Case Study: How DSLs transformed Voluntis in the worldwide leader in Digital Therapeutics algorithms

Strumenta s.r.l. C.so Duca degli Abruzzi 81/bis, 10129 Torino (TO), Italy | info@strumenta.com | https://strumenta.com | VAT Number: IT 11817320010

This case study, in a nutshell

In this single page we provide a few pieces of information, for you to decide if you are interested in reading this.

To who is this case study relevant?

This case study may be relevant if any of this is true:

- You build safety critical software
- You build software which contain complex business logic
- You develop software in close collaboration with some form of domain experts. Examples of domain experts may be engineers, medical-doctors, accountants, and lawyers
- You want to understand if DSL could help in your situation
- You work in digital therapeutics

Executive summary

A company reduced software development times for medical algorithms by around 20 times, reducing significantly errors by adopting a custom language (a Domain Specific Language, DSL).

The case-study details the effects on productivity, time-to market, and error reduction. Problems faced and challenges during adoption are also discussed. Some lessons learned are then shared.

About the client

Voluntis is a French/American company who has been developing digital therapeutics software since 2001. The company went public during the development of this project.

About this case study

This case study is based on declarations of members who were involved in the project at Strumenta or at the client. All quotes are from employees or former employees of the client. The case study was verified and confirmed by the client before being published.

A large number of verbatim citations from personnel involved in the project have been reported.

Problems addressed

Voluntis is a company developing digital therapeutics software. This software includes mobile applications that patients install on their smartphones to monitor their condition and to receive recommendations on measurements to perform (e.g., measure blood pressure or insulin), medicaments to take (e.g., a certain dosage of a painkiller), dietary indications, and other indications which are typically given by medical personnel. This software is obviously safety critical and the industry is heavily regulated (medical device).

Writing digital therapeutics software requires the transpositions of medical competences into software. At the same time all the typical software engineering skills are required to design, develop, and maintain software which has to meet the highest standards.

The Software development process before the DSL

Before the project started the company was developing software with traditional methodologies.

> *Voluntis is a company that does what we call digital therapeutics, which is basically treatments that are supported by software, right? So, sometimes the software itself can be the treatment. […] In general, we have medical treatments or medical protocols, what they like to call them, that medical people come up with, and that we need to translate into executable software. It's not that trivial. When I joined the company, it was like six years ago, and they were doing everything by hand using general purpose languages and they were doing native developments; Android, iOS, .NET.*
>
> *Reported by Patrick Alff*

Typical problems faced in digital therapeutics software

There are typical problems connected with this type of software:

- Misunderstandings between medical doctors and software engineers are very common
- It is hard to extract requirements from medical doctors, due to the fact that they have difficulty in formalizing their thoughts to a level that allows digitalization into algorithms
- Development is slow, because of the necessity of adopting a controlled process, as imposed by regulatory agencies
- Because of development being slow and regulated it becomes very costly
- During development all non-technical people are “blindfolded”: until the application is developed the marketing team cannot see it and doctors are unable to provide feedback
- Time to market is terribly long, which makes it difficult to react to opportunities

Voluntis was experiencing these problems, as the other companies in the field did. In the following paragraphs we examine in detail some of these issues.

Requirements collection

There are several reasons which make collecting requirements particularly difficult in the context of digital therapeutics.

The application must implement a medical protocol, which has to be defined by medical doctors. They have to formalize their way of operating in an algorithmic form, so that it can be translated into software.

This is not trivial for many reasons. One of these reasons is that medical doctors, as many specialists, tend to take for granted knowledge specific to their domains. For example, it is obvious to them that certain ratios between systolic and diastolic blood pressure values are impossible, so this type of constraints are rarely made explicit and this leads to issues uncovered only during the final phases of development.

Another common problem when people with wildly different backgrounds collaborate is the lack of a common vocabulary: medical doctors, analysts, and software engineers think differently and for them the same term could mean different things.

