maybe let's take a look at what code generation actually means.

00:02:51 Speaker 2

Well, the name adds of course.

00:02:52 Speaker 2

In the end, you somehow have code that was written by the machine.

00:02:56 Speaker 2

But what or what is the input, so to speak, with which another piece of software then probably writes the source code?

00:03:04 Speaker 2

So, what's the starting point of all this, so to speak?

00:03:07 Speaker 3

Yes, code generation is basically very diverse, if you would put it formally.

00:03:16 Speaker 3

it's simply about generating something less abstract from something abstract.

00:03:23 Speaker 3

So this can also be a compiler, because it also generates from C.

00:03:28 Speaker 3

Code, such as machine code.

00:03:32 Speaker 3

But the term code generation is also often understood to mean that code is generated from models.

00:03:40 Speaker 3

This means that an abstract model then becomes a program, for example

00:03:46 Speaker 3

Code generated.

00:03:47 Speaker 2

What kind of models do you have there?

00:03:49 Speaker 2

So, is it somehow in a tool, like Enterprise Architect, or how do you build these models?

00:03:54 Speaker 3

Yes, there are various tools that we use.

00:03:58 Speaker 3

So on the one hand, we use Mathwerks' Simulink tools a lot together with Matlab, but we also use our own tools that we wrote ourselves for code generation.

00:04:12 Speaker 2

When we talk about these own tools,

00:04:15 Speaker 2

What do you use as input?

00:04:16 Speaker 2

Is it a visual description of models or have you developed your own language for it in order to ultimately generate the code from it?

00:04:26 Speaker 3

Yes, that's basically both.

00:04:28 Speaker 3

So, you have the choice whether you want to describe something visually, but you also have the choice to store the configuration file in a place that is to be generated and

00:04:44 Speaker 3

In this respect, you are not subject to any particular process.

00:04:50 Speaker 2

What you have already said is that you use code generation primarily in the context of signal processing.

00:04:59 Speaker 2

Now, on the other hand, you have told us that you also use commercial tools, but you have also written some.

00:05:05 Speaker 2

That means we now have such a balance.

00:05:08 Speaker 2

On the one hand, work has to be done to write code generators.

00:05:13 Speaker 2

On the other hand, work would have to be spent to write the source code.

00:05:18 Speaker 2

Now the question arises, when does such an investment pay off?

00:05:22 Speaker 2

In which areas would you particularly recommend generating code?

00:05:27 Speaker 3

Yes, that's basically a very good question or a very versatile question of when it pays off and when it makes sense.

00:05:35 Speaker 3

So for I.M.T.

00:05:37 Speaker 3

was, I mean, writing a code generator

00:05:42 Speaker 3

Only for a single product, that will never pay off, because the effort is far too high.

00:05:48 Speaker 3

But since IMT is active in the project business and IMT is allowed to implement a lot of projects and the same work has to be done over and over again, it makes sense to automate this work, simplify it and ultimately make us more efficient.

00:06:07 Speaker 2

Suppose that if you have such a code generator, i.e. you have made this initial investment, this general investment, then there is still the work to be done to create the models with the help of this code generator.

00:06:21 Speaker 2

Yes, that is, then we now have to find a different balance, namely between modeling in the model and on the one hand and writing the code.

00:06:30 Speaker 2

Are there areas where you would say that modeling is just much faster than writing the source code by hand?

00:06:38 Speaker 3

Yes, I wouldn't confirm that.

00:06:40 Speaker 3

So, experience has shown that you are almost as fast when you model something or program something by hand.

00:06:50 Speaker 3

I see the advantages much more when it comes to the lifecycle, i.e. when you have to make adjustments to the code,

00:07:00 Speaker 3

because, as it turned out, you are much more efficient there.

00:07:04 Speaker 3

The main reason, as I see it, is that graphical modeling gives you very valuable documentation.

00:07:16 Speaker 3

This means that the software is actually very well documented, graphically or normally you document software exactly graphically.

00:07:26 Speaker 3

And if the documentation is actually the same as the code, so to speak, and that is always in sync, that's a huge prejudice.

00:07:37 Speaker 2

Ah, OK, so that's a new insight for me.

00:07:39 Speaker 2

So, it wasn't the point for you that you want to save work on coding, but ultimately actually on documenting and keeping documentation synchronized, i.e. model and code.

00:07:52 Speaker 2

That‘s nice, very nice to hear.

00:07:55 Speaker 2

I had also generated a lot, a lot of code and at this point I actually ran into the opposite problem, namely keeping a generated code synchronized, because it was often difficult to describe business logic.

00:08:11 Speaker 2

How do you manage to keep the code synchronized or phrase the question differently?

00:08:17 Speaker 2

Are you able to see everything in the model

00:08:20 Speaker 2

so that there is no need to intervene in the generated code at all.

00:08:25 Speaker 3

Exactly, that‘s how it is with us.

00:08:27 Speaker 3

So, we do it in such a way that we actually don‘t change the generated code anymore.

00:08:33 Speaker 3

So, we only ever generate in one direction, never back.

