Iris Vessey and Robert Glass

There are many different roads to follow when it comes to problem solving.

STRONG Vs. Weak Approaches to Systems Development

Systems development is, fundamentally, a problem-solving activity. A problem in an application domain is transformed by the systems development process into a solution in the computer's implementation domain.

What can systems development specialists borrow from the traditional literature on problem solving? Are there ideas in this older discipline that can be reused?

There is one particularly vital notion that has important implications for both the theory and practice of systems development. The problem-solving literature distinguishes between "strong" and "weak" problem-solving approaches. Strong methods are those designed to address a specific type of problem, while weak methods are general approaches that may be applied to many types of problems [7].

The words "strong" and "weak" are deliberately chosen. A strong method, like a specific size of wrench, is designed to fit and do an optimal job on one kind of problem; a weak method, like a monkey wrench, is designed to adjust to a multiplicity of problems, but solve none of them optimally.

It is the purpose of this article to pursue this notion of "strong" and "weak" problem-solving approaches more thoroughly from a systems development point of view. Are we making use of this notion in today's theory and in today's practice? If not, how can we change what we are doing to make use of strong problem-solving approaches?

The Nature of Systems Development

Systems development is generally acknowledged to be an intellectually complex activity. This complexity is magnified by the need for expertise in two disciplinary areas—the area of the problem being solved (the application domain), and the area of constructing a software solution (the systems and software discipline).

The application knowledge component of this dual disciplinary problem is significant. Blum [1] states "… much of what we consider to be software development is actually application domain problem solving …" A successful systems developer is one who masters the large amounts of knowledge present in both disciplinary areas.

This complexity inherent in systems development

typically is addressed by employing standardized methods to develop systems.

Method-Based Approaches to Systems Development

Two method-based paradigms for systems development can be identified. The first approach is based on methodology; the second on technique.

Unified methodology approach. Methodologies are a formal attempt to address complexity through the use of standard, predictable approaches to systems development.

Current methodologies tend to focus on principally one unit of decomposition, but they differ on what that unit of decomposition is. Most common methodologies base decomposition on either process or data, or some mixture of the two.

The process approach to handling complexity is seen, for example, in the structured techniques—structured analysis, design, and programming. It is, perhaps, the oldest and most widely used methodology, and in addition is the one most frequently referenced in the information systems literature. The structured techniques all primarily use process decomposition, although the seminal works on structured analysis also included normalization of data as a secondary focus of the methodology.

The data approach to handling complexity is seen in information engineering. It has its origins in the entity-relationship approach to modeling data. Information engineering initially employs data decomposition at the enterprise or organization level to handle problem complexity, the reason being that an enterprise's data is, in general, more stable than the processes used to act on that data. After the initial data analysis, systems projects are developed using process decomposition, which is, therefore, a secondary emphasis.

Thus we see that the difference between process- and data-oriented methodologies is one of initial emphasis. In the end, both orientations must be considered.

The object-oriented approach to handling complexity considers both data and process as a package. An object is a component of the problem's world, a cohesive collection of data coupled with the processes (methods or operations) acting on that data. The act of systems development using the object-oriented approach interleaves analysis and design of objects with analysis and design of the processes relating to those objects. The rationale for the object-oriented approach is that application problems often evolve around real-world objects and the ways in which they interact.

Whether the systems developer employs a process, data, or object-oriented approach, the methodology underlying the approach will be standardized and cohesive. Such a methodology is referred to here as "unified."

Technique approach. Methodologies are, of course, collections of related techniques. Disaggregating most methodologies results in identifying techniques of varying utility—some techniques are exceedingly valuable, some are relatively valuable, and some have only marginal value. For example, an earlier component of the structured techniques was the notion of a "Chief Programmer Team." Over the years, as it became apparent that this was not a successful part of the structured techniques, that notion was eliminated.

Because of this variability in the utility of constituent techniques, there are those who use collections of appropriate techniques rather than unified methodologies. With this approach, systems developers are trained in the use of "best of practice" techniques known to have been successful in solving an enterprise's problems.

Combining Method and Application in Systems Development

In the following, we suggest some ways of identifying and moving toward stronger problem-solving approaches (and away from weaker ones), by using methods that are specifically suited to the needs of the application. But first, to construct a theoretical foundation for this approach, we examine the theory of cognitive fit.

Theory. Cognitive fit is the notion that problem-solving elements such as problem representation, methods, and/or tools should support the strategies (or processes) required to perform the task [10]. If we apply that notion to systems development, we see that the methods used should be chosen to best support the various tasks of systems development.

