Gap Analysis: How to Get Your Information Technology Project Off on the Right Foot!

John C. Goodpasture, Vice President, Legal Services Operations, Lanier Professional Services, Inc.

Introduction

How many times have we heard that information technology projects fail on account of poor requirements? Often, actually. In fact, this a somewhat universal dilemma facing most IT project teams. In a nutshell, the problem for the Project Manager is this: how best to draw the customer into the process of specifying the functionality to be delivered in a new information technology project such that expectations will be satisfied at the end? Especially this is true when the project is to install one of the new "package" enterprise programs that recently have found favor in the data center. Inevitably, the problem arises from the fact that the customer, that is the end-user, has become accustomed to the capability of the legacy system, but it has been decided by others that the legacy must be replaced by a vendor's package software solution, and very often the package offers many new [read: different, unfamiliar, but "wanted"] capabilities. So, many times the challenge faced by the Project Manager is how best to sort through the myriad of opportunities provided by the new package without enduring project crippling scope creep.

Without doing any analysis, the Project Manager can anticipate that functional customers will have four reactions to a new application: First, the new package may provide capability not needed in the business. This is characterized as the *control gap*: "The package has it, and we don't need it." Potentially, that could undermine the business case for the project if there is expensive capability going unused. Second, the legacy has capabilities that have been baked into the organization's business rules, but these capabilities are not in the new package. This is the *release gap*: "We need it, and the package doesn't have it." Of course, the third one is: "We need it and the package has it!"—which is no gap at all. And, fourth: "We could use it if it had it, but it doesn't, and we can wait," which is the *vision gap*. These gaps are summarized in Exhibit 1 and Exhibit 2.

In this paper, a practical methodology will be presented which can be employed by multifunctional teams of information system resources and functional user resources to systematically evaluate the package candidate in terms of the four possibilities.

Gaps Explained

Considering that gap analysis and gap control are important tools in the Project Manager's scope management toolbox, let's take a moment to discuss the three gaps in more detail. Recall that they are the control gap, the release gap, and the vision gap. First, the control gap. Refer again to Exhibit 1. The control gap arises from the fact that in many instances the capability of the package software is different than the business rules for the firm. That is, the firm has chosen to restrict by signature level, by dollar amount, by organization, or by job code, or by perhaps other attributes, the capability of an end-user, and has business rules in place to support these restrictions. But often, the package design does not allow for these restrictions, at least without modification, and so the question arises: should the package be modified to incorporate these restrictions, or should policy and management technique be used instead? And, of course, it is possible to have the flip side of the coin: that is, the package has restrictions built in that are more restrictive than the firm's business rules, and so again: modify the package or modify the firm's rules? Largely, the answer lies in the support strategy of the package, the degree of flexibility the firm has to change, and the culture for accepting adhering to policy. Naturally, the lowest cost of ownership comes from the least modification to the software; the reflection into scope management is that modifications that do not have to be made are no risk at all, cost no money, and consume no time. Just the opposite applies to modifications made to loosen controls: they represent some degree of risk, consume resources, and have the possibility of extending schedule. But, in the final analysis, it's a business decision to go either for the lowest cost of ownership or for making the package conform to the rules of the firm.

Second is the release gap, which was illustrated in Exhibit 2. This gap is characterized by needed modifications to add functionality, which the package does not have, and will not have, in a time frame consistent with need. For the project manager seeking to control scope, it is a matter of obtaining approval for these modifications from a Configuration Control Board and then scheduling these modifications into a release schedule. Naturally, funding

Proceedings of the 30th Annual Project Management Institute 1999 Seminars & Symposium
Philadelphia, Pennsylvania, USA: Papers Presented October 10 to 16, 1999

