Alternative Technologies™

Enterprise Integrity: The Art of Mimicry V

Vol. 8, No. 6

Certain aspects of today's software engineering practices reflect traditional, functional organization with its stove-pipe, top-down control structure and profit-center accounting. They are fundamentally misaligned with the principles and value of the business revolution that results in the service-centric business. As noted in my last two columns, IT has learned to mimic the vocabulary (i.e., the appearance) but not the substance of the service-centric business.

In my last column, I purposely strayed from the proper business terminology of "service-centric" to the IT terminology of "service-oriented", wondering if anyone would notice while hoping my message would not be obfuscated. Businesses cannot be service-oriented, cannot just focus a little more on services: The provision of services must be central to and drive management and operational practices. Service-centricity means that the business is driven by goals, supplying services so that customers can achieve their goals, and aligning business goals with customer goals.

The service-centric business manager knows that satisfactory (let alone exemplary), competitive (let alone innovative) service delivery is the result of a well-managed and integrated suite of business processes. Without successful service delivery into the marketplace, nothing else matters. With it, subsidiary goals ranging from improved market share to greater profit margins can become the subject of rational optimization. When this precedence relationship among goals is turned on its head, as when profitability of profit-centers controls operations, its possible to optimize the business out of existence.

The cost of a service is the result of the cumulative costs of the distributed set of activities that result in that service (including products), i.e., the cumulative costs of a business process allocated by service. Typically, these activities or "functions" are coupled non-linearly, and so cannot be optimized independently. Activities may also contribute to multiple business processes and therefore multiple services. As such, resources (including time) must be "owned by" the business process that consumes them (as contrasted with profit center, product-centric ownership by function or department) if the business process is to be optimized.

Let's compare service-centric business organization to its IT counterpart, SOA. The evolution of SOA has been convoluted, twisting it's way through client/server and object orientation to become a curious hybrid with distributed componentization at its core. Software design and development today is primarily object-oriented, a perspective that gives primacy to the identification of objects and relationships among those objects. The closest thing to services in this view is the object's methods. Determining which class of object should implement a method ties that method inextricably to a set of abstract resources, and to a *functional* decomposition.

But then, who owns the object? Who is responsible for the object's resources (which supposedly correspond to real-world resources)?

An object class defines an abstract data type. Its methods are, in combination, an abstract function. It follows straightforwardly that object design (in particular, establishing the class hierarchy) is merely a high-level or more abstract form of functional decomposition. As noted earlier in this series, functional organizations are necessarily rigid. They assume that functional relationships and priorities will not change often if at all, and yet history shows this assumption could not be more false.

IT services cannot be object-structured methods without imposing rigidity. We must not depend on practices like extreme or agile programming to constantly catch up with a high velocity market, nor does refactoring suffice since it tends to perpetuate a fixed object class hierarchy even when mismatched with service utility. After a while, the object classes become so abstract they bear no resemblance to business entities and resources, and cannot be related to business measures without extreme analytical and development effort. The result is rigid misalignment.

If a business is to be agile, responsive, and adaptable, then how we provide services is key. Flexible service provision must be treated as foremost and composable. Services must be understood as implicitly defining products, not the other way round. Being service-centric makes the goal paramount, rather than some presumed means to achieve that goal. This gives the business flexibility in delivery and resource management. In doing so, the loop between provider and consumer is tightened.

When products, or their IT corollary "objects", are the focus, we ask foolish questions about the "best" design of the object that pertain to its properties rather than its utility. Trying to satisfy a diverse market with this approach necessarily leads to mediocrity and inflexibility. By contrast, supplying composable services with a guaranteed level of satisfaction (i.e., results) irrespective of the means used to meet the contract, promotes user responsibility for identifying what is wanted, while promoting vendor negotiation with the consumer, and cooperative competition among producers over innovative means to meet demand.

The composition of business services should define class hierarchies dynamically and implicitly, reflecting the current, ever-changing, and interacting views of management, employees, suppliers, and consumers alike. Only then can service-oriented IT mimic service centric business, reduce maintenance costs, and maintain agility. Agility cannot be achieved: It is a dynamic balancing act among unpredictable economic forces, intended to keep producer and consumer alike on the tightrope of *enterprise integrity*.

David McGoveran