Approach to Management Information System Design

Joseph George Caldwell, PhD
March, 1993; updated 29 April 2009

Contents

Overview of Approach to Management Information System (MIS) Design

A classical systems and software engineering approach is recommended to assure the development of a management information system that is fully responsive to a client's performance objectives and resource constraints. This approach includes the following major components:

o Systems analysis, which includes information needs assessment, requirements analysis, and requirements specification

o Systems design, which includes synthesis of alternatives, cost-effectiveness analysis of alternatives, specification of criteria for selecting a preferred alternative, selection of a preferred alternative, top-level design, and detailed design

o Systems implementation, which includes forms development, specification of data collection and entry procedures, development of editing and quality control procedures, software coding and testing, development of training materials and training, integration of the software components with other system components (e.g., personnel, communications, data transfer and assembly, report preparation and distribution, feedback), and system-level testing

o Systems operation and support, which includes not only routine operating procedures but also provision for on-going system financing and management, quality control, software maintenance and updating, personnel training, and system maintenance and improvement (including periodic review of system performance and diagnosis and correction of problems)

While the preceding system development phases are completed in sequence, there is some time overlap between them. The following paragraphs discuss aspects of each of the above major components. Our approach to management information system design is based on the modern

software/system engineering discipline, which consists of structured analysis and structured design (top-down design). (See the list of references for several books on the modern systems and software engineering discipline.)

The first step in an MIS development task is the development of an MIS management plan, which describes the major tasks and schedule of work for the MIS activity.

Systems Analysis

Systems analysis includes a review of the present information system to assess its capabilities and shortcomings; specification of system goals, objectives, and constraints; a survey of potential system users to assess their information needs; identification and analysis of alternative system concepts; specification a system concept; and system requirements analysis and specification. This phase includes an analysis of major system functions and the development of a system architecture (identification of the major system components and their interrelationships).

Heavy emphasis is placed on end-user requirements. It is essential to involve the end-user in the system requirements activity, to insure the development of a system that is fully responsive the user's needs. The review of the current system and survey of potential users can be done by a variety of means, including review of documentation, site visits, questionnaire surveys, interviews, and focus-group discussions.

Systems Design

The systems design phase is generally broken into two subphases, top-level design and detailed design. Top-level design consists of the identification of the major system components and their functions. In order to specify the top-level design, a number of alternative system design concepts are synthesized and evaluated in terms of a variety of selection criteria, which include cost (implementation, operation and maintenance), performance, satisfaction of requirements, development risk, flexibility for expansion/upgrading, and political acceptability. The important aspect of top-level design is to present several feasible solutions to the system managers and users, to describe their advantages and disadvantages, and to obtain a consensus on a preferred design concept. An example of a design decision is the decision concerning which functions should be implemented using computers and which should be manual (e.g., should data collected at a regional level and needed at a central level be transmitted via the Internet (e.g., virtual private network or e-mail) or hand-carried on a memory stick).

Detailed design consists of specifying all of the system components and functions in detail. In the detailed design phase, decisions are made concerning what data elements are to be collected, how they are to be coded, how frequently they are to be collected, and at what levels of detail they are to be aggregated. A critical design decision concerns the "units of analysis" -- the item on which the data are to be measured, such as an individual, a household, a school, a clinic, a farm, a village, or a region. The decision on the unit of analysis has a significant impact on both the cost of the system operation (especially the data collection burden) and on the flexibility of ad-hoc reporting. This design decision is particularly important. While it is an easy matter to revise a data entry screen or report format, it is not possible to produce a desired report about a particular type of unit if data on that unit are not included in the data base. For example, if it is desired to produce

a report about the frequency distribution of villages by some characteristic, village-level data must be included in the data base (or capable of being constructed by aggregation of lower-level units). If it is desired to produce a frequency distribution of facilities by size, facility data must be included. If it is desired to produce distributions of families with particular characteristics, data on families must be included in the data base.

It may not be practical to include data in the database on all of the members of a population. If a population is very large (e.g., families, households), consideration should be given to the use of sampling procedures for data collection and reporting. A major advantage of most management information systems, however, is that they include the total population of interest, so that statistical analysis is not required to analyze the data.

