

AUTOMATED APPLICATION AUDIT

Everyone who reads this article, will either own, support, develop, test or use (or all of the above) a large, complicated application written in either RPG, COBOL or CA:2E on an IBM i. As such you will have a vested interest in the design assets that make the application useful to its users. In the two earlier articles I looked at how application maps were built, and how they were used to reveal granular architecture and function, and also how they were deployed to programmatically reengineer those applications. In this the third article in the series of four, we will have a look at a higher-abstraction of an application's design, and how this can be used to extend ROI for many years.

ARCHITECTURAL EROSION

When business applications are first designed and written, well thought out application architecture contributes to their success and resulting life span. Nothing demonstrates this more conclusively than the success and longevity of thousands of IBM i applications over the last 40 years, many of which are still in daily use. However the continual enhancement, syntax and programming style variations, general maintenance, and time and budget pressures conspire to compromise application architecture. As with geological erosion, architectural erosion is not necessarily noticeable or even problematic for many years. In some cases, and given enough time, the quality, efficiency, and maintainability of the application will begin to suffer from this natural evolution of the code base. This will vary in significance from company to company, and application to application. It's not uncommon to hear of cases years of continued enhancements to an IBM i Application, render the application virtually un-maintainable, especially when matched with delivery time expectations of users, and/or development budgets.

THE MODERN TECHNOLOGY TEASE

The last decade has seen the introduction of many powerful enhancements to i/OS, the RPG/COBOL syntax/compiler, and DB2 for i. The benefit of these enhancements has remained tantalizingly out of reach for most of the current applications, for the simple reason that most of the current application code is written using monolithic procedural methods. Integration with other systems, modernizing the user-interface, implementing SOA strategies, all expect a distributed application design, if it is to be done in a sustainable and optimum way. The task of rewriting these entire systems to take advantage of these modern technologies has, for most companies, been too expensive and risky. The optimum approach is to establish what code is useful and relevant and therefore should be rewritten or re-factored into modern constructs and program designs. Even with application mapping technologies, this is still a significant task on any complicated legacy application.

OPTIMIZING AN APPLICATION WITH THE BUSINESS

Generally one will receive varying degrees of consensus about the relevance of a company's legacy application, depending who one asks in the organization. The specific touch points between the application function, and the business process are rarely known in their entirety, and even more rarely documented and therefore, auditable. It's also not uncommon for applications to outlive users at companies by many years. If the original application designers are no longer with the company, it stands to reason therefore that potentially large parts of the application design assets are only known by the application itself.

With the application designs explicitly defined, documented and ready to hand, analysts and architects can map these designs to business architecture and process accordingly. This can also form the basis of subsequent renewal or replacement strategies if applicable. In addition to this, a number of technological project types become feasible even with limited resources. We'll have a look at some of those in some detail a bit later in the article.

Let's now look at the two most important design assets: the referential integrity data model, and the business logic.

DERIVING A REFERENTIAL INTEGRITY DATA MODEL

The most fundamental design asset of an IBM I application is the data model. The data model of an application is not just the design of the files, tables, views and access paths, but includes the foreign key relationships or referential integrity (RI) between database tables.

Basic Referential Integrity Example

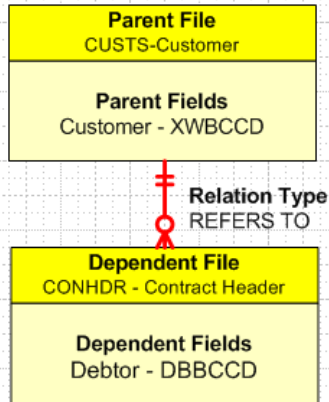


FIGURE 1

The simplest definition of RI is that it defines a relationship between two files in which one is the parent and one is the dependent or child. Records in the dependent file are joined by a unique key in the parent file. For example, the contract header file is a dependent file to the customer master file and records in the contract header file must always contain a valid customer number. Figure 1 shows a diagram displaying the detail of this example RI.

For large and complex applications this task needs to be approached in a structured manner. This can be broken down into a few discrete steps. The first is to establish the physical model by extracting the table, file, view and logical file definitions. This provides both a data dictionary of the database, plus a map of all important keys such as primary or unique identifiers. Taking one table at a time the unique key of the file is compared with all of the keys of other files in turn. Where there is a match between the primary key of the file and any key or at least partial key of the other file a relationship is can be derived. In most cases the analysis is further complicated by virtue of field names being different in different files. In certain cases the difference is simple such as the first two characters are different, while in others the names are completely different. Figure 2 shows a simple example of how two files are joined by a foreign key that is similar in name.

