Update

Enterprise Architecture Advisory Service
Executive Update Vol. 12, No. 15

A Forensic Approach to Information Systems Development:

Part II — Ways to Fix the Problem

by Ian Bailey, Senior Consultant, Cutter Consortium

In the first of this two-part *Executive Update* series,[1] I took a swipe at the currently accepted approach to systems development. My argument was that if a system is to adequately support a business, the information it handles must be rigorously derived from the business itself. By producing a process model, then an information model, then a data model, and then handing it all over to an implementation team, we can end up somewhat removed from the reality of the business. The people responsible for each of these steps in the chain usually don't have a good understanding of each other's specialities, and the result can be "Chinese whispers."

I also noted there are a number of legacy systems out there that are decades old and attempts to replace them with modern technology have failed. The fact that the old systems are so useful is perhaps more of a mystery than the fact that today's technology seems to offer nothing to beat them. Another trait of the these old systems that have stood the test of time is that they (mostly) seem to have been developed inhouse, in the days before there was a specialist IT function in the business. This is even stranger. How can a system that's 20 to 30 years old and developed by a bunch of enthusiastic amateurs outperform the latest technology, designed and developed by highly specialized information technologists and business analysts?

EVOLUTIONARY DEVELOPMENT

It seems these older systems were developed at a time when most work was done on paper. This meant the systems did not become "mission-critical" until well into their operational lifetime, so there was a lot of breathing space in the early years to adapt the systems to business needs. There wasn't a huge requirement to get it right the first time, and the system was allowed to adjust and adapt over time. What emerged from this evolutionary process is a system that provides an accurate model of the business.

The older systems have another advantage over modern systems — they do not have as many complex layers of software to worry about. They don't have middleware and GUIs to recode every time something changes in the database structure; the forms used are, by and large, dictated by the structure of the data. It might not look as nice as a Windows app, but when they were "current technology," it was much easier to make small changes to the information structure. The lack of knowledge about these systems today means that is no longer the case, I should add.

With modern systems, we don't have that option. Electronic information management is the default way of working, and it is not acceptable to deliver a less-than-perfect system and then test some new tweaks every few weeks over a couple of years. In a modern IT environment, we have to deliver a less-than-perfect system and keep it running 24/7. Any tweaks we make to the database will lead to additional work to the middleware and user interface. The tweaks can have unpredictable effects due to the complexity of the systems, often introducing bugs and sometimes even corrupting data. The result is that it is often easier to change the way the users work than to change the system.

Users have to do their day jobs, and whereas in the good old days they'd just tweak the system if they found a problem that was restricting their ability to work, now they have to find workarounds for inadequate systems. Most common among these is the practice of putting information in the wrong field. IT folks

tend to call this a "data quality problem," but it's often because the system does not provide the fields the user needs, so they're forced to break the rules in order to capture essential data. In Part I, I wrote that I was amazed at the quality of the data in the old network databases. A lot of this is due to the need for parsimony: they made extensive use of fixed classification codes, and there were few free text fields. I also think the high data quality has a lot to do with the fact that, in the past, problems were fixed in implementation, whereas now they're fudged in operation.

There are some interesting initiatives around to help make systems more adaptive. SOA is usually sold on the back of an "agile" business justification, and it's true to say SOA should provide a greater degree of flexibility in the business as a whole. SOA doesn't solve the information problem, though, and in some ways is yet another layer of complexity that will be affected by changes to the underlying information structures. In fact, given the opaque, late-bound nature of a true SOA implementation, it could result in an unpredictable chain of dependencies on the information: the more applications that subscribe to a particular information service, the greater the impact of a change in its information model. Dynamic programming languages offer another interesting technology. Languages such as Ruby offer the possibility of changing code in situ. Both approaches have great promise, but they don't tackle the information problem.