For a software subsystem, the structured analysis / structured design approach involves the use of techniques such as data flow diagrams, functional decompositions, and structure charts. Since we recommend making heavy use of fourth-generation database management software, the amount of detail depicted in the detailed software design is generally minimal.

The detailed design phase also identifies the initial reports to be produced by the system (reporting levels, frequency, content, and format). With fourth-generation database software it is an easy matter to change reports or develop new reports, so the specification of the output reports is not critical (since it will almost surely change over time).

The amount of effort expended in detailed software design for management information systems is often not very great, for two reasons. First, through the use of fourth-generation software it is relatively easy to implement modifications of report content and format (as long as the necessary data are available). Second, it is recommended to adopt a rapid-prototyping approach to the software development. Fourth-generation languages are ideally suited to this approach, which consists of developing an initial version of the software, testing it, modifying it, and then producing a second, improved, version. This iterative process is repeated one or more times until a desired version is obtained. With the rapid-prototyping approach, the "design" is continually evolving, and a minimum amount of effort is expended in documenting each of the prototypes.

The system design phase specifies what computer equipment is to be used. Because of the very high computing power (fast speed, large memory, long word-length) of current-day microcomputers, the large capacity of hard drives, the tremendous variety and capabilities of available application software, and the reasonable cost of hardware and software, current microcomputer-based systems will be able to accomplish many of the desired system requirements at acceptable cost and level of complexity. Because of the large diversity of choice, however, and because the acquisition and training costs are not negligible, it is necessary to carefully consider the alternatives and make a good selection. Experience with a wide variety of software and hardware is a valuable asset in guiding the hardware/software selection process.

The significant processing capabilities of microcomputers makes them appropriate candidates for many practical MIS applications. Major categories of software involved in a microcomputer-based MIS system are database software, spreadsheet / presentation graphics, and statistical analysis packages. Depending on the system size and number and location of users, networking may be a useful option.

Larger applications may exceed the processing capabilities of microcomputer-based systems, in which case larger (e.g., minicomputer-based) systems may be appropriate.

The following paragraphs mention some practical aspects of system design.

System Size. Modern microcomputers are so powerful that most MIS applications can be done using commercial off-the-shelf (COTS) microcomputers (e.g., those using the Microsoft Windows operating system). Previously, a major constraint on system size was the computer word length. For example, microcomputers using a 32-bit word length could express single-precision integers only as large as 2,147,483,648, and the maximum file size was also this size (two gigabytes). Scientific computers used 64-bit word lengths back in the 1960s (e.g., Control Data Corporation computers), whereas commercial computers (e.g., IBM, Sun) have been very slow to use large word lengths. Now that desktop microcomputers have moved to 64-bit word lengths, the word length is no longer the constraint that it once was. A desktop computer is now sufficiently powerful to handle almost all MIS applications, from the point of view of data precision, file size, and processing power. About the only reason for using a large database management information system such as Oracle is advanced features, such as security. Small systems such as Microsoft Access or SQL server can accommodate most applications – they are just as fast as the larger systems and far less expensive. A major factor affecting database design is the organizational scale of the application – whether it is a "desktop" application or an "enterprise" application.

Data Models. When someone speaks of a "database" today, he is almost always referring to one based on a "relational" model. There are several types of database models, including indexed-sequential, network, and relational. For general-purpose applications, such as management information systems, the relational model is almost always used. Relational databases are based on a mathematical framework ("relational calculus") that makes it easy to maintain data integrity and perform ad-hoc queries. The only serious drawback of relational databases is that they are not very fast. They are generally not good choices for large-scale, real-time transaction processing systems, such as an airline reservation system. For a manager sitting at his desk, the few seconds required to execute a query or generate a report is not a problem, but for a massive system with thousands of concurrent users, the system simply would not function.

