

Evolution of Decision Support Systems

We are told that the hieroglyphics in Egypt are primarily the work of an accountant declaring how much grain is owed the Pharaoh. Some of the streets in Rome were laid out by civil engineers more than 2,000 years ago. Examination of bones found in archeological excavations in Chile shows that medicine — in, at least, a rudimentary form — was practiced as far back as 10,000 years ago. Other professions have roots that can be traced to antiquity. From this perspective, the profession and practice of information systems and processing are certainly immature, because they have existed only since the early 1960s.

Information processing shows this immaturity in many ways, such as its tendency to dwell on detail. There is the notion that if we get the details right, the end result will somehow take care of itself, and we will achieve success. It's like saying that if we know how to lay concrete, how to drill, and how to install nuts and bolts, we don't have to worry about the shape or the use of the bridge we are building. Such an attitude would drive a professionally mature civil engineer crazy. Getting all the details right does not necessarily equate success.

The data warehouse requires an architecture that begins by looking at the whole and then works down to the particulars. Certainly, details are important throughout the data warehouse. But details are important only when viewed in a broader context.

The story of the data warehouse begins with the evolution of information and decision support systems. This broad view of how it was that data warehousing evolved enables valuable insight.

The Evolution

The origins of data warehousing and *decision support systems* (DSS) processing hark back to the very early days of computers and information systems. It is interesting that DSS processing developed out of a long and complex evolution of information technology. Its evolution continues today.

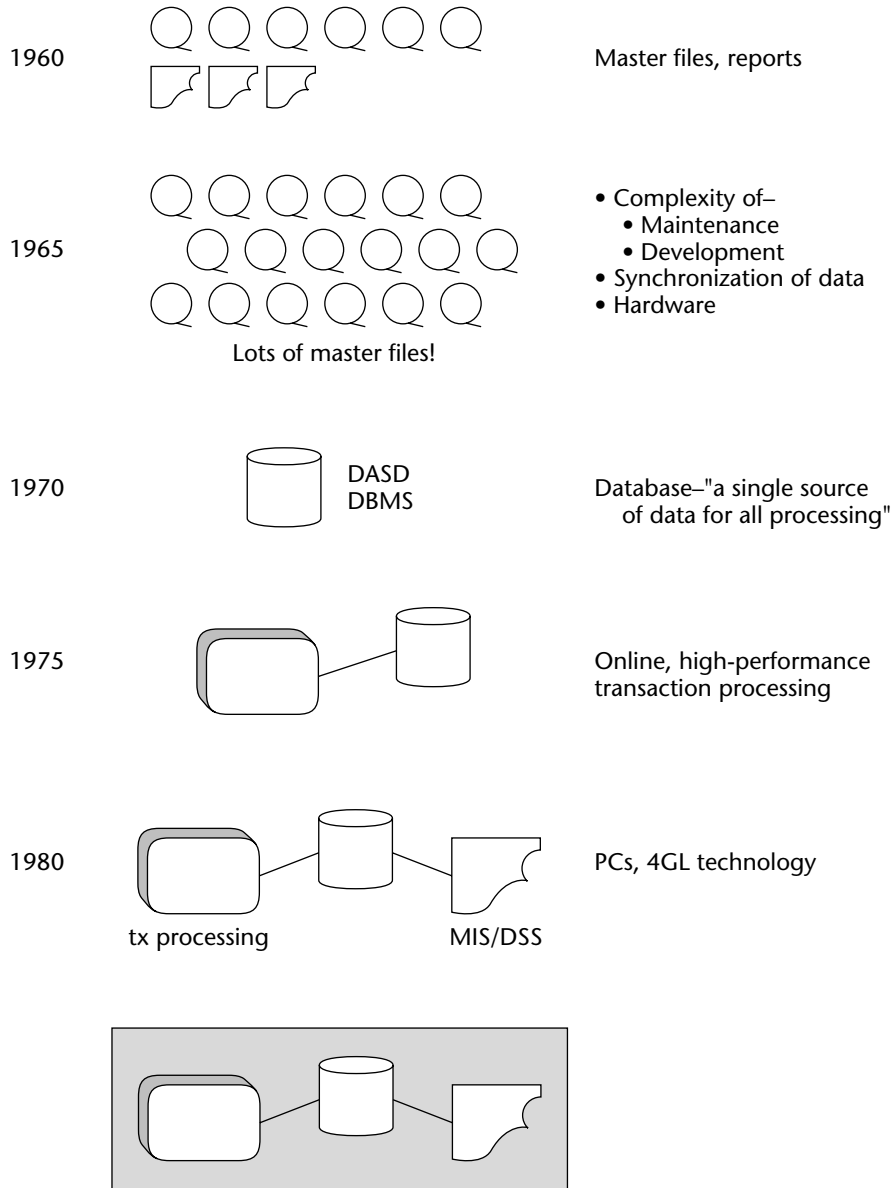
Figure 1-1 shows the evolution of information processing from the early 1960s through 1980. In the early 1960s, the world of computation consisted of creating individual applications that were run using master files. The applications featured reports and programs, usually built in an early language such as Fortran or COBOL. Punched cards and paper tape were common. The master files of the day were housed on magnetic tape. The magnetic tapes were good for storing a large volume of data cheaply, but the drawback was that they had to be accessed sequentially. In a given pass of a magnetic tape file, where 100 percent of the records have to be accessed, typically only 5 percent or fewer of the records are actually needed. In addition, accessing an entire tape file may take as long as 20 to 30 minutes, depending on the data on the file and the processing that is done.

Around the mid-1960s, the growth of master files and magnetic tape exploded. And with that growth came huge amounts of redundant data. The proliferation of master files and redundant data presented some very insidious problems:

- The need to synchronize data upon update
- The complexity of maintaining programs
- The complexity of developing new programs
- The need for extensive amounts of hardware to support all the master files

In short order, the problems of master files — problems inherent to the medium itself — became stifling.

It is interesting to speculate what the world of information processing would look like if the only medium for storing data had been the magnetic tape. If there had never been anything to store bulk data on other than magnetic tape files, the world would have never had large, fast reservations systems, ATM systems, and the like. Indeed, the ability to store and manage data on new kinds of media opened up the way for a more powerful type of processing that brought the technician and the businessperson together as never before.



The single-database-serving-all-purposes paradigm

Figure 1-1 The early evolutionary stages of the architected environment.

The Advent of DASD

By 1970, the day of a new technology for the storage and access of data had dawned. The 1970s saw the advent of disk storage, or the *direct access storage device (DASD)*. Disk storage was fundamentally different from magnetic tape storage in that data could be accessed directly on a DASD. There was no need to go through records 1, 2, 3, . . . n to get to record $n + 1$. Once the address of record $n + 1$ was known, it was a simple matter to go to record $n + 1$ directly. Furthermore, the time required to go to record $n + 1$ was significantly less than the time required to scan a tape. In fact, the time to locate a record on a DASD could be measured in milliseconds.

With the DASD came a new type of system software known as a *database management system (DBMS)*. The purpose of the DBMS was to make it easy for the programmer to store and access data on a DASD. In addition, the DBMS took care of such tasks as storing data on a DASD, indexing data, and so forth. With the DASD and DBMS came a technological solution to the problems of master files. And with the DBMS came the notion of a “database.” In looking at the mess that was created by master files and the masses of redundant data aggregated on them, it is no wonder that in the 1970s, a database was defined as a single source of data for all processing.

By the mid-1970s, *online transaction processing (OLTP)* made even faster access to data possible, opening whole new vistas for business and processing. The computer could now be used for tasks not previously possible, including driving reservations systems, bank teller systems, manufacturing control systems, and the like. Had the world remained in a magnetic-tape-file state, most of the systems that we take for granted today would not have been possible.

PC/4GL Technology

By the 1980s, more new technologies, such as PCs and *fourth-generation languages (4GLs)*, began to surface. The end user began to assume a role previously unfathomed — directly controlling data and systems — a role previously reserved for the professional data processor. With PCs and 4GL technology came the notion that more could be done with data than simply processing online transactions. A *Management Information System (MIS)*, as it was called in the early days, could also be implemented. Today known as DSS, MIS was processing used to drive management decisions. Previously, data and technology were used exclusively to drive detailed operational decisions. No single database could serve both operational transaction processing and analytical processing at the same time. The single-database paradigm was previously shown in Figure 1-1.

