The Data Doughnut and the Software Hole

Neville Holmes
University of Tasmania

In computing, the data is the end, the programs merely the means.

Last September's *IEEE Spectrum* focused on "custom-made enterprise software and its many spectacular failures—the kind that bankrupt companies and cost governments and whole industries tens of billions of dollars a year." I read it at a time when just such a failure—of the Australian Customs Service's Integrated Cargo System—made the headlines here. My memories of the spectacular mid-1970s failure of the Mandata system, intended to keep track of all Australian Federal public servants, came to mind, along with a strong feeling of unease.

Reading the *Spectrum* special report issue's three feature articles left me with the impression we were missing something basic. It occurred to me then that the computing profession was not seeing the woods of what their client enterprises are actually trying to do for the trees of how the profession tries to do it for them.

SOFTWARE FAILURE

Consider the common factors listed in the last of the three articles, "Why Software Fails" by Robert S. Charette (www.spectrum.ieee.org/sep05/1685):

- unrealistic or unarticulated project goals;
- inaccurate estimates of needed resources;
- badly defined systems requirements;
- poor reporting of the project's status;
- unmanaged risks;
- poor communication among customers, developers, and users;
- use of immature technology;
- inability to handle the project's complexity;
- sloppy development practices;
- poor project management;
- stakeholder politics; and
- commercial pressures.

These factors break down into three categories. Most are about *project management*, and their high incidence has strong implications for professional education in a time of declining student enrollment. Many computing professionals work in projects, so their education should include a solid grounding in all phases and aspects of project work.

Sloppy development practices and the use of immature technology are about *project implementation*, the software engineers' bailiwick. *Spectrum*'s "The Exterminators" by Philip E. Ross (www.spectrum.ieee.org/sep05/1454) proclaimed that "A small British firm shows that software bugs aren't inevitable." Although the article was about the benefits of formal methods, it is significant that Praxis, the firm in question, develops "mission-critical systems" rather than custom-made enterprise software.

The third category addresses a project's context: communication among customers, developers, and users; stakeholder politics; and commercial pressures. These might be called *project metamanagement*, the aspects that determine what the project is all about.

The source of my unease was a feeling that these were all symptoms and that the disease lay in seeing a gigantic project as the solution to an enterprise's problems. This kind of project seems all too often to become an end in itself, one with which its proponents identify themselves.

ENTERPRISE SYSTEMS

An enterprise is a system. If a computing system will serve as the basis for enterprise-wide operations, that system is best viewed as something the enterprise uses, not something that defines what the enterprise *can* do. Undoubtedly, an enterprise uses its computing system for collecting, storing, and exploiting data.

One aspect that must be considered is the computing system's structure. Projects pose the problem that their proponents seem always to see them as *monolithic*. Yet, given this word's somewhat pejorative connotation, they call them *integrated* instead.

Projects and their integrated enterprise systems are complex, not simply because enterprises become more complex the longer they last. This is most peculiar, considering that programmers understand that they can

Continued on page 98

Continued from page 100

simplify development and otherwise improve it by dividing the tasks to be done into relatively independent components that interact through the simplest possible interfaces.

Generally, an enterprise's computing system can be divided into three relatively independent segments—one each for importing, storing, and exporting data—that interact with each other and with users' programs through interfaces based on the definition of the data to be passed between the segments. Indeed, an enterprise might benefit from having different users' program suites—specifically, data collection and exploitation systems—handle different parts of its operations, such as their suppliers, customers, and employees, interfacing with the data import and export segments. This approach segregates the users' program suites from the segmented enterprise system, much as clients are segregated from their server in a computer network.

Within the segmented enterprise system, data importation would focus on validating incoming data and its source; data storing on availability, consistency, and expedition; and data exportation on request validation and fulfillment. The main benefit of enterprise systems comes from establishing and maintaining a single database for the entire organization. So in this segmented approach, users and clients of any kind would have to go through data import and export segments to get at the data storage segment and its database.

Provided they establish data definitions first, by adopting a segmented enterprise system developers can implement individual segments independently except for their mutual dependence on the data definitions. Different teams could implement different segments, even a mixture of software vendor teams and in-house teams, and they could develop and run the segments on different computers.

This benefit seems to me of potentially great value, yet I saw nothing in the *Spectrum* special report or the practical *Computer* article titled "A Roadmap for Enterprise System Implementation" (Diane Strong and Olga Volkoff, June 2004, pp. 22-29) to imply that the segmented approach has been considered anywhere. If developers do use the segmented approach, we must wonder why general reporting doesn't make this clear or even emphasized. We must also wonder why developers call enterprise systems *integrated* rather than *segregated* or *segmented*.

