OOPSLA 2007
Domain-Driven Design BoF Session

Recorded and transcribed by
Vladimir Gitlevich

domaindrivendesign.org
vlad@domaindrivendesign.org

DDD BOF AT OOPSLA

EINAR LANDRE

We attended the “40 years of Simula” talk today, and one of those quotes of Kristen Nygaard, the inventor of Simula, is that “programming is understanding”. And I think that the Domain-Driven approach is bringing back the understanding of the domain. And it was also quoted that Simula as a programming language was not invented by computer scientists. It was invented for a particular need related to simulations, basically how to intercept intercontinental ballistic missiles. That was the domain they were working in, and they had a very hard time modeling those things in Fortran. So then they decided to develop a programming language that supports the concept of modeling of behavior. And I think that some of the discussions we have is how to bring back the understanding of the domain, that’s the key point as I see it.

Break in tape

DEEPAK GHOSH

I want to understand on how to get the understanding of the domain which is enough to take me to the next level of the design. What I want to know from you now is, this concept of Ubiquitous Language, what’s the next step? What can we do as notations that we can share straight away with the business and I don’t have to explain to them that, hang on, this has nothing to do with technology, we are just talking in terms of boxes. So what will be the Ubiquitous Language for us?

ERIC EVANS

I don’t think I have a cut and dried answer for this.

(MORE)

ERIC EVANS (CONT'D)
To me, the ubiquitous language is a language, right, and probably it's most important forms would be the spoken form, that is when you are actually talking with your domain experts, what language are you talking in, and the implementation form: is there anything in your code that looks anything like the way you talk to your domain experts. And that's where domain-specific languages come in in my mind, is that it is potentially a way to pull the implementation in the direction of our speech and the richer forms of communication that we have. And I agree with you that the current crop of DSL technologies is so technically demanding that it disrupts the thought process too much, and yet the concept of DSL remains so enticing. And in fact I'd like to hear Martin talk a little on this point because that's of course currently your main area of interest. Obviously the internal DSLs as you call them, where you actually just create carefully written, let's say, Java code, that starts to be so fluent ...

MARTIN FOWLER
...Fluent Interfaces is the term we came up with as I think a kind of analogous term from API-thinking direction.

ERIC EVANS
Yeah. So maybe you can talk to this point a little bit to, just kind of from the DSLs angle.

MARTIN FOWLER
I've been sort of in and out trying to grab Mr. North and get his attention for a few milliseconds and missed a bit of the discussion, but fundamentally I see DSL as the very technique that can help you manifest that ubiquitous language, be a communication technique.
(MORE)

MARTIN FOWLER (CONT'D)
I think the basic DDD idea is more fundamental in the sense that it's more important... you have to have that domain-driven focus and value because otherwise the DSLs are not going to be interesting to you, and you gain a lot of benefit by just generating ubiquitous language and working on that, which is quite enough work on its own for most people, it seems, and seems to be beyond most projects these days, but yet this is really the heart of what the objects are about. In many ways, DDD is just getting back to the core of what objects were about in the minds of the people in Simula and Smalltalk communities in the early days, before we all got distracted with other stuff. And domain-specific languages can help enable beyond that, and it need not be as complicated as a lot of the stuff that the DSM people talk about, I think that community tends to overcomplicate things in many ways because you can do a lot of things much more simply, and the fluent interfaces are a good example of a simple way to think about things in terms of domain-specific languages.

ERIC EVANS
You know, there was one thing that you reminded me. In a way, it's that value that is at the heart of DDD, as you were saying, the value of "we think first about the understanding of the domain, and other things flow from that". And so in a certain sense... and a lot of details flow from that, but people get hung up on a lot of those things. Like, we can spend the rest of our time talking about the difference between an entity and a value object, if we wanted to, but in fact that's really the core of the matter, and then I find that in a very close orbit around that is the language that people use to talk about the domain, so that's the ubiquitous language.

UNKNOWN 1:
I want to say that certainly the unified concept between all these little islands in the archipelago is importance of capturing a domain. Therefore the fundamental activity is modeling. But let's not forget that at the end of the day what is really important now to us is the final customer, the person who wants their problem solved. At that point is where these languages come into play.