> *Yes, so it's two different problems. […] One is two experts not understanding each other, which is **a doctor writes something and there is a certain interpretation of that, of the requirements engineer or the software engineer who writes it as a requirement**. In fact, the implementation itself can differ from requirement as well. So, you have several separate issues there. Test engineer might actually have a misunderstanding as well, so it's like **all the way to delivering the final product, there are different experts who can have different interpretations**. That's one thing.*
>
> *Reported by Patrick Alff*

Development time

Another significant issue is the time needed to complete the development of a single digital therapeutics application. From the time the design is started to the time the application can be verified by doctors a period of several months could pass. While this may be surprising to persons unfamiliar with this particular field, there are reasons for that. One reason is the necessity to ensure the traceability of the software process, which is a requirement for many regulatory agencies. These processes permit to ensure software is developed properly, but they add a significative burden on the organization developing the software, contributing to the explosion of the development time.

> *Well, when you do these things the traditional way then you have like a medical doctor who writes the protocol, I like to call it, and then you have requirements engineers who will translate that into technical requirements, and then you had developers who will implement that, and then you have testers who will test that. **And 18 months later, you will actually have a piece of code that's executable, and that's not doing exactly what the doctor thought it would be doing**. That's in short what I was facing there. This is regulated environment. We need to CE-Mark if you build something for Europe you need an FDA clearance.*
>
> *If you market something on the US market, so that requires compliance standards, very complicated standards that require a lot of traceability, a lot of documentation, design files, and tons of documentation. **It's a very slow process.***

> ***It's very labor intensive, and just to give you an idea, if the software development takes, let's say, one day, all the activities around it take four days. That activity is like testing, writing specifications. Activities like writing regulatory documentation, doing risk analysis, doing human factor analysis. It's mind blowing, really.***
>
> *Report by Patrick Alff*

Feedback cycle

There is a problem that partially derives from the previous one, which is the length of the feedback cycle. From the time the medical doctors describe the application on paper, to the time they are able to verify its actual behavior, a long time passes. Because of this conceptual errors or missing details are captured very late, making it very costly to fix them.

> *You see that, and it's like, "Wow, okay, this is just too complicated. It's like so many different experts who need to work together, and do not understand what they're doing."* ***If the medical doctor had found out the day he wrote his protocol or the next day that he was not going to do what he thought he would be doing, and then he might have changed it right away, and it would've been much less expensive to change it****. That's how I got into this thinking about, "Okay, how can we make this more interactive, faster, less expensive? How can we make it such that the doctor actually comes up with is exactly what he wanted to do and not something he thought he would be doing, right?" That's how we started this whole project.*
>
> *Reported by Patrick Alff*

> ***The main problem we were facing was the fact that the simulation of the algorithm would start too late in the development process.*** *The medical and algorithm design teams would work on paper, and they could not play with a true*

representation of the algorithm, until it was actually implemented in the mobile app. Therefore the "design" of the algorithm was not as much of an iterative process as we wished, where all teams would co-construct in real time something that is clinically and technically valid right out of the box.

Furthermore, because of this limited collaborative process at first, the actual implementation of the algorithm would occur quite late in the development process, whereas the algorithm is part of the first priority from a product perspective (assuming that the main business value of the app is the algorithm).

Reported by Voluntis

The other thing […] it's a very important point: when you draw a diagram of something, and/or you describe a protocol of something that's ruled or something, it's very abstract, right? You're not in execution mode. You would always forget something, or always something would come up that, "I didn't think about that corner-case." ***The sooner you can make it executable and simulated and make it work, the sooner the person who is, let's say, the inventor or the creator of the process or the algorithm, can actually put himself into the shoes of the user, and see, "No, it's not quite doing what I thought it would be doing," and fix it.*** *It's very important to do that very early on in the process, and* ***not wait 18 months for you to go back and say, "No, it's not doing what I thought it would be."***

Reported by Patrick Alff

Building and adopting the DSL

In order to solve the problems reported in the previous section Voluntis decided to build a DSL. The DSL has been then used to describe medical algorithms. The resulting DSL could be then interpreted inside mobile applications.

To make the project successful there were a set of challenges to be faced, some of which are typical in DSL projects:

- Finding expertise to design, develop, and maintain the DSL
- Build internal support, winning resistance from personnel not familiar with DSL
- Considering the specific challenges of a highly regulated sector as medical software
- Have different people with different backgrounds collaborate to combine the necessary domain and technical expertise

In this project Strumenta co-designed the language and have done most of the development and maintenance for several years. Other actors were involved in the design and development of the solution and they are listed in the acknowledgments.