Exhibit 1.Control Gap Defined

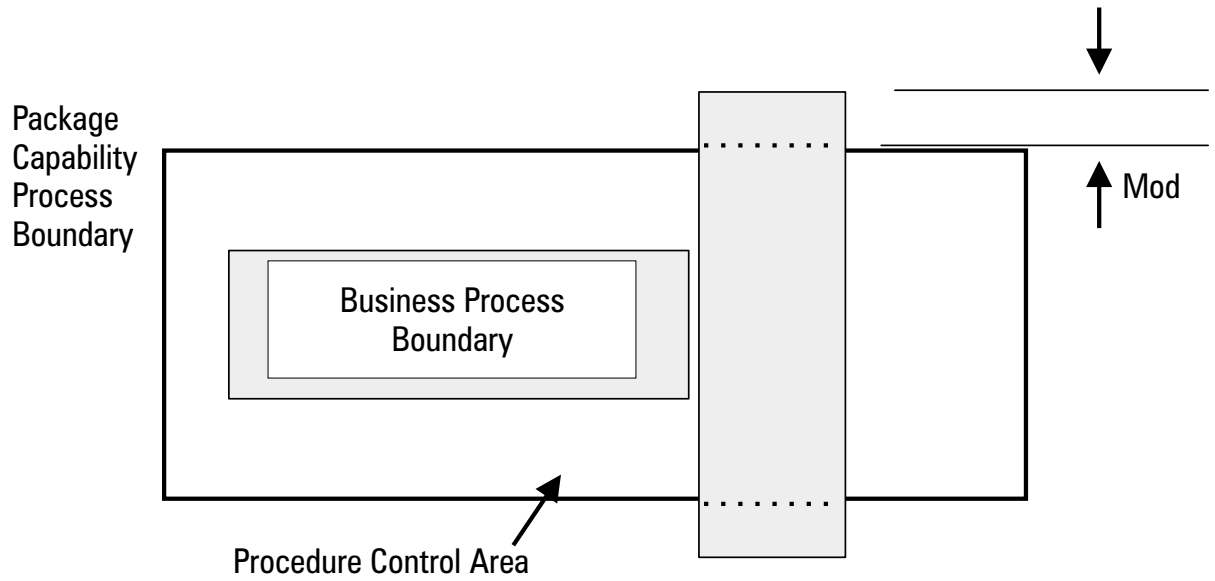
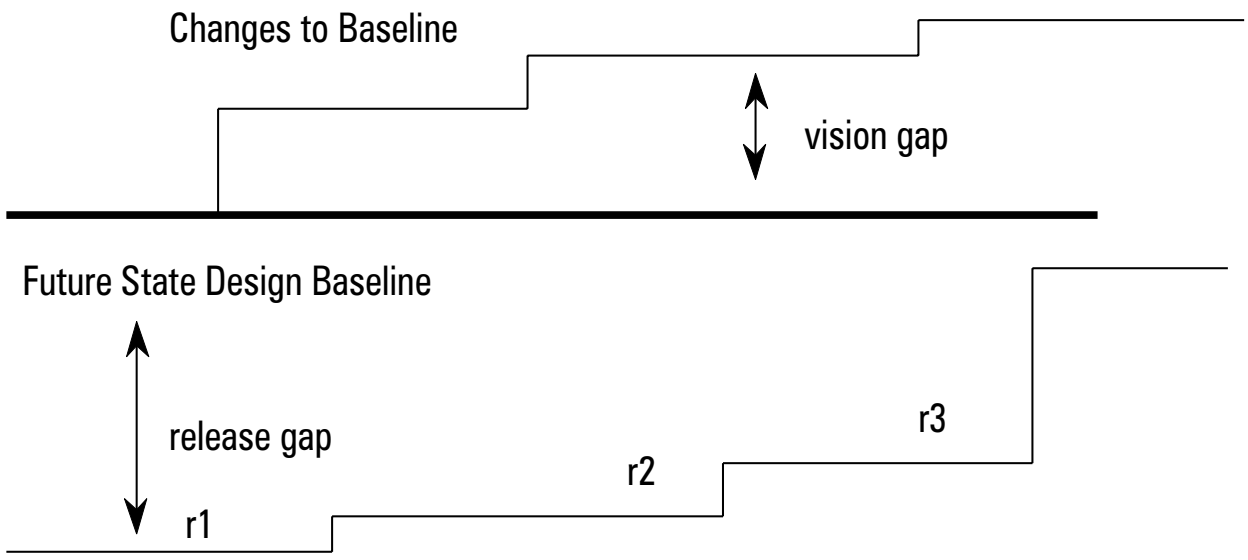