Enter the Extract Program

Shortly after the advent of massive OLTP systems, an innocuous program for “extract” processing began to appear (see Figure 1-2).

The *extract program* is the simplest of all programs. It rummages through a file or database, uses some criteria for selecting data, and, on finding qualified data, transports the data to another file or database.

The extract program became very popular for at least two reasons:

- Because extract processing can move data out of the way of high-performance online processing, there is no conflict in terms of performance when the data needs to be analyzed en masse.
- When data is moved out of the operational, transaction-processing domain with an extract program, a shift in control of the data occurs. The end user then owns the data once he or she takes control of it. For these (and probably a host of other) reasons, extract processing was soon found everywhere.

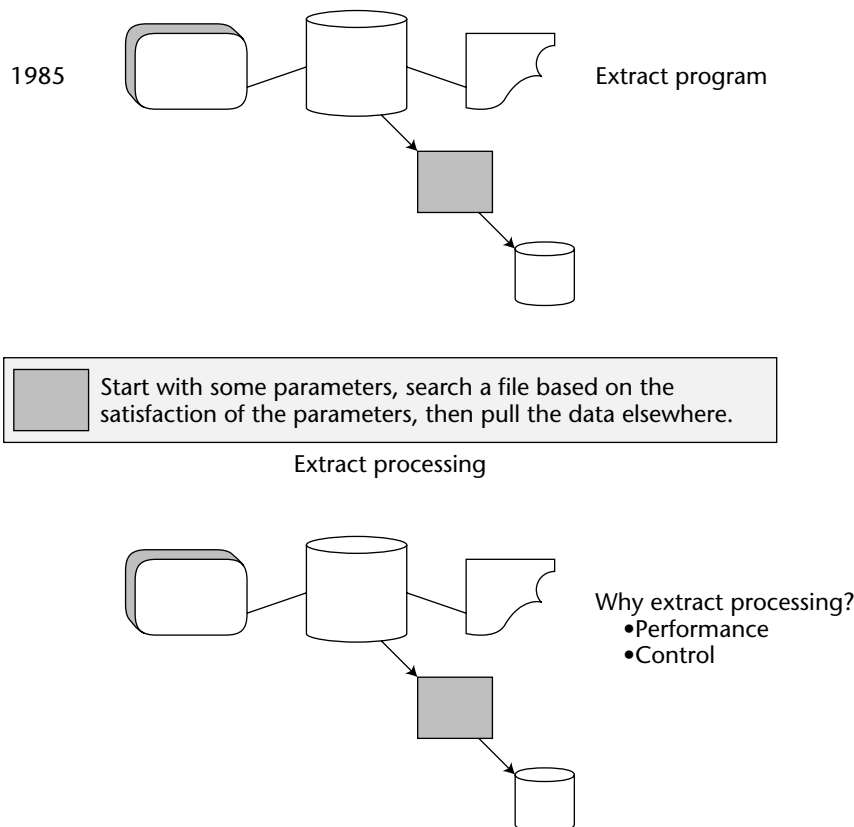


Figure 1-2 The nature of extract processing.

The Spider Web

As illustrated in Figure 1-3, a “spider web” of extract processing began to form. First, there were extracts; then there were extracts of extracts; then extracts of extracts of extracts; and so forth. It was not unusual for a large company to perform as many as 45,000 extracts per day.

This pattern of out-of-control extract processing across the organization became so commonplace that it was given its own name — the “naturally evolving architecture” — which occurs when an organization handles the whole process of hardware and software architecture with a *laissez-faire* attitude. The larger and more mature the organization, the worse the problems of the naturally evolving architecture become.

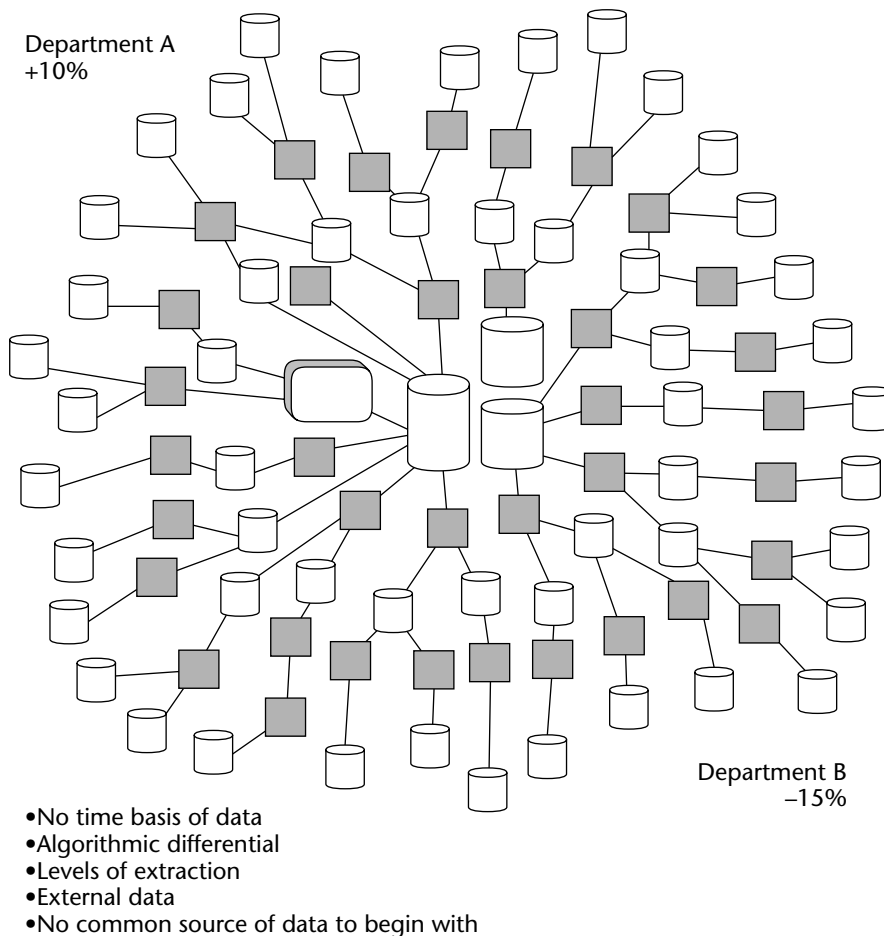


Figure 1-3 Lack of data credibility in the naturally evolving architecture.

Problems with the Naturally Evolving Architecture

The naturally evolving architecture presents many challenges, such as:

- Data credibility
- Productivity
- Inability to transform data into information

Lack of Data Credibility

The lack of data credibility was illustrated in Figure 1-3. Say two departments are delivering a report to management — one department claims that activity is down 15 percent, the other says that activity is up 10 percent. Not only are the two departments not in sync with each other, they are off by very large margins. In addition, trying to reconcile the different information from the different departments is difficult. Unless very careful documentation has been done, reconciliation is, for all practical purposes, impossible.

When management receives the conflicting reports, it is forced to make decisions based on politics and personalities because neither source is more or less credible. This is an example of the crisis of data credibility in the naturally evolving architecture.

This crisis is widespread and predictable. Why? As it was depicted in Figure 1-3, there are five reasons:

- No time basis of data
- The algorithmic differential of data
- The levels of extraction
- The problem of external data
- No common source of data from the beginning

The first reason for the predictability of the crisis is that there is no time basis for the data. Figure 1-4 shows such a time discrepancy. One department has extracted its data for analysis on a Sunday evening, and the other department extracted on a Wednesday afternoon. Is there any reason to believe that analysis done on one sample of data taken on one day will be the same as the analysis for a sample of data taken on another day? Of course not. Data is always changing within the corporation. Any correlation between analyzed sets of data that are taken at different points in time is only coincidental.