Matching Different Field Names

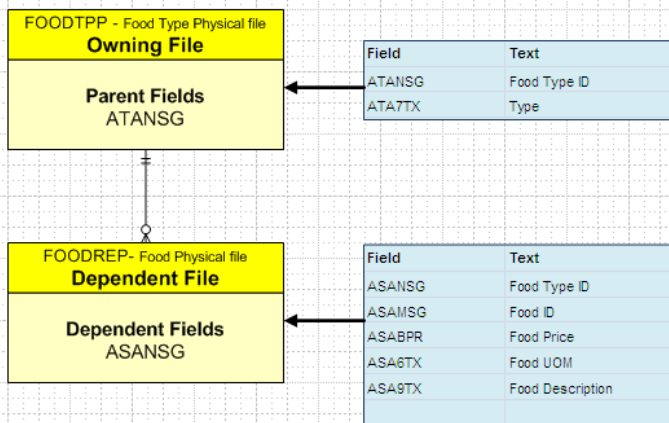


FIGURE 2

Even though DB2/400 was capable of implementing referential integrity (RI) in the database itself since V3R1, virtually zero applications use this approach, even today. In the absence of referential constraints on the tables, RI is managed using program logic. There is nothing wrong with this as a practice, but when one needs or desires to use or visualize the referential data model of such an application, the program source of the system must be analyzed. This analysis serves to validate relationships derived by analyzing the file and field structures, and it can be used to drive relationships that would otherwise not be obvious at all because of file and field

name differences. Programs that use each of the files are analyzed and fields and variables are traced through the source code looking for clues that indicate a relationship between the files. Figure 3 shows a source snippet of field SINIT from the file CNTACS, and the text description gives us a clue that this field means Salesperson.

```
0017.00      A          SINIT          3A          TEXT('Salesperson')
0018.00      A                                     COLHDG('Sales' 'Person')
0019.00      A          STATUS        1A          TEXT('Status')
0020.00      A                                     COLHDG('Sts')
```

FIGURE 3

Figure 4 shows the source code of the file SLMEN and clearly shows that the key of the file is PERSON.

```
0001.00      A          R RSLMEN
-----
0002.00      A          PERSON          3A          TEXT('Person')
0003.00      A          PNAME          34A         TEXT('Full Name')
0004.00      A          K PERSON
```

FIGURE 4

Figure 5 shows a snippet of code from an RPG program that is using the field SINIT read in from the file CNTACS to read the file SLMEN.

```
0193.00      C* Salesman
0194.00      C          if          zsinit <> *blanks
0195.00      C          zsinit      setll(e) rslmen
0196.00      C          if          not %equal(slmen)
0197.00      C          eval          *in37 = *on
0198.00      C          eval          msgid = 'OEM0023'
0199.00      C          callp(e)    rtnmsgtext(msgid:errormsg)
0200.00      C          eval          valid = *off
```

FIGURE 5

This is validation that the SINIT field equals the PERSON field and as such the two files CNTACS and SLMEN have a foreign key relationship. Admittedly this is a very simple example and I knew what I was looking for. In even modest size IBM I applications the task of analyzing an entire system is a tedious and time consuming one, and requires fairly good analytical skills. In the first article in the series titled “Application Mapping on IBM i: Present and Future” I demonstrate how an application map can be used to accelerate analysis tasks by providing mapping between variables and database fields for an entire system. Deriving foreign keys by analyzing source code is a classic use of application mapping technology. It is also possible to write programs that use the application map to analyze the source code and look for clues and proof of foreign key relationships between application files. On large applications it’s virtually a pre-requisite to do this sort of analysis programmatically. The added benefit being it is very easy to keep up to date with a repeatable automated extraction process.

