Databorough

Recovering Business Rules and Data Models

Steve Kilner

WHITE PAPER

Table of Contents

Executive Summary

IT managers who are responsible for legacy systems are both stewards and trustees of highly valuable corporate assets. In the IBM i world these assets typically have a sunk cost, and a replacement cost, of millions, or tens of millions of dollars. Accordingly:

- IT Managers are responsible for ***protecting the value of the legacy asset***
- IT Managers are responsible for ***delivering a good return on the legacy asset***

The challenges to these responsibilities are well known:

- **Growing complexity** of the applications over the years, making them slow, costly and risky to maintain
- **Disappearing knowledge** of the features and rules of the application and how they are implemented
- **Overall manageability** of the application becomes more and more difficult while the demands only increase

Managers who do not address these challenges directly and aggressively create ***latent, growing risks*** for their organization, and thus fall short in their roles as stewards and trustees:

- **The ability to respond quickly and cost-effectively** to business needs gradually degrades until an event triggers an enterprise-wide crisis
- **The complexity of the system** increases the probability of production defects that have a material impact on enterprise customer relationships or financial results
- **The costs of maintaining** the legacy system continually increase relative to the value delivered, thereby depreciating the value of the legacy asset and crowding out other IT opportunities

Legacy applications contain not only huge libraries of source code, they also contain a vast amount of **valuable enterprise knowledge**.

It is through ***re-engineering and design recovery*** that IT managers can successfully **overcome the challenges of complexity and loss of knowledge**. By doing so they deliver greater IT responsiveness, higher quality system resources and increasing value of the legacy asset. They also keep the enterprise **well-positioned for the eventual replacement** of the legacy application.

This paper discusses the concepts of re-engineering and design recovery, how to fit them into the management of legacy applications, and how to build legacy and future success using **Databorough's *X-Analysis Application Discovery* product.**

What is Design Recovery?

Reverse engineering, re-engineering, design recovery, refactoring - what exactly is the difference between all these terms?

It helps to step back and take a look at the big picture. In their paper, *Reverse Engineering and Design Recovery: A Taxonomy,* authors Elliot Chikofsky and James Cross make the case for some clear distinctions between these terms, starting at the beginning:

Forward engineering - the process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.

Reverse engineering - the process of analyzing a subject (physical) system to identify its components and their interrelationships, and create representations of the system in another form or higher level of abstraction. In short, the reverse of forward engineering. It is not to create a new system, but rather to describe an existing system in another form or at a level higher than the source code.

Design recovery - a subset of reverse engineering in which domain knowledge, external information, other documents or reasoning are added to the output of reverse engineering to identify more meaningful higher level abstractions beyond those obtained from direct examination of the system.

Whereas reverse engineering may reveal internal structures, components and elements of the system, design recovery may additionally reveal intentions, goals and actual architectural concepts and designs.

Refactoring - the process of restructuring a system or one of its components to improve its internal "appearance", in a broad sense. By definition it does not change any functionality, typically it changes its maintainability. Examples of this are replacing go-to statements with structured operations, reducing the size or complexity of components, or normalizing database designs.

Reengineering - the process of examining a system followed by refactoring it, i.e., reverse engineering followed by new forward engineering. The refactoring and forward engineering may be to improve the existing system or create a new system or both. Development of new functionality is often added into the reengineering process, but is not, strictly speaking, reengineering.

In this paper we will assume that reverse engineering is being done against legacy IBM i systems, and that there is likely also some design recovery being done, in pursuit of a variety of possible purposes.

Why Reverse Engineer or Recover the Design of an Application?

The same paper mentioned in the previous section outlines six reasons for reverse engineering. It is interesting to note that this paper, written about legacy systems, is at this point a legacy paper, having been written in 1990. It is nonetheless considered the original and authoritative statement on the subject, demonstrating that many historical software problems are very persistent.

Six Conceptual Reasons

The **six reasons for reverse engineering** given in the paper are fairly general, but useful to understand, and are recognizable to most experienced IT managers. After reviewing them we will look at more specific uses of reengineering. The six general reasons:

Cope with complexity - it is no secret that systems become more complex over time and reverse engineering can shed light on that complexity and lead to its mitigation.

