CHAPTER 1
CAUSES OF DATA QUALITY PROBLEMS

Data is impacted by numerous processes, most of which affect its quality to a certain degree. I had to deal with data quality problems on a daily basis for many years and have seen every imaginable scenario of how data quality deteriorates. While each situation is different, I eventually came up with a classification shown in Figure 1-1. It shows 13 categories of processes that cause the data problems, grouped into three high-level categories.

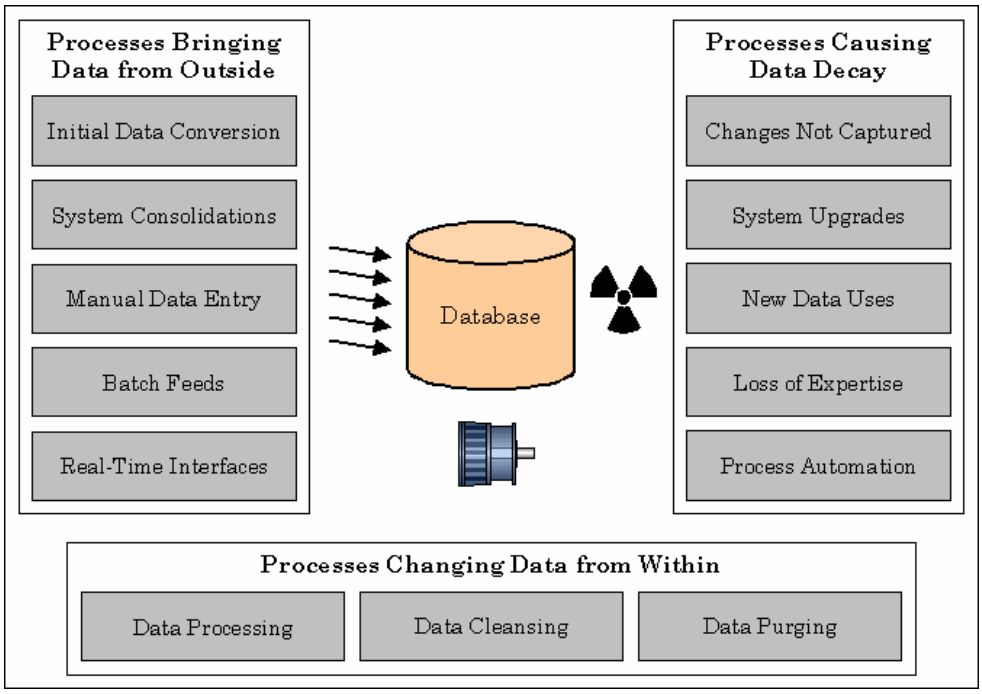


Figure 1-1: Processes Affecting Data Quality

The group on the left shows processes that bring data into the database from outside – either manually or through various interfaces and data integration techniques. Some of these incoming data may be incorrect in the first place and simply migrate from one place to another. In other cases, the errors are introduced

in the process of data extraction, transformation, or loading. High volumes of the data traffic dramatically magnify these problems.

The group on the right shows processes that manipulate the data inside the databases. Some of these processes are routine, while others are brought upon by periodic system upgrades, mass data updates, database redesign, and a variety of ad-hoc activities. Unfortunately, in practice most of these procedures lack time and resources, as well as reliable meta data necessary to understand all data quality implications. It is not surprising, then, that internal data processing often leads to numerous data problems.

The group on the bottom shows processes that cause accurate data to become inaccurate over time, without any physical changes made to it. The data values are not modified, but their accuracy takes a plunge! This usually happens when the real world object described by the data changes, but the data collection processes do not capture the change. The old data turns obsolete and incorrect.

In this chapter we will systematically discuss the 13 processes presented in Figure 1-1 and explain how and why they negatively affect data quality.

1.1. INITIAL DATA CONVERSION

Databases rarely begin their life empty. More often the starting point in their lifecycle is a data conversion from some previously exiting data source. And by a cruel twist of fate, it is usually a rather violent beginning. Data conversion usually takes the better half of new system implementation effort and almost never goes smoothly.