EXTRACTING BUSINESS LOGIC

Over a 20 year period, a company might invest tens of millions of dollars in adding, fine tuning and fixing the business logic in the legacy code. Business Rule Extraction is the process of isolating the code segments which are directly related to business processes. For example: ensuring that when a customer is added to the system, forcing a valid telephone number to be provided by the user, whenever a new customer is added to the system. Figure 6 shows a sample of RPG code used to do this.

```

0169.00      C* Telephone number
0170.00      C          if          ztelno <> *blanks
0171.00      C          ' 0123456789' check ztelno      z1
0172.00      C          if          %found
0173.00      C          eval      *in34 = *on
0174.00      C          eval      msgid = 'OEM0014'
0175.00      C          callp(e)  rtnmsgtext(msgid:errormsg)
0176.00      C          eval      valid = *off
0177.00      C          leavesr
0178.00      C          endif
0179.00      C          endif

```

FIGURE 6

The challenge has always been to identify, isolate and reuse only those designs that are relevant in the new context in which they are desirable. The sheer volume of code, its complexity, and the general lack of resources to understand legacy languages and specifically RPG, represents a tragic potential waste of valuable business assets. The problem is that in the vast majority of legacy RPG and COBOL programs, the business rule logic is mixed in with screen handling, database I/O, and flow control. So harvesting these business rules from legacy applications requires knowledge of the application and the language used to implement it, both of which are steadily diminishing resource.

Once harvested these rules need to be narrated and indexed, thus providing critical information for any analyst, architect or developer charged with renewing or maintaining the legacy application. Figure 7 shows the same piece of code above but with some narrative about the business logic added, along with an index on line 166.99.

```

0169.00      C* Telephone number
0169.96      !=!BRC* If the field ZTELNO is not blank , verify the field ZTELNO against '
0169.97      !=!BRC* 0123456789'. If other values found then the field "Phone" is invalid.
0169.98      %!BRC* Business Rule No. CDEMO/QRPGLESRC/CNTCMAINT/170 Validation.
0169.99      @!BRC* V00002 CNTACS      TELNO      The telephone no. is invalid.
0170.00      B>!BRC          if          ztelno <> *blanks
0171.00      ->!BRC          ' 0123456789' check ztelno      z1
0172.00      ->!BRC          if          %found
0173.00      ->!BRC          eval      *in34 = *on
0174.00      ->!BRC          eval      msgid = 'OEM0014'
0175.00      ->!BRC          callp(e)  rtnmsgtext(msgid:errormsg)
0176.00      ->!BRC          eval      valid = *off
0177.00      ->!BRC          leavesr
0178.00      ->!BRC          endif
0179.00      E>!BRC          endif

```

FIGURE 7

Indexing the business logic in a systematic and structured way by providing reference to its source, the field and file it refers to, and some form of logic type classification, provides some additional benefits. The first and most obvious is that it provides a mechanism to programmatically extract and document the rules in various ways. Figure 8 shows the business logic of the program above documented in MS Word table.

Business Rules for CNTCMAINT, Number of Lines: 6

| Source Member | Narration | Type | Rule No. | Field | File/Program | Rule |
|---------------|--|------|----------|--------|--------------|--------------------------------|
| CNTCMAINT | If the field ZUSERNM is blank then it is invalid. | V | 00001 | USERNM | CNTACS | You must enter a contact name. |
| CNTCMAINT | If the field ZTELNO is not blank, verify the field ZTELNO against ' 0123456789'. If other values found then the field "Phone" is invalid. | V | 00002 | TELNO | CNTACS | The telephone no. is invalid. |
| CNTCMAINT | If the field ZFAXNO is not blank, verify the field ZFAXNO against ' 0123456789'. If other values found then the field "Fax.No." is invalid. | V | 00003 | FAXNO | CNTACS | The fax. no. is invalid. |
| CNTCMAINT | If the field ZSINIT is not blank. | L | 00004 | SINIT | CNTACS | Field logic |
| CNTCMAINT | Verify the field ZSINIT against the file "Salespersons". If not on file then the field "Person" is invalid. | V | 00005 | PERSON | SLMEN | Invalid salesman. |
| CNTCMAINT | If the field ZSTATUS is not blank, set the field Z1 to 1, verify the field ZSTATUS against the array STATUSES from Z1. If not found then the field "Sts" is invalid. | V | 00006 | STATUS | CNTACS | The status is invalid. |

FIGURE 8

The second is the ability to filter cross-reference information about a field in such a way as to show only where business logic is executed against it. Figure 9 shows a spreadsheet of instances across a system where business logic has been applied to a FIELD TELNO. These features can then be put to use by developers wanting centralize business logic across the system by rapidly and accurately accelerating the analysis work required to do this.

