BUILDING SOFTWARE? CURB YOUR ENTHUSIASM

LAUNCHING TOO MANY SMALL APPLICATIONS CAN BE A BIG HEADACHE. SOLUTION: A SHARED INFRASTRUCTURE.

BY PAUL A. STRASSMANN

SOFTWARE DEVELOPMENT IS PROLIFERating. A key reason: It is easier to start a new software project than enhance existing programs. This is how corporations and the government accumulate applications that have a significant amount of common or overlapping code, yet continue as separate systems.

There are justifications for such waste and proliferation. As Capers Jones points out in his book *Software Assessments, Benchmarks and Best Practices* (Addison-Wesley, 2000), when the scope of software applications grows to more than 10,000 function points—a measure of inputs and outputs from a system—36% of projects are late and over budget. The chance of a project being cancelled grows to more than 40%. With odds like that, starting a new application becomes preferable to enhancing the functionality of an existing one.

There are other reasons for launching lots of small projects instead of enlarging current ones. According to Jones, application requirements are seldom more than 50% complete. Therefore, keeping projects small minimizes the chance of setting off in the wrong direction.

Finding and fixing bugs remains the most expensive software activity, and when an application gets too big, the debugging process overwhelms the capacity of programmers to keep track of consequential errors. Testing an application will get out of control as the size of an application increases because less than 50% of the code can be verified as soon as an application passes 10,000 function points. In addition, existing systems usually depend on obsolete technologies and have insufficient funds for a major upgrade of legacy software code.

Yet, the decision to launch yet another software project invites chaos. There are more than 600 programming languages, some 50 metrics for measuring progress, and more than 30 fundamentally different development methods for compliance with any of 20 standards. There are at least 15 different approaches to software testing. Because programmers like to pick the latest method for doing their work, any new software investment will result in increased diversity.

Fragmentation of systems is buttressed by the need to set up special-purpose databases. Special-purpose infrastructures are conceived for each new investment. When subcontractors implement such projects, they tend to deliver results that foster unique solutions. This is how we end up with yet another "island of automation" with its own rules, methods and support staff.

If an organization adds "small" applications over several years, it will wind up with a huge collection of systems that perform comparable but not identical functions. Each of these will consume maintenance costs that cannot be shared across applications.

Thus, maintenance staffs will keep growing and exceed manpower available for new applications. For instance, the Department of Defense has tens of thousands of systems delivering results that a limited number of finance, personnel, logistics and asset management staff could only support properly if a way were found to eliminate the incentives for starting yet another quick answer.

Organizations must stop building diverse infrastructures and focus instead on consolidating applications.

There's no reason why a corporate department must build dozens of finance applications, each generating thousands of on-demand reports. There's no reason why each has to depend on an infrastructure consisting of special-purpose networks, routers, switches, servers and databases.

Organizations must stop building diverse infrastructures; they must focus instead on consolidating applications. Standardization of a corporatewide infrastructure and tight control over communication protocols make it possible to proceed with the merger of applications. The outcome: a unified infrastructure that serves shared needs. Many solutions can then be supported from shared databases. What you get then can be defined as pure service-oriented architecture.

The establishment of a portal can help accommodate local adaptations of applications. Changes can be made only to the small amount of code that controls the user interface, yet allow applications to be accessed. The separation of the portal from the databases and the infrastructure makes it possible to upgrade technologies only when required. Instead of making an entire application obsolete, we can now replace only the code that needs replacing. For instance, upgrading the infrastructure can be done as a service for all apps, rather than as a fix for each one.

By taking such an approach, you can save money. (For an example showing a 39% software cost savings, see "Measuring Results: Is Consolidation Paying Off?" p. 83.) Better still, applications can be slimmed down so that the money saved can be re-applied to innovation.

PAUL A. STRASSMANN, A PROFESSOR AT GEORGE MASON UNIVERSITY, IS A FORMER TECHNOLOGY EXECUTIVE AT XEROX, NASA AND KRAFT. HE CAN BE REACHED AT PAUL@STRASSMANN.COM.

PHOTOGRAPH BY STEVE FREEMAN