Technical note

A proposal to simplify data flow diagrams

by I. Millet

This technical note presents an adaptation of the data flow diagram (DFD) technique whereby each data store symbol represents a database rather than a single table. It is conjectured that this modification makes DFDs easier to create, understand, and maintain. It also reduces an overlap with the entity-relationship diagram technique by curtailing graphical manifestations of the data model in the DFD.

Since the data flow diagram (DFD) technique was introduced in the late 1970s,[1,2] it has become the main process modeling tool for information systems. Recent research has shown that DFDs are the most popular tool taught in systems analysis and design courses: 597 out of 647 schools (92 percent) indicated that they teach DFDs in that course.[3]

Although recent object-oriented design methodologies such as the Unified Method by Booch and Rumbaugh[4] may attempt to replace functional modeling,[5] DFDs seem to have certain advantages. Empirical research by Vessey and Conger[6] shows that DFDs are easier to learn and to use, at least by novice users. Similarly, Agarwal et al. showed that DFDs produce higher-quality solutions in process-oriented tasks and are not inferior to object-oriented methodologies even in object-oriented tasks.[7]

If DFDs are so easy to use, one may ask, where is the problem? Why bother making DFDs even easier and more flexible? There are several reasons for adopting the modification proposed in this note. First, since the DFD is a popular tool, it is easy to justify the effort to improve it. Making DFDs simpler and more flexible may help us to also reduce the tension between *discipline* and *creativity* in the practice of systems development.[8,9] Finally, by removing the data modeling aspect of DFDs we can avoid redundancy and conflict with the popular entity-relationship diagram (ERD) methodology.

The overlap with ERDs

According to Whitten and Bentley,[10] process modeling is a "technique for organizing and documenting the structure and flow of data through a system's Processes and/or the logic, policies, and procedures to be implemented by a system's Processes." The problem is that ERDs[11] already model data structures. As shown below, asking DFDs to depict which tables are required by the system causes duplication of effort, clutter, and inflexibility.

Figure 1 depicts a simple DFD, adapted from Fertuck.[12] According to the Gane and Sarson[2] notation used here, the rounded boxes represent *processes*, such as "enroll students," which transform incoming data flows, represented by arrows, into outgoing data flows. An open-ended rectangle represents a *data store*, typically a database table such as "students," which stores data for use at a later time. A plain rectangle represents a *terminator* or *external entity*, such as "student," which is an external source or destination for information.

0018-8670/99/$5.00 © 1999 IBM

Figure 1 A DFD with data stores for each table

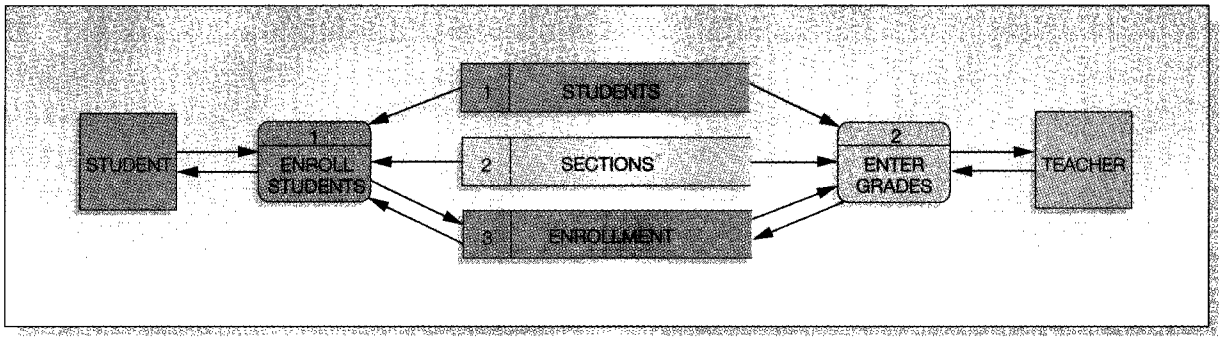
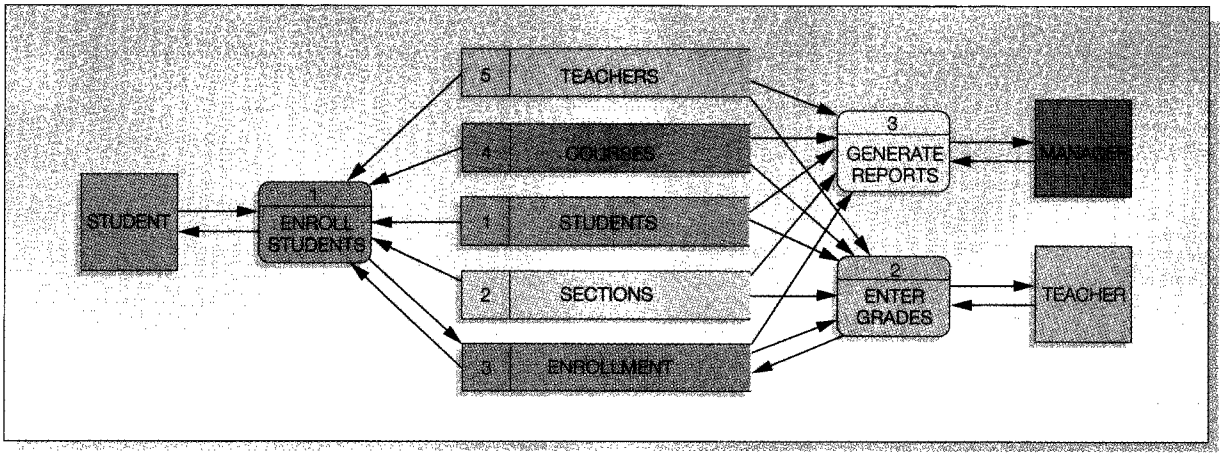