In order to use a relational database system, such as Microsoft Access or Oracle, it is necessary to structure the data in a certain form. This process is called "normalization." There are several degrees of normalization, but the one most commonly used is called "third normal form." The data are stored in a set of tables. The table rows are generally called "records," and the table columns are generally called "fields." It is necessary that the entities stored in the database have unique identifiers, called "keys." For a database to be in first normal form, each field is "indivisible" – it contains a unique type of information, and there are no "repeating groups" of information. For example, one field would not contain a person's complete address (street address plus city) and another field his city – the address would be split into two fields, one containing the street address and another containing the city. Furthermore, a record would not contain multiple fields representing successive product sales – each sale would be stored in a "Sales" table, identified by an appropriate key (linking it to other data tables). For a database to be in second normal form, each table must have a unique identifier, or primary key, that is comprised of one or more fields in the table. In the simplest case, the key is a single field that uniquely identifies each record of the table. (When the value of a primary key of a table is stored in another table, it is called a "secondary key.") The non-key fields provide information only about the entity defined by the key and the subject of the table, and not about entities in other tables. For example, data about the population of the city in which a person lives would not be included in a table of house addresses, because city population is an attribute of cities, not of house addresses. For a database to be in third normal form, each non-key field must be functionally dependent on the key, that is, the values of the key completely determine the values of the non-key fields. Said another way, each non-key field is relevant to the record identified by the key and completely describes the record, relative to

the topic of the table. Most commercial relational database systems require that the database be in third normal form. In some cases, to improve efficiency, a database may be denormalized (e.g., by storing an element of data in two fields), but this should be rarely done (and code should be included in the database to ensure that data integrity is maintained (e.g., in this example by updating the redundant data on a regular basis, and making sure that all queries and reports use the latest data)).

Selection of Software. For most applications, the choice is between using a very expensive system, such as Oracle, or a very inexpensive one, such as Microsoft Access or SQL Server. Another option is to use open-source (free) software, such as MySQL.

For most applications, the Microsoft Access database system is a good choice. Microsoft introduced its Access database management system in 1997, and introduced major upgrades in 2000 and 2003. The 2000 version was prone to fail (catastrophic "freezes" with no way to repair the system but to go back to a much earlier version) for larger applications, but the 2003 version was quite stable. The newer versions introduced in 2005 and 2007 are very similar to the 2003 version. Until recently, Microsoft included Access as one of the modules of its Office suite (with Word word processor, Excel electronic spreadsheet, PowerPoint presentation slides, and Outlook e-mail system). The cost of the entire suite was only a few hundred dollars, so the cost of the Access system was essentially zero (since few purchasers used this module, and purchased the suite for the other programs). Recently, Microsoft has "unbundled" the Access system, so that it must be purchased separately in some instances. Even so, the cost is just a few hundred dollars, for a standalone or small-network system.

Recently, Microsoft has decided to discontinue further development of Access, with the goal of shifting users to the SQL Server. A free version of SQL Server (SQL Server 2007 Express Edition) is available, but the cost of the full product is much higher than Access. This is a very unfortunate move for consumers, given the tremendous power and ease-of-use of Access. After ten years of development (from its introduction in 1997), it had evolved to a reliable, easy-to-use system, and it is a shame to see Microsoft withdrawing support for it. SQL Server is not a good substitute for Access. It may be more powerful in some respects, but it is much less easy to use (and more expensive).

Although open-source software is free to acquire, there are other costs involved. There are two major drawbacks associated with it. First, the major open-source database system is MySQL, and it is a very "primitive" system. It is programmed using the SQL language, and does not contain the useful database development tools that Microsoft and other proprietary systems contain. It is hard to use. The second drawback is that the pool of personnel who are familiar with open-source software such as MySQL is much smaller than the pool of personnel who are familiar with Microsoft systems such as Access. In a developing-country context, this can present serious problems. A firm will typically have a more difficult time recruiting information-technology staff who are familiar with open-source systems than with Microsoft systems. This situation will result in higher staffing and training costs.

Query Design. The major factor affecting database performance is the quality of the design of the queries (procedures used to retrieve data). In a relational database system, queries are implemented using the SQL programming language. Systems such as Access have automated tools for constructing queries, called "query builders." If a query is not properly constructed, the time to retrieve the data could be many hours instead of a few seconds. I was once asked to consult on a project in Egypt, where the performance of a database had declined over the two years since its installation to the point where routine queries that had once taken seconds to do

now took hours. The problem was that the developer had tested the system on a small data set, and had not examined the system performance for large (simulated) data sets. As the set increased over the years (as data were added to the database), the time required to retrieve data increased to completely unacceptable levels. I had a similar experience in Malawi, where the simple process of entering data (done by a few data-entry clerks) was bringing the system (a state-of-the art dual processor computer and an Informix application) to its knees.