GETTING IT RIGHT FIRST TIME

So, given that (at least for now) our options for evolving a modern production system are limited, how can we make sure that the system we develop more closely matches the requirements of the business? Again, I think the answer might lie in the old systems. The really old ones have been honed and adapted to suit the business and often work really well. The more recent legacy systems tend to have data quality issues. I think that in both cases, these systems present a forensic gold mine for systems developers. Instead of producing indefensible process and data models, why don't we look at the data the users produced? I would argue that an extensive set of legacy data (no matter how dirty) provides the best possible indication of what the business does.

The problem with this is that the "gold" is often difficult to identify — it's either buried in some arcane tweak or "bolted" onto the system, or masquerading as an inappropriate data entry in a field. For years, I tended to reengineer these old systems on an ad hoc basis, but now there is at least one formal method available. The one I know best is the BORO[2] method, so I'll use that as the example. I'm sure there are others.

The BORO method thrives on large amounts of complex data. It is a very formal and repeatable method, and its mode of operation is forensic. In a BORO analysis, the data itself is more important than the data structure that stores it. The cynic in me likes to think this is because we assume the old data model is wrong, but in fact the BORO method is entirely neutral. It relies on analyzing the facts that are present in the data, and from those facts, we reconstruct a new model following a repeatable set of steps. The resulting model is ready for implementation. To gain maximum benefit from the reengineered model, it must be implemented in such a way that none of the flexibility is lost.

The analysis is deceptively simple. To be repeatable and defensible, it has to be. The premise of the BORO method is that you can't rely on a term to accurately identify what it refers to. You can never be sure that a given word is understood in precisely the same way by different people or systems. Consider really obvious cases: the word "table" is understood differently in the context of a database from a context involving furniture. Cases can be much more subtle, though; one person's concept of table might include desks but not coffee tables, whereas another person's understanding doesn't include desks. This pattern tends to be reflected in systems, which is why the BORO method relies more on the data itself than on the data structure. To get a clear understanding of what was actually going on, you have to look at the data to see whether the users have included desks and coffee tables.

The BORO method is based on three questions (see Figure 1), asked in order. The first question is about figuring out whether the thing being analyzed is physical. Examples of this would be my car, the Eiffel Tower, the US, me typing this document, and so on. BORO uniquely identifies these things by their spatial and temporal extent. If two things occupy the same space for the

