Relevant Modernization

Mark Schroeder interviews Stuart Milligan from Databorough. They discuss modernization of legacy applications using a rational approach in automated modernization.

Mark Schroeder:
We're going to talk today about software modernization using the Databorough approach. First I'd like you to tell me a little bit about your experience with software development and modernization.

Stuart Milligan:
My software development career started more on the business side, in that I was an implantation and business consultant in textiles. I'm actually a textile engineer originally, by profession. I got involved in textile manufacturing and operational software and ended up working for a company that developing it. I became a developer. Then within two years, it was about '97, we were heavily pressed to actually modernize the technology. Written in Synon and RPG and running on an AS400 back then, about five, six, seven million lines of RPG and Synon code. So there was heavy influence even back then. I got more and more involved in the modernization part because of my engineering background in the sense that you had a problem that was fundamentally technological and an engineering approach would produce far better more consistent and more productive results and that's really how I got into it. Then we got into more specifics in modernization practices where rather than strategic modernization, such as completely re- factoring or re-engineering a system looking at tactical aspects of the technology for specific implementations and that covered a broad set of subjects from RPG re-factoring through to go from 36 to 400 modernization, database modernization, database enhancement, user interface modernization and then re-factoring over all of systems to try and make them run in a more economic fashion from a development perspective, but also from an execution perspective. That really covered a lot of different software aspects because when your working in any RP situation, you have different pressures of different usability issues. What I mean by that is that in the inquiry, sort of in the manage information type mode in any RP application, user interface and quality of data is incredibly important and is more user driven. Whereas when you get into the more batch processing you have application, built up materials and more manufacturing type processes although the batch logic is very, very important. When you do modernization you come at those problems from different angels often and they present different problems to you. I then started working with Databorough on a project, on a big installation and a modernization project that I was doing in India and it solved a big problem for us as that we were localizing a huge ERP system for a big corporate manufacture of textiles vertical plot. All the way though from raw materials to finished goods. Including distribution and logistics and Databorough helped us analyze this gargantuan application in a very, very effective and quantitative way. This meant that we could take very clear and concise management decisions on exactly which parts to attack first and also have a pretty accurate prediction on what it would take and what effort and what times would be required to do what we did. Yeah, so that's sort of it in a nutshell.

Mark Schroeder:
It sounds like you have extensive modernization experience and that's a passion of yours.

Stuart Milligan:
Yes, I'd say you sort of get drawn into it because it's a big challenge. You know when you look at a big program and you look at it from a programming point of view it's interesting, but when you look at it from an assistant point of view when you got thousands of programs, millions of lines of code you've got design of technology, you've got design of system, you've got modernization, you've got the pressures of business, you've got time, you've got people. It's an incredible challenge and it's sort of an engineer's dream in a lot of respects in that a lot of people don't like to work with things that exist, but I do. I like to look at re-engineering as a way of taking something that's very, very difficult and actually doing something fairly unusual with it, so yeah, it is quite a passion.

Mark Schroeder:
That's a big problem right now. You know a lot of companies have big applications out there and they just need to find some way to handle that and move it into a modern world.

Stuart Milligan:
I mean that's it. It's the size of the problem, really that's the big challenge. There are new technologies and new methods and new resources becoming available all the time, but essentially it's the complexity and the dimension of companies applications that really is the big problem. As I said, not many people are really that interested in vesting a huge amount of emotional or intellectual energy in looking at an old problem, but companies have that problem.

Mark Schroeder:
Most of our financial transaction are still accomplished on COBOL. Those are not brand new systems anymore.

Stuart Milligan:
Well, you know what's interesting? I heard a forester research quote a couple of years ago that said "On any given day as much as 80% of any of the world's financial transactions actually takes place on a mid-range system somewhere. The majority of that was either in RPG or COBOL

Mark Schroeder:
That's sure a lot of work to do. A lot of these are IBMi shops. IBMi is a pretty strong work horse out there. What are some of the issues that they face in modernizing their software?

Stuart Milligan:
In terms of priority there's a lot of problems, but I think that the two fundamental ones are that there's a diminishing set of resources that are available to people wise and skill

wise to actually solve the technological problem of understanding theses large complex applications sufficiently to make informed and productive enough decisions to actually take the application asset forward. The other problem is that there really is not the culture of tooling for re-engineering in the world. A lot of people looked at re-engineering back in the early eighties and the philosophy of case from a Ford engineering perspective. A lot of people poo pooed it because they didn't feel that it was possible to produce the optimum system, but in a modernization context, tooling becomes incredibly important. So, most companies really lack resources. They've got a very large and complex problem and generally they lack the tooling. Those are the three most important factors and the challenges that they face.

Mark Schroeder:
There's a lot of information they need in order to determine how to modernize. What kind of information do they need to look at?

