

# Packages Don't Let You Off the Hook!

By

John A. Zachman

Ó 2001 Zachman International

I hear people all the time say, "Well, we bought the XYZ (usually an ERP-type) package so therefore we don't have to worry about Architecture anymore, the package vendor has done it all for us," or something like that. I usually respond by saying, "Packages are really good, but they are NOT a substitute for Architecture." However, before I develop any ideas about packages, let me review some things about Enterprise Architecture.

## The System Is the Enterprise

Those folks who have heard me talk about Architecture have heard me say many times, "the system IS the Enterprise. The system doesn't merely 'support' the Enterprise, it IS the Enterprise. Remember what we have been doing for the last 50 years. We have been replacing the people of the Enterprise with systems. Were those people that were replaced NOT the Enterprise?" No! They were, in fact, the Enterprise. And now, the systems that replaced them have taken their place and have become the Enterprise.

Think about the Framework for Enterprise Architecture. I called Row 6 the "Functioning Enterprise" deliberately. It is clear from looking at the picture that someone (I/S for a lack of any other identifiable group of people) is taking what the Enterprise "Owners" have in mind as to what the Enterprise is, or what they would like it to be (Rows 1 and 2), and, through a series of logical transformations, they (I/S) are transforming Rows 1 and 2 into something very physical as the end result, Row 6, the physical manifestation of what the Owners have in mind, the "Functioning Enterprise."

Granted, some portions of the Enterprise are not what we would normally call "systems," that is, some portions are not "automated." The question is, "What is making the automated portions of the Enterprise any different from the manual, or not-automated portions?" The answer: "It is simply the technology constraints that are being applied by the 'Builder' at Row 4." If the "Builders" are using pencils, paper and file cabinets as the implementation technology, then the resultant portions of the "Enterprise" are what we would call, "manual," or not-automated. On the other hand, if the implementation technology is stored programming devices and electronic storage media, then the resultant portions of the Enterprise are what we would call "automated," or "systems." In either case, the "system" (manual or automated) IS the Enterprise.

When you buy a package, what you actually receive is a diskette that says, "Insert in Drive A, type 'install' and hit 'Enter.'" You are NOT likely getting any Row 1, Row 2, Row 3, Row 4 or Row 5 models. You are getting Row 6, the "Functioning Enterprise,"

the “system.” Therefore, you are actually buying someone else’s design for YOUR Enterprise, because IMPLICIT in the package are all the VENDOR’s ASSUMPTIONS about Row 1, Row 2, Row 3, Row 4, and Row 5 models for YOUR Enterprise. And ... you are buying an AVERAGE Enterprise at that, because the package manufacturer has a marketplace and that marketplace has a median and the package is going to be aimed right down the middle (the median), because that's the way they maximize the profitability of the package and stay in business. Plus YOU (the customer), now have to negotiate changes to YOUR (average) Enterprise with the package vendor on a COST PLUS BASIS, hoping that they remember what assumptions they have made about all the models in Rows 1 through 5, or else they will have to “reverse engineer” the implicit models from the Functioning Enterprise (Row 6, “running system,” “object code”) in order to make changes to the Enterprise, WHICH IS REALLY EXPENSIVE!

### **Reasons to Buy a Package**

Now remember, there are some very good reasons why you would like to buy a package in the first place. They are the same reasons why you would buy ANY standard product off-the-shelf like a toaster, or a Buick, or an IBM ES9000, or an F-14 Tomcat, including:

1. Immediate delivery (that is, off-the-shelf.)
2. Low per unit product cost (the manufacturer spreads the engineering and development costs across a whole bunch of products.)
3. High reliability/availability (standard interchangeable parts, that is, the same parts are used in all the standard products.)
4. Low maintenance costs (the vendor and even “other equipment manufacturers” can make a profit on inexpensive, standard, interchangeable replacement parts as well as on the product itself.)