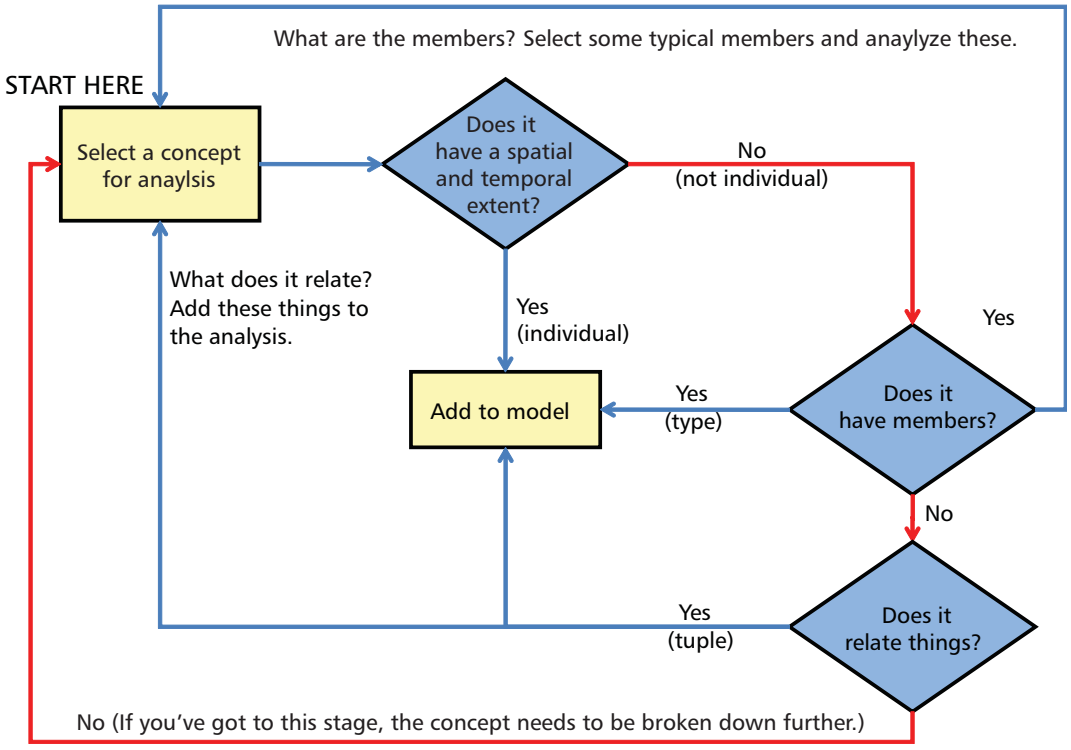

Figure 1 — The BORO Process (simplified).

same time, they are the same thing. This sounds pretty obvious, but you'd be surprised how many people rely on comparing the names of things without actually checking to see whether they *really* are the same. If it's not a physical thing, then maybe it's a type of thing. The test for this is to see whether there are things of *that type* (members). If there are, then it's a type. BORO uniquely identifies types by their members, so it's useful to pick some typical members at this point and add them to the model as documentation examples. As with individuals, names can't be relied on to identify them; the example of tables cited previously illustrates this well. Finally, if it's not an individual or a type, it's probably a relationship (tuple). Tuples are identified by the things they relate (again, not relying on names). If you've got to this stage and have not been able to answer yes to any of the questions, it probably means the original concept is a compound of the things and tuples, so it needs breaking down for analysis.

BORO is about getting away from our reliance on names and moving toward a more accurate way of identifying and categorizing things. Obviously, the names are important, so once we've built our model, we can reapply the names, but we can be sure what we're applying them to. BORO has one more trick up its sleeve. Names have owners, so a given thing in the model might have multiple names, each with different owners. This gives us a really accurate way to compare and integrate data from multiple sources. Two furniture databases with tables called "table" might not be referring to the same things. BORO will not only identify this, but will also show how they're related through simple set theory (see Figure 2).

From what appeared to be a trivial set of legacy data, a richer model of reality has emerged. Not only that, but the BORO naming pattern has provided a way to map the elements in the new model back to the legacy data in a very precise way. What emerges from a BORO analysis is, formally speaking, an ontology. It will be self-defining and extensible.

SUMMARY

Using BORO or a similar method to develop the information models for systems cuts out all the requirements drift that occurs across the analysis phases used in most systems development methods — from process model to information model to data model. BORO can also accurately identify how two apparently similar concepts differ, so it is particularly useful in building systems that replace more than one existing system, or that take feeds from multiple sources.