How it was decided to adopt a DSL

DSLs solutions are not mainstream. That means that most people are simply not aware of the existence of these solutions. Therefore, they rely on traditional techniques, obtaining traditional results. In this case the CTO of Voluntis at that time (Patrick Alff) was already familiar with model-driven development, so he was conscious of the advantages which could be obtained by raising the level of abstraction.

> *Patrick Alff was the one taking the decision to start the MARTE project. Based on his previous experience using DSLs in telecom industry and his previous contacts with Markus Völter, we decided to go for a DSL.*
>
> *The trigger to start doing stuff differently was the fact that with previous apps, the development of the algorithm occurred too late in the development process*

> *So we wanted to find a way to develop the algorithm sooner in the product, and to make this more interactive / iterative.*
>
> *Also, in our plans we were expecting to increase quite extensively the number of medical algorithms we would develop every year (much more algorithms for Oncology than for Diabetes), and this was also a reason for initiating this industrialization of our algorithm design process and tools.*

Reported by Voluntis

Introducing DSL

Introducing a DSL inside an organization is an operation that requires to pay attention to several aspects.

In our experience human factors are very important. It is not uncommon to face resistance from software developers which are used to traditional techniques and are resistant to ideas proposed from outside the company. We faced this particular challenge also in this project.

We were able to ultimately win the resistance thanks to two factors:

1. The technical leadership supported us: the CTO (Patrick Alff) was familiar with these techniques and he contributed to build support inside the organization
2. We strived to include personnel at Voluntis in each phase of the design of the languages, from the earliest occasion. We also organized trainings to make them familiarize with MPS and understand the fundamental aspects of this technology

Regarding the second point, some consultants tend to work in isolation or to put in place mechanism to "protect" their knowledge. In our opinion, this type of approaches undermines the chance of success of DSL projects and should be avoided.

These techniques helped us to ultimately win internal resistance, but we had to face some, specifically from a portion of the software engineers, while the rest of the company was supporting this approach from the earliest stage.

> *I had to convince people, but at a different level. So, because in the, let's say, medical device and pharma industry, people are very theoretical in their approaches, basically. It was not difficult to convince them that, "Hey, this mighty approach was going to be a big game for us." It was way more difficult to solve the software engineers. In particular those people who are in general purpose language and developers who developed for iOS or Android. Usually, they're not very pragmatic. They're very dogmatic, as I call it, about using the design patterns that Apple has imposed on them. And then when you come with something new that they need to include in their app, that's very hard for them.*
>
> *It's been really hard, some of them to accept that, and it got to the point where actually one of the lead developers had to quit the company because he refused to implement […] because the runtime environment was written in C++, and so it was a library that we needed to include and he refused to do that.*

Reported by Patrick Alff

Another challenge to consider is that typically adopting DSLs require obtaining skills that are not present inside the company at the moment the project is started. These competences can be built over time, either by hiring or by training, but at least in the initial phase it makes sense to work with specialists. This may or may not be challenging depending on the culture of the organization.

> *No real challenge. Some new stuff, however. This was the first time we were subcontracting "critical" software development to a 3rd party.*

Reported by Voluntis

Limitations of the solution

While the solution designed and delivered brought significant advantages to the Client there are some limitations. We believe that it is important to share all issues we faced in order to give you an honest and comprehensive report.

The first problem is with the perceived maturity of the underlying technologies. In this project we adopted JetBrains MPS, which is an open-source project supported by a reputable software vendor (JetBrains). While the project has been started over a decade ago it has not yet reached the popularity of more established IDEs, therefore it does not have the same level of support.

We also adopted two open-source components: mbeddr and KernelF. KernelF in particular was introduced after this project was started and it is therefore relatively new.

> *The main problem today is the dependency between the language and MPS / mbeddr: the updates of these tools is not so easy (typically we had some issues when switching to the latest version of MPS).*
>
> *In other words, we have the feeling that the MPS "ecosystem" is not yet as mature as industry standards (a lot of changes in the underlying concepts in MPS, or kernelF, ascending compatibility is not yet mature, leading to painful upgrades sometimes).*
>
> *We'd tend to compare to upgrading Visual Studio/.NET for web dev, or Android Studio/SDK for Android, or XCode for iOS. Obviously, those more widely used platforms have much smoother upgrade experience for developers.*
>
> *There are other items to improve, for sure, but nothing really critical from my perspective.*

Reported by Voluntis

Another perceived limitation is in the cost of evolving the DSL solution. This is due to the fact that we developed a rich solution, with several moving components. They range from the tools to design and simulate the algorithm, to the runtime components distributed with the applications on several platforms, to the interface with web-applications. It is a rich environment and therefore changes in the way the DSL is designed have an impact.