But, the price you pay for all these benefits associated with a standard, off-the-shelf product is, you change the use of the product to fit the product. You DON’T change the product to fit the use. For example, when you buy a Buick off the shelf ... and you want to haul chickens in it ... you haul the chickens in the trunk. What you DON’T do is, you DON’T reverse engineer the Buick into parts and then re-engineer it into a Toyota pick-up truck. That would take you a hundred times as long and cost a hundred times as much as if you went to a job shop and ordered a custom pick-up truck to be built from scratch.

Examining the Enterprise side of the metaphor, the best of all possible worlds would be for you to have all the models of your business and for the package vendor to have all the models of their package. If this were the case, you could compare the models and figure out whether you really wanted to buy the package or not, that is, whether you really wanted to CHANGE YOUR BUSINESS to fit the package or not.<sup>1</sup>

---

<sup>1</sup> Note: if you try to change the package to fit the business, you likely will rapidly lose all the benefits you intended to gain by buying the package in the first place, including: immediate delivery, low per unit product cost, high reliability/availability and low maintenance cost. Plus, you lose the warranty on the package. You would be better off to go to your own applications development shop and ask them to build a custom application from scratch than to take a standard application off-the-shelf and change it into a different application.

For comparing models, you are probably going to have to make the comparison with “primitive” models, in Framework terms, single-cell models, models of one single abstraction from one single perspective. If the package manufacturer was like everyone else, and if they produced any models over the process of building the package, the models they produced were likely “composite” models, models comprised of components from more than one single (Framework) cell. “Composite” models are what are required for implementation purposes whereas “primitive” models are required for architecture. “Primitive” models are defined relative to the Enterprise whereas “composite” models are defined relative to an implementation. You will probably have to factor out the “primitive” models to be able to discern whether you have a “fit” between the package and your Enterprise or not.<sup>2</sup>

If you have already purchased the package, you probably will want to treat it simply as an addition to the legacy, which you will probably have to replace before too long anyway. Major changes are taking place in every marketplace and the half-life of all systems is rapidly declining. Also, the package (any package) is not likely to support the totality of the business. Therefore, there are going to have to be other applications (or other packages) and along with them, all the integration issues with all the other packages, as well as integration of all the packages with the legacy, all of which, if you care about the Enterprise as a whole, will require integration, which will require models, “primitive” models, at the very least, “primitive” Column 1 models.<sup>3</sup>

## **Reasons for Doing Architecture**

For the very large packages, especially those that have given you a range of parameterized options within the package structure in an effort to provide for some flexibility for customization, you may well have to build the models to get them (the packages) configured and installed effectively. In fact, Stuart Macgregor, from South African Breweries, made a presentation at the 1999 Zachman Institute for Framework Advancement (ZIFA) Forum in Scottsdale, Arizona, entitled, “A Model Approach to SAP R/3 Implementations” in which he discussed how they used the Framework and models in their SAP implementation and saved substantial amounts of time and money.

In general, the essential reasons for doing Architecture include:

5. Alignment - ensuring the reality of the implemented Enterprise is aligned with management's intent.

---

<sup>2</sup> For an extensive discussion of “primitive” versus “composite” artifacts, see my 2000 article “Architecture Artifacts versus Application Development Artifacts” which can be found at [www.zifa.com](http://www.zifa.com).

<sup>3</sup> By the way, if it was easy to integrate data after you get the databases built (either package databases or ones that you build yourself), you wouldn't have a legacy today. You could just integrate all the data and the legacy problem would go away. Unfortunately most databases that have been built up to now have been derived from “composite,” implementation models which, if they were not created from “primitive” models, are the likely source of the dis-integration. And, you simply can't change the meaning of the data once you get it implemented in a database!

6. Integration - integration especially in Columns 1, 3 and 6. (Col. 1 - the data means the same thing across the scope of the entire Enterprise. Col. 3 - messages are successfully transmitted and received across the scope of the entire Enterprise. Col. 6 - the business rules are consistently managed across the scope of the entire Enterprise.)
7. Change - Architecture is the baseline for managing change - change to any aspect of the Enterprise.
8. Reduced "time-to-market" - Architecture coupled with assemble-to-order process reduces time-to-market for Enterprise implementations (that is, reduces time-to-market for "application development") to virtually zero.