| A | B | C | D | E |
|--|-------------|---------------|--|-----------------------------------|
| 1 Rule Where Used for TELNO, Lines: 6 | | | | |
| 2 | | | | |
| 3 | Name | Seq No | Business Rule Narrative | Library Source File |
| 4 | NKCUSSEF | 35 | If the field "Phone" is blank go to FINISH, set the specified @FAXNO field to the specified @WADS field, set the specified @WADS field to blank, set the field @T to zero, set the field @X to 1 | CDEMO QRPGRSRC |
| 5 | CUSFMOLD | 212 | If the field ZTELNO is not blank, verify the field ZTELNO against ' 0123456789'. If other values found then the field "Phone" is invalid. | CDEMO QRPGLSRC |
| 6 | CNTCMAINT | 170 | If the field ZTELNO is not blank, verify the field ZTELNO against ' 0123456789'. If other values found then the field "Phone" is invalid. | CDEMO QRPGLSRC |
| 7 | CUSFMAINT | 210 | If the field ZTELNO is not blank, verify the field ZTELNO against ' 0123456789'. If other values found then the field "Phone" is invalid. | CDEMO QRPGLSRC |
| 8 | NKCUSSEF | 35 | If the field "Phone" is blank go to FINISH, set the specified @FAXNO field to the specified @WADS field, set the specified @WADS field to blank, set the field @T to zero, set the field @X to 1 | CDEMO QRPGRSRC |
| 9 | NKCUSF | 318 | If the field DTELNO is not blank, if check(' 0123456789':dstelno) is greater than zero then the field "Phone" is invalid. | CDEMO QRPGLSRC |

FIGURE 9

In addition to these uses for indexed business logic, it is possible to write programs that can extract the indexed logic to create web-service modules, provide documentation for redevelopment in a modern language such as Java or C#, or even populate business rules management systems such as JBOSS DRULES™ or IBM's ILOG™.

UNLOCKING THE POWER OF THE DATABASE

As I mentioned earlier, very few IBM i application databases have any form of data model or schema explicitly defined. As many companies have discovered, this can significantly hinder development initiatives that use direct access to the application data. Here are four of the most obvious areas that benefit from an explicitly defined data model:

USING MODERN INPUT/OUTPUT METHODS IN PROGRAMS

One of the key aspects of RPG is its native I/O access. The terse and simple syntax for this also gives RPG significant development productivity over other languages. In modern languages such as Java and C# the most common practice is to handle database I/O using embedded SQL statements. For simple, single table reads most developers can create SQL statements that meet this requirement. Things get more complicated where tables and files must be joined for reads, updates or deletes. In this context most developers need to understand the data model and must have access to foreign key relationship information, so as to build the JOIN statements correctly in SQL. Figure 10 shows an entity relationship or data model diagram extracted from a legacy RPG application, which is a common way to visualize how files/tables in the application database relate to each other.

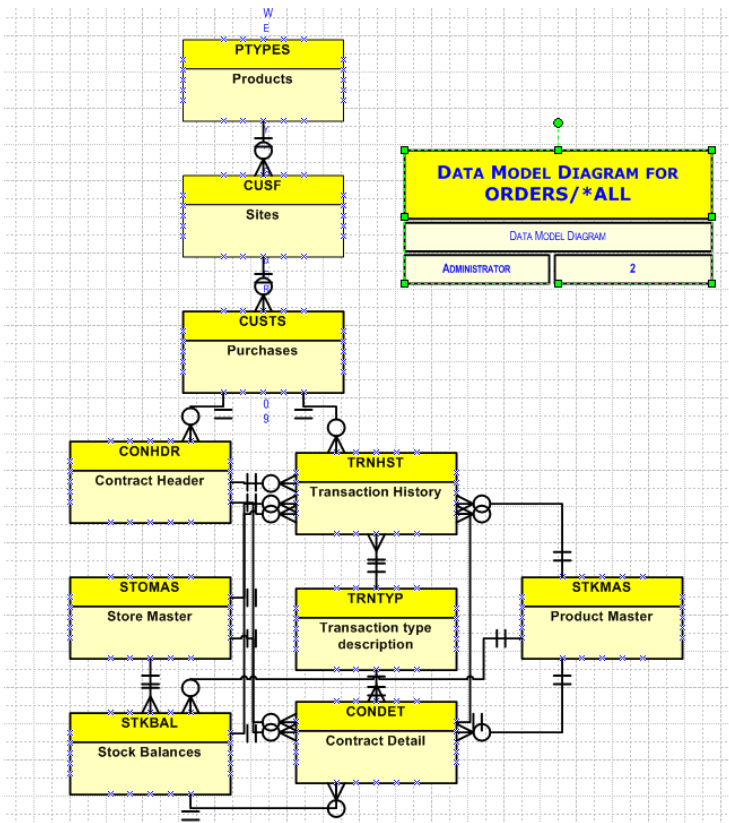


FIGURE 10

Figure 11 in turn shows a spreadsheet with all of the foreign keys that determine the relationships shown in Figure 10.