Stuart Milligan:
Well, this technology is one thing that they need, but really more importantly they need to understand where they are. Nobody can really make an informed decision about where they want to go, unless they know where they are, and that sounds like a very vague statement, but what I mean by that is that you can not plot any real position and root in real predictable and risk adverse terms unless you have a very precise understanding of your exact position. You've got to quantify that. So, it's knowledge about what you've got and how much you've got and how complex it is. It's also facing the reality of relevance. A lot of the companies that I've worked with in installations, in modernizations and various projects over the last 15 years or so…Many of them make assumptions about what goes on in their Legacy application. Then when you run a tool like X-Analysis or you do a bit of analysis work with them, you reveal to them that they had no idea were in their application. For example a big batch calculation does an allocation or an invoicing calculation. You know, maybe the original programmer that wrote that just hasn't been around for ten years and they've just relied upon that technology. Suddenly they're faced with a situation where they say "well, we want to modernize these codes." Well instead of looking at things in terms of bits and bites or lines of source code and guess work, you want to quantify that. So, that's a very, very important aspect of where you should start.

Mark Schroeder:
This is such a complex endeavor you're working on. How can you calculate that complexity and how does that fit into what your gathering and the information you're getting to modernize?

Stuart Milligan:
There are more scientific methods of calculating a complexity. Halstead volume, cyclomatic complexity. These mechanism are really used to look at individual programs if you'd like. You can aggregate them over entire systems, but they look at it in a very, very scientific way. They're probably some of the more extreme approaches to measuring complexity. They're useful and they will give you an indication. Probably a more

commonly used approach is to say “well how many lines of source code have we got? How many objects have we got and how many data files? How many programs, how many display files?” I think there’s also another important factor that you should take into consideration and those are relevant design constructs. In AS400 or IBMi Legacy applications there are things such as copy books which can contribute to the complexity of the number of sub-routines. The lack of sub-routines versus the number of source lines that you’ve got. Implying that you’ve got an incredibly convoluted program that doesn’t have any structure to it or very little structure it. The number of indicators, right hand indicators and left hand indicators. These are all contributing factors, and the important thing in any complexity or subjective analysis if you’d like because complexity is a subjective calculation even if you do it scientifically. What you’re after is something that’s more representative of something that you can do something with the result. In other words when I look at complexity such as Holstead volume, it’s great to tell me that I’ve got a great number of complex programs, but having generic complexity knowledge is not that useful compared to say I’ve got 1500 files, all of which need externalizing or we need to understand the relationships. So what you’re trying to do when you use complexly calculations in a modernization context is trying to find information that more clearly represents the nature and the dimension of the problem that you’ve got in more sizable chunks that you can consume and then use. In other words, you don’t have too much detailed information. At the same time you don’t want too much guess work. It’s somewhere in the middle.

Mark Schroeder:
IBM does help with some of that research. They have some commands such as display program references, display object description, display file description. I assume you use these commands in your process. How do they fit in and how do they help you map an application?

Stuart Milligan:
Well, the nice thing about the 400 is that it’s got a lot of that type of information available to you. So, at an object level it is actually really quite significant. One of the big advantages that companies like us and various other tool vendors have had over the years is using that. The source change management tools use it to make sure they get all the objects in, etc…etc.. Now , it can only go so far though because dealing with things at an object level is fine, but it’s a bit like trying to fix a watch with a barge pole. It’s a bit rough and it’s not as accurate as you’d necessarily want things. One of the reasons for that is the inference of design. What I mean by that is that the source code can infer a relationship between two objects or a variable for example, that maybe significant in business terms and design terms, but isn’t obvious when you look at the object to file or object to object relationship. So, those commands are very useful, but they have a limited amount of use if you really want to get to accurate forensics and analysis results. Just to make a point, the extrapolation of information that is fairly course grained means that you have very little predictability on the probability of it being accurate. So, in other words, if you’re doing a proof of concept where you take a small sample of code and you say this represents things and you extrapolate it out using some linear formula, generally you’re going to get it wrong. The more granular and the more detailed basic forensics that you

can use, the more accurate your result or predictable your result can be. Despite program reference, display object description and all these built in commands are really useful to a point, but you really need to go a lot further and that means looking at the source code.

Mark Schroeder:
Looking at the source code. So, you have a lot of people reading the source code or do you have a way to look through that through the programs?

Stuart Milligan:
Well, there's a good way to represent this. Imagine if you wanted to re-discover America and you needed to get across America quite quickly to get a job done in California. So you arrive in Boston and you get off the ship and you say "right, let's go". Do you want to take a hundred people with you that can write down notes of where they go and how they go about it and then have debates every night on exactly which way you should be going. Or do you want to sit down and say "hang on a minute, let's find a GPS or a map and find out where everything is and then let's just drive around." I think the latter is applicable. The analogy that I'm drawing there is that if you do things manually in an analysis, you will get it done and you will collate information. A much easier way, at a more fundamental level actually have a ready pre-built map of exactly the terrain, the geography and the area that you're going to be transversing or you're going to be assessing. In doing that, it means the group of people that you've got can rather deal with the results of this basic forensics and map, rather than with the pressures of actually building the map itself.