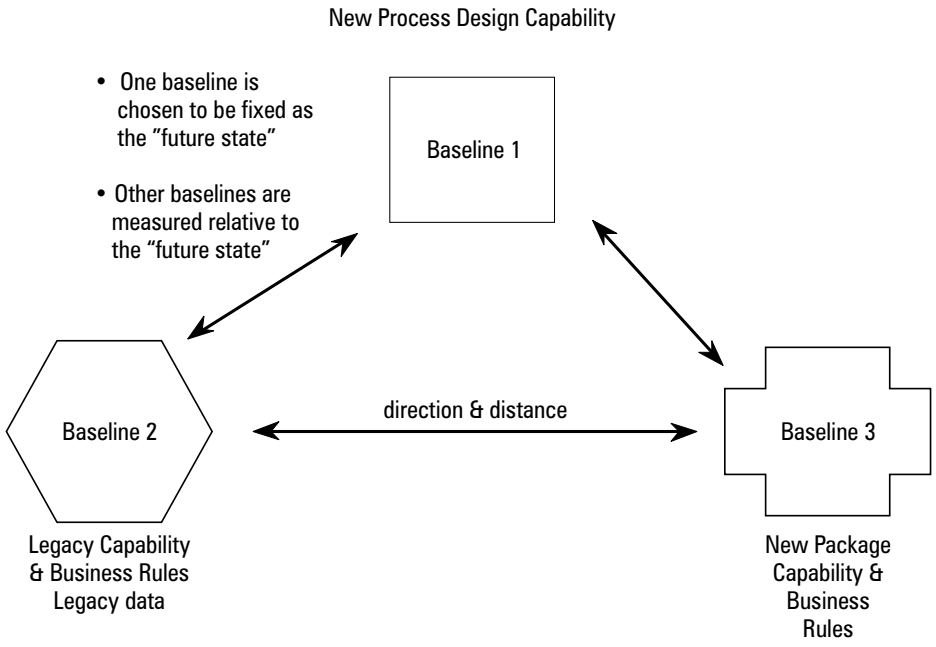
Exhibit 2. Release and Vision Gaps Defined



and resources will have to be scheduled as well. The issue to be handled is the timing of the release: will they be incorporated straight away in release 1.0, or later? This is a scope management question and a risk management question that is often beyond the paygrade of the Project Manager. The project sponsor or the Configuration Control Board may make this decision.

And third is the vision gap, again illustrated in Exhibit 2. Here is where the functionality roadmap to the future is drawn and applied. This is the wish list, prioritized in some manner, which represents where the organization wants the functionality of the package to go, but it is functionality that is not mission critical at this time; therefore, it can wait. For lowest cost of ownership, it is important

Exhibit 3. Measuring the Gap



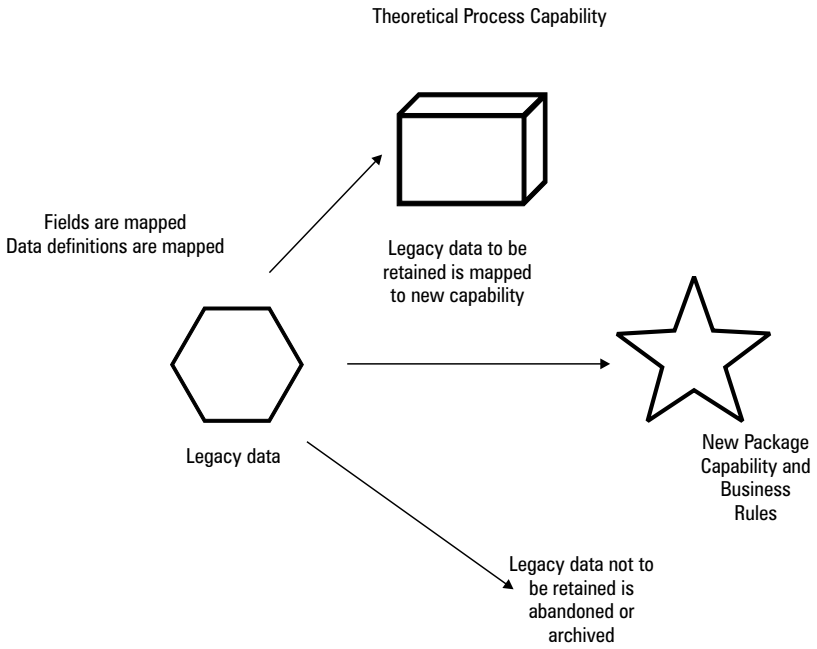
to communicate this roadmap to the package vendor, participate in user group forums, and influence their R&D program through partnership programs whenever possible. Furthermore, having the vision gap list, usually known in the industry as the "dot release" list, provides a place for all functional ideas to go without being lost, and avoids the problem of what to tell the functional team about where their ideas are being placed in the queue of future capability.

