For Enterprise Applications and Data, the Question Is Not Make versus Buy…

…Because, while we can do both, we really need to be good at "assemble" if we're to own our own applications and data destinies

By **Bob Weir** and **Rick Mickool**

When considering strategies for addressing application and data needs in higher education, most vendors and many institutions approach the problem as a choice between make or buy. Should we purchase all-in-one suites offered by solutions vendors such as PeopleSoft, Oracle, and SCT, or should we create what we need? While compelling rationales support both approaches, there are also compelling exposures that render "make versus buy" a moot question because neither make nor buy will solve your problems.

Our answer to this dilemma is a middle ground we call "assemble." In this approach, one buys the best single component for a particular function or task, develops the rare component that does not exist in the market, and focuses internal resources on assembling the pieces. The result is a full-featured, yet ultimately flexible environment specifically tailored to the institution served. Additionally, this approach reduces reliance on any single vendor and seeks to compartmentalize the risk associated with major application and data investments.

What follows is the rationale supporting the assemble approach, as well as the critical success factors that must be addressed to deploy this strategy. These factors span vendor selection and management, institutional policies and practices, and internal information services capabilities.

What's Wrong with Buy?

The attractions of the buy option are well documented by the solutions vendors. Reasons boil down to leveraging the investment streams of large software firms and reducing the institutional investment while delivering best-of-breed solutions—at least according to the vendors. Those investment streams and product functionality are, of course, the result of the aggregate licensing and maintenance fees paid by the vendor's customer institutions minus, it's hoped, healthy profit.

In that last term, healthy profit, lies one of the dangers of the buy option: If you are betting the majority of your application delivery on a single vendor, you want that vendor to be in business and paying attention to you for a very long time. As we've seen over the years, the software business is not for the faint of heart. Whether succumbing to competitive pressures, dealing with customer buying cycles, struggling with technology evolution and delivery, or attempting to expand into new markets, many supposedly solid vendors have stumbled, been acquired, or outright failed.

There is also risk in the implicit assertion that solutions vendors have everything you need. This is rare in industry and impossible in higher education, given the diversity of needs and institutions. Even for those institutions deploying full enterprise resource planning (ERP) solutions, there are two timing problems. The first is that broad ERP implementations often take so long that by the time they are complete, the functional requirements that initially justified the investment have shifted, requiring adjustment either during or after the project. The second timing problem is that solutions vendors focus your attention on the end goal, a fully implemented solution. During the multi-year

implementations, however, you must keep the institution running. Accordingly, you need to integrate the new functionality and data as they come online with the existing functionality and data that have yet to be replaced. In other words, you need to continually integrate or assemble disparate technologies.

The last danger of the buy strategy is the vendor promise, "We're committed to higher education and your institution!" While the intentions of the executives who make that promise are not in question, we find the implications unacceptable. That statement implies that the vendor will be in the business of serving higher education and your institution for an extended period, yet—as discussed above—the market, technology, and management changes make that promise suspect. Additionally, those companies that garner the majority of their revenue outside of higher education may, over time, not be completely reliable, considering the large discounts and more problematic support of the higher education marketplace. The vendor promise is actually closer to "We're committed to your institution so long as we're interested, you follow our lead, and you pay the bills!" Beyond the risk that a vendor will stay in business, commitment to just one or two vendors can lead to forced release migrations, exorbitant maintenance fees, and a myriad of other captive-audience behaviors.

With all of these complications, why would anyone buy? As we'll examine in the next section, the answer is because you cannot make everything you need. The question is not *whether* you buy but *what* you buy—and what you *do* with it. You have no choice.

What's Wrong with Make?

Although making what you need provides the ultimate in control, there are three fatal limitations to using this as your primary strategy: investment, discipline, and support.

From an investment standpoint, a single institution, while perhaps able to address a portion of its application needs through direct development, will never be able to meet increasing internal functional demand or "keep up with the Joneses" in terms of offering competitive services to its constituents. In recent years a number of universities have formed the equivalent of software cooperatives, in which member institutions pool their programming resources and share the fruits of their labors. While we have seen some success with this approach at the component level, the fact remains that this approach also means assembling or integrating those components as if they were purchased.

Software developed for production use must not only support complex functional and business processes, it must also fit within technical and service frameworks and provide a stable, reliable environment for users. Development of this type of software is a complex undertaking requiring disciplined development, testing, documentation, and deployment processes, as well as experienced software developers and management. While there are certainly pockets of this kind of discipline and talent within some universities, no university or cooperative has the processes, retainable skills, or investment capability to build "industrial strength" production software on a scale comparable to its needs.