Mark Schroeder:
So you build a map as one of your first steps in the whole process?

Stuart Milligan:
Absolutely, I mean that's the most fundamental aspect of it. Without the map, any subsequent analysis as I said earlier on, becomes a lot more guess work and you're extrapolating out its averages of averages and really what you want is the most detailed level of basic forensics that you can get and based upon that you can start making better predictions.

Mark Schroeder:
Once you have the map, does that go into the functional organization or does that go into the real guts of the program, or does it do a little bit of both?

Stuart Milligan:
It's a little bit of both. You know, that's a very interesting question because a lot of people think about application mapping and analysis as what some people in the industry currently call program understanding, as a way of mapping an application, and it is to some degree, but it's a little bit sort of basic. It's a bit like hand crafted maps versus a real GPS. A GPS takes pictures from a satellite and it takes them at very fixed and rigid intervals using a very, very rigid method and then calculation in order to bring all of these different grid references together. That's a very sophisticated approach and it means that

you want things at a very detailed level, but you also want them at a higher level at some degree. Building a functional map of a business application means that you've sometimes got to look at object level. Sometimes a particular piece of information that relates to a fundamental design or business rule if you'd like, in the entire system which effects a company might be one line of code. So you do need that map at multiple levels.

Mark Schroeder:
Once you've built the map at the functional level, what information does that help you provide and how does that let you proceed forward?

Stuart Milligan:
Once you've got this map of how every object inter-relates, there is a certain amount of business design or design information that is inherent in the way that an application is architecture. The entry points are usually the starting point in a business process The user interfaces formats for example are usually individual steps in a business process. The text that people assign to a user interface or to a program object usually is a summarized description of the overall transaction or the process. So there's a lot of information embedded in systems that can be used. The trick is to actually put it in a structured and cohesive map if you like use again or provide it and present it to consumers. Those consumers might be analysts, they might be programmers, they might be management. You want to present it to people in such a way that they can actually understand it in context.

Mark Schroeder:
I guess this map is going to go several layers down into the depths of the application. How many layers can you realistically go down and understand and present to someone?

Stuart Milligan:
Well, from an implementation point of view, you've got programs or objects if you like and data files and you've got display files if you'd like at that level. Then you've got source code and then you've got individual lines of source code. One of the things that we've advanced in terms of the science of analysis over the years, which is not just us doing it, it's the industry, is concepts such as business rules. Sub-routines represent usually a cyclic piece of logic that is almost self contained and is reusable and repetitive. I mean it's those kinds of design constructs that represent potentially another layering the system, but it's not always obvious when you read a source code program because of inconsistencies in programming methods and approaches, syntax, you know there are RPG2 all the way through to RPG3 in these systems and you've had upwards of maybe a thousand different programmers over twenty years developing a system. So, you've got to take all those idiosyncrasies away and actually then look at the layers that are in an application from a design perspective. Sometimes the objects of the source code and source lines are relevant to that, but they've got to make sense. So, what you want to build is a structured framework if you'd like information framework that allows you to traverse the layers effectively. The layers can almost be almost infinite, it's not infinite in that it may be a core stack that goes down 14 or 15 layers and then there maybe individual programs with source lines inside of there that you've got to draw down to. It

goes all the way from the bottom upwards from the source lines, individual source lines, all the up to objects, and then aggregating objects together into what we call application areas which usually represent groups of business functions and then going downwards it starts at the complete system. Then you can draw down all the way into say for example objects. Then inside of an object you can say well it's still got an individual screen format because that maps to a business process and then let's go look at business rule and then inside that business rule and then once inside of that business rule, let's go look at one of the conditions. So it's multiple layers and it's from multiple perspectives.

Mark Schroeder:
As your analyzing this, you have the application understood, but then you also have the database side. What aspects of the database do you have to map to understand what is involved in your database structure?

Stuart Milligan:
You need the physical structure, such as what the columns are the fields, you need the files, you need the views or the logical files or the access parts as they're sometimes referred to and these bits of information reveal things about the system. For example, if I look at a table or a physical file and it's got a logical view built over it with a certain number keys, it's a pretty good guess for me that someone wanted a report over that file based upon those keys. Then if I look at the nature of those keys, such as it happens to be customer in order, I know that generally this company that uses the application manages their business by looking at their customer's orders by customer for example. The database can actually reveal that. The other very, very important part is the relation information, the foreign key information. So, when a customer owns an order file, what is the foreign key relationship between those two which enables me then to actually build program constructs that will enable me to navigate it for update purposes for interrogation, for building business intelligence cubes and the various other things that you want to build around an application database. So the important factor, just to summarize then, is not just the physical attributes, but the logical attributes, which is the foreign key relationships.