Methodology for Gap Analysis

Picking the Baseline

Having defined the gaps, a methodology for gap analysis is needed. Such a methodology is a tool for assisting in project scope management, perhaps one of the most important tasks of project management. A couple of key points are these: first, a gap is a difference between two capabilities as already described, but a gap has two attributes: size or degree and direction. Direction is a matter of deciding from which baseline is the gap is to be measured, and size or degree refers to how different the baseline is from the desired state. What is the desired state? There are three possibilities, as shown in Exhibit 3. The desired state could be the functionality, the processes, and the business rules associated with either the legacy, the package, or a third design not specifically in either software system. So, there are several possibilities: measure from the legacy capability baseline, or the package capability baseline, or from the process design? This is not a trivial decision, as there are consequences for picking the direction. If the legacy baseline is picked as the starting point, then what is being said is that the legacy represents the capability and, by extension, the process by which the business is going to be run, so the task is to find a package that has a minimum gap to the baseline. On the other hand, if the package is picked as the baseline, then what is being said is that the business is going to be run the way the vendor's business designers provided for the business rules, and that may take process redesign, job redesign, and other structural or functional changes so as to take advantage of the system. And finally, measurement could be made from the design that is neither the legacy nor the package. There is actually no right answer. It could go either way, but there are other considerations to take into account, most notably the support costs if the first approach is picked. Almost certainly if the baseline is the legacy, then the project is going to be a development project to modify the package to minimize the gap to the baseline to the greatest extent possible. What results is a highly modified package for which it will be

Exhibit 4. Data Mapping



difficult to obtain support from the vendor, thereby making its support an ongoing IT project.

The author has found that picking the package as the baseline usually results in lower cost of ownership and provides the most opportunity to re-evaluate the process design of the business. After all, the package is a mainstream representation of modern business practice. That is what provides the market appeal for these packages; they are the latest and greatest in the average way to go about whatever function they are addressing.

Mapping the Gap

No matter which baseline is the one to map to, or to map from, one thing is certain: there is data in the legacy implementation that must find a home in the new baseline. Not all the legacy data, mind you … just the data necessary to support the new baseline. This data identification is key to getting started. The essential concept is as follows: analysis proceeds directionally from the legacy toward the package (after all, the package is going to replace the legacy, no matter from which direction the gap is measured), identifying to where data must be mapped to support those functions of the package that are going to be in the final implementation. Data not mapped is abandoned. For historical purposes, it may be archived to some permanent storage before the legacy is torn down. Note how central to the technique is the concept of direction. Each legacy data item that is going to be needed in the future state must find a companion capability in the package, but not necessarily the other way around. That is, each possible data item in the package may not be used in the future state, so it is not necessary to populate it with legacy data. On the other hand, there will be data items in the package that need to be populated or configured that do not come from the legacy. These will have to be synthesized. This concept of data mapping is shown in Exhibit 4.

Once the data is mapped, then the functional capabilities of the package are "turned on" in accordance with the process maps, the imported business rules, and the decisions made about the control gap. Further mapping and configuration may be required if modifications are made in accordance with the release gap.

Summary and Conclusions

Replacing a legacy software system used for supporting business processes really can't be done without considering the gaps between what is desired and what is the

legacy with which the project teams begins. It's unrealistic to think that all three gaps will not be encountered. They must be segregated or else the project team could be overwhelmed by problems of scope control. As such, a methodology to deal with them is essential to a successful project. Further, analyzing and controlling the gap is an important component in scope management and risk management, requiring decision making, prioritizing, and business analysis. Employment of the methodology illustrated in this paper will go a long way toward helping with scope management. It has been proven effective in many system deployments by teams seeking to satisfy end-users while keeping to a budget, a schedule, and commitment to management to deploy a modernized system.