Generate alternate views - creating and maintaining documentation is costly and anything that can provide other insight into the system can be useful for maintaining or reengineering it.

Recover lost information - systems often evolve to the point where there is neither knowledge nor documentation of exactly what they do. Reverse engineering and design recovery are processes to rebuild that knowledge.

Detect side effects - both defects in initial development and in subsequent modifications may remain undetected indefinitely. Reverse engineering can help discover those defects before they impact production usage of the system.

Facilitate reuse - widespread, thoroughly institutionalized reuse of software components is perhaps a dream that has never been realized to the hoped-for degree. Reverse engineering tools, if high quality and well-used, can greatly assist in organizing software reuse.

Synthesize higher abstractions - this was a reason described in the paper being discussed, but it's not yet come into existence as envisioned. It's interesting that in 1990 software experts were expecting that specialized expert systems for reverse engineering would be able to automatically deduce and express concepts and abstractions about systems. If there are such expert systems, they are not in widespread use as of 2010.

Twenty Three Practical Reasons

While the foregoing is an interesting set of conceptual reasons for reverse engineering, let's take a look at some fairly specific examples that may be closer to home for many legacy IBM i system owners.

But first, before looking at a long list, let's define some categories. Most of these practical uses fall into one or more categories:

- Management uses
- Design uses
- Maintenance uses
- Knowledge transfer or communication uses

The 23 practical uses:

1. Break down projects for enhancement or rewriting into smaller, more manageable, less risky pieces
2. Identify application and sub-application boundaries for interfacing to other systems
3. Assist with developing coherent test databases for enhancement or reengineering projects
4. Identify changes needed for modernization of the database design
5. Analyze data availability and structure for building Reporting/BI/Data Warehousing applications
6. Measure the referential integrity of the database
7. Assist with designing archival databases
8. Map business processes as UML through analysis of the application source code
9. Harvest model-view-controller design components from legacy applications for reengineering
10. Centralize the logic, i.e., business rules, in an application
11. Populate a rules engine system as part of refactoring or reengineering efforts
12. Inventory and document functionality in existing applications for comparison to functionality in packaged products being considered for purchase
13. Develop test cases for regression testing as part of enhancement, refactoring or reengineering efforts
14. Audit critical business rules for regulatory or quality control purposes
15. Create system documentation specifically for use by business analysts
16. Add information and insight to the planning process of projects for enhancement or reengineering projects
17. Feed into automated modernization tools, such as Databorough's Reengineering suite

18. Inform the design of projects for enhancement or reengineering projects
19. Inform the design of projects for interfacing or SOA "wrapping"
20. Facilitate discussions with users for developing new requirements
21. Document the system for disaster planning, or audit and due diligence purposes
22. Facilitate reuse of existing functionality
23. Facilitate application support and enhancement design and programming tasks

No doubt there are more uses, but this list should make it clear that reverse engineering and design recovery can play a significant role in improving IT processes.

What can be Recovered that's Useful?

In the book, *Legacy Systems: Transformation Strategies*, by William Ulrich, the author makes the case for four critical tools and their information output that aid reverse engineering and reengineering projects:

- **Legacy Componentization -** attempts to bring more modularity to legacy procedural code; ideally this produces a hierarchy of systems, subsystems and components
- **Data Reverse Engineering -** captures physical and logical data definitions; typically such a tool goes beyond merely looking at data definition source code and may also inspect program code and data content to dig deep into the data model structure
- **Business Rule Capture -** identifies and slices logic paths to extract business logic; typically an inventory is built of business rules which are traceable back to their source code implementation
- **Transaction Flow Analysis -** captures information and produces documentation that describe system flows; typically in UML Action or Sequence diagrams

Databorough's *X-Analysis Application Discovery* product provides all of the functionality mentioned above using the following terminology, and adds one more useful function:

- Subdividing the software into application areas and sub-application areas
- Recovering the data model
- Recovering the business rules
- Recovering business processes
- Recovering the user interface

In the remainder of this paper we will drill down further and further into what X-Analysis provides and how it facilitates this type of work. Let's first look at each of these areas in a little more detail.