Regardless of where an institution acquires its production software, ongoing defect and technical support is imperative, given that all software has bugs. Additionally, evolution of technology and regulatory changes (for example, financial aid) require continual adaptation of production software. Vendor support—while far from perfect—is preferable to local or cooperative support from individuals who have nonsupport responsibilities and priorities. Locally or cooperatively developed software components are often the products of one or two individuals; if they leave the university, their knowledge of the production system often goes with them, leaving the software, the project, and support at considerable risk.

So does all this mean you should never make? No. Whether for base functionality or for integration, you must build some software because you cannot buy everything you need. The question is not *whether* you make it but *what* you make, *how* you make it, and what you *do* with it. You have no choice.

Why Assemble?

At Northeastern University, we have been deploying an assemble strategy for the past three years, and the observations noted are hard won. Today we support a 50,000-customer environment integrating PeopleSoft, SCT, Computer Associates, College Board, and homegrown legacy administrative systems, among others, as well as a myriad of academic enabling software and tools. We support DB2, Oracle, Microsoft SQL, and Computer Associates' Integrated Database Management System (IDMS) databases across IBM, Sun, and Compaq servers accessing our 11-terabyte, mirrored EMC enterprise storage solution.

While this environment is extremely complex, the main benefit we have enjoyed in deploying our assemble strategy is unprecedented control of our own destiny. This has included creating a vendor-competitive environment within Northeastern, which has driven down licensing and support costs and increased our component and tool choices. The assemble strategy has also allowed us to significantly reduce the scope of new projects, improving granularity of investment while significantly reducing the risk associated with any one project, service, or vendor.

What's Wrong with Assemble?

Plenty. The assemble approach forces you to deal with all the exposures of both make and buy. In assemble, you make what you can't buy and you integrate what you can buy. This requires implementation rigor analogous to software development. Also, in assemble you buy what you can, albeit in smaller pieces. As a result, for a given capability you have all the functional-process and vendor-reliance issues associated with implementing a single vendor's total solution. In fact, assemble introduces its own unique problems, spanning multi-vendor relationships to disparate technologies. There are management and technical costs in working across

multiple vendors, as well as the risk that all fingers point back to the institution should something break, particularly at an integration point. Given that, the assemble strategy is not the Holy Grail of application strategies but rather the least of the evils. Whether you make or buy applications, you will assemble. You have no choice, so you might as well get good at it.

What's Right with Assemble?

Plenty. There are numerous advantages to the assemble strategy. You can enable each of the major functions within your institution (enrollment management, finance, advancement, and learning management, to name a few) with the software components that most closely meet their functional needs and preferences. Despite the disparity of applications and vendors, you have the capability to serve your varied constituencies (students, faculty, staff) with a consistent experience through the proper application of enterprise-portal and data-management tools. You have the ability to swap out components as your institution's needs or preferences change without ripping up a major solution suite. Your risk is compartmentalized, both in terms of reliance on a single vendor and of failure of a single project or product. Finally, you cannot be held hostage for your entire applications platform due to escalating costs or the vendor's timetable for upgrades. All these advantages add up to giving you and your institution maximum flexibility while minimizing risk.

What Does It Take to Assemble?

Plenty. Each of the preceding sections—buy, make, assemble—ended with the same phrase: "You have no choice." In any complex environment you must do all three. The question is how to focus on assemble as your primary strategy and make or buy componentry within that context.

We have learned that there are three critical success factors for the assemble strategy: institutional management and funding, internal information services (IS) capabilities, and vendor selection and management.

Institutional Management and Funding

The first institutional success factor is actually more about a service approach that maximizes the benefits of the assemble strategy than about successful deployment. Institutions that view themselves as single electronic communities consisting of varied but interrelated groups and individuals have the most to gain from an assemble strategy. In fact, institutional portals are the best example of the obvious need to assemble multiple applications into integrated application and collaborative environments. Given the diversity of constituents served by higher education portals, the assemble strategy is required for broad-based yet individualized support of an institution that views itself as a single electronic community.