The main benefit of enterprise systems comes from establishing and maintaining a single database for the entire organization.

THE PROJECT APPROACH

Enterprise system development projects exhibit an even greater peculiarity in that, according to Strong and Volkoff, "an organization can only realize an ES's benefits by going through a lengthy and costly implementation process—one that almost guarantees spending more time and resources than for any project the organization has yet undertaken."

Consider a highway system. Every bit as complex as an enterprise system, it moves freight and people from one place to another. Yet changes to any highway system have always been made incrementally. The system's design allows making significant changes such as the addition of bypasses, bridges, and toll roads with little disruption to traffic and without the complete abandonment of its existing structure.

Why shouldn't enterprise systems be like highway systems and be developed incrementally? Simply dividing the enterprise computing system into relatively independent segments and segregating the users' program suites takes a first step in that direction. With a design of this kind, developers can add new user interfaces to data import and export segments, and even new data import and export segments themselves, at any time, without changing the data storage segment.

A different problem arises when the data storage segment needs changing, which requires changing the data definitions. The solution lies in making the interfaces between the system components symbolic by coupling data fields to names. How this would work can perhaps be seen best by explaining yet another precedent from the mid-1970s.

When display screen terminals began replacing typewriter terminals, Cobol programs became overloaded with code for formatting and filling out the display screens and for getting data from the keyboard through fields defined on the screen. Developers commonly solved this problem by having a screen definition file mediate between the Cobol program and the terminal driver. The screen definition file determined what the terminal driver would do with named data coming from or going to the Cobol program: Data moving between the terminal user and the program was named so that the program didn't need to deal with the screen handling and so that different users could have different screen definition files to suit their needs.

Likewise, if similar data definition files specified the interfaces between the components of a segregated and segmented enterprise system, adding definitions to the database need not affect the use of existing interface definitions or the operation of existing programs. Further, if the interface definitions included specification of representations, the interface process could adapt to representation changes in the database. If developers defined new data names, however, new or modified import or export segments would be needed, but the use of symbolic interfaces would greatly simplify transition.

DATA AND INFORMATION

The segregated approach lets users' programs communicate with the enterprise's database across an interface to the segments mediating their

database use. The users' programs present data to and elicit data from human users as effectively as possible. The emphasis should be on properly informing and effectively empowering all users, making sure they understand what is going on and required of them. In contrast, the mediating segments protect the database within its data storage segment from improper use and invalid data.

The users' programs perform *information* management by informing users and getting data from them. The three kinds of segments perform *data* management by ensuring the enterprise database's health and welfare.

Approaching enterprise systems in this way involves several important implications that spring from seeing that information management, being concerned primarily with the human user, contrasts with data management, which primarily concerns automatic data manipulation. Given this contrast, people in the information management area need a different background and different skills than people in the data management area.

For the enterprise, this implies that their information people can and even should be separated from their data people. The data people belong with the machinery in a back room, while the information people belong with the users in the shop front. Indeed, much can be said for an enterprise that recruits information professionals from their user employees. Much too often, an enterprise treats its employees as unthinking automatons to be driven by the mighty Integrated Enterprise System. Surely this is at least partly responsible for many of the system failures that *Spectrum* lists. Users and information professionals must work in close partnership for both to be fully effective.

For the computing profession, this implies the need for bifurcation because few people have the talent and education to succeed professionally in dealing with data and information and with machinery and its users. Ironically, this bifurcation has happened in some universities where information systems departments compete with computer science departments for students. Cooperation on the basis of a split between information and data professionals should replace such competition.

When private or government enterprises set out to spend vast sums on computing technology to improve their functioning in one way or another, they should look primarily for a data handling system, not for a lump of software. Their data and their users are what they must look after, not the programs and their suppliers. Data is the substance of their doughnut; their programs merely shape the doughnut and supply it to the users.

A curious development in the computing industry has been the dropping of its once universal cognomen, *data processing* or *DP*, in favor of the more pretentious *information technology* or *IT*. What we really need is both, with the proper distinctions enforced.

Neville Holmes *is an honorary research associate at the University of Tasmania's School of Computing. Contact him at neville.holmes@utas.edu.au. Details of citations in this essay, and links to further material, are at www.comp.utas.edu.au/users/nholmes/prfsn.*