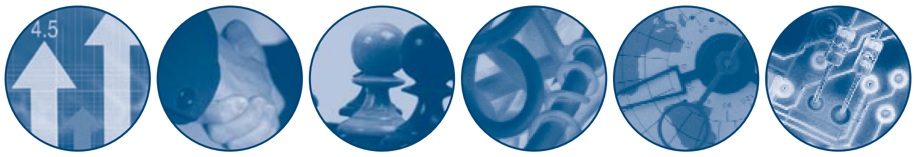

Enterprise Architecture
Advisory Service
Executive Update
Vol. 10, No. 13

Update

Enterprise Architecture and Business-Focused Change Management: Part I

by Sebastian Konkol,
Senior Consultant,
Cutter Consortium

In thinking about enterprise architecture and the value it provides, one can see the focus is moving toward supporting big IT-enabled endeavors, which means providing them with the possibility of improved alignment with the existing information systems environment as well as the projects just being rolled out. However, the IT organization often struggles with the relatively small number of big projects aimed at rolling out a new IT system along with the relatively big number of small "projects" aimed at changing something in the existing IT systems.

While I have not come across many methods to deal with such organizational complexity, I have worked for companies suffering from such circumstances — a kind of inability to manage such a scenario more efficiently and keep it under control. Consequently, I have done work on overcoming this issue. This *Executive Update* series will present my main findings.

THE PROBLEM

To set the scene, I should first explain the meaning of a "big number of small 'projects.'" In practice, this is a set of change requests being realized in parallel with many information systems, typically organized within some kind of release management process. Such change requests generally represent modifying an algorithm calculating certain data, adding a data field to some form, or altering some aspects of business process support being done by a particular IT system. Due to a shortsighted required level of cost efficiency, it is quite common in such circumstances that a change is being implemented as only locally optimal and in as simplified a way as possible. Since the broader consequences of such change implementation are not being analyzed and documented, it is much more probable that they would be uncovered at the most inappropriate moment — while trying to make a subsequent change, for example, probably in the same manner.

In such an environment, there is no time (or, at best, a very limited amount of time and resources) for software refactoring; the business does not want to understand that a developed solution may have to be altered due to other changes implemented into the information systems (nor does the business want to pay for such changes). Things that were locally implemented and should be corrected (I'll call them "refactoring needs" as far as an IT dictionary is concerned) are being uncovered during the development work of a subsequent change request implementation, and at that stage, it is not acceptable to return to analysis to deal directly with that need. Consequently, the problem uncovered is not being resolved but rather is being subsequently patched and buried down under a new layer of changes. The refactoring needs are not being registered and described by any means so it is not possible to plan for satisfying them later. Thus, the vicious circle is being closed, not to mention the growing costs associated with resolving the initial issue with each subsequent "patching."

THE IDEA

To solve this puzzle, there are two questions to be answered:

1. How does IT convince the business that facing such problems is unavoidable if the business is to be properly supported by the IT systems it funds?
2. How does IT actually do the corrections in a cost-efficient manner?

While here I mainly focus on answering the second question, I do first offer a brief response to question number one.

Economy of Patching

I strongly believe that convincing businesspeople is a quite straightforward task. All you have to do is to collect objectively applicable arguments and present them using the language that businesspeople understand: exchangeable currency or its equivalent.

There is nothing wrong with a "not so perfect" implementation of a change to a certain system as long as the business needs are fulfilled and the emerging tradeoffs that pose some sort of constraint on the business capabilities are accepted. Tradeoffs may happen due to a variety of circumstances (e.g., short implementation time required). The drawback is that when such tradeoffs made during that implementation are forgotten, the constraints posed locally tend to propagate out of control and "infect" other parts of the business and will strike during the next change being made to the same piece of software (or the infected ones). It is much cheaper to correct the temporarily acceptable constraint implementation that will fix that piece and then deal with the additional infected ones later. It is far wiser to adjust such tradeoffs during "spare" time than while working under pressure (e.g., short implementation time allowed). Both tasks — healing infected pieces of the system and not being able to do the required things during a subsequent time-pressured effort — can be presented as additional cost elements, which should be a sufficient enough argument for the businesspeople.

Visibility and Manageability Constraints

As far as refactoring efficiency is concerned, the basic idea is quite easy and consists of: (1) the enterprise architecture as the map upon which the refactoring needs (things to be corrected) are pinned down and thus made visible; and (2) agile techniques needed to direct and manage the correction development effort.

Things to be corrected can emerge for various reasons. They can be the result of agreed-upon tradeoffs during some implementation effort (e.g., when time constraints are so stringent that some flexibility or configurability has to be sacrificed) or can represent constraints artificially imposed on some system function. They can also result from implementing many changes in parallel (locally optimal) that are not being perfectly coordinated and integrated (which is hardly possible to achieve); this results in globally suboptimal or constrained designs. In addition, things to be corrected can be the result of changes in the environment surrounding IT that were not possible to envision during the initial development effort. Finally, they can be the result of human error, for example, designers' or developers' mistakes; by the way, this last example is the only case among the above mentioned solely attributable to an error by IT.