One of the things we talked about in this panel that happened this afternoon, "Domain-Specific Languages: another silver bullet?" is, for instance, how do you determine importance of DSLs. One concept that was put forward was "is there a quantum leap in productivity", is there a quantum leap in the quality of the things that are produced. So the thing is that we should not forget that there are different components that need to be placed. Yes, it is important to understand the model, and I would argue that we don’t know yet how to do that. Just take a look at the word “Ontologies” that is so tightly associated with the domains, and that opens the whole can of worms or Pandora’s box if you will. But let’s not forget the person for whom in a certain way we are working, which is essentially the person who is paying for our time to solve the problem.

ERIC EVANS
Naturally, and of course one of the things I often point out is that I don’t think deep modeling is necessarily the right answer to every situation. I think that in fact we have overused the idea of modeling, and in the process we’ve watered it down. So we go into every project and say “everything is an object”, “everything will be modeled”, etc.
(MORE)

ERIC EVANS (CONT'D)
But what a model really most useful for? It is really most useful when you have sort of intricacy to the problem, and usually that intricacy isn't throughout, it is usually knotted in certain hot spots. And so one of the things that the technology is not really good at is allowing us to apply this kind of deep analysis in certain areas, and in other areas just slam some kind of 4GL type of solution.

Fitting those two things together unfortunately is very difficult with the common toolset. Obviously we have to solve customer's problem, in fact I would also say that one thing to keep in mind is that for any problem there is not a model, there is some infinite number of possible models, or, if it's not infinite, it's extremely large. And we are just choosing some model that addresses the particular problems that we are trying to solve right here, and that's one reason that I am always harping about concrete scenarios: I don't want to model this thing in general, I want a small set of concrete scenarios that tells me what's the hard thing that you want to attack.

MARTIN FOWLER
...and that's the linking to behavior-driven stuff. The way I look at behavior-driven design is that there are two components, two parts to it. One is just replacing the word "test" with the word "should" in your unit test which is boooring. But then, there is another part, which is the much more getting into the scenarios and stories and trying to express those in the way that we connect to the domain theme.
(MORE)

MARTIN FOWLER (CONT'D)
Because in the end, the biggest issue in software development, in my mind, is the gap between developers and domain people, the “yawning crevasse of doom” as Dan and I have talked about. And the key to Domain-Driven Design is that it attempts to cross that by building a bridge through the ubiquitous language. And everything that’s a part of that is the key part. And that’s what really, really interesting, and that’s what brings that whole notion of scenarios. So I wanted to make a connection through that to Behavior-Driven Development, we talked about it earlier on, and that to me where the link comes in.

PETER BELL
That’s something that we noticed: when developers first come across it, they get so excited about the idea of doing this ubiquitous language, but they go and try to learn the entire domain, and then they’re saying “wait a minute, what use cases do we have to deliver, how much of the domain do we need to understand to have some balance” - you can model too little, but I think it is also possible to model too much. One of the issues with that, though, and that’s I think where DSL people will have something to talk about as well, is how do you handle model evolution, because if you only model enough for a use case at a time, you are going to find that your domain concepts are going to evolve. Let’s say you have ten thousand statements in a DSL or code or how else you’re implementing that. How do you allow your grammar and your concepts to evolve in a way that you then can automate the transformation of your existing code so you are not going to break stuff. A specific problem we had - we were developing a set of domain-specific languages for generating rich web applications.
(MORE)

PETER BELL (CONT'D)
The problem was, I got to the point that I just didn’t want to do any more because I was finding that every time I change the grammar of my domain-specific language I had to throw away thousands of statements that I had carefully developed because I couldn’t automatically migrate them. So, I think it also ties into the idea of agility in how do you create tooling and approaches for evolving your grammar, whether you are implementing DSLs or however you handle that so that you are not scared to model just a little bit and you can use YAGNI and just develop one step at a time.

ERIC EVANS
I couldn't agree more with that. As I say again and again, you do not understand your domain very well in the beginning. If you are stuck with the model that you came up with at the beginning of your project, you are locking in your ignorance from that day one. Modeling is a learning process, and that would be a tremendously valuable thing that the technology developers could give us, would be to make something easier to change. If you use internal DSL, the refactoring tools do give you some of that, at least it is easier to rename things and a few other changes, but refactoring tools don’t really have the concept of a language, the kind of transformation they got are more conventional way of thinking, it sort of stimulates you to think “wow, I wonder if there are going to be refactorings that might be more language-like?”.