Figure 2 Adding two data stores and a process to Figure 1 leads to a cluttered DFD



This is a rather simple case and it is further simplified by the omission of other necessary data stores such as "teachers" and "courses." Still, this approach of assigning data store symbols to every table makes this diagram more complex and less flexible than it should be.

If the analyst realizes that certain tables should be added to the system, the change will not be limited to the ERD; this DFD would also have to be redrawn and would become even busier. Similar changes will have to be made throughout all levels of the DFD hierarchy. After one or two cycles of such changes, the analyst will probably be less inclined to use DFDs in future assignments.

Furthermore, consider adding a third process to this diagram, say "generate reports." Since this third process may require access to many tables, we immediately have either data flows crossing one another or data store replicas cluttering the diagram. Figure 2 demonstrates how the addition of two more data stores and one more process causes a rapid deterioration in the visual appeal of the DFD. Things can become much uglier when designing DFDs for more complex situations.

These limitations are self-imposed due to the insistence on using DFDs to model not only processes, but also data structures.

Proposed adaptation. The solution to these problems is to let DFDs and ERDs serve different purposes. Allow ERDs to focus on modeling data, and let DFDs focus on modeling processes. If we adopt a guideline whereby each data store can represent a whole

Figure 3 DFD obtained when Figure 1 is modified such that there is one data store for a whole database

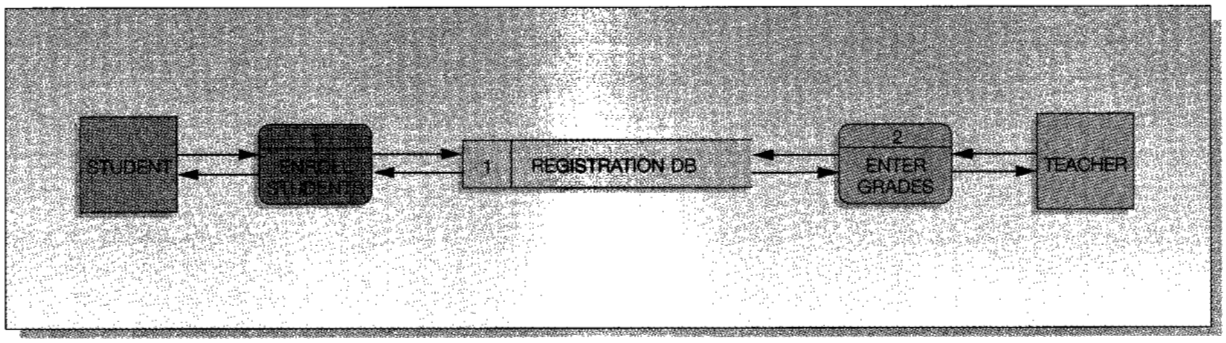
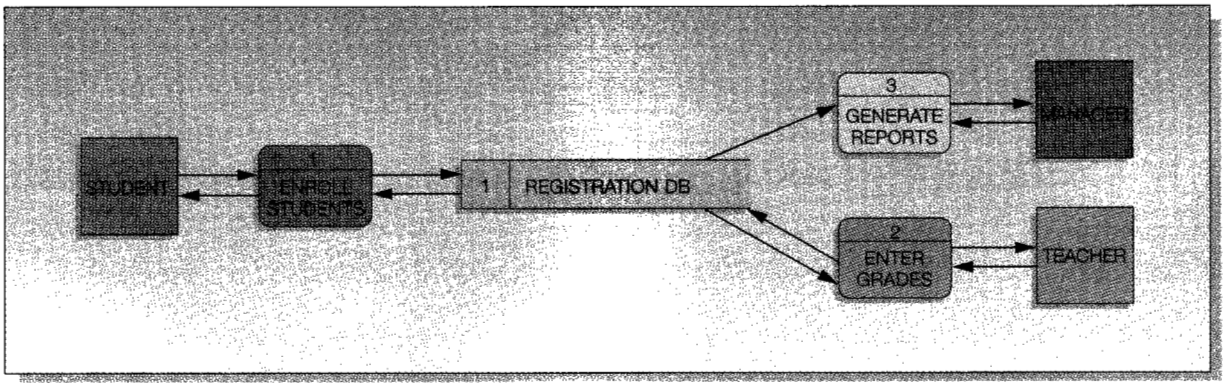


Figure 4 The DFD of Figure 3 remains simple when adding tables and processes



database, then we can model the original system (Figure 1) using the DFD depicted by Figure 3.