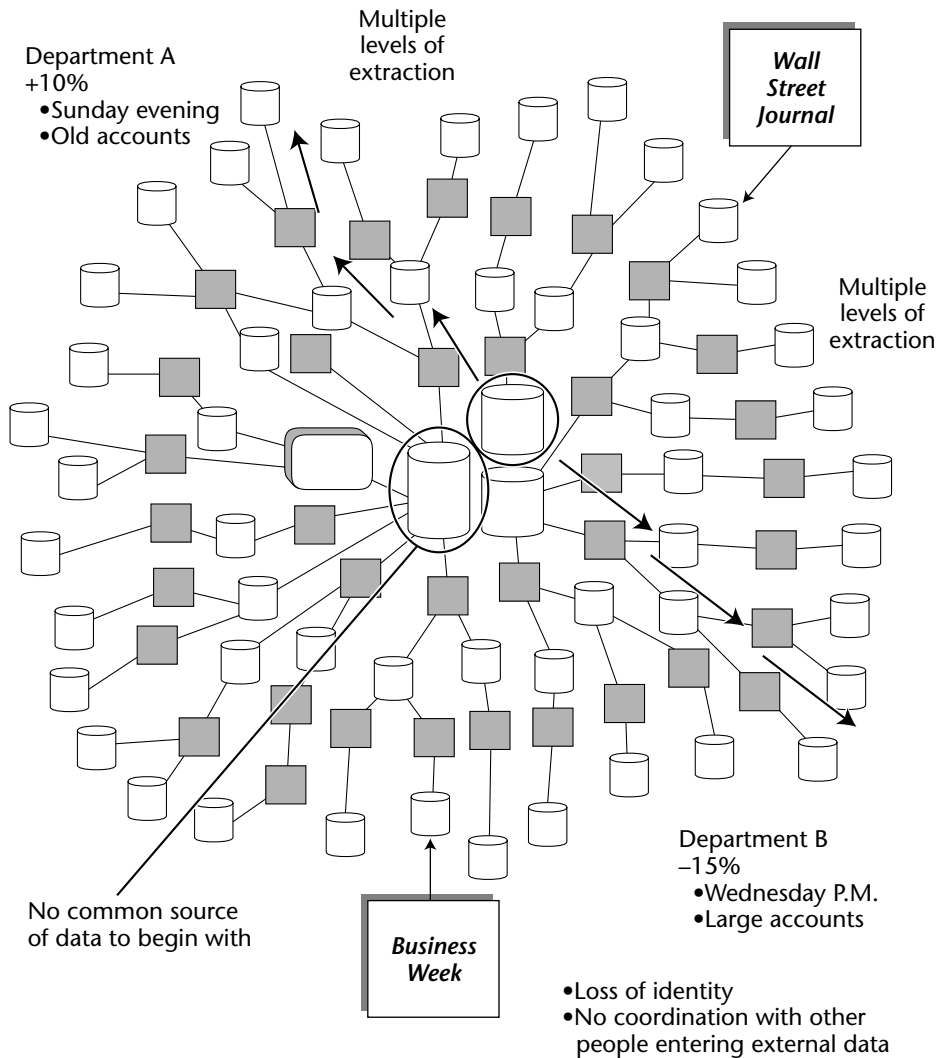


Figure 1-4 The reasons for the predictability of the crisis in data credibility in the naturally evolving architecture.

The second reason is the algorithmic differential. For example, one department has chosen to analyze all old accounts. Another department has chosen to analyze all large accounts. Is there any necessary correlation between the characteristics of customers who have old accounts and customers who have large accounts? Probably not. So why should a very different result surprise anyone?

The third reason is one that merely magnifies the first two reasons. Every time a new extraction is done, the probabilities of a discrepancy arise because of

the timing or the algorithmic differential. And it is not unusual for a corporation to have eight or nine levels of extraction being done from the time the data enters the corporation's system to the time analysis is prepared for management. There are extracts, extracts of extracts, extracts of extracts of extracts, and so on. Each new level of extraction exaggerates the other problems that occur.

The fourth reason for the lack of credibility is the problem posed by external data. With today's technologies at the PC level, it is very easy to bring in data from outside sources. For example, Figure 1-4 showed one analyst bringing data into the mainstream of analysis from the *Wall Street Journal*, and another analyst bringing data in from *Business Week*. However, when the analyst brings data in, he or she strips the external data of its identity. Because the origin of the data is not captured, it becomes generic data that could have come from any source.

Furthermore, the analyst who brings in data from the *Wall Street Journal* knows nothing about the data being entered from *Business Week*, and vice versa. No wonder, then, that external data contributes to the lack of credibility of data in the naturally evolving architecture.

The last contributing factor to the lack of credibility is that often there is no common source of data to begin with. Analysis for department A originates from file XYZ. Analysis for department B originates from database ABC. There is no synchronization or sharing of data whatsoever between file XYZ and database ABC.

Given these reasons, it is no small wonder that there is a crisis of credibility brewing in every organization that allows its legacy of hardware, software, and data to evolve naturally into the spider web.

Problems with Productivity

Data credibility is not the only major problem with the naturally evolving architecture. Productivity is also abysmal, especially when there is a need to analyze data across the organization.

Consider an organization that has been in business for a while and has built up a large collection of data, as shown in the top of Figure 1-5.

Management wants to produce a corporate report, using the many files and collections of data that have accumulated over the years. The designer assigned the task decides that three things must be done to produce the corporate report:

- Locate and analyze the data for the report.
- Compile the data for the report.
- Get programmer/analyst resources to accomplish these two tasks.

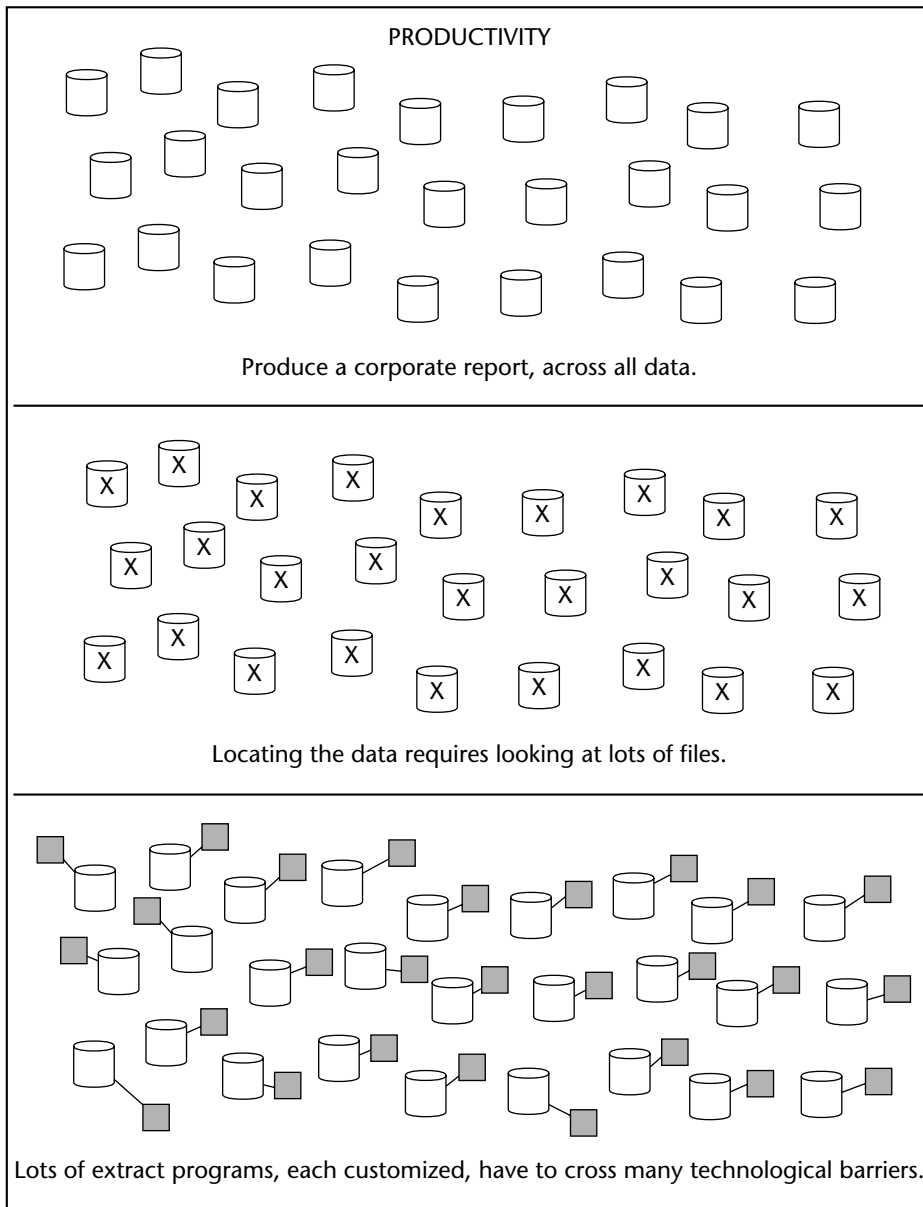


Figure 1-5 The naturally evolving architecture is not conducive to productivity.

