Architecture Artifacts
Vs
Application Development Artifacts

By
John A. Zachman
© Copyright 2000 Zachman International

All of a sudden, I have been encountering a lot of confusion between Enterprise Architecture and classic Application Development Work Products, probably because the stark reality of the Information Age is upon us! There are likely several reasons for the confusion. One thing is, historically, in the Industrial Age, we have not been very precise about the definition of “Enterprise Architecture.” To some people, Architecture is simply a high level description (model) of the system to be built. To others, Architecture is conceptual or logical as opposed to physical. To others, Architecture is “requirements” and to others, Architecture is simply “principles.” Probably the most confusing factor is that some of the work products (models) themselves that may be produced over the process of building traditional applications systems may actually look like architectural representations and it is hard to tell the difference.

Based on the Framework for Enterprise Architecture, I would suggest that Enterprise Architecture is the set of primitive, descriptive artifacts that constitute the knowledge infrastructure of the Enterprise. It is purely structural. I would further suggest that ultimately, the Enterprise will require that these artifacts must necessarily be graphically presented because at the point in time when you will need the artifacts, you won’t have time to read through a thousand pages of text to attempt to discern their implications. I am confident that at some point in time, the Enterprise is going to wish it had all of those design artifacts (models, cells of the “Zachman Framework,” the Framework for Enterprise Architecture) made explicit, Enterprise-wide, horizontally and vertically integrated at excruciating levels of detail because these models have everything to do with managing complexity and high rates of change, as those of you who have heard me talk have heard the strength and force of that argument.

Now at any given point in time, this total set of models may or may not be perfectly conceived or may or may not be complete or defined to an excruciating level of detail. However complete or incomplete they might be, to qualify for being called Enterprise Architecture, they must, by definition, be descriptive of the Enterprise, not merely descriptive of an implementation within the Enterprise. Since they, in their aggregate, would be descriptive of the Enterprise in its entirety, they could be recombined for any (or many) implementations within the Enterprise. This is the reason they could be called, “infrastructure.” They are defined (or designed) for reuse, or interoperability in their creation. The essence of infrastructure is that it is something that is going to be used...
In contrast, Application Development Work Products are created as inputs or outputs for the application development process. They are created for implementation purposes. They are process specified. They are created at and for a given point in time. They constitute the actual work products for building systems.

The basic question is, “are these application development work products being derived from architectural primitives that were designed with the entire Enterprise in mind, or are they being created to specifically serve the implementation process at hand?” If they (the work products) are being created for a specific implementation, the likelihood of their being interoperable or being reusable in subsequent implementations is low to zero, and the probability of changing them with minimum time, disruption and cost is low to zero as well. On the other hand, if the work products have been derived from architectural primitives, they are simply one temporal manifestation, that is one combination, or one composition (out of a virtually infinite number of manifestations, or combinations, or compositions) possible from the primitives. They are “composites,” likely composed of components from more than one primitive cell model of the Framework.

Reuse, or interoperability, does not happen by accident. It is the result of engineering. That is, reusability or interoperability has to be engineered at the outset. It occurs at the level of the primitive, not the composite. It is extremely unlikely that something that is already built (composed, assembled) is going to be able to be reused or interoperable in a different context, even within the same Enterprise. This is so easy to see in physical objects but we seem to have a great deal of difficulty envisioning this limitation for intangible objects like Enterprises. No one would even dream of being able to manufacture a bunch of carburetors and then expect to attach them at random to any Ford, or any Chrysler, or any Buick or any Toyota engine or any whatever. No. A carburetor has to be engineered to fit on a specific Ford engine in a specific Ford automobile. If you want the carburetor to fit on more than one Ford engine, it has to be engineered that way in the first place. After you get the carburetor built, if you haven’t anticipated “reusing” it in other “applications,” that is, if you haven’t engineered it for multiple uses in the first place, there is not much you can do to make it fit on any different engine, short of throwing it (and/or the engine) away and starting over again.

Why would someone think that data, for example, is any different than carburetors? If data is designed to be used only with a specific process, why would one think it is going to be able to be interoperable (reused) with a different process? To generalize a little further, if data, or function, or distribution, or presentation, or event/cycles or business rules (each of which represents one of the six columns of the Framework) are designed to be used in a given implementation, why would one think that they could be interoperable (reused) in different implementations? Interoperability (reuse) is engineered into the primitive components, not into the implementation.