MARTIN FOWLER
But it’s also about how you build the pieces. At the moment we are still in the situation where most people don’t know how to build DSLs very well because the information isn’t out there.
(MORE)

MARTIN FOWLER (CONT'D)
Just as people have struggled with a lot of domain-driven development stuff because Eric's book wasn't out there and there wasn't very much advice as to how to do that.

DAN NORTH
Ubiquitous is a really dangerous word, I think, and it is in danger of becoming massively overused, like using a golden hammer to beat people with. The big thing that Domain-Driven Design gave me is this idea, at least the idea of the ubiquitous language, gave me was that it allowed me to articulate what I was trying to do with BDD.

What I am trying to do is create a language that describes how software gets written, trying to create a way of describing scenarios, and scenarios themselves, the words that I use to describe scenarios are a domain language of the domain of writing software. For instance if I am describing the behavior of a web app I immediately got two domains I am describing. I am describing the behavior of actually operating a browser, so selecting things, entering text, clicking on things, and above that the domain of the problem I want to solve. I want to log in as Bob. Logging in as Bob involves coming up with a sequence of web-type instructions. If I have those two domains, I understand that those domains are related. If I then want to replace my web app with a rich UI app, I still have the domain idea, the ubiquitous idea of logging in as Bob, it just now means that I do it though a different way, I am still expressing the same intent.
(MORE)

DAN NORTH (CONT'D)
And when you get this type of web driver type languages, if your language is “here are thirty things you can do with the browser”, and now we want to change the things we can do, and now I have to go through and change all of our scripts - doh! If instead you say “these are the kind of the atomic things I can do with the browser”, now I can start tracing the abstractions over those things. So entering a name and address is quite a common operation. If I have entering a name and address as a thing I can do, and that enters my vocabulary, then the way I implement that is the beginning of regular software development now, the beginning of good abstractions.

ERIC EVANS
Yeah, but you are talking about building abstractions out of other abstractions...

DAN NORTH
...language abstractions out of a language...

ERIC EVANS
...which is so key to modeling processing. It is one of the weaknesses of the external DSL techniques that have been put out so far. One of the nice things about the internal DSL is that you have all of the usual abstraction tools of the language, so if I create this kind of internal DSL, I can easily use terms in that to define new terms, so building abstractions out of other abstractions, including...

DAN NORTH
...internal DSL. It’s lovely to do. There’s a friend of mine, Simon Stewart, has written a thing called “Web Driver”. It’s a really simple Java interface that describes how you drive a Web browser. And using implementation of that will drive Internet Explorer ...
(MORE)

DAN NORTH (CONT'D)
headless mode. You can run all of your functional browser tests through this Java thing, you just code it. But then, because it’s just Java, it’s an internal DSL. I can create abstractions over that, and I can put those things together as scenarios. And it’s so expressive when you are looking at it, and you can refactor it. I can pull out big lumps of it and turn it into something else. Wow, we are suddenly getting refactoring at a language intent level. And that to me is ... language ubiquity. If we mean the same things by the same things.

There is something I wanted to pick up on what you said earlier. It’s not just doing the same things better, it’s doing different things. You get this emerging behavior, people stop being able to talk very differently.

DEEPAK GHOSH
I think one of the things that stood out completely for me in Domain-Driven Design is human-to-human interaction. It enabled human-to-human interaction, which usually reduces as I see people more and more going into their own silos. The way I practice Domain-Driven Design is on the white board. So I white board it with people. Unless I want to interact with people who are the stakeholders and with their own agendas... it enables the human interaction, and that’s why it works. I am still far away from DSL and all that stuff. I think what is working for me is this collaboration that it enables automatically. But then, I have some limitations that I found. This limitations are around... when I use DDD and these modeling technique, I can figure out the key entities and the surrounding entities of the business system.
(MORE)

DEEPAK GHOSH (CONT'D)
And it also helps me to understand the associations between them, how these entities collaborate. But the business process still seems to be on the dark side. I could not excavate the business process very clearly.

MARTIN FOWLER
Well how long have you been modeling it?

DEEPAK GHOSH
With the Domain-Driven Design? It’s been a year..

MARTIN FOWLER
After a year I’d think I’d begin to understand something. This is not a short process, this is not going and building a model in a few months. Eric talks a lot about how the key insights come a year, a year and a half into building the thing. This is something you have to evolve over time. It is really important to understand that, because I think people sometimes get a sense that “oh, I can just go in and model and quickly understand and run with that model. And “quick” is not a part of this process as far as I can tell.