In order to locate the data, many files and layouts of data must be analyzed. Some files use the *Virtual Storage Access Method (VSAM)*, some use the *Information Management System (IMS)*, some use *Adabas*, some use the *Integrated Database Management System (IDMS)*. Different skill sets are required in order to

access data across the enterprise. Furthermore, there are complicating factors. For example, two files might have an element called `BALANCE`, but the two elements are very different. In another case, one database might have a file known as `CURRBAL`, and another collection of data might have a file called `INVLEVEL` that happens to represent the same information as `CURRBAL`. Having to go through every piece of data — not just by name but by definition and calculation — is a very tedious process. But if the corporate report is to be produced, this exercise must be done properly. Unless data is analyzed and “rationalized,” the report will end up mixing apples and oranges, creating yet another level of confusion.

The next task for producing the report is to compile the data once it is located. The program that must be written to get data from its many sources should be simple. It is complicated, though, by the following facts:

- Lots of programs have to be written.
- Each program must be customized.
- The programs cross every technology that the company uses.

In short, even though the report-generation program should be simple to write, retrieving the data for the report is tedious.

In a corporation facing exactly the problems described, an analyst recently estimated a very long time to accomplish the tasks, as shown in Figure 1-6.

If the designer had asked for only two or three man-months of resources, then generating the report might not have required much management attention. But when an analyst requisitions many resources, management must consider the request with all the other requests for resources and must prioritize the requests.

Creating the reports using a large amount of resources wouldn’t be bad if there were a one-time penalty to be paid. In other words, if the first corporate report generated required a large amount of resources, and if all succeeding reports could build on the first report, then it might be worthwhile to pay the price for generating the first report. But that is not the case.

Unless future corporate reporting requirements are known in advance and are factored into building the first corporate report, each new corporate report will probably require the same large overhead. In other words, it is unlikely that the first corporate report will be adequate for future corporate reporting requirements.

Productivity, then, in the corporate environment is a major issue in the face of the naturally evolving architecture and its legacy systems. Simply stated, when using the spider web of legacy systems, information is expensive to access and takes a long time to create.

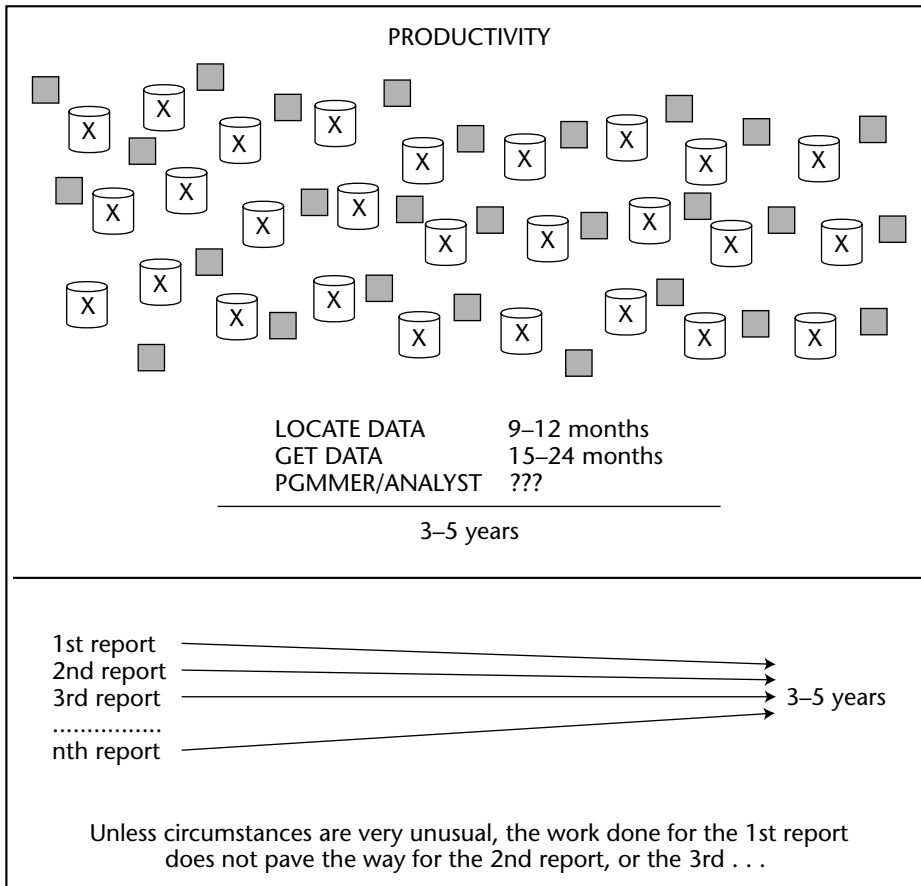


Figure 1-6 When the first report is being written, the requirements for future reports are not known.

From Data to Information

As if productivity and credibility were not problems enough, there is another major fault of the naturally evolving architecture — the inability to go from data to information. At first glance, the notion of going from data to information seems to be an ethereal concept with little substance. But that is not the case at all.

Consider the following request for information, typical in a banking environment: “How has account activity differed this year from each of the past five years?”

Figure 1-7 shows the request for information.

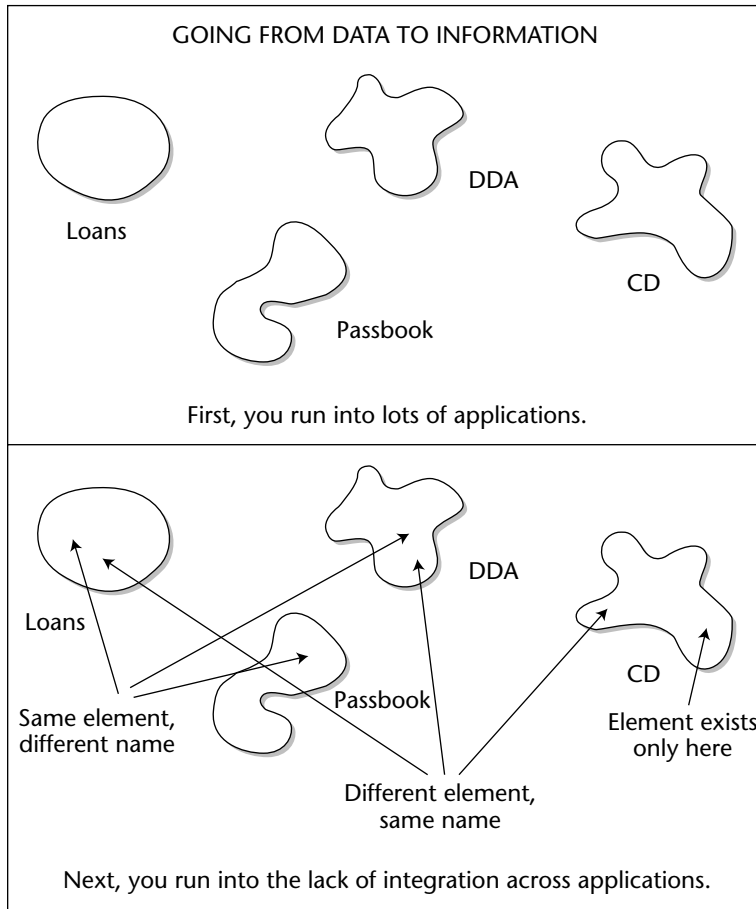


Figure 1-7 “How has account activity been different this year from each of the past five years for the financial institution?”

The first thing the DSS analyst discovers in trying to satisfy the request for information is that going to existing systems for the necessary data is the worst thing to do. The DSS analyst will have to deal with lots of unintegrated non-integrated legacy applications. For example, a bank may have separate savings, loan, direct-deposit, and trust applications. However, trying to draw information from them on a regular basis is nearly impossible because the applications were never constructed with integration in mind, and they are no easier for the DSS analyst to decipher than they are for anyone else.

But integration is not the only difficulty the analyst meets in trying to satisfy an informational request. A second major obstacle is that there is not enough historical data stored in the applications to meet the needs of the DSS request.

Figure 1-8 shows that the loan department has up to two years' worth of data, passbook processing has up to one year of data, DDA applications have up to 30 days of data, and CD processing has up to 18 months of data. The applications were built to service the needs of current balance processing. They were never designed to hold the historical data needed for DSS analysis. It is no wonder, then, that going to existing systems for DSS analysis is a poor choice. But where else is there to go?