Mark Schroeder:
One of the ways that you do this is by looking at the DDS. What kind of information do you extract out of the DDS and what kind of diagrams do you create with that?

Stuart Milligan:
From the DDS you can get a certain amount, but in RPG based systems, by design most of the field names are different, so the RPG can handle it. That means that by just looking at the field it's not explicit that there is a foreign key between two files for example. Usually it's a pre-fix, so in other words the last four digits of the six digit field will be the same between the files, but the first digits may vary, but on big systems you get a roll around. In other words they run out of field names and so that becomes irrelevant as well. So from the DDS you can get a certain amount, but then you need to get into the RPG code or the COBOL code because the implementation of the referential integrity in almost every single IBMi, RPG or COBOL system in the world is implemented in the

actual program code itself. So what you really need to do is read the RPG or COBOL code to derive those foreign key relationships.

Mark Schroeder:
You have to have that combination and the DDS itself then can't be directly converted into SQL, but you have to use a combination of analysis if I understand right.

Stuart Milligan:
Well, you can actually convert DDS to SQL without any problem, but the result it produced is limited by what you put in. DDS has a limit on what it has in it. It doesn't have the referential integrity…and also the tooling that's provided doesn't really allow you to view the foreign key relationships because they aren't explicitly defined. So one of the points that you asked that I didn't actually answer was what diagrammatic constructs. Entity relationship diagrams are one of the fundamental tools that any business intelligence consultant will actually use in order to build a data warehouse. For an analyst to understand how a business application works if it's a database application, which almost every single application is, the main part of the architecture, 80% if you'd like of contribution of the actual business architecture is in the data model itself. So entity relationship diagrams, class diagrams and diagrams that explain the database structure are very, very important. So taking it from DDS to SQL, you can do it, but you want to enrich that with other information such as long field names and things like that.

Mark Schroeder:
You would include your data model and your DDS together to convert to SQL?

Stuart Milligan:
Absolutely, that's exactly what we do so you end up with an incredibly rich application database architecture.

Mark Schroeder:
Application like 2E, I do quite a bit of development in that and the names for your fields and files are so cryptic. It's nice to be able to move to something that means something.

Stuart Milligan:
Well, you know what's interesting on the subject of 2E and modernization. Quite a few of our customers are actually rebuilding their underlying databases using our tooling by extracting the information from the 2E model and generating DDL from that.

Mark Schroeder:
And then they take the DDL and that goes directly into what format?

Stuart Milligan:
You just run it as a script, you can just run it in almost any database management system. You can run it in as MSQL, on Oracle, My SQL. You just run the DDL's. The DDL is actually data definition language and it is SQL code. It's the implementation, how you

implement the actual structures, the schemers themselves. So you run it in the script engine or the database management system that you're using or any other available tool and it will actually create the physical database for you. So, of course if you're enriching your DDL code or the SQL code if you'd like with all of this inferred design information that you've gleaned from the RPG code or COBOL code or the Scion model, you end up with very, very well articulated databases as a result.

Mark Schroeder:
We've talked about the application map. We've talked about creating the SQL. The application map, it can be used as an input into your re-engineering. How does this fit? What do you do once you have your application map?

Stuart Milligan:
You've got to sort of break it down into further constructs because modern applications really use what you might call model based programming. What I mean by that is not using a model to generate the code, but I mean that it goes to a pattern. A model view controller for example is generally an industry standard or some patterns may be model and viewed and may be controller and model in a batch type application or a web service based application. So what you're looking to do with the application map, is to abstract some of the detailed information up higher so you end up with constructs such as a data model or an object relational map…which brings together..an object relational map is a way of translating a database a relational database into an object oriented style architecture. So modern object oriented programmers and technologies can consume the data from there in a way that is more suitable for the object oriented approach to development…and so that's one of the inputs. So, in other words the data model, relational model is the fundamental way of actually mapping an object relational map. You then combine that with the information about how those entities or those physical files and views are used in a given program, which you could call a transaction for example. You then can build an object relational map that has a transaction dimension to it if you'd like. So in other words you know where the data for a given transactions and order entry program. You know where the data comes from. You know how it relates, you know all the join rules to actually build the different views and you can build the object relational map in a product like Hibernate for example or other modern industry standard like in hibernate which is the porting of Hibernate to C-Sharp. With these what they call persistence frameworks it makes the development process much easier. So in terms of taking that concept further, you can then say well if I've got this explicitly defined set of DDL that we spoke about a moment ago and I can generate the database can I use DDL to actually build an object relational map in a modern application framework like Hibernate?...and the answer is yes, you can import it directly. You can create the entire Java persistent API framework and all the classes and all the configuration files just by importing the DDL. So it's a nice stage and that's a simple example of how it can be used. You can take it a step further and say OK, in my Legacy application that I'm trying to modernize one of the other critical elements that is of interest to me is all these business rules that I've been programming for the last 20 - 30 years. They really represent how my business has evolved and how we've learned to do things more cleverly and it's almost a software implemented version of my business specification if you'd like. The