> *Also, even if I would not say it is a problem today, we know that one of the limitation of a DSL is the fact that you will need to update the language if you want to introduce new concepts in your algorithm, which means ultimately that 1. You need to update the language, 2. You need to update the RTE, 3. You need to update your algorithms, 4. You need to integrate in your app. As for any "framework", this creates some dependencies that could create some frustration and some delays (I also know that there are many advantages however).*
>
> *Reported by Voluntis*

In this case, what we did was to focus on unit tests and continuous integration from the very beginning to ensure any issue is caught as early as possible. With several 1000s of unit tests, the run-time interpreter has a code coverage of more than 90%.

Advantages provided by the DSL solution

Building Domain Specific Languages can traditionally bring a set of advantages.

They main ones are:

- Reduced development time
- Significantly reduced feedback cycles
- Testing made easier
- Reduced rate of errors

These advantages have been confirmed by the client in this case, in additional to a few more specific to the solution delivered.

> *In addition, the use of the DSL was also a way to avoid implementing the algorithm in multiple languages, which would ultimately accelerate the effort for development + reduce the cost to test +reduce the number of bugs, etc.*
>
> *Reported by Voluntis*

In this particular case we have an advantage due to the fact that from a single DSL we can obtain applications which run on different platforms, guaranteeing the same logic is implemented on each of them, by construction. The advantages in reduced development time, easier testing, and reduced rate of errors have been confirmed.

Let's now dive more deeply in some of these advantages.

> *Main advantage is: Faster design for new medical apps, reducing costs and improving scalability & reliability*
>
> *To be a little more specific:*

- *Design/simulate/test/produce algorithm documentation in one integrated environment*
- *Allow Design team to check the result of their work immediately*
- *Design-time consistency checking thanks to projectional editor and "on the fly constraints checking"*
- *Simulate & iterate quickly with doctors/customers/formative study testers*
- *Enforce "Good Coding Practice" (State Machines, Decision Trees & Tables)*
- *Embedded testing (record test scenarios, import test scenarios from Excel, run 100s of test scenarios in minutes, get algorithm tests coverage)*

Reported by Voluntis

One advantage which emerged in this particular project is the possibility of **generating automatically meaningful documentation from the algorithm**. That same documentation can be used for the filing to obtain approval from the regulatory agencies. Because the documentation is generated from the DSL code which is ultimately executed, we are always guaranteed that the documentation is aligned with the implementation, as it is impossible for them to diverge. This means that it is more reliable and there is no effort necessary for trying to manually maintain the alignment between code and documentation, as it is normally the case.

Another advantage is the **much greater agility** obtained. This is another common consequence of adopting DSLs. Indeed, we perceive DSLs as enabler of strongly agile processes, as they permit the collaborative evolution of software at a pace which is very hard to imagine with traditional development techniques.

> *And as mentioned above, this platform-based DSL approach has reframed our whole approach to medical algorithm design for the better, from a very Water-fall approach, to a much more iterative/incremental process.*
>
> *Reported by Voluntis*

Results obtained

There are some results we can traditionally expect from the adoption of a DSL. They derive from the advantages we described in the previous section.

The exact results are often difficult to quantify. When it is possible to do so the clients may not want necessarily to share them. This makes it typically difficult to provide concrete, actual results, in a form that can be shared. We are trying to do this in this section.

> *We know that the time of development, the overall cost, and the quality of the algorithm has improved but It's really hard to quantify.*
>
> *Reported by Voluntis*

Improved testability

> ***Tests execution that was previously manual for one of our app, and now completely automated with MATE went from 3 days to 5 minutes.***
>
> *Reported by Voluntis*

The resulting system permitted to easily develop a larger number of tests. We cannot disclose the exact techniques we adopted to reach this goal. However, we can state that it was done by writing a specific DSL to represent test-case, in a way that they were readable by medical doctors. The system permitted to verify code-coverage and it permitted to execute automatically all the tests regularly, guaranteeing that the algorithms worked as expected on the different platforms.