The systems found in the naturally evolving architecture are simply inadequate for supporting information needs. They lack integration and there is a discrepancy between the time horizon (or parameter of time) needed for analytical processing and the available time horizon that exists in the applications.

A Change in Approach

The status quo of the naturally evolving architecture, where most shops began, simply is not robust enough to meet the future needs. What is needed is something much larger — a change in architectures. That is where the architected data warehouse comes in.

There are fundamentally two kinds of data at the heart of an “architected” environment — *primitive data* and *derived data*. Figure 1-9 shows some of the major differences between primitive and derived data.

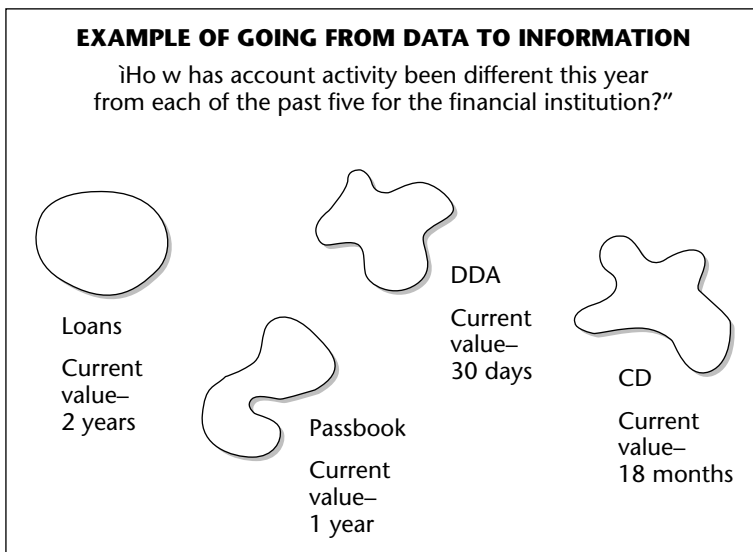


Figure 1-8 Existing applications simply do not have the historical data required to convert data into information.

A CHANGE IN APPROACHES	
PRIMITIVE DATA/OPERATIONAL DATA	DERIVED DATA/DSS DATA
• Application-oriented • Detailed • Accurate, as of the moment of access • Serves the clerical community • Can be updated • Run repetitively • Requirements for processing understood <i>a priori</i> • Compatible with the SDLC • Performance-sensitive • Accessed a unit at a time • Transaction-driven • Control of update a major concern in terms of ownership • High availability • Managed in its entirety • Nonredundancy • Static structure; variable contents • Small amount of data used in a process • Supports day-to-day operations • High probability of access	• Subject-oriented • Summarized, otherwise refined • Represents values over time, snapshots • Serves the managerial community • Is not updated • Run heuristically • Requirements for processing not understood <i>a priori</i> • Completely different life cycle • Performance relaxed • Accessed a set at a time • Analysis-driven • Control of update no issue • Relaxed availability • Managed by subsets • Redundancy is a fact of life • Flexible structure • Large amount of data used in a process • Supports managerial needs • Low, modest probability of access

Figure 1-9 Differences between primitive and derived data.

Following are some other differences between the two.

- Primitive data is detailed data used to run the day-to-day operations of the company. Derived data has been summarized or otherwise calculated to meet the needs of the management of the company.
- Primitive data can be updated. Derived data can be recalculated but cannot be directly updated.
- Primitive data is primarily current-value data. Derived data is often historical data.
- Primitive data is operated on by repetitive procedures. Derived data is operated on by heuristic, nonrepetitive programs and procedures.
- Operational data is primitive; DSS data is derived.
- Primitive data supports the clerical function. Derived data supports the managerial function.

It is a wonder that the information processing community ever thought that both primitive and derived data would fit and peacefully coexist in a single database. In fact, primitive data and derived data are so different that they do not reside in the same database or even the same environment.

The Architected Environment

The natural extension of the split in data caused by the difference between primitive and derived data is shown in Figure 1-10.

There are four levels of data in the architected environment — the *operational level*, the *atomic* (or the *data warehouse*) *level*, the *departmental* (or the *data mart*) *level*, and the *individual level*. These different levels of data are the basis of a larger architecture called the *corporate information factory (CIF)*. The operational level of data holds application-oriented primitive data only and primarily serves the high-performance transaction-processing community. The data-warehouse level of data holds integrated, historical primitive data that cannot be updated. In addition, some derived data is found there. The departmental or data mart level of data contains derived data almost exclusively. The departmental or data mart level of data is shaped by end-user requirements into a form specifically suited to the needs of the department. And the individual level of data is where much heuristic analysis is done.

The different levels of data form a higher set of architectural entities. These entities constitute the corporate information factory, and they are described in more detail in my book, *The Corporate Information Factory, Second Edition* (Hoboken, N.J.: Wiley, 2002).

Some people believe the architected environment generates too much redundant data. Though it is not obvious at first glance, this is not the case at all. Instead, it is the spider web environment that generates the gross amounts of data redundancy.

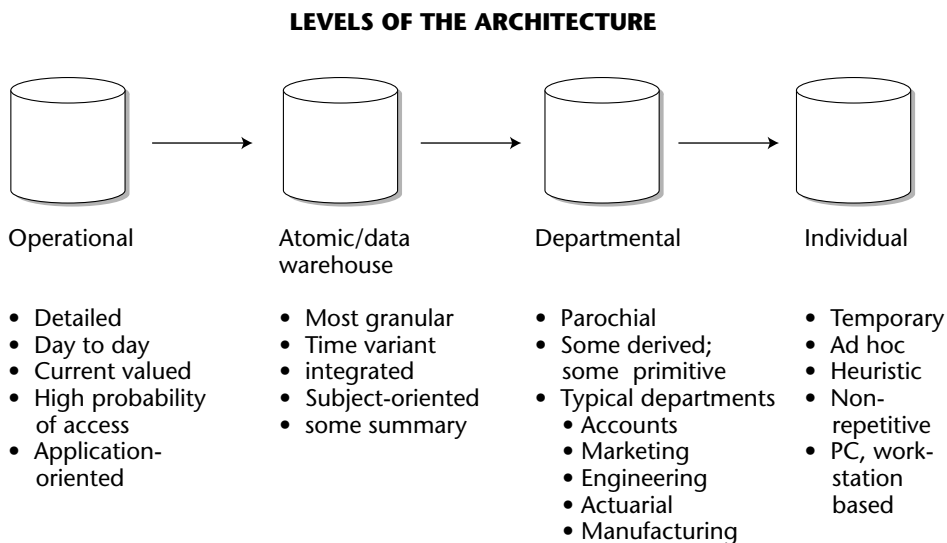


Figure 1-10 Although it is not apparent at first glance, there is very little redundancy of data across the architected environment.

Consider the simple example of data throughout the architecture, shown in Figure 1-11. At the operational level there is a record for a customer, J Jones. The operational-level record contains current-value data that can be updated at a moment's notice and shows the customer's current status. Of course, if the information for J Jones changes, the operational-level record will be changed to reflect the correct data.

The data warehouse environment contains several records for J Jones, which show the history of information about J Jones. For example, the data warehouse would be searched to discover where J Jones lived last year. There is no overlap between the records in the operational environment, where current information is found, and the data warehouse environment, where historical information is found. If there is a change of address for J Jones, then a new record will be created in the data warehouse, reflecting the from and to dates that J Jones lived at the previous address. Note that the records in the data warehouse do not overlap. Also note that there is some element of time associated with each record in the data warehouse.