All of these benefits derive from Architecture and ONLY from Architecture. None of them derive from technology or packages or from any systems implementations, regardless of their origin.

### **Disturbing Perceptions of Today's Large Packages**

I recently received a letter that mentioned two very disturbing perceptions of today's large packages. The rationale for raising the issues in the letter was that since they (the Enterprise) already own the package, they didn't have to be concerned with Enterprise Architecture anymore.

Here are the two disturbing perceptions from the letter along with my responses.

Perception 1. "XYZ (the package) has already usurped half those cells (of the Zachman Framework) and therefore the package is our architecture."

My response: Oh???? Clearly, the package now IS your architecture. In fact, it is your ENTERPRISE! Have you seen any of those "usurped" cells, the package architecture that has now become YOUR Enterprise?? Who has them?? Do YOU have them?? Does the vendor have them?? Was the package actually "architected"?? That is, did the vendor actually do ENGINEERING work, that is, build the architectural models before they built the package ... or did they use the old "you start writing the code and I will go find out what the users (you) have in mind (later) approach?"

In fact, if I was buying a package today, I would really want the models, not simply the code. If you had the models, the code would be easy. And, if you had the models, you could derive the values of having Architecture (above) including: alignment, integration, change, and reduced time-to-market. Conversely, if you don't have the models, how do you intend to address alignment, integration, change and reduced time-to-market???

Perception 2. "mere mortals cannot presume to understand the intricacies of XYZ's internal integration."

My response: Good night!!! What IS the internal integration in the package? Does ANYBODY understand its internal integration? Do

THEY (the vendor) understand the internal integration? Where are the models that describe the integration? And ... who in the world would want to base their Enterprise's success on unintelligible, intricate and undefined integration especially when the implementing Enterprise is in an uncertain environment and change is the name of the game ... and you are dependent on the package vendor to maintain/change your Enterprise???

### **Assume the Best**

Let's assume the best for a moment ... let's assume that the package manufacturers figure all this out that:

1. What the customers really want/need are the "primitive" models.
2. If the package "primitive" models and the customer "primitive" models are a match, they have a guarantee of a happy customer every time.
3. "Primitive" models are the way to make integration understood and implemented effectively within the package and the only way to even understand (let alone do anything about) the integration issues with other packages and/or other legacy applications.
4. "Primitive" models are the baseline for managing change, whether the change is initiated by the customer or by the package manufacturer themselves, and
5. They can deliver their products (packages) ... and new products (new "composite" models) faster and cheaper if they had an inventory of "primitive" models (Architecture).

Let's go even further and assume that the Enterprise Architecture state of the art improves and enables us to identify the Enterprise's Business Rules, factor them out in such a way that we can manage them (as "primitives") quite independently from their implementation. This would give us flexibility to change the Business Rules independently of the semantic structures, the business processes, the Enterprise logistic distribution, the work flow, the events/cycles and the business objectives/strategies (Columns 1 through 6). (Note: this goes far beyond what is typically available today with parameterized options within the defined structure.) If a package vendor truly gave us this facility to manage the rules as independent variables ("primitives"), that would give us the flexibility to customize the package to fit the business simply by defining our own Business Rules to the package. That, of course, presumes that we know what the Business Rules are, which presumes that we had defined the Column 6 models and understood their relationship with the other Columns!

The Business Rules are a very interesting part of the puzzle because what is making one business different from another business in the same industry appears to be the Business Rules and "the devil is in the details!" If this is the case, the above Architecture/Package strategy is likely to be very effective. In the interest of accurately setting expectations, it is necessary to observe that if you change the fundamental concept of the industry, or if you change to a different industry, you may well impact the semantic structures, the

business processes, the logistic distribution, the work flow, the business events/cycles and the business objectives/strategies. Simply factoring out the business rules and managing them independently from their implementation would not be adequate. In this case, you likely would need a different package. Once again, if you had the models of your Enterprise and the models of the package, it would be relatively easy to discern whether the changes could be accommodated simply by changing the rules or whether you would actually have to have a different package.