This issue is at the heart of the confusion between Enterprise Architecture and
Application Development work products. How can you tell by looking at a model whether it is Enterprise Architecture, engineered to be used in more than one context, or whether it is a work product being used for one specific implementation?

The first test might be, “is it a single variable model (primitive, all components occurring within a single cell of the Framework), or is it a composite, multi-variable model (comprised of components from more than one cell of the Framework)?”

A single variable model would basically be:

- a model of THINGS
  Thing – Relationship – Thing (e.g. a “semantic” model).
- Or, a model of PROCESSES
  Input – Process – Output. (e.g. a “transformation” model).
- Or, a model of LOCATIONS
  Node – Line – Node (e.g. a “connectivity” or “network” model).
- Or, a model of PEOPLE
  People – Work – People (e.g. a “work flow” model).
- Or, a model of TIME
  Event – Cycle – Event (e.g. a “dynamics” model).
- Or, a model of MOTIVATION
  Ends – Means – Ends (e.g. a “business rules” model).

A composite model would be a model that includes several primitives in the same model, for example, it might include some data being transmitted down some lines connecting to some points which include some processes updating some files, etc., etc, that is:

Some THINGS transformed by some PROCESSES in some LOCATIONS by some PEOPLE at some TIME for some REASONS. (Content from more than one column of the Framework in the same model.)

If you are looking at a composite model, the high probability is that you are looking at an application development work product because that is the kind of thing you want to know, indeed you have to know in order to actually build a “system.”

The next test you might apply is, “is this composite model being composed from primitive architectural representations (in which case, ‘show me the primitive models!’) or is the composite simply representing a point-in-time application solution?” If there are no primitive models in evidence, then it is certain that you have an application development, point-in-time, work product in hand. A very simple illustration is a typical application development work product that shows some data
elements, their identifiers and definitions, the process that creates them, the process that uses them and maybe some editing specifications. This is a composite in that it has both some data and some processes specified. It defines the “requirements” for writing a program, but the question is, “where is the data model from which the data is being derived?” If there is no data model, then it is very misleading because what you have is not really data as it relates to the Enterprise, but “information,” that is, data defined by its use in the context of a process, a “view” of the data, which is highly subject to change, as well it should be.

There is nothing the matter with application development work products. They are helpful, useful, even mandatory for getting actual work done. They are just not “Architecture” in and of themselves. Architecture, in its entirety, is the set of primitives from which application development work products could be derived.

Here are the problems with composite, application development work products that have no underlying architectural primitives:

1. They are only good as long as nothing changes. As long as the same THING is transformed by the same PROCESS in the same LOCATION etc., etc., you are OK. However, if anything changes, you are into, “throw that one away and start over again.”

2. They are misleading. If you presume that the model actually represents Enterprise Architecture primitives, you are going to be surprised when you find they are not interoperable, and they are not reusable, as there are an infinite variety of combinations and permutations which look to be about the same, but are not exactly the same, that is, not same enough to be reused.

3. They are complex. There are too many different things and different relationships being depicted in the same diagram and therefore they are very difficult to understand. By definition, they have to be limited in scope for comprehension purposes, therefore, inevitably represent “stovepipes.”

4. They are narrow in scope, that is, a “stovepipe,” and therefore you cannot tell whether the subset of any one primitive depicted in the composite represents an Enterprise-wide perspective or not.

5. They are limiting. It is hard to see the array of alternatives when presented with a single composite model, representing a single (current) view at a given point in time.

If the work products are assembled from primitives on the other hand, they are likely flexible or adaptable, etc. because changing them is simply changing relationships between the primitives, kind of like the idea of the old “table-driven” implementations of
However, even if the work products are derived from architectural primitives, they still may not be reusable or interoperable which brings us to the last test, “are the architectural primitives designed such that they are Enterprise-wide in scope?”

If the architectural primitives are defined (or designed) for something less than Enterprise-wide scope … or if you enlarge the scope of the Enterprise after you get the primitives defined (e.g. “value-chain,” merger, acquisition, etc.), then they are only going to be reusable or interoperable within the scope for which you originally designed them.

It is increasingly significant to understand the scope of the Enterprise (or the changing scope of the Enterprise) because it is the Enterprise that is critical in the Information Age, not simply the implemented systems.