Subdividing the Software into Application Areas

In one of the authoritative books on working with legacy code, *Working Effectively With Legacy Code*, author Michael Feathers describes what he calls "The Legacy Code Algorithm":

- Identify change points
- Find test points
- Break dependencies
- Write tests
- Make changes and refactor

While this process is intended for maintaining legacy code it is conceptually just as appropriate for any sort of reverse engineering or reengineering effort. What is interesting to note is that that three of the steps of this Legacy Code Algorithm relate to *finding critical locations of boundaries in the code*. As any experienced legacy developer knows, it can be extremely difficult, and seemingly impossible to separate a legacy application into sub-applications.

And why is this important? As we saw in the previous section this can be important for such activities as breaking up large projects into smaller ones, creating interfaces into subsystems, devising manageable test plans, designing SOA-type wrappers, etc.

Recovering the Data Model

Throughout its history the IBM i has always supported a relational database, but it has never provided a complete set of data modeling facilities. A good data modeling facility provides information to application managers and designers at four levels:

- Data definition - the physical files and their fields, plus logical files, or views
- Data relationships - the foreign key relationships that exist between files
- Data integrity - the actual use of the data definitions and data relationships in practice and reality
- Application usage - where and how the data definitions and relationships are used throughout the application software

Recovering and understanding the complete data model is useful for a number of activities, such as modernizing the database, interfacing to other systems, developing test data and so on.

Recovering the Business Rules

Some estimates put business rules as comprising around 30% of the content of legacy systems that are coded with procedural languages, such as RPG. The flip side of the coin is that those business rules comprise perhaps 80% of the value of such legacy systems. Unfortunately, with legacy procedural systems the business rules are tangled in with screen management, control flow and database access code. Extracting the core value of a legacy system is a major challenge.

The essential tasks involved with recovering business rules are:

- Identifying and scoping business rules in the source code
- Correlating instances of the same business rule coded in different areas and recognizing them as being the same rule
- Building an indexed repository of the business rules
- Cross referencing the repository business rules back to the source code

Recovering the business rules and building a repository of them assists with such activities as centralizing key logic, aiding the design of reengineered systems, facilitate code reuse. Etc.

Recovering Business Processes

Business process recovery and modeling take place on a couple of levels: 1) identifying what business processes are encoded in a system, and 2) identifying what comprises a given business process. Creating a list of business processes is more relevant to the first of the four sections, *Sub-dividing the Application*. Here we are more concerned with describing what goes on inside of business processes.

Such descriptions are often represented as UML diagrams, such as Activity Diagrams or Use Cases Diagrams. The information obtained from this modeling typically includes:

- Descriptions of user-driven or system-driven activities
- Descriptions of the flow of control between these activities
- Correlation of these activities to the system implementation, such as particular screens or business rules

Knowledge of the business processes in a system are useful for helping analysts and programmers define enhancements to the system, for communicating with users in terms more understandable to them, and for providing a basis for reengineering projects.

How can the Recovered Design be Used?

Formatting and Delivery Options

Most design recovery data in Databorough's *X-Analysis Application Discovery* product can be directly exported from the user interface as:

- PDF files
- Excel spreadsheets

Design recovery data that relate to UML specifications is also available in XML-formatted UML files.

Additionally, much of the data relating to application areas, business rules and the data model is available in IBM i tables which are accessible to X-Analysis users. See the *X-Analysis Technical Guide* for more information.