When I think of data conversion, my first association is with the mass extinction of dinosaurs. For 150 million years, dinosaurs ruled the earth. Then one day – BANG – a meteor came crashing down. Many animals died on impact, others never recovered and slowly disappeared in the ensuing darkness. It took millions of years for flora and fauna to recover. In the end, the formerly dominant dinosaurs were completely wiped out and replaced by the little furry creatures that later evolved into rats, lemurs, and the strange apes who find nothing better to do than write data quality books.

The data conversion is no different. Millions of unsuspecting data elements quietly do their daily work until – BANG – data conversion comes hurling at them. Much data never makes it to the new database; many of the lucky ones mutate so much in transition that they simply die out slowly in the aftermath. Most companies live with the consequences of bad data conversions for years or even decades. In fact, some data problems can be traced to "grandfathers of data conversions," i.e. conversion to the system from which the data were later converted to the system from which the data is converted to the new system…

I still vividly remember one of my first major data conversion projects. I was on a team implementing a new pension administration system. Among other things, we needed to convert employee compensation data from the "legacy" HR database. The old data was stored in much detail – by paycheck and compensation type. The new database simply needed aggregate monthly pensionable earnings. The mapping was trivial – take all records with relevant compensation types (provided as a list of valid codes), add up amounts for each calendar month, and place the result into the new bucket.

The result was disastrous. Half of the sample records I looked at did not match the summary reports printed from the old system. The big meeting was called for the next morning, and in the wee hours of the night, I had a presence of mind to stop looking for bugs in the code and poke into the source data. The data certainly did not add up to what was showing on the summary reports, yet the reports were produced from these very data! This mathematical puzzle kept me up till dawn. By then I had most of it figured out.

Half a dozen compensation codes included in the aggregate amounts were missing from our list. In fact they were even missing from the data dictionary! Certain codes were used in some years but ignored in other years. Records with negative amounts – retroactive adjustments – were aggregated into the previous month, which they technically belonged to, rather than the month of the paycheck. Apparently the old system had a ton of code that applied all these rules to calculate proper monthly pensionable earnings. The new system was certainly not programmed to do so, and nobody remembered to indicate all this logic in the mapping document.

It took us eight weeks of extensive data profiling, analysis, and quality assessment to complete this portion of the project, whereas one week was budgeted for. We were lucky, though, that the problem was relatively easy to expose. In many conversion projects, the data is converted based on the mapping specifications that are ridiculously out-of-sync with reality. The result is predictable – mass extinction of the data and the project teams.

So what is it that makes data conversion so dangerous? At the heart of the issue is the fact that every system is made of three layers: database, business rules, and user interface. As a result what users see is not what is actually stored in the database. This is especially true for older "legacy" systems. During the data conversion it is the data structure that is usually the center of attention. The data is mapped between old and new databases. However, since the business rule layers of the source and destination systems are very different, this approach inevitably fails. The converted data, while technically correct, is inaccurate for all practical purposes.

The second problem is the typical lack of reliable meta data about the source database. Think about it, how often do we find value codes in the data that are missing from the mapping documents? The answer is: All the time. But how can we believe any meta data when even such a basic component is incorrect? Yet, over and over again, data conversions are made to the specifications built on incomplete, incorrect, and obsolete meta data.

To summarize, the quality of the data after conversion is directly proportional to the amount of time spent to analyze and profile the data and uncover the true data content. In an ideal data conversion project, 80% of time is spent on data analysis and 20% on coding transformation algorithms.

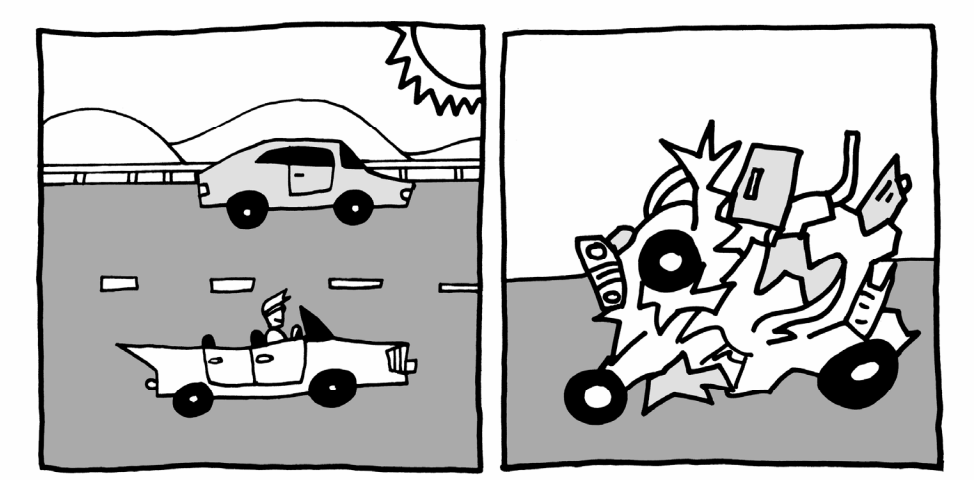
So far I have talked about the data problems introduced by the conversion process; however, the source data itself is never perfect. Existing erroneous data tends to mutate and spread out during conversion like a virus. Some bad records are dropped and not converted at all. Others are changed by the transformation routines. Such changed and aggregated errors are much more difficult to identify and correct after conversion. What is even worse – the bad records impact conversion of many correct data elements.