This small change has reduced the number of data stores from three to one, and the number of data flows from twelve to eight. As one more example, Figure 4 shows how the DFD in Figure 2 becomes much simpler when we apply the new guideline.

By comparing the DFD in Figure 4 to the one in Figure 2 we can see that using a single data store to depict the whole database allows us to add more processes without resorting to spaghetti data flows or to data store replicas. We reduced five data stores to one, and 23 data flows to 11. The new process can find easy access to the single data store symbol since the database is not surrounded by other data store symbols.

Although the diagram in Figure 4 is simpler and easier to understand, the most important impact is the isolation from changes in the data model. Adding or dropping tables would have no impact on the new DFD, unless these changes also reflect changes in process design. For example, adding a "classroom" table to the database would require no changes to the DFD in Figure 4. The single data store symbol encapsulates the database structure and shields the process model from such changes.

Concluding remarks

At the lowest level of decomposed DFDs, primitive process specifications (PPSs) identify records and data elements used as input and output to processes. This is a valid area of overlap between data models and process models. Should DFDs then model data structures after all? The answer lies in the integration provided by modern CASE (computer-assisted software engineering) tools. The same repository used by the ERD tool to maintain information about tables and

record structures can be used by the DFD tool to specify records and data elements as input, output, and data flow components. The key then is not the isolation of DFDs from data aspects, but the isolation of the graphical DFD representation from such issues.

Although object-oriented analysis and design methods are adding useful techniques to our systems analysis toolbox, we still lack descriptive and prescriptive research on the application of these tools. Such research can increase the likelihood of teaching and practicing effective systems analysis techniques.

For the last five years I have been teaching students the DFD technique using this adaptation. The feedback has been very positive. I must concede, however that, while the examples above seem compelling, the benefits of the proposed adaptation are mere conjecture at this stage. I can only hope that those who try this technique report that indeed it makes DFDs easier to create, understand, and maintain.

Cited references

1. T. DeMarco, *Structured Analysis and System Specification*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1978).
2. C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1979).
3. R. McLeod, Jr., "Comparing Undergraduate Courses in Systems Analysis and Design," *Communications of the ACM* **39,** No. 5, 113–121 (May 1996).
4. G. Booch and J. Rumbaugh, *Unified Method for Object-Oriented Development*, Technical Report, Rational Software Corporation, Cupertino, CA (1995).
5. I. M. Holland and K. J. Lieberherr, "Object-Oriented Design," *ACM Computing Surveys* **28,** No. 1, 273–275 (March 1996).
6. I. Vessey and S. A. Conger, "Requirements Specifications: Learning Object, Process, and Data Methodologies," *Communications of the ACM* **37,** No. 5, 102–113 (May 1994).
7. R. Agarwal, S. P. Atish, and T. Mohan, "Cognitive Fit in Requirements Modeling: A Study of Object and Process Methodologies," *Journal of Management Information Systems* **13,** No. 2, 137–162 (Fall 1996).
8. R. L. Glass, "The Faking of Software Design," *Journal of Systems and Software* **26,** 101–102 (1994).
9. R. L. Glass, "A Theory About Software's Practice," *Journal of Systems and Software* **28,** 187–188 (1995).
10. J. L. Whitten and L. D. Bentley, *Systems Analysis and Design Methods*, McGraw-Hill, Inc., Boston (1998), p. 121.
11. P. Chen, "The Entity-Relationship Model—Towards a Unified View of Data," *ACM Transactions on Database Systems* **1,** No. 1, 9–36 (March 1976).
12. L. Fertuck, *Systems Analysis and Design with Modern Methods*, Wm. C. Brown Communications, Dubuque, IA (1995), p. 358.

Accepted for publication September 16, 1998.

Ido Millet *Penn State Erie, The Behrend College, School of Business, Station Road, Erie, Pennsylvania 16563-1400 (electronic mail: ixm7@psu.edu).* Dr. Millet is Assistant Professor of Management Information Systems at Penn State Erie, The Behrend College. He received a B.S. in industrial engineering and management from the Technion–Israel Institute of Technology, an M.B.A. from Tel-Aviv University, and a Ph.D. in information systems from the Wharton School, University of Pennsylvania. He has 12 years of industrial experience, including systems analysis and project management for large-scale information systems, consulting, and development of PC-based management information systems. He is the author of articles in *Systems, Objectives, Solutions, Information and Management*; *Interfaces*; *Journal of Information Technology Management*; *CIO*; *Journal of Multi-Criteria Decision Analysis*; *PowerBuilder Developers Journal*; *Journal of Systems and Software*; *Journal of Business Ethics*; and the *European Journal of Operational Research*. His research interests include management reporting systems, issue management systems, and the analytic hierarchy process.

Reprint Order No. G321-5700.