| | A | B | C | D | E | F |
|----|---|-------------------|---------------|------------------|-------------------|-------------------|
| 1 | Relations for MVCPROCESS/*ALL, Total Relations: 22 | | | | | |
| 2 | | | | | | |
| 3 | Rel No. | Dependent File | Relation Type | Parent File | Dependent Fields | Parent Fields |
| 4 | 1 | Transaction Histo | ACCESSES | Contract Detail | Contract, Product | Contract, Product |
| 5 | 2 | Transaction Histo | ACCESSES | Contract Header | Contract | Contract |
| 6 | 3 | Transaction Histo | ACCESSES | Customer Groups | DGrp | CusGrp |
| 7 | 4 | Transaction Histo | ACCESSES | Purchases | Debtor | Customer |
| 8 | 5 | Transaction Histo | ACCESSES | Salespersons | Rep | Person |
| 9 | 6 | Transaction Histo | ACCESSES | Stock Balances | Product, Store | Product, Store |
| 10 | 7 | Transaction Histo | ACCESSES | Product Master | Product | Product |
| 11 | 8 | Transaction Histo | ACCESSES | Store Master | Store | Store |
| 12 | 9 | Contract Detail | OWNED BY | Contract Header | Contract | Contract |
| 13 | 10 | Contract Detail | ACCESSES | Stock Balances | Product, Store | Product, Store |
| 14 | 11 | Contract Detail | ACCESSES | Product Master | Product | Product |
| 15 | 12 | Contract Detail | ACCESSES | Store Master | Store | Store |
| 16 | 13 | Contract Detail | REFERS TO | Transaction type | Trn Hst Trn Type | Transaction type |
| 17 | 14 | Purchases | ACCESSES | Sites | Prospect No | Cus. No. |
| 18 | 15 | Purchases | ACCESSES | Customer Groups | CusGrp | CusGrp |
| 19 | 16 | Purchases | ACCESSES | Distributors | Distributor | Code |

FIGURE 11

An explicitly defined application data model described in DDL can be imported directly into persistence frameworks such as Hibernate and nHibernate. These open source Object Relational Maps allow Java™ and C#™ developers to access DB2/400 databases without needing to know the architecture of the database or JDBC or ODBC technologies, thus greatly simplifying their work. DDL can also be imported by all popular application modeling such as Rational Modeler, Borland Together and Eclipse. Microsoft Visual Studio also allows the import of DDL for building Data Projects.

Over many years of application use, enhancement, upgrades and fixes, it is only natural that the referential integrity will suffer. This is not true of systems that implement RI in the database itself but as mentioned, very little if any IBM i applications use this facility. With an explicitly defined data model of an IBM i application database, database records can be tested for referential integrity programmatically producing a report of orphaned records as an output. Simply explained the program starts at the bottom level of the hierarchical data model, in other words those files/tables that have no children/dependents, and looks for corresponding records in owning files/tables using the foreign keys provided by the data model. It carries out this test for all files/tables in the application one after the other.

AUTOMATED TEST DATA EXTRACTION

The need for accurate and representative test data is a requirement as old as application development. Most companies use copied production data to fulfill this need. There are a few problems associated with this such as keeping test data current, length of time to copy production data, disk space requirements, length of testing over complete data sets versus limited data. Another well used method for creating test data is to use simple copy statements in the OS for the required files only. This approach works fine but is still quite labor intensive and can be prone to error. An increasing popular approach is to select specific master or control records from the production database, and then write a program to copy the related records from the other files, using the foreign keys provided by the explicitly defined data model. This produces small and current test data quickly and with guaranteed referential integrity. It is often used by ISV's and support organizations to assist with customer testing of changes and enhancements of base systems.