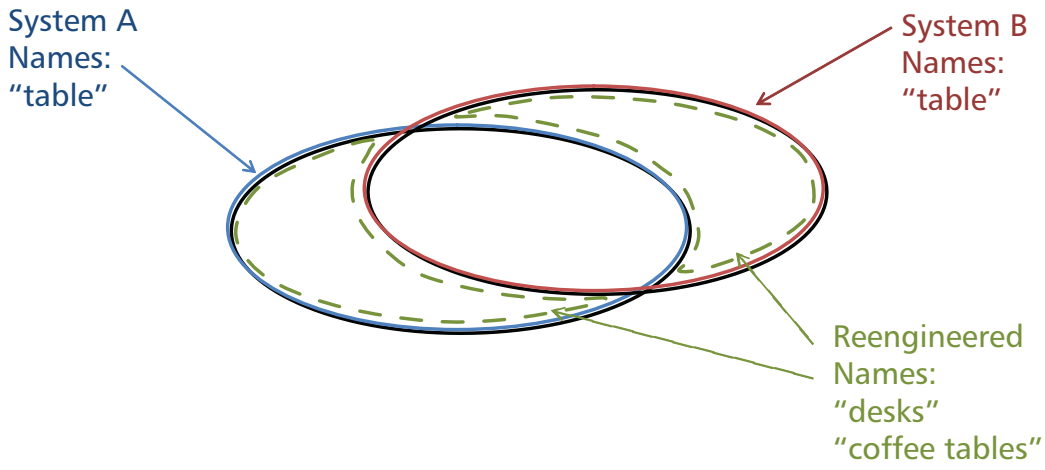

Figure 2 — Simple Venn diagram showing naming of sets (types).

I don't claim BORO and similar methods can solve everyone's problems. I chose to use BORO because I like its precision and lack of ambiguity. It also seemed to put a lot of the work I'd previously done with systems integration on a much more formal footing. It provides a repeatable, defensible process, grounded in sound mathematics and physical principles. BORO is a steep learning curve, and in many ways, it is more of a craft than a science. It has proved to be especially hard for IT professionals and information scientists to come to grips with; they have "unlearn" a lot of what they currently do. I've used BORO on a number of projects now and found that non-IT people get into the swing of it much more quickly.

The challenge, as we complete this series, even with BORO, lies in making sure that implementers don't cut corners. The BORO analysis will have exposed subtleties in the underlying data that were not previously explicit. The ontology it produces will be flexible and extensible. All this benefit can be undone at the point of implementation, so it is vital to ensure that the model is implemented in such a way that no flexibility or detail is lost. This *is* possible in relational databases but requires some lateral thinking. It is also possible to use an ontology engine to implement the model (several scalable, commercial-grade systems are out there).

In summary, then, I believe that methods such as BORO can really improve the systems development process — not only simplifying and reducing the development lifecycle, but also improving the quality of the systems that are developed.

ENDNOTES

[1]Bailey, Ian. "A Forensic Approach to Information Systems Development: Part I — Describing the Problem." Cutter Consortium Enterprise Architecture *Executive Update*, Vol. 12, No. 13, 2009.

[2]BORO Program (www.boroprogram.org); and Partridge, Chris. *Business Objects: Re-engineering for Re-Use*. 2nd edition. The BORO Centre, 2005.

ABOUT THE AUTHOR

Ian Bailey is a Senior Consultant with Cutter Consortium's Enterprise Architecture practice. He is an expert in military and government enterprise architecture (EA) who has done extensive work in applying EA techniques to information assurance and security. Dr. Bailey was the Technical Lead in the team that originally developed the UK Ministry of Defence (MOD) Architecture Framework (MODAF); he also led the development of the subsequent v1.1 and v1.2 releases. Dr. Bailey is the MODAF subject matter expert for the Unified Profile for DoDAF/MODAF (UPDM), representing UK MOD at OMG. He is also an expert in ontology development. Having worked with formal 4D ontologies since the 1990s, such as EPISTLE/ISO 15926, Dr. Bailey specializes in novel ways of implementing and applying ontologies. Currently, he is the Technical Lead in the IDEAS Group, a five-nation ontology development project that supports interoperability between national EA frameworks.

Dr. Bailey has worked on several EA, ontology, and information management projects for customers, such as AstraZeneca, BAE Systems, Boeing, BP, Ericsson, NASA Jet Propulsion Laboratory, Nokia, Shell, Statoil, Swedish Defence, Thales Group, UK MOD, UK NHS, US DoD, US National Reconnaissance Office, and Volvo. Prior to his MODAF work, he was Editor of the ISO/INCOSE AP233 standard for systems engineering. Dr. Bailey has a degree in engineering and management from Brunel University and Henley Business School and a PhD in systems integration, also from Brunel. He can be reached at ibailey@cutter.com.