00:08:38 Speaker 3

And yes, this requires or that of course requires that all application logic can also be described visually or in the model.

00:08:49 Speaker 3

Otherwise, of course, it will be difficult.

00:08:52 Speaker 2

Could you give us an example of such a model element?

00:08:57 Speaker 2

So, what kind of notation elements does it insist on, so that we can imagine what you can express with it?

00:09:03 Speaker 3

There are a lot of different model languages on how to describe what.

00:09:10 Speaker 3

So, these can be filter elements that we can drag in.

00:09:16 Speaker 3

But it really goes so far that we have auditions

00:09:19 Speaker 3

graphically.

00:09:21 Speaker 3

We can describe various other mathematical operations.

00:09:26 Speaker 3

Then there is also the possibility that we can describe state diagrams, yes, how to scan from UML.

00:09:34 Speaker 3

Then you can define the logic directly in the model with a scripting language and then generate the code accordingly from the whole construct.

00:09:46 Speaker 3

So in this respect, you can really

00:09:48 Speaker 3

also perform very low level operations.

00:09:52 Speaker 3

What is not possible, however, is direct access to hardware, and there is always a need for an interface between generated code and handwritten code.

00:10:01 Speaker 3

That's also something we typically do.

00:10:05 Speaker 3

Well, we never actually generate projects holistically, but there is still a certain part that is handwritten.

00:10:15 Speaker 3

the important thing is simply that you define and adhere to clear interfaces, that you really don't have to adapt the generated code anymore, but that this code then simply communicates with other handwritten code, which then accesses the hardware accordingly.

00:10:35 Speaker 2

Mhm, you just hinted that even if the notation elements are not sufficient, so to speak, you also have the possibilities to add to it with the scripting language,

00:10:45 Speaker 2

Is this a self-developed scripting language or have you put one on it that already exists, like no idea, JavaScript for example.

00:10:52 Speaker 3

Yes, in our case this is now in the range of Simulink and and Matlab it is simply the M.

00:11:01 Speaker 3

Script that you can use there and from this M.

00:11:05 Speaker 3

Script is then co-generated accordingly.

00:11:09 Speaker 3

But you also have the option to play directly in C.

00:11:13 Speaker 3

syntax, which was then taken over accordingly.

00:11:19 Speaker 2

Will be.

00:11:19 Speaker 2

We have already talked about the topic of documentation, and it is one of the main regulatory requirements for medical devices or for the development of medical software.

00:11:32 Speaker 2

Now we also have regulatory requirements for code generators.

00:11:38 Speaker 2

Which one would you like

00:11:39 Speaker 2

as particularly relevant and how did you meet this requirement?

00:11:44 Speaker 3

So, the most important for us is certainly or most important in connection with code generation is certainly the 13485 and the 62304, which gives us certain specifications.

00:12:00 Speaker 3

Now in the field of 13485, it is certainly the requirement that the standard sets that we tools

00:12:09 Speaker 3

And validate tools, i.e. a co-generator, as long as it is involved in the development process for the medical device.

00:12:19 Speaker 2

And that‘s obviously what you did.

00:12:22 Speaker 2

Now you have also mentioned 62 304 again.

00:12:25 Speaker 2

This is actually a standard that takes care of software that goes into the medical device and that is not the case with a code generator.

00:12:34 Speaker 2

From a 62 304 perspective, how do you handle the generated code?

00:12:39 Speaker 2

Do you use it as a soup or do you say, no, we actually treat it like self-written code and then apply all the elements of quality assurance that 62 304 provides for to it?

00:12:54 Speaker 2

Which of the versions do you use?

00:12:57 Speaker 3

Yes, so basically as a soup, I think it will be relatively difficult, because by definition a soup is something that

00:13:08 Speaker 3

was not originally developed for the product to be integrated into it, but yes, a SUP is typically a library or whatever.

00:13:22 Speaker 3

Sure, you can try to argue with a lot of creativity, but I doubt that an auditor will accept that.

00:13:31 Speaker 3

From this point of view, we submit to the generated code

00:13:37 Speaker 3

Yes, quite normal the 62304, as it would be written by hand, and perform exactly the same activities.

00:13:47 Speaker 3

In other words, we carry out a static code analysis with appropriate metrics and we do dynamic tests of the individual components and we also ensure appropriate traceability between requirements

00:14:05 Speaker 3

and code.

00:14:06 Speaker 3

But it‘s true that you don‘t necessarily have to do that.

00:14:10 Speaker 3

So in the example of the compiler, which is also a code generator, it is not the case that we check the machine code again and carry out all the checks.

00:14:26 Speaker 3

But the fact is that we can argue differently.

00:14:32 Speaker 3

This means that the

00:14:34 Speaker 3

With a compiler, you can refer to the fact that the probability is very, very, very low that it will introduce an error into the application because it has been used thousands of times.

00:14:49 Speaker 3

and So we can do a little validation there, that of the compiler and that's good.

00:14:55 Speaker 3

On the other hand, with a code generator, especially a code generator that was written by the user,

00:15:05 Speaker 3

you can almost not verify that from a blackblock point of view, because you can't see into the system code generator and from that point of view it becomes difficult to argue.