Continuing to assume the best, that:

- a. the package manufacturers figure out that Architecture is imperative, and t
- b. the Architecture state of the art improves to allow us to manage Business Rules as an independent variable;

it still will not absolve us of the responsibility for Architecture.

One thing this would do however, would be to free us to concentrate our Architectural energy on Rows 1 and 2 of the Framework, that is, to concentrate on deciding what the Enterprise is and how it relates to its external environment as well as on defining it to a level of specification that it actually can be accurately transformed into the reality of a “Functioning Enterprise.”

### **Managing the “Primitives”**

It may always be advantageous to buy some “slivers” of some cells of the Framework, even of Rows 1 and 2, assuming they adequately represent our intent for our Enterprise. I doubt if it will ever be practical to expect any one package manufacturer to supply enterprise-wide models of all the cells of all the Rows and all the Columns for any given Enterprise. In this case, if there is more than one source for various slivers of various cells, you will need the “primitive” models to decide about and/or to effect integration where it is necessary.

For sure, I would want to control whatever slivers of cells in Rows 1 and 2 that I buy from a supplier. In fact, I don’t think I would ever want to lose control of the Row 3 models either, as they are the key to changing the technology with minimum disruption, time and cost. If you have no Row 3 models and you want to change the technology constraints, you are faced with a complete redesign and reconstruction of the systems. On the other hand, if you are managing your Enterprise Architecture at the logical level (Row 3) and you want to change the technology dynamically, it will not be effortless, but it would be doable, requiring only a re-transformation of Row 3 models to Row 4 under a new set of technology constraints (minimizing the time, disruption and cost of making the change).

In fact, I also think I would always want to know enough about the Row 4 models to evaluate a “General Contractor’s” ability to successfully transform my Row 3 models to Row 4 and to assess the cost, time and impact of doing so.

## **For the Time Being (Assuming Anything But the Best!)**

For the time being, my advice regarding packages is:

1. If you are going to buy packages, go in with your eyes open. Go in with the
2. intent to change the Enterprise to fit the package (not to change the package to fit the Enterprise!)<sup>4</sup>
3. Buy packages with models, “primitive” models.
4. Before you buy a package, at a minimum, compare your logical data model with the package logical data model. If you have semantic structural incompatibility, that is what you absolutely DON'T want to change in the package and that is also where you experience substantial pain in having to change the Enterprise to fit the package.
5. Get Architecturally competent ... this may take a while as it is likely to be culturally inconsistent with your current I/S paradigm plus, it takes practice (i.e. experience).
6. If you already have the package or are unable to influence the decision for acquiring the package, I would treat the package as one more legacy system. (I personally like the “Data Warehouse” as a strategy for migrating out of the legacy environment into an architected environment, but that is the subject of a different article.)

In conclusion, packages are NOT, and will NEVER BE a substitute for Architecture. If I understand the "Information Age" correctly, complexity and change are the name of the game and Architecture is the price of admission. Do NOT relax on Architecture! Now is the time to make your move!! Packages do not let you off the hook!!!

*John A. Zachman  
Zachman International  
2222 Foothill Blvd. Suite 337  
La Cañada, Ca. 91011  
818-244-3763 (Phone and Fax)  
johnzachman@compuserve.com*

---

<sup>4</sup> I heard a presentation at the June 2001, Province of Ontario's Showcase Conference by Debbie Barrett, CIO of the City of Mississauga, who said it cost the City of Mississauga \$1.5 Million to implement SAP whereas it cost one of the other cities \$42 Million and the other city finally gave up without getting it implemented. Debbie said Mississauga's secret is, in general, they only have “standards of 1” and for the SAP implementation, they changed the City to fit the package.