There are two things that must be done, in order to keep query times fast. The first is to create "indexes" for keys that are used in the processing of data. The second is to perform all "selects" and "aggregates" before performing any table "joins." (A "select" is the process of selecting records from a table; an "aggregation" is the process of aggregating data in a table, such as calculating totals or means of subsets; a "join" is the process of combining data from different tables (which are related by keys). The process of joining tables can be very time-consuming for large tables. In an uncritical approach to constructing a query, the user may construct a single SQL statement that includes the selection, aggregation and joining. For large tables, such queries can require much time, even if the keys are indexed. It is important to break the query into a series of simpler queries, such that the join is made on as small a table as possible. Most queries do not involve accessing all of the data in the database. If indexing is done and if selections and aggregations are performed before joins, the tables to be joined are often small, and the query is very fast. Expensive database systems such as Oracle make a big deal out of "optimizing" SQL statements, and they charge massive amounts of money for such systems. This is wasted money. In all cases, an analytically minded database user can break a query into a series of simpler queries and accomplish the same level of performance.

There are many other factors to be considered in conducting the detailed design of a management information system, such as whether data should be archived year by year or included in a single large database (to facilitate time series analysis); whether the data should be entered at a central facility or on-line via many distributed workstations (e.g., in different districts); procedures to be used for primary data collection (e.g., questionnaires); data-cleaning routines; and security.

Systems Implementation

Systems implementation consists of developing all of the system components -- data collection forms; data collection, transfer and processing procedures; data entry procedures and screens (including on-line edit checking); software; report forms; report distribution; quality control procedures. As mentioned, we recommend the use of an iterative, rapid-prototyping approach to the software implementation. It is highly recommended to field-test major systems in a single geographic area before going full scale. This field testing involves not only software, but all aspects of the system (e.g., data collection procedures, training, quality control).

The importance of allowing for prototyping and field testing cannot be minimized. A key problem faced in developing countries is data integrity. From previous experience, we know that users become much more interested in data integrity after seeing the data they reported in a printout. The system procedures will allow for regular feedback of data to the source levels, for review and correction.

In a developing country, a considerable amount of time must be allowed for implementation. Data that are taken for granted in a highly developed country often do not exist in a developing country. For example, in a recent MIS development effort in Egypt, there did not exist a correct, complete

list of settlements. Since settlement-level statistics were desired (e.g., percentage of settlements without piped water), it was necessary to construct a settlement list before the data collection effort could begin.

Because of the many uses to which an MIS may be applied, all of the collected data may not be included in a single file. Instead, there may be a variety of data files, involving a variety of units of analysis. For example, there may be national-level, regional-level, and local-level files; facility files, personnel files, program files and client files. Data from these files may be aggregated and combined as desired, as long as there is a "key" (identifier) linking the file records. An essential element in linking geographically-oriented data files is a the availability of a geographic code, or "geocode." With a suitably defined geocode (containing subparts referring to the political subdivisions), it is possible to perform a wide variety of geographic-oriented analyses. Many useful analyses can be readily performed with a standard data base management. Many more can be efficiently conducted if a microcomputer-based geographic information system (GIS) is available. The decision to require the use of geocodes is an example of a system design decision that has a profound effect on the kind of questions that can be answered (i.e., the type of ad-hoc reports that can be generated) after the system implementation effort is completed.

System Operation and Support

System support includes all of the resources required to operate, maintain, and improve the system. A crucial aspect of the system operation phase is the fact that the MIS design team will, at some point, be ending its involvement in the MIS. At that point, it is important that the client have the personnel know-how and resources to continue operation of the system, maintain it, and improve it, without further assistance from the contractor. Because of the known eventual departure of the system development contractor, it is necessary to develop a system that is easy to use and improve, and that will not collapse with the departure of a key government person. To accomplish this goal, training materials and procedures will be developed that can be used to assure the continued operation of the system in the future.