problem is it's all spread around your system, it's buried inside of RPG, maybe CL, maybe COBOL code, inside of Scion action diagrams. So what you want to do is distill that out into a form of business rules database but keep the context because the context is important. It's pointless doing a price calculation if your not putting an order in. So it's bringing the application map together with modern constructs. So you're bringing business rules together with the context of a particular transaction say an order entry and then if you can distill that by the application map which knows where all these applications are in such a way that you can then put it into a new tool or put it into the mind of a new developer through some descriptive process, a picture of a data model, a class diagram an activity program, a narrative that describes what the RPG code is doing. A picture and a narrative of exactly what the action diagram is doing for that given user interface that it's going to be used for in a modern application in a database that we've modernized now and put on my SQL or have just put a new version on the IBMi. So bringing those constructs together you can now generate the code to build the application framework from it. The last bit that I just mentioned is that the user interface information is very important The problem that you have with Legacy applications from 2E all the way through RPG COBOL is that it's procedural and modern development requires vent driven architecture. That's what model view controller actually is. In other words the user interface from some other interaction with the user either pressing a tab button or selecting a key or something like that will actually go away and actually fire off an event which may then execute a piece of business logic. The user interface precision because it is driven by a procedural program is actually quite different to get out and re-structure, but if you can distill the user interface information into a form of metadata and describe it, it means that you've actually got a very, very powerful bit of information that can be used to generate the skeleton. At a more fundamental level at what I would call fundamental modernization where you want to keep your application in RPG or COBOL, then what you would do is you can use some more fundamental constructs such as sub-routines. You can externalize your sub- routines and procedures and modules and then replace the execute sub-routine statements in your original program with cores to theses procedures. Then you can build simple stub programs if you'd like that then attach to the individual procedures in the modules and you can create a web service that then exposes all of those sub-routines and all of those procedures or sub-procedures if you'd like as a form of web service. That means that you've deconstructed the application and you've done all of this by analyzing the application model seeing where you've got the sub-routines, understanding their context, looking at the business rule and then re-achitetecting using the analysis information that you had.

Mark Schroeder:
That initial analysis you just mentioned really opens up a lot of open doors for you as far as what direction you want to go from that point forward.

Stuart Milligan:
Absolutely, it's quantitative isn't it? It's once you know where you are and you understand the value and the relevance. You can't ask a business user how important the tax calculation is unless you know what the tax calculation is.

Mark Schroeder:
One of the things that we do in programming and engineering is we look back at time and we've learned some things from one of the events that we call Y2K. What kind of things do you find relevant now for modernization that we learned back in those days?

Stuart Milligan:
Interestingly it's almost the same problem in a way. It depends what you mean by modernization. In completely rebuilding a system is an incredibly complicated and big task. In a more conservative context, such as something simple, you get a new standard a regulatory compliance requirement that dictates that you've got to expand a field. Now that's a modernization project and it can sometimes cripple a company for the simple reason that if I've got five or six thousand programs and I've got to expand my product code for internationalization or some regulatory requirement it might take me a couple of years to get through the coding just to make that simple change and that's very analytic of what happened with Y2K. You had a relatively simple problem that manifest itself in a very, very large way. So what we learned from Y2K, which was great for us as a company is that the majority of companies just weren't prepared. What they weren't prepared for is understanding the scope of the problem that they had and there after being able to technologically fix that problem…and that's exactly what we did and you know having a technology that builds an incredibly detailed application map, sort of an for an entire system if you'd like, with all of the ability to drill down and have all of the information at your fingertips. It's kind of straight forward then to write algorithms that can pro grammatically change the system. That's what we learned from Y2K, the fact that people weren't prepared that they didn't think that they had that much of an analysis problem until they were confronted with something as so significant as Y2K and you see this on a more regular basis so if you then interpolate it or retro fit it into the modernization question, when someone says "well how much or how long will it take to re-factor this system to make it so I can build a modern interface over it. People are doing the same sort of guess work that they were doing back in Y2K. When people would look at a problem with a database change they'll say well I've done the display program reference and I think that's enough, and they've missed all of the subtle inferences that might have made the project ten times bigger and so they're over run by ten times the amount of time

Mark Schroeder:
In a lot of companies they're still approaching re-engineering in a manual approach. What are the steps that are involved in a manual approach?

Stuart Milligan:
Well, if you compare the two different approaches it's sometimes easier to show this in a picture, but if you imagine the way that a programmer works is that they have a set of ideas of what they want to do. Then they attack one program at a time and fix that program and go down until they've fixed all the problems in that particular program, test it and move on to the next one. When you compare that to a re-engineering approach what you do is you find out all of the problems and then you look at them across all the

systems and all the programs that you've got, the individual. You then break those programs down into type, or the problems down into type and then you go and you change one problem at a time across all programs. The manual approach generally means that you're doing one program at a time and it can produce a lot of problems but that's generally the way that people do it. They have a general brief of what they want to find, they go ahead and they fix their program one at a time and then they go forward from there.