Feeding Processes

Design recovery data can be used to facilitate the population and development of reengineering designs in several ways:

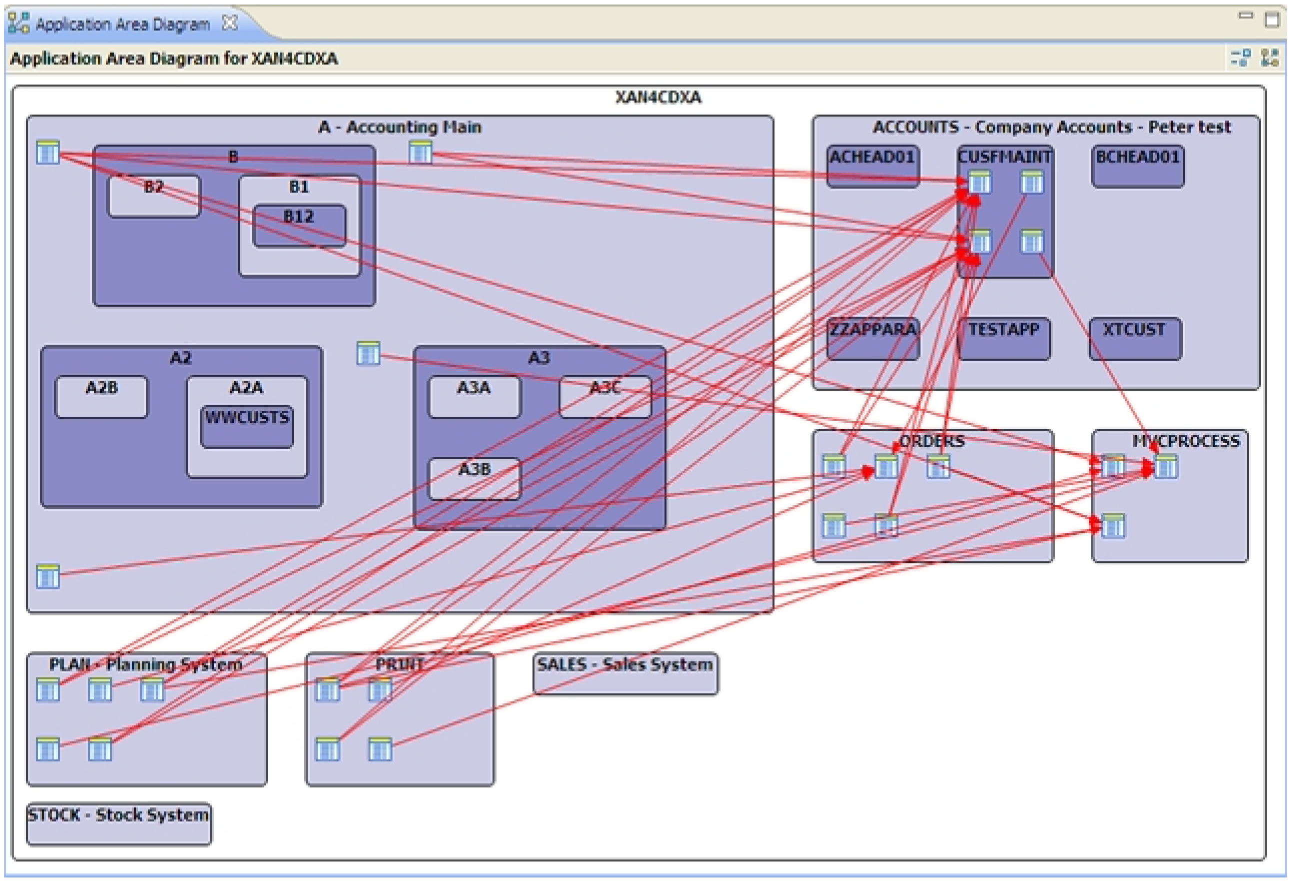
- The recovered IBM i data model can directly feed the specifications of a new data model using standard DB/2 SQL.
- The recovered business rules can be converted and populated into a business rules management system such as IBM's iLog or JBoss BRMS
- The recovered UML specifications can be fed into any UML-based tool such as IBM Rational Rose, Eclipse, Microsoft Visio and so on.