To conclude, data conversion is the most difficult part of any system implementation. The error rate in a freshly populated new database is often an order of magnitude above that of the old system from which the data is converted. As a major source of the data problems, data conversion must be treated with the utmost respect it deserves.

1.2. System Consolidations

Database consolidations are the most common occurrence in the information technology landscape. They take place regularly when old systems are phased out or combined. And, of course, they always follow company mergers and acquisitions. Database consolidations after corporate mergers are especially troublesome because they are usually unplanned, must be completed in an unreasonably tight timeframe, take place in the midst of the cultural clash of IT departments, and are accompanied by inevitable loss of expertise when key people leave midway through the project.

An old man once rode his Pontiac three miles in the oncoming traffic before being stopped. He was very surprised why everybody was going the wrong way. That is exactly how I feel when involved in a data consolidation project.



Instead of two small cars we have one big pile of iron, plastic, and rubber.

Data consolidation faces the same challenges as initial data conversion but magnified to a great extent. I have already discussed why conversions cause data

quality problems. The idea of consolidation adds the whole new dimension of complexity. First of all, the data is often merged into an existing non-empty database, whose structure can be changed little or none whatsoever. However, often the new data simply does not fit! The efforts to squeeze square pegs into round holes are painful, even to an outside observant.

More importantly, the data in the consolidated systems often overlap. There are duplicates, there are overlaps in subject populations and data histories, and there are numerous data conflicts. The traditional approach is to setup a winner-loser matrix indicating which source data element is picked up in case of a conflict. For instance, date of birth will be taken from System A if present, from System B otherwise, and from System C if it is missing in both A and B. This rarely works because it assumes that data on System A is always correct – a laughable assumption. To mitigate the problem, the winner-loser matrix is usually transformed into a complex conditional hierarchy. Now we take the date of birth from System A for all males born after 1956 in California, except if that date of birth is January 1, 1970, in which case we take it from System B, unless of course the record on System B is marked as edited by John Doe who was fired for playing games on the computer while doing data entry, in which case we pull it from Spreadsheet C…

At some point the winner-loser matrix is so complex, that nobody really understands what is going on. The programmers argue with business analysts about the exact meaning of the word "unless," and consumption of antidepressants is on the rise. It is time to scrap the approach and start over.

I will discuss the proper methodology for data consolidation in the next chapter. For now we just conclude that data consolidation is one of the main causes of data problems and must be treated with great fear. Walking a tightrope is child's play in comparison.

1.3. MANUAL DATA ENTRY

Despite high automation, much data is (and will always be!) typed into the databases by people through various forms and interfaces. The most common source of data inaccuracy is that the person manually entering the data just makes a mistake. To err, after all, is human! People mistype; they choose a wrong entry from the list or enter right data value into the wrong box. I had, at one time, participated in a data-cleansing project where the analysts were supposed to carefully check the corrections before entering them – and still 3% of the corrections were entered incorrectly. This was in a project where data quality was the primary objective!

Common error rate in data entry is much higher. Over time I collected my personal indicative data from various databases. My collection includes eight different spellings of my first name, along with a dozen of my last name, and four dates of birth; I was marked as male, female, and even the infamous 'U'.

Convoluted and inconvenient data entry forms often further complicate the data entry challenge. The same applies to data entry windows and web-based interfaces. Frustration in using a form will lead to exponential increase in the number of errors. Users often tend to find the easiest way to complete the form, even if that means making deliberate mistakes.