Mark Schroeder:
What kind of problems can the manual approach present to the company?

Stuart Milligan:
Well human error is the most obvious one. It's very time consuming. I mean a factory that manufactures cars is productive because they mass produce individual components one at a time. So the concept of automated re-engineering is not a new one and why should the software industry be any different? Because people want to be creative, because they want to finish one because it's personal preference? No, it's a commodity, you need to be productive, you need to get the job done, the company needs the result and so why not apply engineering principles to these changes?…and that's the most important problem you have with manual changes is the time constraints. The human errors are introduced. The other problem that you've got is you may take a particular approach and you may get half way through and then someone comes up with a better idea or they face a particular problem that actually could manifest itself in the programs that have already been done and you can't start all over again. So it means that you work and working in a linear way you have to stay in a linear way. So it means that you can't repeat things and you can't do them iteratively.

Mark Schroeder:
What are the benefits that you get from an automated approach and how does it solve these problems?

Stuart Milligan:
Well, you can do things again and again. You simplify each and every problem that you've got in a very straight forward fashion…because you're doing it in a very scientific and engineered approach it means the way that you quantify both how long it might take and what effort it might require, it means that you can have a much higher degree of predictability…and because you're automating it means that you'll need less people. It means that the result can be a lot more consistent. You have to test less. A lot of people don't realize that well if you get it wrong then it's wrong everywhere, but if you get it right once in one place, then everywhere is correct so you have to test less. That's another important factor. The bottom line just goes down to cost, if you can get a program to do something, that's the whole reason information systems exist. It's a lot cheaper than getting a human to do it.

Mark Schroeder:
Cost is really an important part in every company.

Stuart Milligan:
That's it…a lot of people don't get uncomfortable with upfront cost because they feel that if you do things gradually you just consume salaries of people that are employed anyway. A lot of companies manage their business on that basis is that it's just a … you know it's not a direct cost that doesn't go on the balance sheet. It's just an indirect overhead from a salary that's been paid…but the bit that they miss is that those people could actually be used to do a lot more productive and more innovative things.

Mark Schroeder:
Exactly, they know the business already, so let them produce better software with their business instead of re-writing all the old stuff.

Stuart Milligan:
Well, that's right …I mean how often do you go into an IT department especially in the S400 world and somebody's doing something that they think is a technical junket if you'd like and there's the in tray is piled high with request for reports that the users want.

Mark Schroeder:
Right, right, very often. You mentioned earlier a task, a modernization task of re-sizing the field. How can you determine what scope this project would take?

Stuart Milligan:
One of the problems that you have in any integrated large complex system is that you have to trace the impact right down to a variable level..and so when you establish the scope you can't just do it at an object level…you've actually got to go down into where for example if I am trying to change the customer number and the customer number is then used to chain off to another file to go and read the orders for example and then that's used to go away and actually build some statistics in another file…and you think that well I'm only changing the customer file, but suddenly you've got to change the orders file then you've got to change the statistics file. These fields may have a different name and once you change the statistics files you've got to change all the order of the statistics programs that read it….and so it's very difficult and that's just at a file level and what about the variables that it uses. Imagine you've got a batch program that's doing some kind of statistical calculation or tax calculation based upon the value of order lines and you change the price field for example. All that calculation, all those arrays, all those variables are also actually not going to work. So it's like in a way it's Y2K all over again. In fact that's exactly what it is…and so if you have an application map and when I say an application map, I'm not talking about something from the stratosphere. This is where you look at the application map at the most detailed possible level which is the variable to variable relationship in context so I know that the relationship between that variable and that variable is of this nature. It's an update, it's a move, whatever it might be it's a definition…if you've got that type of map instantly available in a structured format, it means that when you ask this application map a question about the scope of a problem like a database change it can go across the entire system and literally come back in seconds…I mean I've had instances where I've done projects with people, where they ask

me the question , you know when we install our products and somebody says well how would I do this?..we've just finished this exercise and even with an older cross referencing tool and it took us four weeks and we do it literally in eleven seconds in our product because the application map has got it available there. You've got the information at your fingertips. It's a bit like Google earth, ..how do I get there, what's down there, where's the car parked. You can press the button and zoom and you can jump in your car and off you go.

Mark Schroeder:
That saves a lot of time then because if someone was going to look at all of that manually, they'd spend hours and hours and never finish it and your tool is able to do that type of cross referencing?

Stuart Milligan:
Well, that's what the application map is used for and then we've written inquiries over this application map. So there's sort of the scientific approach if you'd like of what we do is broken up into two main sort of sections. One is building the application map and that means reading RPG, COBOL and Sign On systems at a very, very detailed and comprehensive way and understanding the interferences that is made by compute statements and action diagrams and foreign keys and primary keys and all that sort of stuff and putting it in the map and the other thing that we do is write technologies that we then use to write in that reports ..to re-engineer systems, to tell you where things are. To draw color coded, interactive diagrammatic pictures of the architecture of a series of programs and that;a how we actually bring that together.