When the types of information in the method and the application match, the problem solver can use processes that emphasize this type of information, facilitating the construction of a mental representation of a solution. Hence, cognitive fit leads to an effective and efficient problem solution. If cognitive fit is lacking, it does not mean that the problem cannot or will not be solved; it simply means that the solution will be less effective and/or less efficient. Cognitive fit, then, embodies Newell's notion of a strong approach to problem solving, because it is based on matching solution approaches to the problem at hand.

Matching methodology to application. Traditionally there has been tacit acknowledgment in the literature that we need to consider the application when choosing a systems development method. For example:

Jeffries et al. [6] state " … the formal literature on software design lacks a mapping between the types of problems and the appropriate design methodology."

Jackson [5] writes: "Again and again writers on

development methods claim to offer an analysis of a problem when in fact they offer only an outline of a solution, leaving the problem unexplored and unexplained."

Potts [8] says: "... we may have made as much progress as we can reasonably expect in domain-independent languages, methods, and tools. Nobody talks about domain independence in hardware engineering because its disciplines differ so greatly in their underlying principles and skills. The same is true of many application domains that are suited to software implementations."

In general, however, only lip-service has been paid to the issue of which problem representations, methods, tools, techniques, or methodologies are best used for a given set of circumstances.

To use the theory of cognitive fit to select an existing methodology for developing a given application, we would prefer first to develop classes of applications, and then consider which methodology is best suited to each class. But little research has been conducted on the nature of applications [3], whereas there has been at least some attempt to classify methodologies. Therefore, in what follows we classify methodologies into process, data, and object-oriented approaches, and consider which applications might be best handled by each methodological approach.

What applications might lend themselves to starting with an analysis of processes? Application problems that are more about functionality (processes or algorithms) than the data manipulated by the processes. Most scientific-engineering applications, for example, are of this kind. In the information systems area, payroll, inventory, accounts receivable, and accounts payable are often characterized in this way.

What applications might lend themselves to starting with an analysis of data? Those where the data itself is more complicated than the processes to be performed on it. Applications that deal with record keeping, such as medical records systems, are frequently of this type. The processes for these types of applications may be relatively simple, but the organization and access of the system data may be quite complex.

What applications might lend themselves to starting with an object-oriented view? Here, the answer is less clear. Advocates of the approach tend to claim it is application-independent, that is, suitable for all applications. However, there is little research to explore either that thesis or its opposite. One might speculate that applications for which the processes and data are intimately related (they occur and are needed together) would be the most appropriate. For example, such an approach might be appropriate in a library book check-out system, where objects (books, clients) are subject to well-defined transactions (check-out, return). Some say real-time systems, characterized by objects such as devices and the operations they initiate or perform, are also best approached in this way.

Matching technique to task. Recall that earlier in this article we proposed the use of disaggregated techniques, rather than unified methodologies, as another approach to problem solving.

Using this approach, systems developers can, at any point in the process, use whatever technique is best suited to the task at hand. Such an approach can lead to greater flexibility, and therefore to greater power.

For example, Sanden [9] discovered that, for a particular real-time application, a combination of Jackson Structured Design and object-oriented analysis could overcome deficiencies that each of the methodologies had when used alone: "Rather than arguing about which one design method is best, we should take an eclectic view and use any combination of approaches that yields important results in a given situation."

Other authors have proposed so-called "multiparadigm" approaches to systems development. With this approach, different paradigms (procedural, functional, object oriented, and so forth) might be used on different portions of a system. Zave [12] states: "(E)ach paradigm is too narrowly focused to describe all aspects of a large, complex system.... The purpose of multiparadigm programming is to let us build systems using as many paradigms as we need, each paradigm handling only those aspects of the system for which it is best suited."

Zave moves very close to the notion of cognitive fit when she says "...application domains must be studied in a new, multiparadigm perspective, producing understanding that eventually leads to standardized paradigm interfaces for each application domain."

Discussion

The pursuit of the notion of using strong rather than weak problem-solving approaches in systems development has led us in several interesting directions. For instance, greater attention is paid to the application domain. It is clear that systems development complexity lies in both the application problem domain and in the software solution domain. But traditional views of the systems development process have concentrated almost entirely on the solution domain, and as a result have produced weak problem-solving approaches that are too general to be very powerful.