BUILDING BI APPLICATIONS OR DATA WAREHOUSES

There has been a big push over the last couple of years in the area of Business Intelligence. By using the data in the application database more and more effectively companies expect to attain and sustain a real competitive edge. The technologies available to facilitate this are not very new or even that complicated for the most part. They all have a fundamental requirement for use: that the application database design be defined or described to them in detail. This is not in and of itself a problem, but when you consider that virtually no IBM legacy applications have an explicitly described relational data model design, it can become one for IBM i users. With access to an explicitly defined data model of the legacy application, these tools can be used much more productively and help provide increased and ongoing ROI from the legacy application, even with relatively small development and support teams.

A good example of this is with IBM's DB2 WebQuery product. A great tool and natural successor to IBM's Query/400 product, it comes with many power BI type features. To really get the full benefit of all of this rich functionality in DB2 WebQuery, one needs to populate the meta-data repository with the DB2 application database design including the foreign key information. The explicitly defined database supplies this information and can be used to create an entire met-data layer in DB2 WebQuery. Figure 12 shows an example of the source and model views of the meta-data of IBM's DB2 WebQuery. The model view shows how a file is joined to the other files in the database. This meta-data was created automatically from an explicit data model derived from an IBM i DB2 database

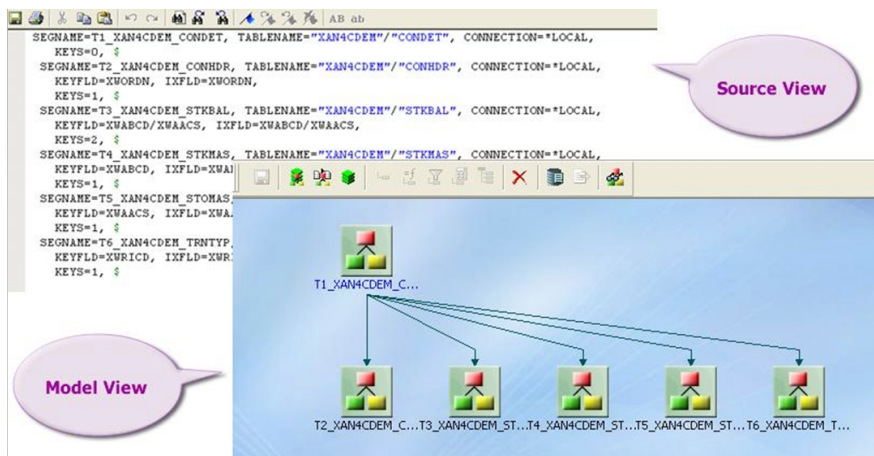


FIGURE 12

REDUCING RISK AND MAXIMIZING ROI

IT departments throughout the world are struggling to balance day to day support with a backlog of new user requirements often operating under severe headcount and cost restrictions. Compliance and regulatory pressures have increased over the last ten years or

so making large 'build it from scratch' type projects too risky to contemplate. From this unpromising start-point it is possible to make headway against the storm of conflicting demands by making use of the proven business processes contained in your existing systems.

If we can extract business logic and data model then a world of new possibilities open up and a layer of risk and uncertainty around potential projects is reduced - as you have access to what your systems really do, as opposed to what people think they do or the outdated documentation says they do.

If this business logic and data model extraction processes can be automated, it follows that many project types become feasible with limited resources. These can range from quick solutions such as making an order enquiry process available as a web service so customers can integrate into their own processes. Recovered business rules can be recycled and reused in Business Rule and Process Management systems like JBoss Drools or workflow systems. Through smaller scale developments like using the business rules and processes of a CRM system to build a new Java based web application - secure in the knowledge that the processes contained in it are proven, reducing the risk and the time taken to develop accordingly. Finally this process translates to large scale reuse precisely because it can be taken step by step in manageable chunks and delivers working prototypes quickly from the recovered business rules and model - there is no need for development to be done in a vacuum starved of the oxygen of user feedback. The next article in the series covers design recover and application renewal using extracted logic in more detail.

Application development and modernization is all about designs - not pure code philosophy.

Finally, from a compliance or audit point of view, compare the confidence levels of two IT Directors one who has used an automated business rule and data model recovery system to reuse and rebuild systems, with one who has used a conventional approach of manually designing and building new functionality to match the functionality of an existing system by looking solely at user requirements documentation and source code of the existing systems.

One has an auditable and repeatable process to recover rules and processes and build new systems, the other is solely dependent on the skill and experience of the people building his new system and their interpretation of the existing systems scope and eccentricities.