A common data entry problem is handling missing values. Users may assign the same blank value to various types of missing values. When "blank" is not allowed, users often enter meaningless value substitutes. Default values in data entry forms are often left untouched. The first entry in any list box is selected more often than any other entry.

Good data entry forms and instructions somewhat mitigate data entry problems. In an ideal fantasy world, data entry is as easy to the user as possible: fields are labeled and organized clearly, data entry repetitions are eliminated, and data is not required when it is not yet available or is already forgotten. The reality of data entry, however, is not that rosy (and probably won't be for years to come). Thus we must accept that manual data entry will always remain a significant cause of data problems.

1.4. BATCH FEEDS

Batch feeds are large regular data exchange interfaces between systems. The ever-increasing number of databases in the corporate universe communicates through complex spiderwebs of batch feeds.

In the old days, when Roman legions wanted to sack a fortified city, they hurled heavy stones at its walls, day after day. Not many walls could withstand such an assault. In the modern world, the databases suffer the same unrelenting onslaught of batch feeds. Each batch carries large volumes of data, and any problem in it causes great havoc further magnified by future feeds. The batch feeds can be usually tied to the greatest number of data quality problems. While each individual feed may not cause too many errors, the problems tend to accumulate from batch to batch. And there is little opportunity to fix the ever-growing backlog.

So why do the well-tested batch feed programs falter? The source system that originates the batch feed is subject to frequent structural changes, updates, and upgrades. Testing the impact of these changes on the data feeds to multiple independent downstream databases is a difficult and often impractical step. Lack of regression testing and quality assurance inevitably leads to numerous data problems with batch feeds any time the source system is modified – which is all of the time!

Consider a simple example of a payroll feed to the employee benefit administration system. Paycheck data is extracted, aggregated by pay type, and loaded into monthly buckets. Every few months a new pay code is added into the payroll system to expand its functionality. In theory, every downstream system may be impacted, and thus each downstream batch feed must be re-evaluated. In practice, this task often slips through the cracks, especially since many systems, such as benefit administration databases, are managed by other departments or even outside vendors. The records with the new code arrive at the doorsteps of the destination database and are promptly dropped from consideration. In the typical scenario, the problem is caught after a few feeds. By then, thousands of bad records were created.

The other problem with batch feeds is that they quickly spread bad data from database to database. Any errors that somehow find their way into the source

system will usually flow immediately through the batch feeds like viruses and can blend well enough with the rest of the batch data to come unnoticed and cause the greatest damage.

The batch feeds are especially dangerous because newly arrived records do not sit quietly. The incoming transactions usually trigger immediate processing in the target database. Even during loading, existing data might be changed to reflect new transactions. Thus more data is immediately corrupted. Additional processing can be triggered, creating more and more errors in an avalanche of bad data. For example, erroneous employee termination records arriving to a benefit administration system will initiate a sequence of benefit calculations. The results will be forwarded to the benefit payment system, which will create more wrong data and initiate more wrong activities. The cost of a single bad record can run in to thousands of dollars. It is hard to even visualize the destructive power of a batch feed full of erroneous data.

1.5. REAL-TIME INTERFACES

More and more data is exchanged between the systems through real-time (or near real-time) interfaces. As soon as the data enters one database, it triggers procedures necessary to send transactions to other downstream databases. The advantage is immediate propagation of data to all relevant databases. Data is less likely to be out-of-sync. You can close your eyes and imagine the millions of little data pieces flying from database to database across vast distances with lightning speed, making our lives easier. You see the triumph of the information age! I see Wile E. Coyote in his endless pursuit of the Road Runner. Going! Going! Gosh!

The basic problem is that data is propagated too fast. There is little time to verify that the data is accurate. At best, the validity of individual attributes is usually checked. Even if a data problem can be identified, there is often nobody at the other end of the line to react. The transaction must be either accepted or rejected (whatever the consequences). If data is rejected, it may be lost forever!

Further, the data comes in small packets, each taken completely out of context. A packet of data in itself may look innocent, but the data in it may be totally erroneous. I once received an email from a Disney World resort thanking me for

staying there. The text was grammatically perfect and would have made me feel great, except I did not go to Disney that year.