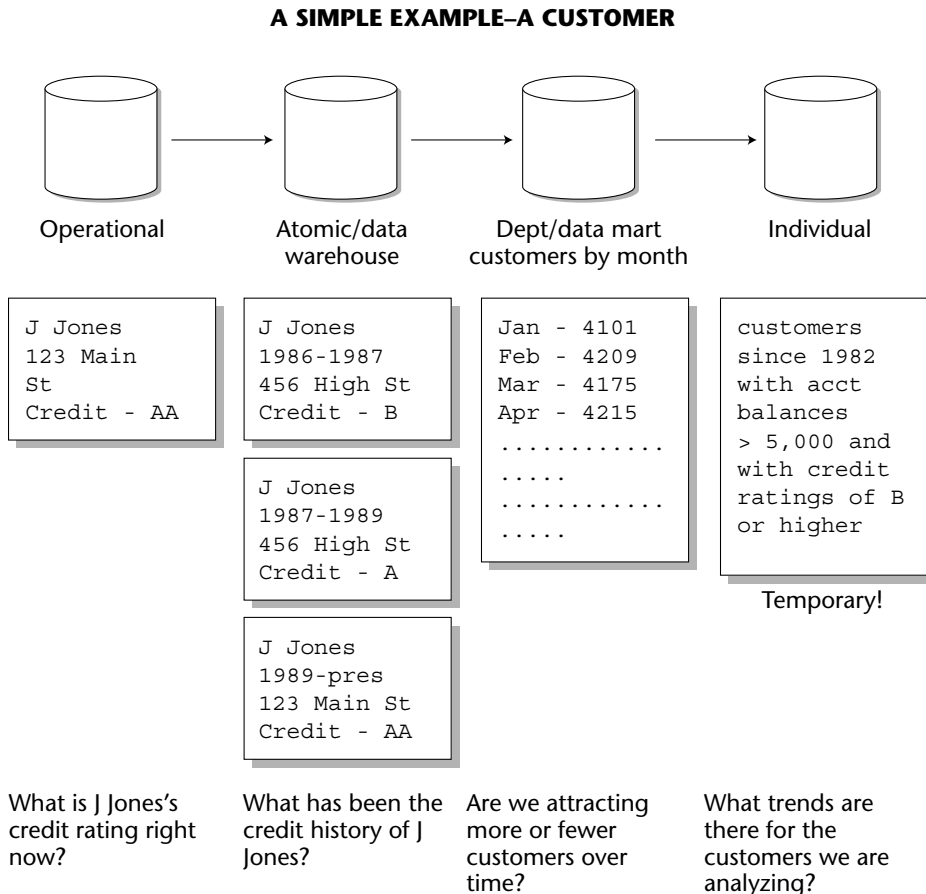


Figure 1-11 The kinds of queries for which the different levels of data can be used.

The departmental environment — sometimes called the *data mart level*, the *OLAP level*, or the *multidimensional DBMS level* — contains information useful to the different parochial departments of a company. There is a marketing departmental database, an accounting departmental database, an actuarial departmental database, and so forth. The data warehouse is the source of all departmental data. While data in the data mart certainly relates to data found in the operational level or the data warehouse, the data found in the departmental or data mart environment is fundamentally different from the data found in the data warehouse environment, because data mart data is denormalized, summarized, and shaped by the operating requirements of a single department.

Typical of data at the departmental or data mart level is a monthly customer file. In the file is a list of all customers by category. J Jones is tallied into this summary each month, along with many other customers. It is a stretch to consider the tallying of information to be redundant.

The final level of data is the individual level. Individual data is usually temporary and small. Much heuristic analysis is done at the individual level. As a rule, the individual levels of data are supported by the PC. Executive information systems (EIS) processing typically runs on the individual levels.

Data Integration in the Architected Environment

One important aspect of the architected environment that was not shown in Figure 1-11 is the integration of data that occurs across the architecture. As data passes from the operational environment to the data warehouse environment, it is integrated, as shown in Figure 1-12.

There is no point in bringing data over from the operational environment into the data warehouse environment without integrating it. If the data arrives at the data warehouse in an unintegrated state, it cannot be used to support a corporate view of data. And a corporate view of data is one of the essences of the architected environment.

In every environment, the unintegrated operational data is complex and difficult to deal with. This is simply a fact of life. And the task of getting your hands dirty with the process of integration is never pleasant. To achieve the real benefits of a data warehouse, though, it is necessary to undergo this painful, complex, and time-consuming exercise. *Extract/transform/load* (ETL) software can automate much of this tedious process. In addition, this process of integration has to be done only once. But, in any case, it is mandatory that data flowing into the data warehouse be integrated, not merely tossed — whole cloth — into the data warehouse from the operational environment.

A SIMPLE EXAMPLE-A CUSTOMER

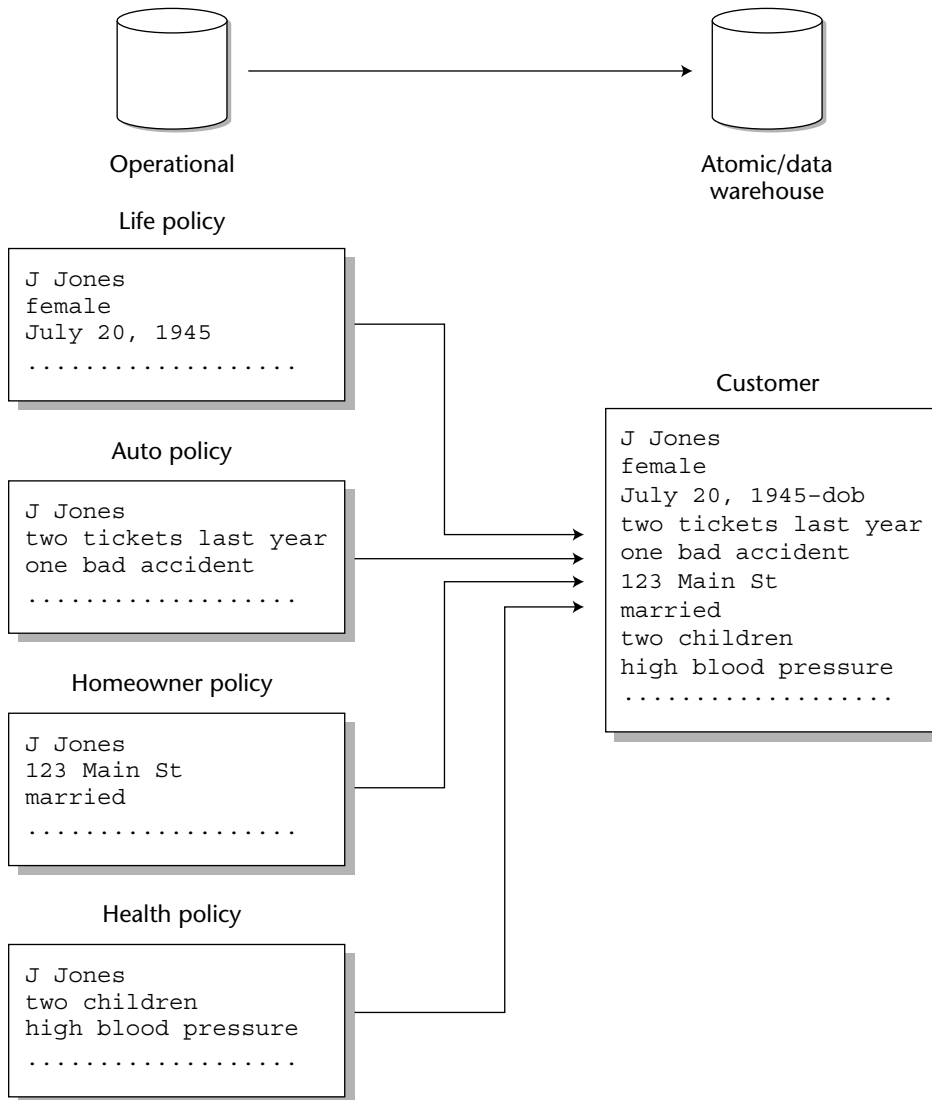


Figure 1-12 As data is transformed from the operational environment to the data warehouse environment, it is also integrated.

Who Is the User?

Much about the data warehouse or DSS environment is fundamentally different from the operational environment. When developers and designers who have spent their entire careers in the operational environment first encounter the data warehouse or DSS environment, they often feel ill at ease. To help them appreciate why there is such a difference from the world they have known, they should understand a little bit about the different users of the data warehouse.

The data-warehouse user — also called the *DSS analyst* — is a businessperson first and foremost, and a technician second. The primary job of the DSS analyst is to define and discover information used in corporate decision-making.

It is important to peer inside the head of the DSS analyst and view how he or she perceives the use of the data warehouse. The DSS analyst has a mindset of “Give me what I say I want, and then I can tell you what I really want.” In other words, the DSS analyst operates in a mode of discovery. Only on seeing a report or seeing a screen can the DSS analyst begin to explore the possibilities for DSS. The DSS analyst often says, “Ah! Now that I see what the possibilities are, I can tell you what I really want to see. But until I know what the possibilities are I cannot describe to you what I want.”