References

1. Whitten, Jeffrey L, Lonnie D. Bentley, and Victor M. Barlow, *Systems Analysis and Design Methods*, 2nd ed., Irwin, 1989

2. DeMarco, Tom., *Structured Analysis and System Specification*, Yourdon Press (Prentice-Hall), 1979

3. Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, McGraw Hill, 1982

4. Martin, James and Carma McClure, *Software Maintenance: The Problem and Its Solution*, Prentice-Hall, 1983

5. Martin, James and Joe Leben, *Fourth-Generation Languages*, Prentice-Hall, 1986

6. Martin, James and Carma McLure, *Structured Techniques for Computing*, Prentice-Hall, 1985

7. Martin, James and Carma McLure, *Diagramming Techniques for Analysts and Programmers*, Prentice-Hall, 1985

8. Boehm, Barry W., *Software Engineering Economics*, Prentice-Hall, 1981

9. Ullman, Jeffrey D., *Principles of Database Systems*, Computer Science Press, 1980

10. Davis, Alan M., *Software Requirements: Analysis and Specification*, Prentice-Hall, 1990

11. Kruglinski, David, *Data Base Management Systems: MS-DOS: Evaluating MS-DOS Data Base Software*, McGraw-Hill, 1986

12. Dumas, Joseph S., *Designing User Interfaces for Software*, Prentice-Hall, 1988

13. Kernighan, Brian W. and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976

14. Martin, James, *System Design from Provably Correct Constructs*, Prentice-Hall, 1986

This piece was written some time ago (1993), and the preceding references are dated (all prior to 1991). More importantly, they address only the "theoretical" aspects of database and MIS design – that is why they are still relevant. In practice, the developer will have to be very skilled in the particular database development system being used, such as Microsoft Access, Oracle, or the open-source MySQL.

The programming language used for most relational database system is SQL. A reference for SQL is *SQL in a Nutshell: A Desktop Quick Reference* by Kevin Kline with Daniel Kline (O'Reilly, 2001). SQL is both a data definition language (DDL), i.e., it can be used to create, alter and delete tables, and a data manipulation language (DML), i.e., it can be used to select records, insert records, delete records, and update records in tables.

Most of the following technical programming references are for Intel-bsed microcomputers. All of the database development references refer to SQL-based systems. A database development system that was widely used formerly was the "X-Based" language developed by Jet Propulsion Labs. Examples include Borland's dBASE and Paradox, and Microsoft's FoxPro. The X-Based languages are easy-to-use and powerful, but they are procedural, that is, they involve coding of detailed steps. SQL is a nonprocedural language – the programmer specifies the result that he wants (e.g. a query), but not the detailed steps on how to obtain it.

A fundamental decision concerning any application is whether it will be a relatively simple application that works on a single desktop computer (workstation) or a small network, or is a major "enterprise-level" application. If the latter, more effort will go into data modeling, and a development system may be used for that. For simple systems, the data modeling may be done using Visio or Microsoft Access (which can construct ant print out entity-relationship diagrams). For large applications, more time will be invested in data modeling, and a data-modeling language such as UML may be used. References for UML include: *UML Distilled Third Edition: A Brief Guide to the Standard Object Modeling Language* by Martin Fowler (Addison-Wesley, 2004); *Use Case Driven Object Modeling with UML: A Practical Approach* by Doug Rosenberg with Kendall Scott (Addison-Wesley, 2000); *UML for the IT Business Analyst: A Practical Guide to Object-Oriented Requirements Gathering* by Howard Podeswa (Thomson Course Technology, 2005); *The Elements of UML 2.0 Style* by Scott W. Ambler (Cambridge University Press, 2005); and *Data Modeling with ERwin* by M. Carla DeAngelis (Sams, 2000).

There are a large number of "after-market" books on database systems such as Microsoft Access or SQL Server. Anyone undertaking database development work should purchase three or four of them for whatever system he is using. A whole new set of them is issued whenever a new version is issued. For Microsoft Access 2003, examples (from my personal library) are: *Microsoft Office Access 2003* by Virginia Anderson (McGraw-Hill Osborne, 2003; *Access Database Design and Programming*, 3rd edition by Steven Roman (O'Reilly, 2002, 1999, 1997); *Access 2003 VBA Programmer's Reference* by Patricia Cardoza, Teresa Hennig, Graham Seach, and Armen Stein (Wiley/Wrox, 2004); and *Microsoft Office Access 2003 Inside Out* by John L. Viescas (Microsoft Press, 2004). Texts written for earlier editions are also useful, such as *Access 2000 Developer's Handbook, Volume 1 Desktop Edition and Volume 2 Enterprise Edition* (Sybex, 1999); *Sams Teach Yourself Microsoft Access 2000 in 21 Days* by Paul Cassel and Pamela Parker (Sams, 1999); and *Running Microsoft Access 2000* by John Viescas (Microsoft Press, 1999). The reason why multiple references are required for the developer is that the "on-line" help manuals provided with the systems are not very helpful, and no single text contains all of the information that is necessary to make the systems work as advertised.