Enterprise architecture is able to model the relations between various pieces of the information systems environment and is described in various dimensions: business, information, applications, and technical architectures. Knowing these relations, it is possible to identify the elements likely to be affected by the planned change to a particular piece of the architecture. In order to create the ability to precisely register the things to be corrected in an appropriate context, the organization should build all the perspectives of an EA (business, information, applications, and technical architectures). The EA should also map from the technical to the information and applications architectures and from this perspective to the business architecture. Such mapping should allow pinning any refactoring need to the technical or information and applications architectures and thus be rolled up to the business architecture level.

Any refactoring need detected (or created) and not being corrected during a certain development effort should be registered as soon as it is discovered. It should be pinned down to the element of EA (technical, information, or applications architecture) that it affects and be described in simple but meaningful terms. For that purpose, I find the concept called the "user story" known from agile methodologies to be very applicable; I call it the "refactoring story." Such a simplified description can take various forms depending on the way the enterprise architecture is being documented. If the EA is modeled and documented in Microsoft Word, the refactoring story can take the form of a comment in that document. In other cases, when the enterprise architecture is being kept in a specialized repository, the refactoring story can take the form of some special attribute able to be assigned to each object comprising the EA. If the EA is being documented as a diagram printed and displayed on a wall, the refactoring story can take the form of a yellow note pinned to a certain part of the printed architecture model.

Regardless of the documentation mechanics used, the result should visualize the state of the enterprise architecture and its constraints. The EA business architecture should be used as a continuum framework, allowing for the refactoring needs to be presented in a manner understood by the business. The number of "temporarily acceptable" solutions (which is the same as the refactoring needs for IT people) should be published periodically to build an understanding among businesspeople about the state of the information systems environment. It should be communicated to and discussed with business since it presents real constraints for future projects (i.e., the next changes in software), possibly impacting business cases if there are change requests in particular pieces of the EA business architecture.

While working on a feasibility study stage of a subsequent project preparation, when some idea of the required changes to the enterprise architecture is being determined, all these registered refactoring needs relevant to the planned change to the EA should be taken into account and their impact on the work required should be evaluated. The decision should be made whether particular refactoring stories will get included in project scope. Each refactoring need can represent both the technical risk and the cost to be incurred; if known and controlled, the risk can be properly mitigated and the cost can be added to the budget, which makes project planning much more reliable.

Depending on the software development methodology practiced in the organization, the refactoring stories to be included in the project scope can be dealt with differently. In the case of agile software development methods, refactoring stories integrate seamlessly into the process; they should be added to the list of user stories and handled (prioritized, developed, released) in exactly the same way. Where agile methods are not being exercised, the refactoring stories should become requirements and as such should be taken into the analysis phase of the project or its iteration.

Apart from the implementation of project-based changes, the state of the enterprise architecture supplemented by refactoring needs represents a perfect assessment and justification for doing some tuning to the information systems environment.

Being able to present existing constraints applicable to business processes means being able to judge the necessity of initiating some technical work aimed at business adaptability as well as getting to the point where future predictable changes will be possible to achieve. For that purpose, registered refactoring stories should be judged against their impact and ranked accordingly, for example, according to the importance of a process being constrained. In such a case, IT would know what to do and how to optimally spend the time (and funds) it would be granted for that purpose.

It used to be rather difficult to convince decision makers that appropriate information systems tuning (aka refactoring cycle, technical release) should be done from time to time between business releases of a company's information systems environment. The framework presented here can be of help in that matter.

THE RESULTS

Various benefits can be achieved by accepting such an approach. Due to the visibility and comprehension of the complexity rooted in technology and its utilization, the level of trust between IT and businesspeople tends to grow. Registering refactoring stories builds awareness of technical risks and constraints and gives some idea of the implementation problems that must be addressed. This in turn gives more confidence to budget planning and work scheduling.

Enterprise architecture brings value not only to big IT-enabled projects being implemented but to the change management discipline as well. In conjunction with agile methods of software development, it is possible to achieve much wiser control over the quality of software being developed.

In the remainder of this *Executive Update* series, I will focus on each specific area mentioned here (project feasibility, software development, release management), presenting a more detailed approach of each element.

ABOUT THE AUTHOR

Sebastian Konkol is a Senior Consultant with Cutter Consortium's Enterprise Architecture and Enterprise Risk Management & Governance practices and a Senior Consultant with Cutter's partner in Poland, Infovide-Matrix S.A. Mr. Konkol specializes in strategic IT planning, business-technology partnership, and fostering new technology management methods supported by social psychology. He is coauthor of the Enterprise Architecture Management (EAM) approach and an expert in IT governance. He divides his time among running strategic IT consulting endeavors, managing projects, and sharing his expertise in EA management and IT governance. Currently, Mr. Konkol is working on *frameworks for strategic technology management, binding company strategy, enterprise architecture, and corporate governance, as well as concepts that apply contemporary social psychology tools to the field of technology management. Prior to joining Infovide in 2003, he worked, since 1997, for a mobile telecom operator as project leader in the field of GSM network surveillance and later as program manager responsible for running projects aimed at creating and providing GSM value-added services to the mass market. He has authored numerous publications dealing with IT, telecom information environment management, and IT organization management. Mr. Konkol can be reached at skonkol@ cutter.com.*