We've also reconsidered the notion of a single, standard methodology. We have seen ample reason why a single, application-independent methodology may not be the optimal solution for all systems development projects. But what are the alternatives? We present several answers to that question:

- ***Matching methodology to application***. Employ several methodologies in an enterprise and base the choice of which one to use on the application at hand. Note that this is a single paradigm approach; once the choice is made, it is used throughout the project.
- ***Disaggregated methodologies***. Identify a selection of appropriate systems development techniques, and for a particular project choose those techniques best suited to the application at hand. Note that this is, in a sense, a no-paradigm approach, because the techniques do not necessarily fit into any overall model of how things should be done, and because for any one subsystem different collections of techniques may be used.
- ***Multiparadigm approaches***. Employ several paradigms in an enterprise, and for a particular project use that mixture of them matching the unique demands of the application at hand. Note that this is called a multiparadigm approach because, for any one large, complex system, different subsystems or even different life-cycle phases may each use a different paradigm.

The dilemma with any of these approaches, of course, lies in establishing criteria for choosing among, and interfacing schemes for meshing, the various constituent methods. To overcome this dilemma, one must keep in mind that today's problem-solving approaches, though general, are weak, and that solving the dilemma is the price to be paid for moving toward strong approaches.

Implications for Research

The findings of this study suggest significant research redirection. It has been the goal of most methodological studies in the past to define a domain-independent, general approach to systems development problem solving. The work presented here suggests that, although this approach has led to impressive gains in software productivity through the years, further potential gains within that framework are limited.

Research, these findings suggest, should focus on systems development approaches based on cognitive fit, the matching of methods to tasks. Methods should be studied from the viewpoint of their benefit to particular classes of applications, to facilitate matching method to application task.

Recent research into "method engineering," the idea that standardized methodologies must be tailored to meet specific needs [2], appears particularly promising in this regard.

Implications for Practice

Most enterprises have, by now, adopted some sort of standard methodological approach to systems development, and provided the necessary training and infrastructure to implement it. The findings of this article suggest, however, that these approaches, though useful, are far from optimal.

What is needed are strong problem-solving approaches, based on matching method to task. But there remain many unanswered questions. Which methods or techniques are best for which applications? How are training and infrastructure to be created for the more complicated world of application-focused solution approaches? Where is the research information to help with the needed practical decisions? There are no good answers to any of these questions.

Recent findings that practitioners are not using methodologies as defined by their originators, but rather are tailoring them (see, for example, [4, 11], provide interesting evidence that practice may be beginning to move in application-specific directions, without waiting for research to show the way.

Is the systems development field ready for the dramatic paradigm shift implied in this article? Are researchers and practitioners alike willing to reach as far as is apparently necessary to move in this new direction?

The answers are, at best, unclear. C

References

1. Blum, B.I. A paradigm for the 1990s validated in the 1980s. In *Proceedings of the AIAA Conference*, American Institute for Aeronautics and Astronautics, 1989, pp. 502–511.
2. Brinkkemper, S. and Joosten, S. Special issue on method engineering and meta-modeling. *Info. Softw. Tech.* (Apr. 1996).
3. Glass, R. and Vessey, I. Contemporary application-domain taxonomies. *IEEE Software 12*, 4 (Jul. 1995), 63–76.
4. Hardy, C.J., Thompson J.B. and Edwards, H.M. The use, limitations, and customization of structured systems development methods in the United Kingdom. *Info. Softw. Tech. 37*, 9 (Sept. 1995), P. #.
5. Jackson. M. *Software Requirements and Specifications.* Addison-Wesley, Reading, Mass. 1995, p. 1
6. Jeffries, R., Turner, A., Polson, P. and Atwood, M. The processes involved in designing software, in J. R. Anderson, ed., *Cognitive Skills and Their Acquisition*, Erlbaum, Hillsdale NJ, 1981.
7. Newell. A. heuristic programming: Ill-structured problems, in J. Aronofsky, ed., *Progress in Operations Research*, John Wiley & Sons, New York, 1969.
8. Potts, C. Software engineering research revisited. *IEEE Software 10,* 5 (May 1993), 19–28.
9. Sanden, B. The case for eclectic design of real-time software. *IEEE Tran. Softw. Eng. 15*, 3 (Mar. 1989), 360–362.
10. Vessey, I. Cognitive fit: A theory-based analysis of the graphs vs. tables literature. *Decison Sci. 22,* 2 (Spring 1991),219–240.
11. Vlasbom, G, Rijsenbrij, D. and Glastra. M. Flexibilization of the methodology of system development. *Info. Software Tech. 37*, 11 (Nov. 1995).
12. Zave. P. A compositional approach to multiparadigm programming. *IEEE Software 6*, 5 (Sept. 1989), 15–25.

Iris Vessey (ivessey@indiana.edu) is a professor of Information Systems at Indiana University.
Robert Glass (rglass@indiana.edu) is the publisher of the *Software Practitioner* newsletter and editor of Elsevier's *Journal of Systems and Software.*