00:15:20 Speaker 3

Yes, it won't introduce us any errors, because it is not used as thousands or millions of times as a compiler and from there.

00:15:33 Speaker 3

In the end, the only option left for us is to really check the generation accordingly and make sure that the function for which it is there actually fulfills.

00:15:44 Speaker 2

Yes, thank you very much.

00:15:45 Speaker 2

So maybe 2 thoughts on this and another question.

00:15:50 Speaker 2

First thought, so with sub

00:15:53 Speaker 2

you can not only include libraries, but also self-developed code for which there were no quality assurance measures.

00:16:00 Speaker 2

That's exactly what SUP from O.T.S.

00:16:02 Speaker 2

differs.

00:16:04 Speaker 2

Second thought, what they do is a risk-based approach and that's actually exactly what you want, namely that you think about it, with what probability can a mistake slip through again.

00:16:18 Speaker 2

And the way I've heard, they're on the safe side they're moving towards,

00:16:23 Speaker 2

that they validate the code generator and not only as a black box, as I have just understood it, and that they also subject the generated compilation, i.e. the source code that their code generator spits out, to complete quality assurance, as they did for their own code.

00:16:43 Speaker 2

So double mowed, so to speak, is better here.

00:16:45 Speaker 2

One more question, you just said that as part of this

00:16:51 Speaker 2

check, i.e. the quality assurance of the generated code, also do a static code analysis.

00:16:57 Speaker 2

Why can't something like this already be checked at the level of the code generator?

00:17:03 Speaker 2

Or to put it another way, how can it be that the code generator generates code that does not meet the requirements of a static code analysis?

00:17:13 Speaker 2

How could something like this happen?

00:17:15 Speaker 3

So, that can happen in the fact that

00:17:20 Speaker 3

an error in which code generator exists that generates something that is not allowed, such as dividing by 0 or nullpointer access or whatever.

00:17:32 Speaker 3

And from that point of view, yes, if you don't have validation from the code generator and can really execute such low-level instructions in the model, then

00:17:47 Speaker 3

yes, you are not immune to it.

00:17:49 Speaker 3

So, for example, you can very well define things at the model level that are not good, i.e. that are dangerous, because the problem is a bit that many co-generators already have explicit checks that check something like that.

00:18:13 Speaker 3

But what will happen then

00:18:15 Speaker 3

resulting in potentially inefficient code.

00:18:19 Speaker 3

This means that if the code generator assumes that the code generator or if the code generator wants to protect the user from itself results in some way, that's a lot of additional checks that are generated.

00:18:35 Speaker 3

And if you can't assume endless performance of the processor, you're

00:18:44 Speaker 3

depending on the case, do not have such checks generated for this purpose.

00:18:49 Speaker 2

I see, that is, it was more or less a trade-off where you put these final checks and came to the conclusion that it can sometimes be more elegant not to build this into the generator or into the evaluation of the model, but ultimately in, for example, a static code check of the generated code.

00:19:10 Speaker 2

O.

00:19:11 Speaker 2

K.,

00:19:11 Speaker 2

Yes, you can already see how much experience something like this is necessary not only to model, but also to buy or build appropriate generators.

00:19:22 Speaker 2

And you obviously have a lot of experience in this area.

00:19:26 Speaker 2

With which projects could you help other companies that may now be in a similar situation particularly well?

00:19:34 Speaker 2

So, we would also link your contact details.

00:19:38 Speaker 2

For whom does it make particular sense for him or her to contact you?

00:19:41 Speaker 3

Basically, especially if you were to have something developed in the field of embedded devices.

00:19:50 Speaker 3

So we use our co-generator mainly for embedded devices.

00:19:55 Speaker 3

There, they are part of our standard, unless the customer explicitly does not want to.

00:20:02 Speaker 3

But we are very much exaggerated by the advantages.

00:20:07 Speaker 3

which brings us these code generators.

00:20:10 Speaker 3

What we also have, as I mentioned before, is this own tool that can also generate code.

00:20:21 Speaker 3

And we decided on this tool because it is very important in our development process.

00:20:30 Speaker 3

This tool we decided to analyze this commercial.

00:20:35 Speaker 3

under the company Dataflow A.

00:20:38 Speaker 3

G.

00:20:38 Speaker 3

Yes, you can take a look at it and see if that's something.

00:20:45 Speaker 2

This means that you even have 2 ways to support others and let them benefit from their experiences.

00:20:51 Speaker 2

Namely, on the one hand, by really helping with the engineering directly and on the other hand, by making the tool, which has obviously already proven itself for you, available to others.

00:21:02 Speaker 2

Yes, great.

00:21:03 Speaker 2

I would like to thank you very much.

00:21:05 Speaker 2

Mr. van der Steyr was a lot of fun.

00:21:08 Speaker 2

For all those who want to know more about the topic, they are welcome to take care of it with confidence.

00:21:13 Speaker 2

Otherwise, I have to say thank you very much again and see you next week.