The point is that "faster" and "better" rarely go hand-in-hand. More often quality is the price paid for faster delivery. Real-time data propagation is no exception – it is a liability from the data quality perspective. This does not make it any less valuable. Real-time interfaces save millions of dollars and significantly improve efficiency of the information systems. But data quality suffers in the process, and this has to be recognized. When an old batch feed is replaced by a new real-time interface, the potential cost of data quality deterioration must be evaluated and weighed against the benefit of faster data propagation.

1.6. DATA PROCESSING

Data processing is at the heart of all operational systems. It comes in many shapes and forms – from regular transactions triggered by users to end-of-the-year massive calculations and adjustments. In theory, these are repetitive processes that should work "like a clock." In practice there is nothing steady in the world of computer software. Both programs and underlying data change and evolve, with the result that one morning the proverbial sun rises in the West, or worse yet, does not rise at all.

The first part of the problem is the change in the programs responsible for regular data processing. Minor changes and tweaks are as regular as normal use. These are often not adequately tested based on the common misconception that small changes cannot have much impact. Of course a tiny bug in the code applied to a million records can create a million errors faster than you can read this sentence.

On the flip side, the programs responsible for regular processing often lag behind changes in the data caused by new collection procedures. The new data may be fine when it enters the database, but it may be different enough to cause regular processing to produce erroneous results.

A more subtle problem is when processing is accidentally done at the wrong time. Then the correct program may yield wrong results because the data is not in the state it is supposed to be. A simple example is running the program that calculates

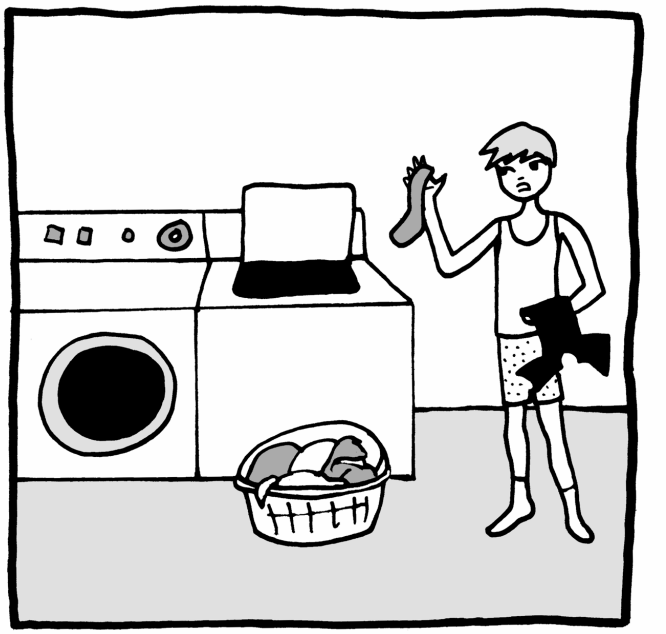
weekly compensation before the numbers from the hours tracking system were entered.

In theory, documenting the complete picture of what is going on in the database and how various processes are interrelated would allow us to completely mitigate the problem. Indeed, someone could then analyze the data quality implications of any changes in code, processes, data structure, or data collection procedures and thus eliminate unexpected data errors. In practice, this is an insurmountable task. For that reason, regular data processing inside the database will always be a cause of data problems.

1.7. DATA CLEANSING

The data quality topic has caught on in recent years, and more and more companies are attempting to cleanse the data. In the old days, cleansing was done manually and was rather safe. The new methodologies have arrived that use automated data cleansing rules to make corrections *en masse*. These methods are of great value and I, myself, am an ardent promoter of the rule-driven approach to automated data cleansing. Unfortunately, the risks and complexities of automated data cleansing are rarely well understood.

The reader might ask in surprise, "How come that data cleansing that strives to correct data errors may instead create new ones?" Those who, like me, in their college years mixed whites and colors in the laundry machine will know how hopelessly "dirty" the white shirts become after such cleansing. And so, despite the noble goal of higher data quality, data cleansing often creates more data problems than it corrects. This situation is further complicated by the complacency that commonly sets in after the cleansing project is "completed."



Data cleansing is dangerous mainly because data quality problems are usually complex and interrelated. Fixing one problem may create many others in the same or other related data elements. For instance, employment history is tightly linked with position history, pay rate history, and many other employment data attributes. Making corrections to any one of these data categories will make the data inconsistent with all other categories.