The attitude of the DSS analyst is important for the following reasons:

- It is legitimate. This is simply how DSS analysts think and how they conduct their business.
- It is pervasive. DSS analysts around the world think like this.
- It has a profound effect on the way the data warehouse is developed and on how systems using the data warehouse are developed.

The classical *system development life cycle* (SDLC) does not work in the world of the DSS analyst. The SDLC assumes that requirements are known (or are able to be known) at the start of design, or at least that the requirements can be discovered. In the world of the DSS analyst, though, new requirements usually are the last thing to be discovered in the DSS development life cycle. The DSS analyst starts with existing requirements, but factoring in new requirements is almost an impossibility. A very different development life cycle is associated with the data warehouse.

The Development Life Cycle

We have seen how operational data is usually application-oriented and, as a consequence, is unintegrated, whereas data warehouse data must be integrated. Other major differences also exist between the operational level of data and processing, and the data warehouse level of data and processing. The underlying development life cycles of these systems can be a profound concern, as shown in Figure 1-13.

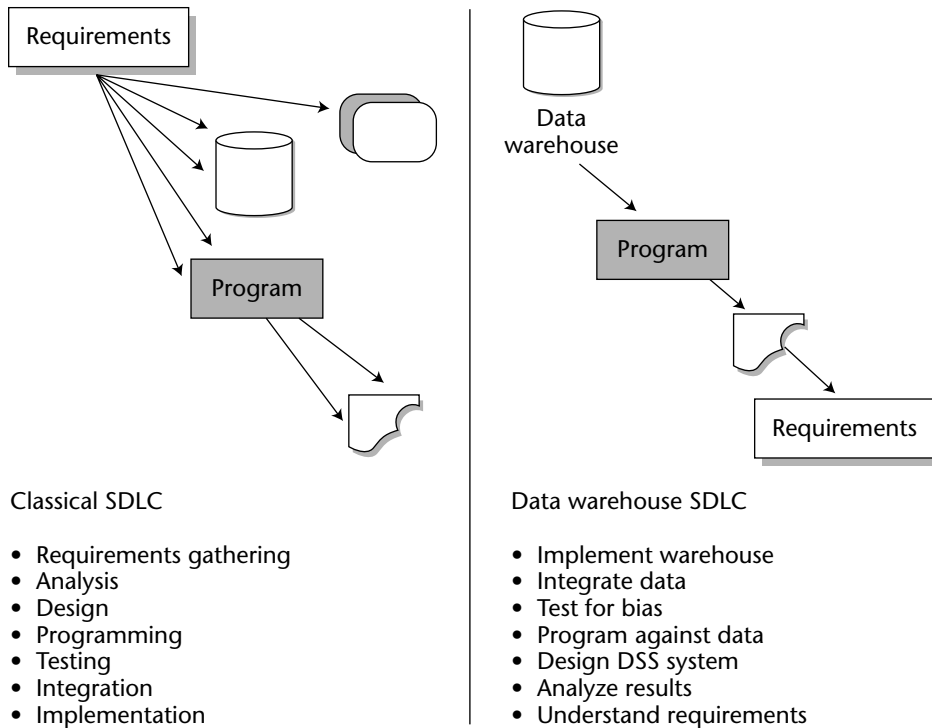


Figure 1-13 The system development life cycle for the data warehouse environment is almost exactly the opposite of the classical SDLC.

Figure 1-13 shows that the operational environment is supported by the classical systems development life cycle (the SDLC). The SDLC is often called the “waterfall” development approach because the different activities are specified and one activity — upon its completion — spills down into the next activity and triggers its start.

The development of the data warehouse operates under a very different life cycle, sometimes called the CLDS (the reverse of the SDLC). The classical SDLC is driven by requirements. In order to build systems, you must first understand the requirements. Then you go into stages of design and development. The CLDS is almost exactly the reverse. The CLDS starts with data. Once the data is in hand, it is integrated and then tested to see what bias there is to the data, if any. Programs are then written against the data. The results of the programs are analyzed, and finally the requirements of the system are understood. Once the requirements are understood, adjustments are made to the design of the system, and the cycle starts all over again for a different set of data. Because of the constant resetting of the development life cycle for different types of data, the CLDS development approach is usually called a “spiral” development methodology.

The CLDS is a classic data-driven development life cycle, while the SDLC is a classic requirements-driven development life cycle. Trying to apply inappropriate tools and techniques of development results only in waste and confusion. For example, the *Computer Aided Software Engineering* (CASE) world is dominated by requirements-driven analysis. Trying to apply CASE tools and techniques to the world of the data warehouse is not advisable, and vice versa.

Patterns of Hardware Utilization

Yet another major difference between the operational and the data warehouse environments is the pattern of hardware utilization that occurs in each environment. Figure 1-14 illustrates this.

The left side of Figure 1-14 shows the classic pattern of hardware utilization for operational processing. There are peaks and valleys in operational processing, but ultimately there is a relatively static and predictable pattern of hardware utilization.

There is an essentially different pattern of hardware utilization in the data warehouse environment (shown on the right side of the figure) — a binary pattern of utilization. Either the hardware is being utilized fully or not at all. It is not useful to calculate a mean percentage of utilization for the data warehouse environment. Even calculating the moments when the data warehouse is heavily used is not particularly useful or enlightening.

This fundamental difference is one more reason why trying to mix the two environments on the same machine at the same time does not work. You can optimize your machine either for operational processing or for data warehouse processing, but you cannot do both at the same time on the same piece of equipment.

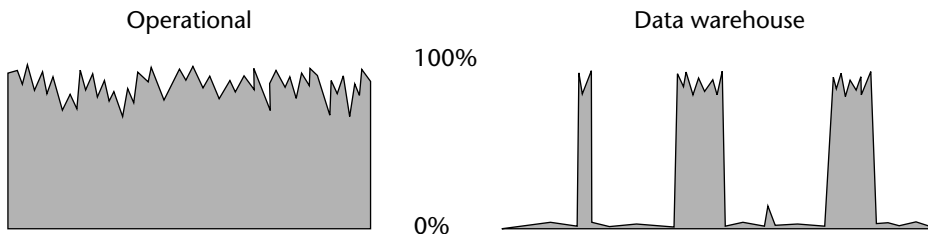


Figure 1-14 The different patterns of hardware utilization in the different environments.

Setting the Stage for Re-engineering

Although indirect, there is a very beneficial side effect of going from the production environment to the architected, data warehouse environment. Figure 1-15 shows the progression.

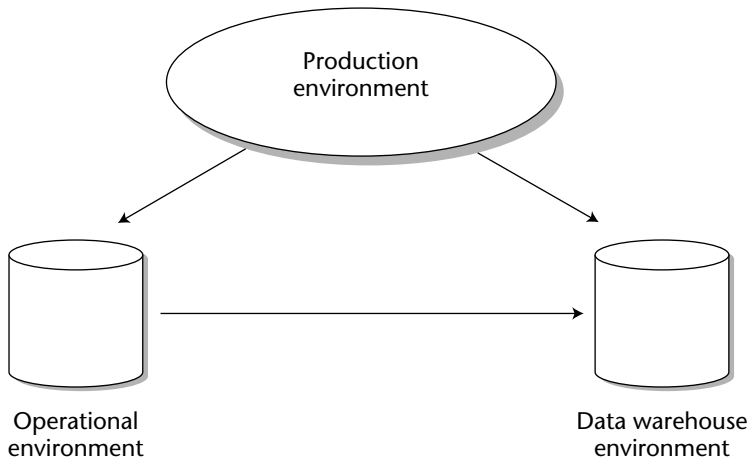


Figure 1-15 The transformation from the legacy systems environment to the architected, data warehouse-centered environment.

In Figure 1-15, a transformation is made in the production environment. The first effect is the removal of the bulk of data — mostly archival — from the production environment. The removal of massive volumes of data has a beneficial effect in various ways. The production environment is easier to:

- Correct
- Restructure
- Monitor
- Index

In short, the mere removal of a significant volume of data makes the production environment a much more malleable one.

Another important effect of the separation of the operational and the data warehouse environments is the removal of informational processing from the production environment. Informational processing occurs in the form of reports,

screens, extracts, and so forth. The very nature of information processing is constant change. Business conditions change, the organization changes, management changes, accounting practices change, and so on. Each of these changes has an effect on summary and informational processing. When informational processing is included in the production, legacy environment, maintenance seems to be eternal. But much of what is called maintenance in the production environment is actually informational processing going through the normal cycle of changes. By moving most informational processing off to the data warehouse, the maintenance burden in the production environment is greatly alleviated. Figure 1-16 shows the effect of removing volumes of data and informational processing from the production environment.