DAN NORTH
You may find you start introducing a different language to describe the interactions between these characters in your system. I do a lot in investment banks, so I have things like a trade and portfolio, and this kind of stuff. But you also then have a settlement process. And when you start looking at the life cycle of a trade, what happens to it, you introduce a whole new vocabulary. And it is interesting to a whole different bunch of people. Because a trader doesn’t care. A trader goes “capture the data - bang! - I’ve just made a bunch of money. Next phone call”.
(MORE)

DAN NORTH (CONT'D)
There is a bunch of very serious legal people who do care. And for them... this is the thing again, this is the danger of ubiquity: there is number of languages, there is a number of models that will overlay each other. And yet, absolutely, your characters need to understand your domain, the static domain. Then you need to see what happens to them, how they interact. And that may well introduce a whole other domain. And that's ok, I think.

ERIC EVANS
I think that the point you are both making is that these dynamic aspects, they add into this model. I mean, I would say that I view that as one model, the static and dynamic parts are woven together. A rich language describes those two things fluidly in a way that you become less aware, I think, or less conscious of what's dynamic and static. But I just wanted to follow up on the point that Dan made about the multiple models, because I realized after a couple of years that a lot of the contents in chapter 14 probably should have been in chapter 2. Nobody ever gets to chapter 14. So they read the chapter 2 about ubiquitous language, but they never get to chapter 14, where it talks about how there will be multiple models on the project, which carry different languages with them, and the importance of recognizing them and bounding them and clearly identifying their relationships. I think that that's one of those things that, as Dan was saying, as the term "ubiquitous language" becomes ubiquitous, I hope that this idea of bounded context of a model starts to travel along with that term, so that people realize that any kind of set of concepts exists within some defined context.

DAN NORTH
...not necessarily. An example I use is booking a holiday. Now, when I book a holiday, I think of a holiday, of vacation as, like, a beach, tequila, that kind of stuff. When my HR department things of booking a holiday, they are thinking of People Soft, the whole business process they do. I don't even want to know what that world is like, I necessarily want those to be different domains.

ERIC EVANS
Well, they are different models of the same domain.

DAN NORTH
Yes, yes, totally. So I just said that the concept of holiday, a vacation is shared.

ERIC EVANS
...and if you mix them together you will have a mess, but if you have these two as distinct models within distinct contexts, then...

MARTIN FOWLER
...and that's why the global object model, global data model is doomed.

ERIC EVANS
Yes.

UNKNOWN 1
So there is this very, very tough problem that I have still unsolved and I haven't seen practical solutions to it, which is a problem of interoperability among models. You can live within the "comfort zone", where you develop your own model, you see your versions, and everybody means the same when they talk about the same concept, but then you get out of the comfort zone and you get people who mean completely different things, and guess what? In any organization you can have many different software products.
(MORE)

UNKNOWN 1 (CONT'D)
You have invested money in them, you have invested money in a workflow system, you have invested money into an ERP system, and you want the workflow system to send information to the ERP system. How do you solve it?

ERIC EVANS
Yeah, this is exactly the context map problem. Those two systems speak different languages, they are based on different conceptual models, whether they have been explicitly modeled or not, they are based on different conceptual models. So when you send a message from one to the other, that message has to be translated. And I think we have to just say there is just no such thing as something that means the same to everybody. And if just let go of that idea, and concentrate on what is the context within which I can interpret this language, and then I have some translations between this language and whatever other language I need to translate into, then things return to sanity.

MARTIN FOWLER
Makes me think actually of a fascinating conversation a while ago with... I won’t mention his name... one of the leading SOA authors... he was talking about the importance of modeling the domain, getting every and all of the inconsistencies ironed out so you get a single consistent map for the whole domain before you go and begin your SOA work. And it struck me that there are two very different attitudes here. There is one attitude that says, in order to handle the world, we must simplify and make... not necessarily simplify, but make everything coherent and logical.
(MORE)

MARTIN FOWLER (CONT'D)
And there is another view of the world, which is we can’t make everything coherent and logical because the world isn’t that way, we just need to figure out how to cope with incoherent and illogical world. I am going to be on the panel tomorrow about silver bullets and things . What strikes me is that one of the best allies that the werewolf has is the desire for the silver bullet to make everything coherent and logical, because that causes even more trouble, one of the best things the werewolf has.