I also must mention that automated data cleansing algorithms are implemented by computer programs, which will inevitably have bugs. Bugs in these algorithms are very dangerous because they often impact thousands of records.

Another problem is that data quality specifications often do not reflect actual data requirements. As a result, data may be brought in compliance with some theoretical model but remain incorrect for actual use. For example, in one of my early projects the client – a large corporation with a history of acquisitions – requested to cleanse employment history on their HR system. One of the major problems was missing or incorrect original hire date for many employees, used to calculate amount of retirement pension benefits. I had access to several "legacy" data sources and was able to devise a series of algorithms to correct the problem for over 15,000 employees. Unfortunately, many of the employees were not

originally hired by my client but came through numerous acquisitions. The pension calculations performed by the HR system were not supposed to use the period of employment with the acquired companies prior to the acquisition. Therefore, what the system really expected in the original hire date field for the employees from acquired units was the acquisition date. However, the data quality specifications I was given did not reflect that. As a result, many corrections were wrong. Since I had a complete audit trail of all data changes, it was not too difficult to fix the problem. Many data cleansing projects do not have the happy ending, and newly created errors linger for years.

To summarize, data cleansing is a double-edged sword that can hurt more than help if not used carefully. I will discuss the proper methodology for data cleansing in the next chapter.

1.8. DATA PURGING

Old data is routinely purged from systems to make way for more data. This is normal when a retention limit is satisfied and old data no longer necessary. However, data purging is highly risky for data quality.

When data is purged, there is always a risk that some relevant data is purged by accident. The purging program may simply fail. More likely, the data structure may have changed since the last purging due to a system upgrade, data conversion, or any of the other discussed above processes. So now the purging may accidentally impact the wrong data. More data than intended can be purged. Or alternatively less data than intended might be purged, which is equally bad since it leaves incomplete records in the database.

Another factor that complicates things is the presence of erroneous data in the database. The erroneous data may accidentally fit the purging criteria and get removed when it should be left alone, or vice versa. For example, if the HR system is setup to purge data for all employees that were terminated over five years ago, then it will wipe out records for some employees with incorrectly entered termination dates.

Since purging often equals destruction, it has to be exercised with great care. The fact that it worked reasonably well last year does not guarantee that it will work

again this year. Data is too volatile a compound to be fooled around with. This requires more sophisticated design of the purging programs than is often used for such a trivial technical task. After all, it seems quite easy to just wipe out a few millions of records. So we live with the data quality consequences of data purging in almost every database.

1.9. CHANGES NOT CAPTURED

Data can become obsolete (and thus incorrect) simply because the object it describes has changed. If a caterpillar has turned into a butterfly but is still listed as a caterpillar on the finch's menu, the bird is in her right to complain about poor data quality.

This situation is very commonplace in human affairs, too, and inevitably leads to gradual data decay. The data is only accurate if it truly represents real world objects. However, this assumes perfect data collection processes. In reality, object changes regularly go unnoticed to computers. People move, get married, and even die without filling out all necessary forms to record these events in each system where their data is stored. This is actually why, in practice, data about

same person may be totally different across systems, causing pain during consolidation.

In this age of numerous interfaces across systems, we rely largely on the fact that a change made in one place will migrate to all other places. This obviously does not always happen. As a result, changes are not propagated to all concerned databases and data decays. For instance, interfaces often ignore retroactive data corrections. Alternatively, IT personnel may make changes using a backdoor update query, which, of course, does not trigger any transactions to the downstream systems.

Whether the cause is a faulty data collection procedure or a defective data interface, the situation of data getting out of sync with reality is rather common. This is an example of data decay inevitably leading to deterioration of the data quality.

1.10. System Upgrades

Most commercial systems get upgraded every few years. Homegrown software is often upgraded several times a year. While upgrades are not nearly as invasive and painful as system conversions and consolidations, they still often somehow introduce data problems. How can a well tested, better version negatively impact data quality?

The culprit here is the assumption that the data complies with what is theoretically expected of it. In practice, actual data is often far different from what is described in data models and dictionaries. Data fields are used for wrong purposes, and some data is missing while other was massaged into a form acceptable to the prior version. Yet more data just exists harmlessly as an artifact of past generations but should not be touched.