A Closer Look at Reengineering with X-Analysis

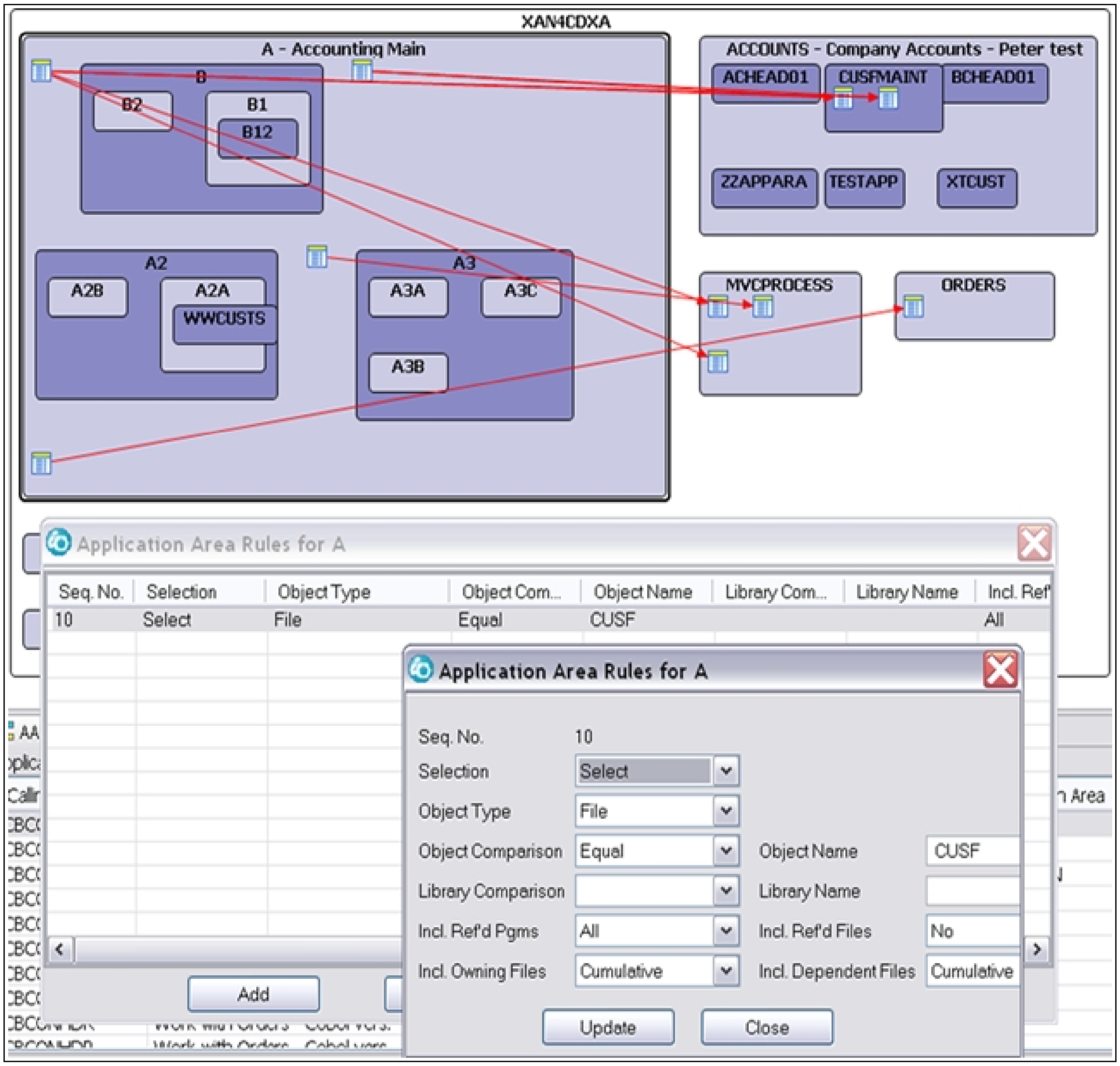
Subdividing an Applications into Sub-Applications

Entire legacy applications are often too large to effectively comprehend or implement wholesale changes. For this reason it is often necessary or helpful to sub-divide a system into application areas. The reasons and specifications for these may change with time. X-Analysis provides facilities for subdividing an application area into groups of objects that meet user defined selection criteria. These criteria might be based on functions or even generic names. X-Analysis then uses its internal sophisticated cross-reference information and Data Model relationships to include, automatically all related elements such as programs, displays, or files in the application area.