DAN NORTH
...something I’ve noticed ... in the last couple of years is the gaps between the contexts are themselves a context. So in other words you get people who are experts at the translation you are describing, and I go to those people because I know they know far more about how to make “a trade” over here mean the same as “a booked deal” over here. And I don’t understand the subtleties and the nuances and the business rules, and they spent their life knowing that stuff. So there is like a really deep domain of being good at the cracks. And that itself is an emergent vocabulary. ... They don’t necessarily need to know what you are going to do with that information, they need to know that this translates to that, and that you find it useful.

EINAR LANDRE
...we were able to say “divide and conquer” and isolate, that defines the major contexts, and also the interfaces between the contexts ... can be domains in themselves. Of course, they are so thick that there is too much going on, and there are issues regarding how to place the boundaries, so you make the interfaces as thin as practically possible.
(MORE)

EINAR LANDRE (CONT'D)
We put together two experience reports published at OOPSLA last year discussing some of the experiences we had with architecting an information system we had build and also how we applied the same technique for ... evaluation to find out how these models in that package fit to the environment we should deploy it into, and it basically rules out ... all packages because it didn't fit. So that's the context mapping, but we probably didn't say that. It's the hidden secret of Eric's book...

ERIC EVANS
...hidden on page 400 or something.

CHRISTOPHER O'CONNOR
I really like the idea of that man in the middle [Einar] because I think I heard a lot of discussion here at the SOA workshop and other places that are talking about getting enough semantics ... and that's a part of the context, so if you have two different contexts and you have semantics associated with both of them, that's a really important concept that you have somewhere that's in between that somebody that knows enough about both to feel comfortable getting in there, and that's another aspect to keep track of ... [some gibberish I can't make sense of]

DAN NORTH
I think there is a really dangerous vendor anti-pattern in that space. So I will go back to Martin's lovely metaphor earlier in the year with ferrymen and bridge builders ... we got this gap and we got geeks on one side and business people on the other side and we wanna get these people talking the same language. You can have a ferryman going across so you can say "we have to translate", we have to translate between these domains. You know what?
(MORE)

DAN NORTH (CONT'D)
We call it “enterprise service bus”, I’ll sell you one. It costs a small fortune. And that’s what it does, it does translation. All we can say, we want to make the problem of translation go away by collaborating and getting these people speaking in a shared understanding of one another’s context, and kind of making the gap really, really really tiny. So there are kind of two ways to solve that...

ERIC EVANS
...there is pragmatic balancing of those things. If you say “oh well, that’s just another context” every time anyone disagrees about anything you get this fragmentation, pretty soon you can’t say a complete sentence without switching languages in the middle. And on the other extreme you have the enterprise model attempt to force Esperanto onto the world. So I think there is pragmatic middle ground where you say, do we have a big win out of making these people speak one language; do we have a practical possibility or are these people never going to? Because it takes a lot of work to maintain a ubiquitous language, so you are going to declare your context boundary to encompass both of these groups. It means they have to put in the work to agree on a single language and then maintain it as it evolves, and that’s a lot of overhead. So I think this is a pragmatic choice to be made, depending on how much value there is and how much difficulties there is.

DAN NORTH

...I am not trying to suggest we bring them together and give them a common language, just that they understand enough of each other's domain, enough of each other's language to understand what happens at the edges, but they don't need to know anything other than that.

ERIC EVANS

...Although I am big on constraining that knowledge implicit in a piece of software, like I want my objects to be ultra-specialised, I want to have these distinct contexts and within them to have only one model , and that model is very de-coupled from other models in other contexts and so on, but I don't think that way at all about the people involved. I mean I want the person to know as much about the whole situation as possible. There is no need for people to respect the boundaries of a bounded context in the knowledge that they acquire. I mean, we are way to compartmentalized as people. We have to keep reminding ourselves that my boundaries should be much broader than the boundaries of the software I am writing. I need to understand the context within which this software operates, and that's much bigger than the software. And I think it's weird, I mean saying it out loud it sounds ridiculous, but I've observed that people really do make decisions about what knowledge they should acquire very much in alignment with the knowledge that will be in their software. For example, one of the strict things that we do in agile is we don't over-engineer, we don't try to accommodate cases beyond the ones we've got. But people extend that to saying "I shouldn't understand those other cases, I shouldn't understand the business beyond these little cases, these little stories".

(MORE)

ERIC EVANS (CONT'D)
I couldn’t disagree more, I think in order to understand a story, really understand a story you are working on right now, you need to know a lot about environment that this story is happening in. So I think people should know as much as possible...