Once the production environment undergoes the changes associated with transformation to the data warehouse-centered, architected environment, the production environment is primed for re-engineering because:

- It is smaller.
- It is simpler.
- It is focused.

In summary, the single most important step a company can take to make its efforts in re-engineering successful is to first go to the data warehouse environment.

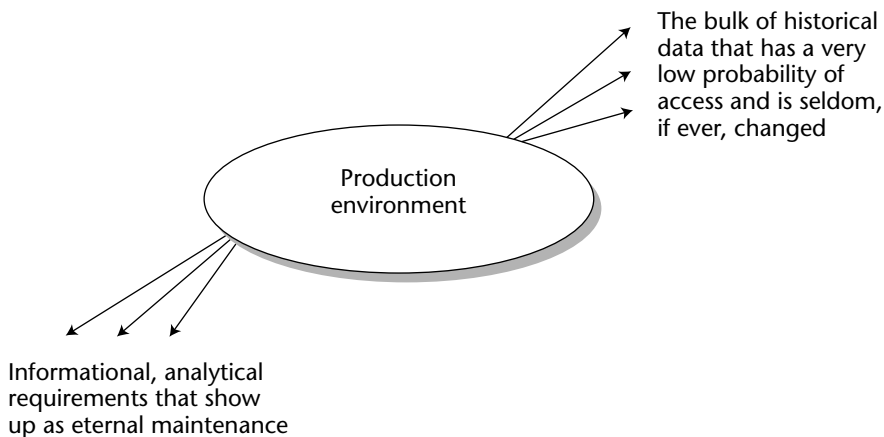


Figure 1-16 Removing unneeded data and information requirements from the production environment – the effects of going to the data warehouse environment.

Monitoring the Data Warehouse Environment

Once the data warehouse is built, it must be maintained. A major component of maintaining the data warehouse is managing performance, which begins by monitoring the data warehouse environment.

Two operating components are monitored on a regular basis: the data residing in the data warehouse and the usage of the data. Monitoring the usage of the data in the data warehouse environment is essential to effectively manage the data warehouse. Some of the important results that are achieved by monitoring this data include the following:

- Identifying what growth is occurring, where the growth is occurring, and at what rate the growth is occurring
- Identifying what data is being used
- Calculating what response time the end user is getting
- Determining who is actually using the data warehouse
- Specifying how much of the data warehouse end users are using
- Pinpointing when the data warehouse is being used
- Recognizing how much of the data warehouse is being used
- Examining the level of usage of the data warehouse

If the data architect does not know the answer to these questions, he or she can't effectively manage the data warehouse environment on an ongoing basis.

As an example of the usefulness of monitoring the data warehouse, consider the importance of knowing what data is being used inside the data warehouse. The nature of a data warehouse is constant growth. History is constantly being added to the warehouse. Summarizations are constantly being added. New extract streams are being created. And the storage and processing technology on which the data warehouse resides can be expensive. At some point, questions arise such as, "Why is all of this data being accumulated? Is there really anyone using all of this?" Whether there is any legitimate user of the data warehouse, there certainly is a growing cost to the data warehouse as data is put into it during its normal operation.

As long as the data architect has no way to monitor usage of the data inside the warehouse, there is no choice but to continually buy new computer resources — more storage, more processors, and so forth. When the data architect can monitor activity and usage in the data warehouse, he or she can determine which data is not being used. It is then possible, and sensible, to move unused data to less-expensive media. This is a very real and immediate pay-back to monitoring data and activity.

The data profiles that can be created during the data-monitoring process include the following:

- A catalog of all tables in the warehouse
- A profile of the contents of those tables
- A profile of the growth of the tables in the data warehouse
- A catalog of the indexes available for entry to the tables
- A catalog of the summary tables and the sources for the summary

The need to monitor activity in the data warehouse is illustrated by the following questions:

- What data is being accessed?
 - When?
 - By whom?
 - How frequently?
 - At what level of detail?
- What is the response time for the request?
- At what point in the day is the request submitted?
- How big was the request?
- Was the request terminated, or did it end naturally?

Response time in the DSS environment is quite different from response time in the OLTP environment. In the OLTP environment, response time is almost always mission critical. The business starts to suffer immediately when response time turns bad in OLTP. In the DSS environment, there is no such relationship. Response time in the DSS data warehouse environment is always relaxed. There is no mission-critical nature to response time in DSS. Accordingly, response time in the DSS data warehouse environment is measured in minutes and hours and, in some cases, in terms of days.

Just because response time is relaxed in the DSS data warehouse environment does not mean that response time is not important. In the DSS data warehouse environment, the end user does development iteratively. This means that the next level of investigation of any iterative development depends on the results attained by the current analysis. If the end user does an iterative analysis and the turnaround time is only 10 minutes, he or she will be much more productive than if turnaround time is 24 hours. There is, then, a very important relationship between response time and productivity in the DSS environment. Just because response time in the DSS environment is not mission-critical does not mean that it is not important.

The ability to measure response time in the DSS environment is the first step toward being able to manage it. For this reason alone, monitoring DSS activity is an important procedure.

One of the issues of response time measurement in the DSS environment is the question, "What is being measured?" In an OLTP environment, it is clear what is being measured. A request is sent, serviced, and returned to the end user. In the OLTP environment, the measurement of response time is from the moment of submission to the moment of return. But the DSS data warehouse environment varies from the OLTP environment in that there is no clear time for measuring the return of data. In the DSS data warehouse environment, often a lot of data is returned as a result of a query. Some of the data is returned at one moment, and other data is returned later. Defining the moment of return of data for the data warehouse environment is no easy matter. One interpretation is the moment of the first return of data; another interpretation is the last return of data. And there are many other possibilities for the measurement of response time; the DSS data warehouse activity monitor must be able to provide many different interpretations.

One of the fundamental issues of using a monitor on the data warehouse environment is where to do the monitoring. One place the monitoring can be done is at the end-user terminal, which is convenient — many machine cycles are free here and the impact on system-wide performance is minimal. To monitor the system at the end-user terminal level implies that each terminal that will be monitored will require its own administration. In a world where there are as many as 10,000 terminals in a single DSS network, trying to administer the monitoring of each terminal is nearly impossible.

The alternative is to do the monitoring of the DSS system at the server level. After the query has been formulated and passed to the server that manages the data warehouse, the monitoring of activity can occur. Undoubtedly, administration of the monitor is much easier here. But there is a very good possibility that a system-wide performance penalty will be incurred. Because the monitor is using resources at the server, the impact on performance is felt throughout the DSS data warehouse environment. The placement of the monitor is an important issue that must be thought out carefully. The trade-off is between ease of administration and minimization of performance requirements.

One of the most powerful uses of a monitor is to be able to compare today's results against an "average" day. When unusual system conditions occur, it is often useful to ask, "How different is today from the average day?" In many cases, it will be seen that the variations in performance are not nearly as bad as imagined. But in order to make such a comparison, there needs to be an average-day profile, which contains the standard important measures that describe a day in the DSS environment. Once the current day is measured, it can then be compared to the average-day profile.

Of course, the average day changes over time, and it makes sense to track these changes periodically so that long-term system trends can be measured.

Summary

This chapter has discussed the origins of the data warehouse and the larger architecture into which the data warehouse fits. The architecture has evolved throughout the history of the different stages of information processing. There are four levels of data and processing in the architecture — the operational level, the data warehouse level, the departmental or data mart level, and the individual level.

The data warehouse is built from the application data found in the operational environment. The application data is integrated as it passes into the data warehouse. The act of integrating data is always a complex and tedious task. Data flows from the data warehouse into the departmental or data mart environment. Data in the departmental or data mart environment is shaped by the unique processing requirements of the department.

The data warehouse is developed under a completely different development approach than that used for classical application systems. Classically, applications have been developed by a life cycle known as the SDLC. The data warehouse is developed under an approach called the *spiral development methodology*. The spiral development approach mandates that small parts of the data warehouse be developed to completion, and then other small parts of the warehouse be developed in an iterative approach.

The users of the data warehouse environment have a completely different approach to using the system. Unlike operational users who have a straightforward approach to defining their requirements, the data warehouse user operates in a mindset of discovery. The end user of the data warehouse says, “Give me what I say I want, and then I can tell you what I really want.”