If you are using Microsoft SQL Server, examples of after-market books include *Microsoft SQL Server 2005: Database Essentials Step by Step* (Microsoft Press, 2007); *SQL Server 2005 Express Edition Starter Kit* by Rajesh George and Lance Delano (Wiley/Wrox, 2006); *Microsoft SQL Server 2005 Unleashed* by Ray Rankins, Paul Bertucci, Chris Gallelli, Alex T. Silverstein et al. (Sams, 2007); *Beginning SQL Server 2005 Programming* by Robert Vieira (Wiley/Wrox 2006).

While queries, forms and reports are found in virtually all database applications, a major component of any larger database is program modules, usually coded in Visual Basic. Examples of reference texts in Visual Basic include: *Start-to-Finish Visual Basic 2005* by Tim Patrick (Addison-Wesley, 2007); *Mastering Microsoft Visual Basic 2005* by Evangelos Petroutsos (Sybex, 2006); *Visual Basic 2005 Programmer's Reference* by Rod Stephens (Wiley, 2005); *Visual Basic 2005 for Programmers* 2nd edition by Paul J. Deitel and Harvey M. Deitel (Prentice Hall, 2006); *Microsoft Visual Basic 2005 Express Edition Programming for the Absolute Beginner* by Jerry Lee Ford, Jr. (Thomson Course Technology, 2006); *Beginning Visual Basic 2005 Databases* by Thearon Willis (Wiley/Wrox, 2006); *Visual Basic 2005: The Compete Reference* by Ron Petrusha (McGraw-Hill/Osborne, 2006); and *Expert One-on-One Visual Basic 2005 Database Programming* by Roger Jennings (Wiley/Wrox, 2006).

If your organization is heavy into software development, it will probably use an integrated development environment (IDE) such as Microsoft Visual Studio, and you will likely make use of web-based technology. Reference texts on this subject (for Microsoft developers) include *Professional Visual Studio 2005* by Andrew Parsons and Nick Randolph (Wiley/Wrox, 2006); *Sams Teach Yourself ASP.NET 2.0 in 24 Hours* by Scott Mitchell (Sams, 2006); *Murach's ADO.NET 2.0 Database Programming with VB 2005* by Anne Boehm (Mike Murach & Associates, 20007); *Sams Teach Yourself Visual Studio .NET 2003 in 21 Days* by Jason Beres (Sams, 2003); *Sams Teach Yourself ASP.NET in 21 Days* 2nd edition by Chris Payne (Sams, 2003); *Microsoft Office FrontPage 2003 Inside Out* by Jim Buyens (Microsoft Press, 2004). A popular programming language for web-based applications is Java, developed by Sun Microsystems (it is an alternative to Visual Basic and the more powerful C++ programming languages). References on Java include *Sams Teach Yourself Programming with Java in 24 Hours* 4th edition by Rogers Cadenhead (Sams, 2006); *Sams Teach Yourself Java 2 in 21 Days* 4th edition by Rogers Cadenhead and Laura Lemay (Sams, 2004); *Java 2 Unleashed* by Stephen Potts, Alex Pestrikov and Mike Kopack (Sams, 2002); *J2EE 1.4 The Big Picture* by Solveig Haugland, Mark Cade and Anthony Orapallo

(Prentice Hall, 2004); *Sams Teach Yourself J2EE in 21 Days* by Martin Bond, Debbie Law, Andy Longshaw, Dan Haywood, and Peter Roxburgh (Sams, 2004).