MARTIN FOWLER
Encapsulation is for objects, not for people :-) I have to head off...

ERIC EVANS
I’m glad you could join us.

UNKNOWN 2
...in many fields the domain experts will not speak any programming language. So if a DSL is to be a language for modeling the domain expert’s view of the world, it mustn’t be a programming language but an analytic language, and evaluation of this language means to project from analysis to procedure, to operational logic, starting with a language that has a declarative logic known to your domain experts and that exists independently from your software.

ERIC EVANS
Exists independently from the software? Can you give an example?

UNKNOWN 2
Yeah, sure. An evolutionary biologist tells me how the population he wants to study looks like in terms of how many genes are involved and so on, I can make him a computer simulation, and if we agree about aspects of this model, which he can vary in terms of ...
(MORE)

UNKNOWN 2 (CONT'D)
how many genes are involved,.., if we can agree on this sufficiently narrow descriptive language for specializing his scientific theory, I can build a meta-program that will produce a computational simulation for every scenario he is interested in, and at the same time he has the terms that already have a definite meaning in his scientific community, but he has no idea about any kind of software, or at least he doesn't need to have.

ERIC EVANS
So, I do agree that domain experts are not likely to learn a programming language, and this is one of the things that I think, I think that the idea that wish that the purposes of a domain-specific language will ultimately be that domain experts could write their own programs, I think that's a red herring. No, I am agreeing with you, I agree that the goal is not to make a programming language that domain experts understand, or let's say that they can write, anyway. I don't think I entirely followed the whole thing you are saying...

UNKNOWN 2
...let me try it this way. If I think about this kind of analytic language, I would rather think about profiles about set calculus or algebra than profiles for UML. At least in such mathematical disciplines as engineering or biology the people are absolutely able to give you sufficiently precise and formal model of the scenario they are interested in to allow the automatic production of a computer program they want to address some of the questions they ask. Of course you have to know what kind of questions they're asking...

ERIC EVANS
How is that not a programming language then?
(MORE)

ERIC EVANS (CONT'D)
If you state something that fully specifies a program, then it’s a programming language.

UNKNOWN 2
It doesn’t fully specify. Maybe something that is more commonly known. Say these guys describe an engine, a combustion engine...

PETER BELL
To me the real distinction with this is that when you are working with business experts often, as we found out, you don’t really know what you are doing. They don’t really know how to say what they want. Perhaps the distinction is that in certain scientific communities...

Break for tape change

DAN NORTH
... if actually the problem I’ve got is to have some really useful defaults in a bunch of fields on this green screen, and that would make me three times as productive, then don’t re-write the entire look [??] but give me a super-duper swanky web app thing.

ERIC EVANS
Right. Although that last bit there is a good example of what I meant when I said that not every situation calls for DDD. If what you want to do is to streamline data entry, you go out there and do more of a usability study and look at the problems that the data entry people are encountering. They are probably not deep domain problems, they are little things like entry fields aren’t in the right order, or this thing can be calculated from these two fields, and little stuff which...

DAN NORTH
...you say they are not domain problems, I say they are, I think they are symptomatic of poor...
(MORE)

DAN NORTH (CONT'D)
If the guys designing the system had realized these two things are actually the same thing, you would have got [??]...

ERIC EVANS
That’s true, and of course my bias is towards seeing everything as a domain problem [laughter], so I am probably trying here to adjust my own bias.

I want to go back for a second to this politics question because I honestly agree with him when he said you are just not going to get the politics out of this. What we are doing goes too close to the bone of how people’s businesses work, and therefore we are just in the middle, but on the other hand if you can’t avoid something, then maybe you can face it head on and manage it. That’s another thing about the context mapping. I would encourage people to take a look at that chapter 14, even if they have to skip over the 200 pages before that which they haven’t read because it really talks about, like, there are models within contexts but what are the relationships, because a lot of that revolves around politics. Just to take one example: I see a fairly common occurrence of a team that says, well, ok, to do what we need to do for out little bit would be very nice to have this additional feature over here. So let’s go talk to these guys. “We would like this additional feature” - “Oh, that makes a world of sense, yeah I can see why you need it, well, we’ll make it for you”. And then they proceed and they write their stuff, and at some point, boom! - their project is delayed, because these people didn’t do that. And why didn’t they do that?