Effectiviness

> - *Lines of code: We believe that the number of line of codes is 5 to 10 times less if you compare an algorithm written in a general-purpose language vs a DSL*
>
> *Reported by Voluntis*

Very simply, the DSL code permits to define higher-level concepts, so it is naturally more concise. As consequence there is simply **much less code to write, less code to debug, and less code to maintain**. This is a strong impact on the whole organization.

Improved marketing effectiveness

One interesting result obtained in this specific case was the usefulness of the DSL for marketing. Why is it the case? Because by adopting the DSL solution it was possible to **prototype the algorithm very quickly**. That algorithm could be then used inside the simulator, giving a way to the marketing department to show something which would resemble the final application very quickly. In this way the marketing department was in the position to work with leads, clients, and potential partners already during the development phase.

Time to market

A DSL is substantially a much more efficient language to be used for a specific goal. As a consequence, it permits to reduce significantly development time and the related costs. This means being able to deliver solutions faster, evolve them more quickly, and react to opportunities as they present themselves.

> *Yeah, as you know, […] the Voluntis solution in DSL was actually interpreted in runtime environment, and so we actually built a simulator where the medical staff and the consumer could simulate the algorithm right away. That was very useful. First, as good engineers, of course, we all thought that, "Okay, this is drastically going to reduce the time to develop an algorithm. And then once the*

> *algorithm is ready we just drop it in the application, and hey, magic happens and it works."*

Reported by Patrick Alff

Increased richness of the algorithms

In this specific case it meant that the company was able to go from 1, maybe 2 iterations of a specific algorithm to up to 18 iterations. This had a side-effect: as domain experts were now able to do so much more, they did so much more! As a consequence initially the complexity of algorithm went up, as they were now able to create richer solutions, which before could not be created because of the effort necessary to create even simple algorithm. To control the increase in complexity we introduced the concept of a complexity score. We identified suggested limits for complexity in our guidelines, as a way to avoid that perfectionism went out of hand, leading to solutions too rich.

Now, think that to get 18 iterations with the original development speed it would have taken around 27 years. While it is now taking a few months. That means being able to create medical algorithm of a quality that was simply not affordable before a DSL was adopted.

> *Yeah, so in fact the algorithms became way more advanced and complicated than before. Now, because we had to deal with this complexity that emerged from it, we still have to reduce the time to develop. We're not talking 18 months any more to do a development, but we still have a lot of iterations, a lot of interactions. The good thing is also with this kind of approach it's quite easy to validate an algorithm with a board of doctors, because usually we have one or two medical people actually would work on protocol, right? And then they need to validate it with what they call key opinion leaders and different people.*

Reported by Patrick Alff

Then what happens, of course, is that the medical people got way more involved in the design of the algorithm. Whereas before we had basically one iteration with maybe one second iteration to fix whatever was really broken at the very end after 18 months. Now, they would get immediate feedback on how that algorithm worked, right? They could play with it in simulator as if they had it on the patient's app, and then we would say, "No, no, we've got this corner-case that we need, and this other corner-case, etcetera."

So, what happened then is that medical people were just like, "Keep adding stuff to the algorithm modifying it, changing it. We had one algorithm where we had 18 iterations just to get one algorithm solidified. And then we faced another problem which is that in the end the algorithm is so complex that we as engineers felt very uncomfortable putting it out there. It was like, "Wow, how are we going to test this? How are we going to make sure it works?" This is way too complicated. Then the solution to that was, "Okay, well, that medical people can't understand, we'll tell them, 'Well, if it's too complicated, there's a big risk that they will make mistakes and things go wrong, so we need to simplify it."

But then they don't have no sense for what is a simple and what is a complicated algorithm? So, they have to basically implement the complexity index that was computed based on the different branches and operations that were in the algorithm. And then we told them, "Well, you cannot see this particular level of complexity for this type of risk in an algorithm." And that worked. They scaled back and they said, "Okay." We trimmed the algorithms, and then we scaled it back slightly so that it would fit the criteria that we had defined. But the difficulty becomes now, how do you test it? Because we would have thousands and thousands of possibilities and datasets that we need to crunch through the algorithm to verify that it's working properly, because for the FDA, for the regulators you need to prove that every branch of the algorithm and different combinations have been tested and that we have a high level of confidence that the algorithm will works as designed.