If you are planning to use open-source (free) software (which I don't recommend) such as MySQL, you will need a whole library on development tools. First, if you are involved with open-source systems, you will probably not be using the Microsoft Windows operating system – you will be using an open-source version of Unix. Reference on current popular open-source operating sytems include: *Moving to Linux Kiss the Blue Screen of Death Goodbye!* 2nd edition by Marcel Gagné (Addison-Wesley, 2006); *The Official Ubuntu Book* by Benjamin Mako Hill and Jono Bacon, Corey Burger, Jonathan Jesse, and Ivan Krstić (Prentice Hall, 2007); *SUSE Linux 10 Unleashed* by Michael McCalister (Sams, 2006); *Red Hat Fedora 4 Unleashed* by Andrew Hudson, Paul Hudson, Bill Ball and Hoyt Duff (Sams, 2005); *Red Hat Fedora and Enterprise Linux 4 Bible* by Christopher Negus (Wiley 2005); *A Practical Guide to Red Hat Linux* 2nd edition by Mark G. Sobell (Prentice Hall, 2005); *Knoppix Pocket Reference* by Kyle Ranking (O'Reilly, 2005).

If you are using open-source software, you will probably be using the complete "LAMP" suite, which consists of Linux operating system, Apache web server, and PHP and MySQL for web-based databases. The open-source Java integrated development system is Eclipse; a reference is *The Java Developer's Guide to Eclipse* 2nd edition by Jim D'Anjou, Scott Fairbrother, Dan Kehn, John Kellerman and Pat McCarthy (Addison-Wesley, 2005). Books on the LAMP suite include the following: *Beginning MySQL* by Robert Sheldon and Geoff Moes (Wiley/Wrox, 2005); *MySQL: The Definitive Guide to Using, Programming and Administering MySQL 4.1 and 5.0* 3rd edition by Paul DuBois (Sams, 2005); *PHP & MySQL* by Vikram Vaswani (McGraw-Hill/Osborne, 2005); *Web Database Applications with PHP and MySQL* by Hugh E. Williams and David Lane (O'Reilly, 2004); *PHP and MySQL for Dynamic Web Sites* 2nd edition by Larry Ullman (Peachpit Press, 2005); *PHP and MySQL Web Development* 3rd edition by Luke Welling and Laura Thomson (Sams, 2005); *Sams Teach Yourself PHP, MySQL and Apache All in One* by Julie C. Meloni (Sams, 2005); *MySQL and JSP Web Applications* by James Turner (Sams, 2002); *Tomcat: The Definitive Guide* by Jason Brittain and Ian F. Darwin (O'Reilly, 2003); *Ant: The Definitive Guide* 2nd edition by Steve Holzner (O'Reilly, 2002, 2005); *Apache: The Definitive Guide* 3rd edition by Ben Laurie and Peter Laurie (O'Reilly, 2003, 1999,1997); *Professional Apache Tomcat 5* by Vivek Chopra, Amit Bakore, Jon Eaves, Ben Galbraith, Sing Li, and Chanoch Wiggers (Wiley/Wrox, 2004); *Struts in Action: Building Web Applications with the Leading Java Framework* by Ted Husted, Cedric Dumoulin, George Franciscus and David Winterfeldt (Manning Publications, 2003); *Sams Teach Yourself HTML and CSS in 24 Hours* 7th edition by Dick Oliver and Michael Morrison (Sams, 2006); *No Nonsense XML Web Development with PHP* by Thomas Myer (SitePoint, 2005); *Sams Teach Yourself Networking in 24 Hours* 3rd edition by Joe Habraken and Matt Hayden (Sams, 2004); *Beginning Perl* 2nd edition by James Lee with Simon Cozens and Peter Wainwright (Apress, 2004); *CGI Manual of Style* by Robert McDaniel (Ziff-Davis, 1996).

On the surface, it may seem that it is a great idea to use an open-source suite, such as LAMP, instead of a proprietary system such as Microsoft operating system and Access or SQL Server. Most of my experience in database development has been using Microsoft Access. The only time I used Informix was to convert a very unhappy Informix user to Access. The problem with MySQL is that the user has to use the SQL programming language. Working with MySQL after working with Access is like returning to the Dark Ages. Access has very easy-to-use graphics-based tools for constructing and modifying tables, queries and reports, whereas in MySQL it is necessary to use SQL directly to do these things. While this may be fine for the skilled programmer who is setting things up for permanent use (e.g., a web-based application that allows a few simple predetermined queries), it is frightening to the non-programmer who wishes to construct an ad-hoc query and must do it using SQL. Another drawback of open systems, in a developing-country

setting, is that more people are trained in Microsoft products, and they do not see that becoming involved in open-source work enhances their careers.