Mark Schroeder:
And then with that information, then you have enough information to write a program that actually does the work for you is that right?

Stuart Milligan:
Absolutely, remember that I said a minute ago that in an assembly line, in a car manufacturer you make all the wheels in one part of the factory. You make all of the cylinder heads in another part of the factory and then you assemble them in another one. Well, then we take the concept of knowing what the change is in very clearly articulated terms. We understand by virtue of the application map exactly what that means at every single line and every single program across the entire system. It's very easy then to write single programs that go and do simple changes at that level across all of the programs one at a time. When I said all of the programs one at a time, I meant one type of problem fixed across all programs one at a time. Then what you're left with is just measuring and reporting where you've got exceptions where you might have a problem. That's the concept of writing re-engineering technologies.

Mark Schroeder:
Sounds like it would save a lot of time for a lot of companies to use some of these products.

Stuart Milligan:
It's interesting, once I think people have used an application map approach down to the level of detail that we're talking about, you never turn back. If you look at all of the case studies that we've got on our web site, just speak to any one of our customers and their like "wow, I wish I had this six years ago. I mean imagine living without a GPS, without a map. Imagine an airplane flying without a GPS system. You just wouldn't do it.

Mark Schroeder:
As you go forward and continue to add to your application, having this map is just going to help you understand the application better.

Stuart Milligan:
Well, that's right. There's an important part looking at the reverse of that is that, you know, with a diminishing set of resources and a lot of companies looking to off shore to actually reduce costs you increase risk. You increase risk because you've got less resources to do more things because you've got more code because every year your writing more code and you're making enhancements. The minute you go off shore you're saying, " well, I'm giving up my application knowledge base because these people don't know my application, but if you've got an application map that can give someone that hasn't seen the system before, very detailed and very clearly, concise graphic color coded information about the design of an application instantly as it currently stands…it means that you've solved that problem too.

Mark Schroeder:
Another area of modernization is going to Unicode. That's something new out there that a lot of companies are having to go to. Is Unicode pretty much the same as modifying a field or are there other steps involved with that?

Stuart Milligan:
It's almost identical actually. You know it was interesting, we have one technology, the X-resize Component which was sort of a descendant, if you'd like of our year 2000 product that we had and Unicode was a spawned out of it X-resize too. It's just another form of it because what you're looking to do is add the necessary code to the database itself, and then you're looking for the display files and wherever it might be relevant in a program to actually update the necessary CCSID codes in there too. So what you're looking at in pure engineering terms is you're looking to do a little bit of intelligent search and replace. That intelligent part is the important part, that's why you can't use a normal set of text or any of the current set of RPG code editors to do it, because to do it across the entire system it has to understand context and more precisely what you need to change. Essentially it's the same principal as field expansion and or Y2K.

Mark Schroeder:
All right, well you have a certain set of tools at Databorough that help companies modernize. Can you give me a little bit of overview? You know we've talked about them already throughout this discussion. But how your tools help with defining, building and implementing a good modernization approach?

Stuart Milligan:
I think the two sort of aspects, the way to represent the technology is in two sort of ways. One is active and the other one is passive. The most common use of our products over many, many years has been the passive use and that's the analytical aspect. In other words building a very, very structured map and then allowing people to go in and actually map their business to it. So your doing what I call a top down analysis and mapping it with the bottom up…and so X analysis which is our flagship product has been around for 23 or 24 years is really what does that. It something that builds the application map and then it gives you client based tools that enable you then to go and draw pictures, instant documentation, look at business rules in structured English. See pictures of structure charts, data flow charts, entity relationship diagrams and all that. So you've got this incredibly powerful, and it is a bit like a GPS Imagine that the repository that we build over the business system that we're analyzing, it's rebuilt, it's ready to use and that's the information a GPS would use for example…and then x analysis the client enables you to then draw down and look at this interactively and navigate your way around it or produce reports that show you the completeness of structure. In other words all of the changes that need to be made all of the data modeling relationships, all of the user interface formats and their structure. The structure of programs, where an object is used and all that sort of classic sort of stuff in an interactive and graphical way…and just another point to make that a lot of people miss the relevance of…color coding is a very, very important aspect in anybody's life. Our brain responds to color before it actually understands what the eye is actually seeing. So before your brain can conceive what your eye is actually looking at, your brain will already respond to the color that it sees. So having graphical documentation over a large complex application can mean that you can assimilate architectural information that helps you decide on a modernization strategy in terms of what does this program do, is it relevant, what is the underlying architecture, how is this code written. All of that information is much easier assimilated by looking at graphically and color coding. In terms of the active side of our business that's using an incredibly detailed application map and an active context to do things like converting the database from DDS to SQL. We can do that literally over a weekend for very, very large complicated systems and you don't have to recompile a single RPG program. This is not rocket science in a sense that if you read the IBM red book it tells you that you can do that….But because you've got large complex systems and because we subtly bring all the information about the system together and we automate, in what is essentially a simple process…But because the systems are large, because they're complicated, because there's quite a few got yas in such a process, it means using an automated tool you get less risk, you get a higher level of automation, you've got less testing requirement and you've got high levels of confidence and that's how we can use our tools actively. Including things like the field expansion where we can take a field and expand it and automate it across the entire system and literally save companies millions of dollars. I know the one case study that we've got at the moment, the company used the product last year, the previous year, they did a field expansion project and they reckoned it saved the m at least 13 to 15 man years of work by using our technologies to do it.