Application area filters can then be used through the X-Analysis Solution Sets to view, document or reengineer as opposed to individual objects.



The Application Area diagram in X-Analysis is interactive and by clicking on different parts of your system you can see the relationships between either all parts or just the area you've clicked on and the areas it relates to. You can see in the following image the options that went into defining a sample application area, here, centered around one file, CUSF.



Application Area Rules for A

Seq. No.	Selection	Object Type	Object Com...	Object Name	Library Com...	Library Name	Incl. Ref'
10	Select	File	Equal	CUSF			All

Application Area Rules for A

Seq. No.	10		
Selection	Select		
Object Type	File		
Object Comparison	Equal	Object Name	CUSF
Library Comparison		Library Name	
Incl. Ref'd Pgms	All	Incl. Ref'd Files	No
Incl. Owning Files	Cumulative	Incl. Dependent Files	Cumulative

Update Close

Recovering Data Model

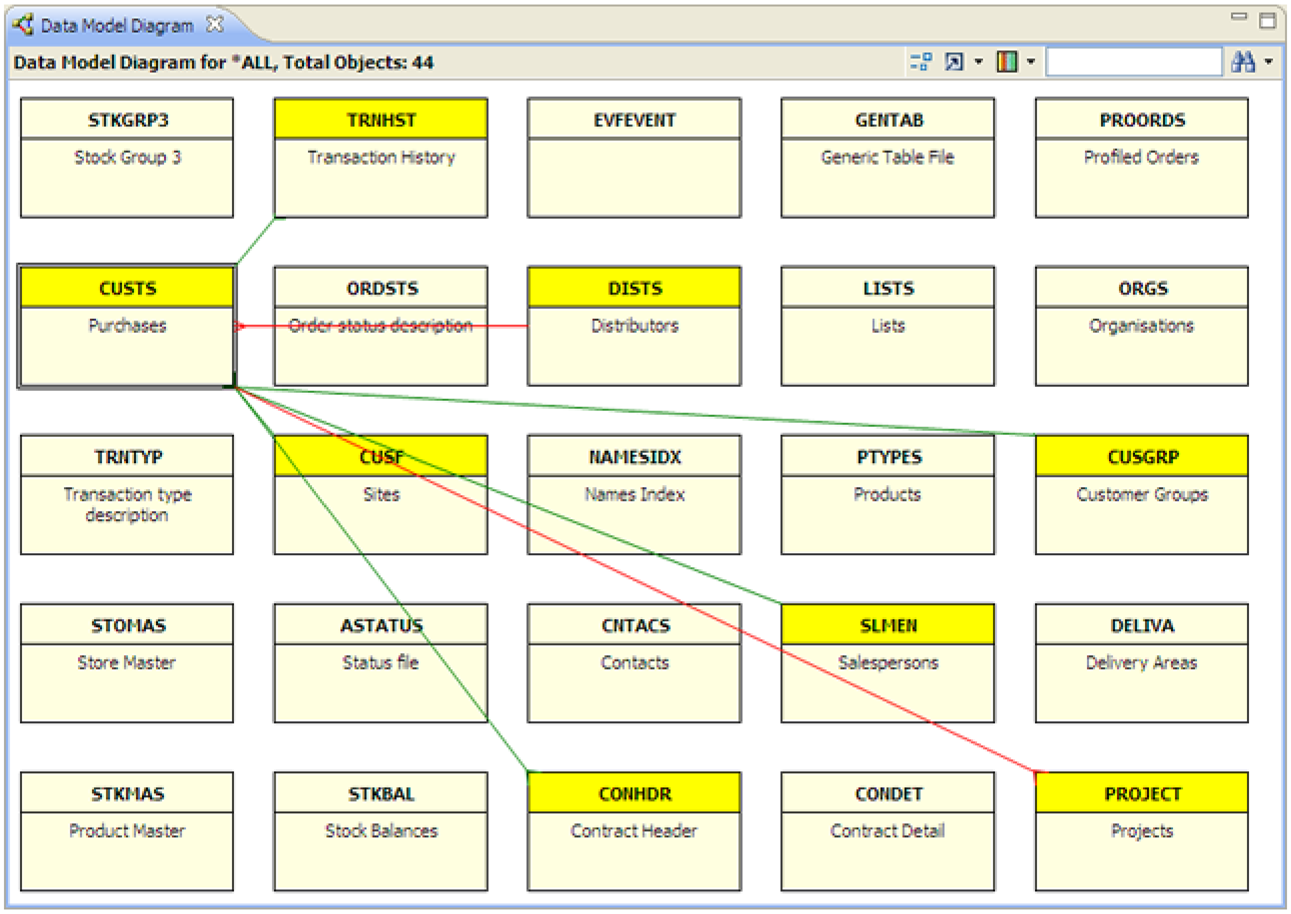
The relational database model of an enterprise application is an extremely powerful piece of information and potentially valuable asset to the organization. Unlike 2E systems, for almost all RPG or COBOL applications running on System i, there is no explicit data model or schema defined. By the term model, we are referring to the foreign key or relational model, not just the physical model of the database. The relational model or architecture of the database can be reused in a number of scenarios including:

- Understanding application architecture
- Data quality analysis - referential integrity testing
- Automated test data extraction, masking and aging
- Building BI applications or Data warehouses

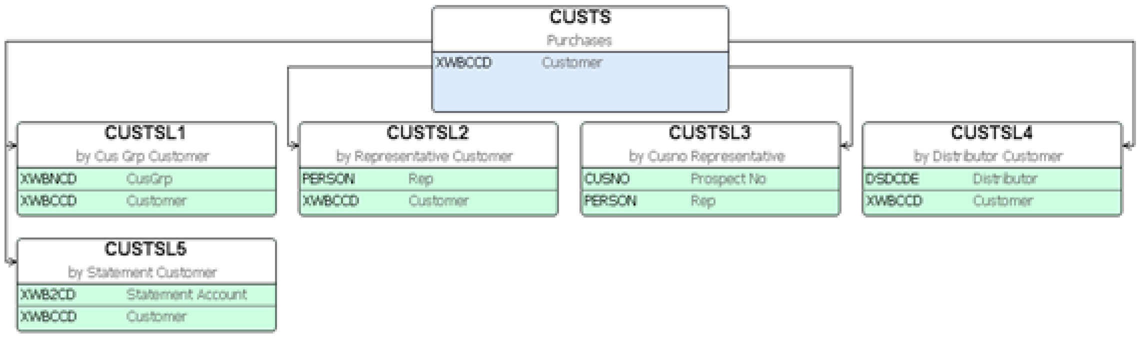
X-Analysis has the unique capability of automatically deriving the explicit system data model from a legacy RPG, COBOL or 2E application. Let us have a look at this and the model reuse capability in a bit more detail.

Deriving the Legacy Data Model - X-Analysis accomplishes this by analyzing the data structures of the physical and logical files, but it then programmatically traces these through all programs that use them to verify the existence of any cross-file relationships or foreign keys. These derived relationships can also be verified by the product by performing an integrity check on the actual data. This ensures that the data of the dependent file makes a reference, to data records from the owning file. In this way, the automated reverse engineering can fully extract the data model from even the most complex legacy system.