The second institutional success factor is having senior administrators who understand and support several key concepts and are willing to subvert their individual unit priorities to the overall benefit of each customer constituency. One key concept is that constituencies and individual customers are also the customers of the balance of the institution's functional areas and are best served through an enterprise portal delivering integrated services tailored to the individual. Another key concept is a shared responsibility with the portal delivery team to own not only the transactions and data within a functional part of the portal, but also the content and the currency of the information channels associated with any given function. Lastly, senior functional administrators must fully engage with the technology team and own the resulting customer experience. This engagement includes articulating and integrating business processes, including both online and offline components, and helping to select the software componentry required to support those processes.

The final institutional success factor we will mention is funding justification. We believe that, done right, the assemble strategy is the most cost-effective approach, particularly over the long term, as needs escalate, technology evolves, and vendors come and go. The funding-justification challenge of the assemble strategy is that a significant portion of the cost is justified by the customer and constituency experience versus functional capabilities. Historically, application initiatives have been justified along functional lines, such as a new student services suite or an improved HR capability. While funding justification of components will continue to be articulated functionally, institutions will have to recognize, through resource allocation, the priority of the customer-centered, cross-functional experience.

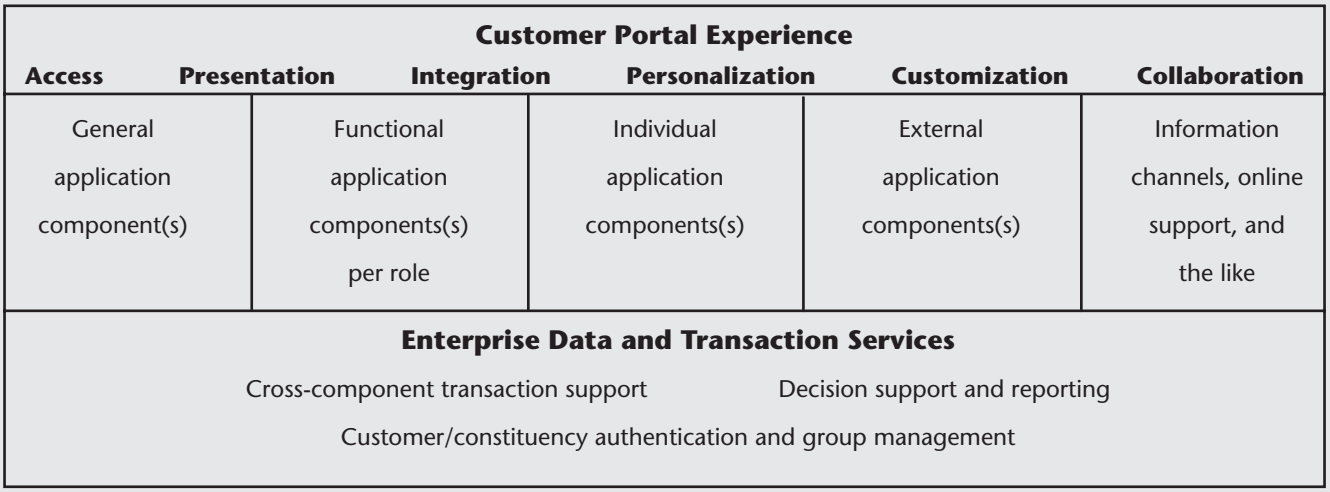
Internal IS Capabilities

A number of success factors fall to the internal IS team, the first of which is discipline. Key to this discipline are the core processes of production software development, such as business-requirements articulation, professional project management, adherence to architectural and technical standards, and maintaining separate development, test, and Q/A functions and environments. Other important core processes are systems integration and environment management, including change control, infrastructure availability, and service measurement. These processes must be applied to both component development and integration management. By analyzing every project against these comprehensive processes, teams can deviate as required for their success at project definition, but only with the direct engagement and agreement of their key constituents and management. In this way, tradeoffs and shortcuts are taken explicitly before the project begins versus as the project hits the inevitable snags.

Figure 1 represents a generic assemble architecture with individually purchased or made components providing a broad range of functionality. These components are joined from an authentication, data, and transaction perspective as well as in the presentation to each customer. We use this diagram to set the stage for the balance of the information services and vendor discussions.

Figure 1

Generic Assemble Architecture



The second internal IS success factor is talent, both technical and managerial. In addition to the technical competence and customer focus associated with any good IS team, the people required for a successful assemble strategy must have the ability to center on integration, versus creation, as their prime motivation. Accordingly, they must be willing to align their efforts with the overall application, data, transaction, and business-process standards required by this strategy, along with the insistence that every component, whether bought or made, must "plug and play" with the balance of the environment. With plug and play, the team must be constantly flexible and willing to embrace change. Also, the core team must be permanent employees rather than consultants in order to reduce the cost per person and to reduce the project and solution risk introduced by turnover.