Mark Schroeder:

Wow, it sounds like your tool paid for itself there over again many times.

Stuart Milligan:
Yeah, I think we should ask for a higher price.

Mark Schroeder:
Aside from your tools, when someone comes in and becomes a customer of yours. What kind of support do you provide them?

Stuart Milligan:
There's two parts, you know this is really an engineered and scientific approach to software maintenance and modernization..and a lot of people aren't that familiar with using tools, power tools like this in their consciensness. It's not proprietary technology or proprietary methods we're trying to teach people. We can help people utilize the tools by getting them involved more effectively and we do that through basic training. Another thing that we do is that on many customer's systems there are idiosyncrasies with the coding constructs that are just unusual. You know COBOL is more typical of this but often in RPG we find that too. We have a very, very good reputation for coming and looking at systems and coming up with work around and fixes and enhancements to our own product sets that make it really work as productively as any system using our products without any real glitch and we've got a fantastic reputation for that..The last thing that we're actually really good at is if a company gives us their system, because we've been analyzing systems, literally thousands of them must be trillions of lines of code over many, many years..and we're able to look at systems and come up with pretty good code review and assessments on tactical approaches and even strategic approaches on how they can modernize their code, re-engineer their system and but also in terms of how to use the product to map the relevance of the software that they've got to their business process and what I mean by that is that when a company is looking at their business and saying, "well what are my business requirements and then turning around and saying what does the software currently do and how does it map to that. We can help them with that process too.

Mark Schroeder:
So in the process of their modernization, they could actually end up with a lot better business logic and business application then they had before?

Stuart Milligan:
Oh absolutely, well that's the whole point. You don't want to do things that are unnecessary but you don't want to over-engineer what you're trying to achieve as well..Often what happens is because people try to do things one at a time is they tend to…and when I use the term over engineer. What I mean is they're trying to perfect what they're trying to do in one hit. Now that's a typical technological approach, but if you take an engineered approach what you're trying to do is get the actually lower hanging fruit. So you get as much modernization done for as little effort as possible and therefore get the maximum gain in the shortest amount of time and that's what we're really good at. We've got people at our organization that have done huge re-engineering projects that

we‘ve produced often. But sometimes we’ve done with our products and haven’t….and that expertise is quite unique and I think that is something that companies value, especially in the context of a very large complicated application with potentially millions and millions of lines of code that they’ve got to do something significant with, in terms of a change..and so to come up with a quantified intelligent re-engineering approach is really, really worth it’s weight in gold for a lot of people.

Now when they’re re-engineering is there a target language that you make them go to or can they use any of the acceptable modern languages of this
time?

One of the things that is very important to us and always has been is that we follow mainstream. We can’t and we don’t want to build anything that’s proprietary, because the industry that we work in now moving into mid-range systems if you’d like in VB and Java, but really it’s about S400 application code. People are looking for industry standard technologies and so if you’ve got an application map, if you’ve got business rules database, if you’ve got an entity relationship diagram, if you’ve got a DDL, if you’ve got a class diagram, if you’ve got an activity diagram, it’s got to be presented and communicated and utilized in such a way that you can generate anything out of it. Otherwise, you know, we would have a very limited use for people.

Mark Schroeder:
Sounds good..it really provides customer’s some good flexibility then.

Stuart Miligan:
Absolutely, it’s just a good foundation and a way to start. You know the foundation product, the passive use of X analysis and getting a very clear picture of your data model, your business rules, the user interface or what you might call the process logic from a user perspective of your system is an incredibly important piece of your system to quantify.

Mark Schroeder:
Now that people are all ready to go and buy your product..what’s the best way for them to get a hold of you?

Stuart Milligan:
Probably through our web site, we have various partners around the world. A number in the U.S., IBM, there are a number of IBM experts. IBM Rochester have copies of our products, various ISV’s but the easiest way is just to contact us directly through the web site www.databorough.com and then just look for someone in your region or just contact us directly.

Mark Schroeder:
Stuart, thank you for your time. I've enjoyed the conversation, I've learned a lot. I look forward to following your product and seeing how it can increase companies in their modernization approach.

Stuart Milligan:
Thank You very much.