Upgrades expose all these problems. More often than not, they are designed for and tested against what data is expected to be, not what it really is. Once the upgrades are implemented, everything goes haywire. People lose their hair trying to figure out why the system worked in the past, and the new version did beautifully in the testing environment, yet all of the sudden it breaks on every step.

System upgrades usually impact data quality through the described above process of data decay. However, they often require real restructuring and mass updates of

the existing data. Such changes coupled with lack of reliable meta data lead to huge quantities of data errors.

1.11. New Data Uses

Remember that data quality is defined as "fitness to the purpose of use." The data may be good enough for one purpose but inadequate for another. Therefore, new data uses often bring about changes in perceived level of data quality even though underlying data is the same. For instance, HR systems may not care too much to differentiate medical and personal leave of absence – a medical leave coded as a personal leave is not an error for most HR purposes. But start using it to determine eligibility for employee benefits, and such minute details become important. Now a medical leave entered as a personal leave is plain wrong.

The new uses may also put greater premium on data accuracy even without changing the definition of quality. Thus, a 15% error rate in customer addresses may be perfectly fine for telemarketing purposes, but try to survive with that many inaccurate addresses for billing!

Besides accuracy, other aspects of data quality may differ for various uses. Value granularity, or data retention policy, may be inadequate for the new use. For example, employee compensation data retained for three years is adequate for payroll administration but cannot be used to analyze compensation trends.

1.12. Loss of Expertise

On almost every data quality project I worked, there is Dick or Jane or Nancy whose data expertise is unparalleled. Dick was with the department for the last 35 years and is the only person who really understands why for some employees date of hire is stored in the date of birth field, while for others it must be adjusted by exactly 17 days. Jane still remembers times when she did calculations by hand and entered the results into the system that was shut down in 1985, even though she still sometimes accesses the old data when in doubt. When Nancy decided to retire, she was offered hourly work from home at double her salary. Those are true stories.

Much data in databases has a long history. It might have come from old "legacy" systems or have been changed several times in the past. The usage of data fields and value codes changes over time. The same value in the same field will mean totally different thing in different records. Knowledge of these facts allows experts to use the data properly. Without this knowledge, the data may be used literally and with sad consequences.

The same is true about data quality. Data users in the trenches usually know good data from bad and can still use it efficiently. They know where to look and what to check. Without these experts, incorrect data quality assumptions are often made and poor data quality becomes exposed.

Unfortunately much of the data knowledge exists in people's minds rather than meta data documents. As these people move on, retire, or simply forget things, the data is no longer used properly. How do we solve this problem? Besides erecting monuments honoring Dick, Jane, and Nancy, what we need is obviously a well-designed and maintained meta data repository and data quality meta data warehouses. This is a great dream to have, and maybe with luck, some day, our names will be etched on the monuments too. In the meantime, we must deal with the consequences of lost expertise in the form of data decay.

1.13. Process Automation

With the progress of information technology, more and more tasks are automated. It starts from replacement of data entry forms with system interfaces and extends to every layer of our life. Computer programs process and ship orders, calculate insurance premiums, and even send spam – all with no need for human intervention. Where in the past a pair (or several pairs) of human eyes with the full power of trained intellect protected the unsuspecting customers, now we are fully exposed to a computer's ability to do things wrong and not even feel sorry.

A human would automatically validate the data before using it. Computer programs take the data literally and cannot make a proper judgment about the likelihood of it been correct. Some validation screens may be implemented in the automated processes, but these will often fail to see all data peculiarities, or are turned off in the interest of performance. As a result, automation causes data decay!

Another aspect of technology development is greater data exposure to broader group of users. For instance, over the last 15 years it has become possible to publish HR data for employee access via voice response systems and later intranet. Employees can check their eligibility for benefits, various educational programs, and query other information. All of the sudden erroneous HR data became exposed, causing floods of employee complaints. The data did not change, but its perceived quality deteriorated.

SUMMARY

We have discussed various processes that affect data quality. In some cases, bad data comes from outside of the database through data conversions, manual entry, or various data integration interfaces. In other cases, data deteriorate as a result of internal system processing. Yet in many situations, data quality may decline without any changes made to the data itself – the process we referred to as data decay. Each of these problems must be addressed if we are to assume the data quality management responsibility. The next chapter will discuss how it can be done through a comprehensive data quality program.