Reported by Patrick Alff

Verification

Verification was made significantly easier because of the presence of a simulator. Medical doctors could simply reproduce scenarios of interest inside the application to verify the recommendations actually proposed by the applications matched their expectations. The system had also a mean to record such interactions and provide automated tests, in a way that was extremely intuitive for the medical doctors to define.

> *It's so much easier when we can show them how it behaves in the real world, and then showing them tons of paper on how it works, right? All that was great for Voluntis is that in the end Voluntis is recognized as the world leader in algorithms for digital therapeutics. […] Voluntis has the most advanced algorithms in oncology together with this stuff. That's the good side effect that we have.*
>
> *Reported by Patrick Alff*

Lessons Learned

At the end of each project we aim to distill some lessons learned, because we believe in continuous improvement. We did so also in this case and these are some of the considerations which emerged.

Different domain experts can use DSLs differently

In some domains the domain experts can directly write the application using the DSL. This is the case for example for telecommunication engineers, as reported by one of the participant in the case-study, based on his previous experience. In other domains the analysts could do that, and the domain experts can work with them, reading the DSL. They can understand it; but they have problems formalizing processes on their own. In this scenario pair development with domain experts is possible. In addition to this we can retain the benefits of a reduced feedback cycle: analysts can show the application in a simulator minutes after the doctors describe it.

> *Because there's one big difference between what happened in the telco industry and what happened in the medical industry. In the telco industry they were all engineers, and so they had different disciplines, but they were all engineers. Now, in the medical world, you have medical people and you have engineers. The brain is not wired the same way, I can tell you that. So, they really have a hard time understanding each other, so I think it's not realistic to believe that a doctor, an average doctor, can actually write code using a DSL. But what you can do is you can have an engineer write code on a DSL that's specific to the medical domain that you're working with, and you can have the medical person understand that DSL or that program that's been written in the DSL. That is what we've been trying to do in Voluntis, and that worked.*
>
> *Reported by Patrick Alff*

Finding the right persons

Adopting DSLs can significantly change the processes of an organization, for the better. It will make the organization more efficient by empowering certain roles to be more independent and to contribute to the development of the software, something that was traditionally the exclusive responsibility of software engineers. However, depending on the nature of the DSL, software engineers may not be the best persons to use the DSL. In our experience other engineers, or analysts are the best users of DSLs. This seems confirmed in this case.

> *Actually, if I can maybe elaborate a little bit on that, it is important also that we do not try to convert somebody who is an iOS developer or an Android developer or software developer to be using your DSL because most of these people don't want to do that. So, it's much better if you bring in an engineer, like in our case things like health engineers. Those are people who usually work, I don't know, scanners or imaging or some kind of diagnostic tools or whatever. But they know enough about medical world, and they know also about software. This kind of people are very eager to have the programming language that's more restrained and that's very focused on what we need to do, and they are really good developers, with DSLs. I highly recommend taking that past reasoning trying to convert the GPL developer to DSL.*

Reported by Patrick Alff

When you do NOT need a DSL

We love DSLs and we believe they can change many organizations. However, because we love them, we do not want to see them misused. There are situations in which DSLs are really not the right choice. In this instance we describe one of these cases.

> ***I would say if you cannot see any communication problems between different expertise, then you don't need DSL.*** *It's domain-specific, that's what the DS stands for. If there's nothing domain-specific you need to solve, and if everybody in your team can understand whatever you're using, whether it's HTML or Java*

> *or whatever, then it's not an issue.* ***But if you need to communicate your code or co-design your code, I mean the code not the GUI, with people of a business or often industry who are not in the software industry, then you should consider a DSL****. That would be my short answer to this.*
>
> *Reported by Patrick Alff*

Integration is key

In our experience a DSL becomes a key component of a larger system. It becomes frequently a connector between other pieces. For this reason, it is key to devote the proper amount of energies to the integration of the DSL with the rest of the system.