(MORE)

ERIC EVANS (CONT'D)
Because they had good intentions and they wanted to help, they wanted to make that feature, but because of the political dynamics of the whole relationship was not such that it was their primary focus, and they didn't get around to it, and not by any kind of in-fighting, in fact, these guys actually wanted to. And they go back and they say "you haven't done it!", and they say "oh, we feel terrible! Ok, we'll get to that, we are going to get to it right away!", and then they still don't do that, and they just feel guiltier and guiltier and guiltier, but that doesn't make it get done. What makes it get done is either you change the structure of things politically in terms of who they report to, what their assignments are, or, if that's not going to happen, then you change the relationship, the context, say "we will do this ourselves, within our own model, it won't be as good a solution, theoretically, but it will actually happen". And if you look at the realistic lay of the land, you can make better decisions about stuff like that.

DAN NORTH
The word "politically" has a bunch of negative baggage with it. There is one other thing I've seen, really just makes me smile with domain-driven design, and that is particularly my current plan is investment bank, we are doing the service tier ... but we are trying to do it by talking to the guys using the service tier and engaging them... is in trying to get a shared domain, in trying to understand what this language looks like, we've uncovered loads and loads of bogus received wisdom, a bunch of waste, extra effort stuff in their existing process.
(MORE)

DAN NORTH (CONT'D)
It’s not a complicated system, it’s capturing trade data and pushing it through to some system that’s going to do some settlements, but there are lots of little nuances to it. And it was when we were trying to get a shared vocabulary to describe these nuances, we were like “but why is it even there??” - that’s just a huge piece of work we don’t have to do. And as you were saying about silos - the person confines themselves with they work with in their little silo world, and in fact this is how they used to develop software: the manager would know how the whole thing was going to fit together, like “Eric, you are going to work on these four classes” - “Ok, I’ll just go and type that, shall I”, and so everyone was in these silos, there was no communication across the silos. So we came in and started to ask a bunch of dumb questions, like “what does that mean?”, this vocabulary si changing form bit to bit, how do these things work? And a bunch of duplication just fell away, it was joyful. Literally, we deleted about 30% of this code base in the last four months, and it works, it does more quicker...

ERIC EVANS
Deletion is so satisfying! That’s great!

EINAR LANDRE
Basically what you hit on there is to identify the problem, and very often, there is one old architecting heuristics that you should never accept a requirement as valid up-front. And the guy who said it was the chief designer of the F-16 aircraft, because the US Airforce, they requested a Mac-3 fighter, and that was impossible to build within a reasonable cost. And he went back to the generals and asked them, “why do you need a Mac-3 aircraft??” “We want to escape from airfight”.
(MORE)

EINAR LANDRE (CONT'D)
Ok, that boils it down to thrust versus weight ratio, which is an engineering problem we can solve. We need a light aircraft with a really big engine. And that was that. And you need, you have that type of process where something is stated. And even though a domain expert states that - airforce generals probably have a good idea of what they want from their aircraft - they were not able to articulate what they really wanted.

DAN NORTH
Yes, rather than giving me a list of features I have to deliver, tell me a list of problems I have to solve, because maybe between us we can come up with a really clever way of doing it. I had a system recently, or rather one of my clients had a system recently where every couple of days it would crash. And it has some memory instability, and every couple of days it would crash, and it was a huge issue because it was a live trading system supporting very expensive very unhappy traders. So what he did was he rang an operations desk and he said, every night around 2 a.m. you should restart the system [?]. And they said, sure!

Problem solved in a stroke because it always lasted longer than a day. So now the system’s stable again, now the developers can work out what is wrong, like do memory profiling. So the outcome was to stop that thing crashing rather than the traders coming out screaming at the developers “you have to fix it NOW!”, and he solved the right problem.

UNKNOWN 2
This raises question how come domain-specific model capture what he told you about the problem he wants to solve, but does it capture what you design to solve his problems. Is it in the problem domain, does it mean you need a formal transformation into the solution domain, or if your domain model is the model of your solution.

ERIC EVANS
I know that people make that distinction and talk about the model and the solution domain and the problem domain. I think we should throw that out. Honestly, I think we should have one model, and that model would be harder to find because we want one that describes the problem well and works well as a solution. In other words, I want it all, and I want it all in one. And it’s hard to find, but that’s where the big gain comes from, because if you have two, and you have to translate between the two, how do you ever know that you actually did it right, how do you know that your thing really means the same thing as his thing.

UNKNOWN 2
Actually, I would agree that we need one model, but everything that happens after having formulated the model should be formal and automatic.