Is it possible to actually identify Enterprise boundaries and build Enterprise-wide models and manage architectural primitives? Not only is it possible, but early evidence (numbers) indicate that because of the reuse, interoperability, flexibility and so on, it is actually cheaper and faster by more than an order-of-magnitude to take top-down, Enterprise-wide, model-driven, architectural approaches than to take the traditional, “you start writing the code” approaches that characterized the Industrial Age application development culture.

The Framework for Enterprise Architecture can be a helpful analytical tool for evaluating your design artifacts. Are they work products, or are they architecture? Are they composites or are they primitives? Are they point-in-time solutions or are they Enterprise-wide? You could graphically depict the answers to these questions by overlaying the artifacts against the Framework.

Many people want to try to “put” existing, traditional (or even new) application development work products (and/or other documents and various other things) into the Framework. Actually, the idea is not to “put” work products into the Framework. Remember, the Framework is not a database for storing work products (or storing anything else, for that matter.) It is a schema for classifying the primitive architectural constructs. At best, the Framework as an “implementation” could be used as a kind of index (a classification scheme) to keep track of your artifacts, to remind you of the locations where they are stored, to analyze your artifacts to determine which primitives they contain or relate to … kind of like a card catalog in a library. (Framework Software’s Structure product uses the Framework in this fashion.) But if used in this fashion, the “Librarian” would have to really understand the contents of the artifact, understand the logic of the classification scheme (the Framework) and understand where the artifact was being “shelved” (stored) in order to correctly index the artifact against the cell or more likely, cells to which it relates.

The Framework is simply an analytical tool to help you figure out the composition of your work products or other artifacts in terms of the architectural primitives and to
anticipate any potential deficiencies. You might have a work product that spans the scope of more than one cell (or row, or column, or framework, for that matter.) The question is, from what primitives is it being derived … or more fundamentally, do any source primitives even exist?? … or better yet, is anyone working on them?? And further … what confusion or misunderstanding is potential from the composite nature of the work product?

Application Development work products or just plain old models for whatever reason are not Enterprise Architecture. They might look like Architecture. They might be helpful, useful or even mandatory for getting some kind of work done or getting something implemented, but if they have not been derived from some architectural primitives that were designed for the Enterprise as a whole, the work products are only going to be good for the job for which they were produced. They are not going to be reusable, interoperable, flexible or adaptable except by some miracle or act of fate.

Of course, architectural primitives are not mystical or magic either. They are only going to be as good as the quality and experience of the Architect that produces them. At the same time however, if it is reusability, interoperability, flexibility, adaptability, reduced time-to-market and so on that the Enterprise really requires, then Enterprise Architecture, the architectural primitives, are the engineering prerequisites! In this case, the crisis is going to be enhancing the quality and experience of the people building Enterprise Architecture in as short a time as possible. It would be much better to be learning and gaining experience long before the crisis occurs.

In conclusion, producing application development work products for implementations was adequate, even exemplary, during the days of the Industrial Age. In those days, the end object was to get the Enterprise automated, replace people with machines, improve quality, reduce time and cost, that is, to improve productivity.

However, the Information Age demands knowledge to manage complexity and respond to the dynamically changing marketplace. Reusability, interoperability, flexibility, adaptability, reduced time-to-market are the by-words of today’s “agile” Enterprise. The gate you are going to have to go through to realize these modern Enterprise attributes is Enterprise Architecture, that is, actually producing (“engineering”) the architectural primitives that constitute the knowledge infrastructure of the Enterprise.

Application development work products (even if they are “models”) are not Enterprise Architecture, unless they are being derived from Architectural primitives, primitives that have been designed with the Enterprise in mind.

Someday, you are going to wish you had all those models made explicit, Enterprise-wide, horizontally and vertically integrated at excruciating level of detail. (From these primitives, you could very quickly derive an infinite number application development work products.)
Architecture is the finite set of primitives. Work products are the infinite set of compositions. Architectural primitives are the “elements.” Work products are the “compounds.” Architecture is structural in nature. Work products are process derivatives, the inputs and outputs of the application development process.

Enterprise Architecture and Application Development work products are both relevant and both necessary, but Application Development work products are neither a substitute for nor a source of Enterprise Architecture!

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