> *The first implementation of MARTE in a real app was somewhat painful. Integrating the RTE in a real app was not part of the MARTE project, but the iOS engineer in charge of that was not a great believer in DSLs, and left the company. Integration in a mobile app was postponed to an actual real integration in the product. This led us to find some integration issues late in the project.*
>
> *If I were to start again, I'd make sure that the integration in a demo app with a real (therefore simple) algo is part of the Project, and part of the acceptance criteria of the language and of the RTE.*
>
> *Reported by Voluntis*

Involve the testers sooner

A DSL could never be built in a vacuum, without the close collaboration with the intended users and the people intimately familiar with the domain. That said it is not always trivial to get enough access and collaborations from all the different roles. In this case the project would have been benefitted by including the V&V sooner.

> *Also, I'd make sure to involve someone from V&V to help in the design of MATE far sooner in the project. To this point, I think that the "integration in a demo app" would have helped.*
>
> *Reported by Voluntis*

Consider impact on processes

Adopting a DSL can transform an organization by making it more efficient. As consequence of this it could change processes and this may have an impact on many different roles. Those persons have to be involved and supported, so that they can take advantage from the benefits brought by the new solution.

> *Finally, we could have done better in the adoption phase of the RTE at first. We did not allocate enough resources to accompany the actual implementation and use of the RTE in a real product. So the project team was not very positive, because we did not accompany them enough while we were changing their way of working. But we corrected that eventually and the integration ended up going really well.*
>
> *Reported by Voluntis*

Conclusions

Building Domain Specific Languages proved to be a good business decision for Voluntis.

This project was challenging for several reasons, but in particular because it was about changing how medical software is developed. That required tackling several challenges at one and reacting to the impact of the changes introduced in the organization.

While the project put the company in a better position there were a number of issues to face and solve. In this case study we did our best to disclose them and describe how we solved them.

The results are in our opinion very interesting: the reduction in development time, in the number of errors, in feedback cycles had a long lasting effect on the whole business and in the end all the persons we talked to enthusiastically confirmed that it was well worth the effort to embark in this project. This, we think, is the most important measure of what we do.

Learning more about this case study

If you are interested in learning more about this project you can contact us at federico@strumenta.com, we will share what we can, in accordance with the confidentiality we owe to the client.

You may also be interested in reading a scientific paper of the techniques used to develop a safety-critical DSL. You can read *Using Language Workbenches and Domain-Specific Languages for Safety-Critical Software Development* on the website of Markus Völter: http://voelter.de/data/pub/MPS-in-Safety-1.0.pdf

By the way, that paper was recognized as Best Paper 2018 by Software and Systems Modeling.

You may also want to listen to interview recorded by Federico Tomassetti and Sergej Koščejev together with Patrick Alff. While the interview has a broader scope, significant parts of it are about this project: http://beyondparsing.com/interview-with-patrick-alff/

Acknowledgments

We believe that value is tremendously easier to create when working with partners. For this project to be successful and for this case study to be written we have worked with a number of people who we would like to thank.

This case study has been possible thanks to the generous collaboration of Patrick Alff, Etienne Vial, and Laurent Wiart. In particular the quotes by Patrick Alff are from an interview we did with him as part of a podcast. The full interview is accessible here: http://beyondparsing.com/interview-with-patrick-alff/ . We then obtained his authorization to include his remarks in this case study.

We prepared a set of questions for Etienne Vial and Laurent Wiart. They were kind enough to provide detailed answers in writing. Those declarations constitute a significant part of this case study.

It was a pleasure working with them on this project and we would like to thank them to have generously agreed to participate in this case study. Thank you!

We would also like to thank Nikhil Khandelwal, who has been working closely with us to ensure the DSL was built to fit exactly the needs Voluntis has. It has been a pleasure working with him during this time.

This project was not developed exclusively by Strumenta. Beside the team at Voluntis there were key members who participated in this project. It has been a great experience working with them and a significant portion of the merit for achieving these results is due to them.

Among them we want to cite:

- Markus Völter, who is a worldwide expert in Domain Specific Languages. You can learn more about his activities on his website: http://voelter.de/
- Sergej Koščejev, an expert in JetBrains MPS Consulting. You can find more about his work at: https://koscejev.cz/

- Klaus Birken and Wladimir Safonov, who contributed significantly during the development of the solution

We would also like to thank JetBrains because they created and released JetBrains MPS, which has been an invaluable tool for this project. Their team offered help when it was needed to ensure the results we needed.