ERIC EVANS
Yes, that’s the idea...

UNKNOWN 2
The idea that the problem model somehow supported transformation to a solution model ... but the question is, do we have a rich model that allows us to define our solution of the problem as a part of the model? Or is this a part of the transformation that is defined in the meta-model?

ERIC EVANS
Yeah... and you are describing, I think, a kind of elegant and very complete solution which for most projects right now isn't within reach. But there are not as elegant, not as correct, but nonetheless very useful ways of writing, let's say, a Java program that really does express faithfully a set of concepts which we have expressed in richer formats like conversations in English, for example, or more rigorous things, so I think that right now this situation we have, we have to deal with expressing our ubiquitous language in ordinary programming languages. But that's what we would really like from these new technologies, of course. By the way, I think one of the keys is that we have to get there in steps. I have seen attempts to do what you just described. And they haven't worked out. And I think part of the reason is that we are trying to leap to far at once. Even with some of the things the MDSD people came up with, which are somewhat more incremental. But yet, they make me give up too much. I look at some of these things and I say "this is really nice, I would like to have this capability, and all you are asking me to give up in exchange to this capability is EVERYTHING that I've ever made work in the past. Everything that I know actually I can trust I have to give up in order to have this thing you have given me". And so if we can have the tool people think about, how can I add without taking away so much, that would really be helpful, I think for them to have this mentality of saying "these guys have a lot of tools they've learnt to use and make work..."

DEEPAK GHOSH
I have another horror story, probably the last one to share.
(MORE)

DEEPAK GHOSH (CONT'D)
That is, again, on domain knowledge excavation. There are two types of difficulties that I've faced, I keep on facing. One is - the person doesn't speak up. There could be various reasons why he is not speaking out the right things, and the next, which is more horrible story, is that the domain has got embedded into a software system, so it is enclosed there, it is not in people's heads any more, because people who were there at the time, they have left the company. So when I ask for a requirement, they say "go to the SQL database and start firing queries so you find the rules". I say - HELLO, I cannot excavate. So I think I fall short of one chapter that you should be writing now, which is domain excavation, domain knowledge excavation.

ERIC EVANS
That's a good one. I look forward to your first draft of that. And I'm kind of serious about that. I think that other people starting to write about Domain-Driven Design now, other people beside me, and that's good because it takes a community of people to make something like this, not just me.

By the way, I wanted to introduce Dan North, kind of little late now, but he is Behavior-Driven Development guy, and remember I was saying there were all these drivens and different domains, so that's one of them. It was really good to have him in this conversation.

DAN NORTH
Quick plug: I am doing a Behavior-Driven Development tutorial tomorrow afternoon. It says with JBehave, but it's mostly about BDD and a little bit about JBehave, and probably some stuff in Ruby as well.

(MORE)

DAN NORTH (CONT'D)
The reason Eric and I were talking quite a lot is, like I said, Domain-Driven Design has given me a way to articulate what I am trying to do. I had this huge realization. I was trying to coach TDD and I was having real problems with it. I'm a sort of neuro-linguistic kind of behavioral psychology type person, I like all that stuff as well, how people think. And I though I change the words, so I changed the words and started talking about software behavior, saying "we are not writing tests, we are writing examples of behavior, we are writing enough examples of behavior, and then write enough software that the examples of behavior work. And then I had this kind of ah-ha moment, with a guy Chris Matt, he is a business analyst, where we realized that you can use this behavior-driven kind of approach, so you capture stories and features at a business level, or stakeholder level, and now I can capture requirements in terms of behavior, and I could automate those, and then suddenly I am moving into automated acceptance testing space, and that's quite exciting. And my model was that there was this continuum of behavior, from the kind of code object interacting level right up to domain application enterprise interaction, and there was this continuum, and all you have to do is stand back far enough and squint. And it turns out I was wrong. What I realized is that it's not a continuum, it's a bunch of kind of shells if you like of different domains, and they flow one into the other, they translate one to the other, or they encapsulate one another. The example I was using earlier on is the business objective is to log in, is to identify myself to a system. The application interactions may be a bunch of clicking and typing things in.
(MORE)

DAN NORTH (CONT'D)
The code level behavior may be a whole bunch of stuff altogether. So I've got three levels of description of behavior to get stuff done. Now, without the vocabulary of DDD, how can I possibly start to explain that? So it's helped me to articulate what I am trying to do. And... yeah, thanks!