The third internal IS success factor is responsive support. The combination of plug-and-play functional componentry and personalizable portal interfaces means that every customer's experience is unique, and thus every service call has the potential to require the complex triage of a one-of-a-kind environment. In addition to renewed emphasis on customer training reflecting this dynamic environment, customer service representatives, whether in the call center or visiting desktops, must be trained to quickly assess and resolve situations where there is little certainty in terms of the customer's configuration. Additionally, WYSIWYG (what you see is what you get) tools that enable a remote customer service representative to witness a caller's problem or scenario will dramatically reduce this support challenge.

Vendor Selection and Management

This last critical success factor is often the most difficult because it is where the IS executive, despite being a paying customer, has the least control. There will be dozens of products from many vendors, all of which must share common characteristics if an assemble strategy is to succeed.

The first success criterion in vendor selection has to be a healthy business model with enough profit to allow the vendor to focus on its customers and product line rather than continually focusing on the bottom line. Churn in product strategies, support structures, and markets is often driven by the bottom line and is always disruptive. Additionally, a vendor's competitive and customer strategy must be structured to allow the vendor to serve and compete at the component (versus solution) level and thereby avoid some of the temptations of manipulating customers within a proprietary or closed environment.

Second, it is critical that the terms and conditions of the vendor's offerings fully support a component-oriented strategy. From a pricing point of view, granularity of both individual components and concurrent (versus authorized) user charges is essential. In terms of maintenance pricing, support structures, and trouble-call management and charging, support offerings must also be granular, such that an institution need only pay for support for the components actually in production. Additionally, license charges for development and test environments (distinct from production) must be zero or near zero to enable the disciplined integration and deployment processes noted above.

The final success criteria is the vendor's technology itself. The vendor's components need to be "plug-able." This includes open interfaces for transaction processing, data structures, and external authentication for both directory management and remote program calls. It also includes standards-based support for both Web-browser access (including Internet Explorer and Netscape) and full-client interface support across PC, Mac, UNIX, and Linux workstations and, eventually, handheld devices. Optimally, a vendor will also present the institution with an open data model and a choice of database and server platforms. Of course, the products need to be current in terms of their product life

cycles and underlying technologies.

A final point regarding vendor technology and the assemble strategy takes open Internet, authentication, data, and programming standards to their next logical step: Web Services architecture. As noted in an *EDUCAUSE Quarterly* Viewpoint by Bernie Gleason of Boston College, the promise of Web Services architecture is to enhance the interoperability and minimize the integration expense associated with assembling disparate applications.

We have focused on what is required for a successful assemble approach in lieu of a broadly accepted Web Services architecture, which is still years away. Although the practical application of Web Services is in the future, it should influence your decisions today. Institutions such as Northeastern will give preference to vendors supporting both the concept and eventual rollout of open (versus proprietary) Web Services architecture standards.

In Conclusion

Our goal in this discussion has been to demonstrate that all institutions must assemble and to outline what it takes to succeed with this application and service strategy. We do not attempt to address the reality that within an assemble approach there is actually a continuum ranging from "buy as much as you can, then assemble" to "make as much as you can, then assemble." Each institution must base its make/buy decisions on a myriad of institutional, functional, constituent, and internal technical criteria not examined here.

As we have demonstrated, the current and expanding demand for enterprise portals and data requires IS organizations to provide unprecedented levels of integration and customization, which are best enabled by an assemble strategy. Continued volatility and consolidation of software providers serving higher education requires that IS leaders ensure independence and flexibility going forward—attributes best protected by an assemble strategy. Most importantly, because no single vendor will have all the capability required to serve a university, and, conversely, because no university will have the capability to build all it needs, the assemble strategy is the most effective strategy for competitive advantage and a robust, sustainable technology environment. We have no choice. *e*

Acknowledgments

We would like to express our appreciation to Larry Conrad of Florida State University, Frank Urso of Harvard Business School, and Charlie Moran of Blackwell Consulting, as well as numerous colleagues at Northeastern University, for their insights, review, and comments.

Bob Weir (bobweir@neu.edu), formerly of IBM, is Vice President and Rick Mickool (r.mickool@neu.edu), formerly of Babson College, is Executive Director, Information Services, at Northeastern University in Boston.