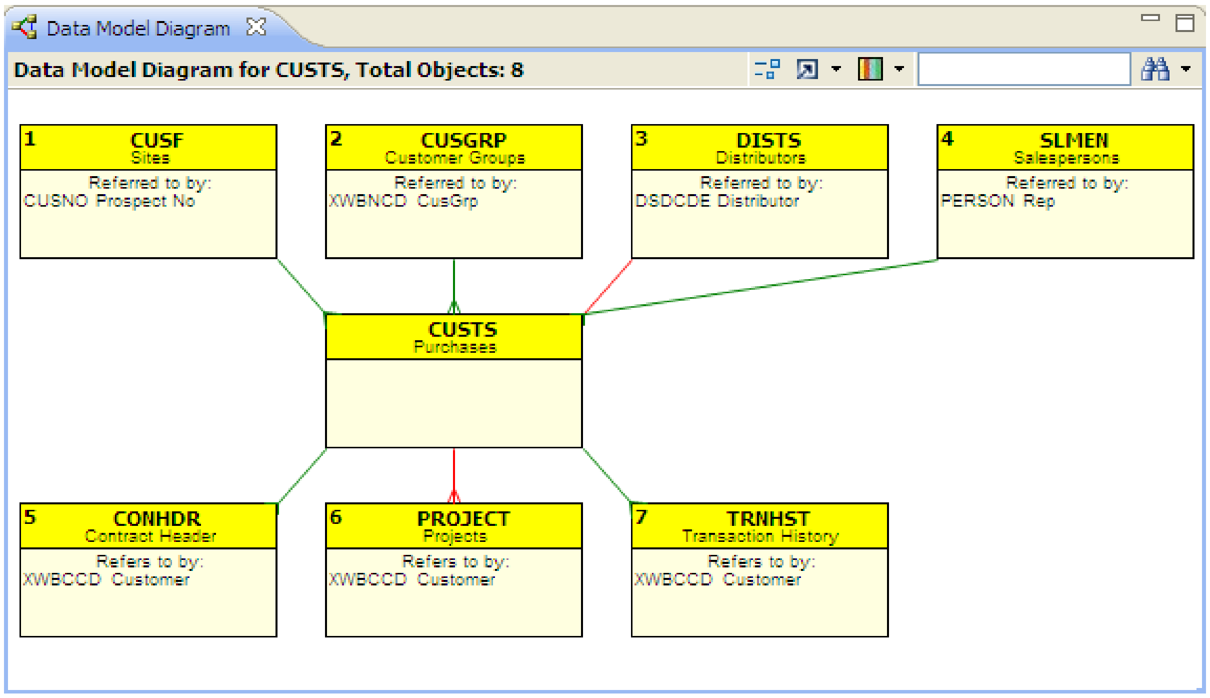
In the following image we see one section of the data model diagram, and in this case, the CUSTS file has been clicked on, showing all the files that have parent, child or foreign key relationships with it.



Taking this a step further for the CUSTS file we use the context menu to view its access paths:



With another click we can drill down into the foreign key relationships:



Test Data for Modernization & Maintenance Projects - Creating and managing test data can be a labor-intensive and costly task. As a result of this, many companies resort to creating copies of entire production systems. This approach in itself can produce its own set of problems, such as excessive storage demands, longer test cycles, and often a lack of current data for testing. The relational data model is used to automatically extract records related to those specifically selected for testing. In this way smaller, accurate test subsets can be extracted quickly and respectively, with additional functionality for scrambling sensitive production data and aging the dates in the database forwards or backwards after testing.

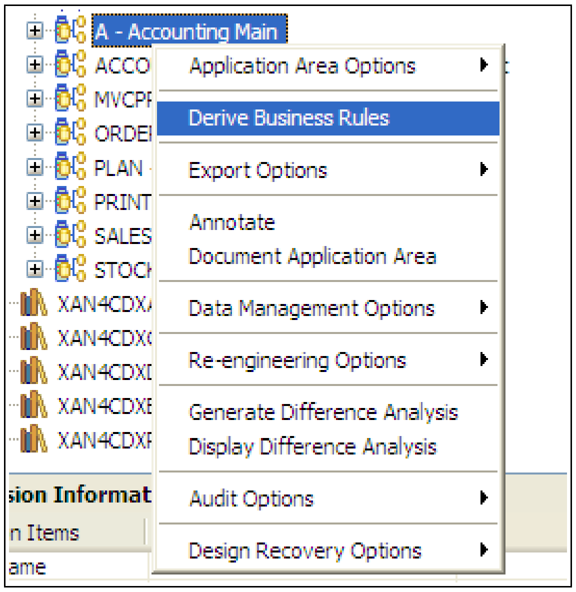
Recovering Business Rule Logic

Traditionally business analysts find the business rules for a new application by organizing workshops and interviews and then manually writing use cases to describe the rules as text. However, for a legacy application all the rules are already fully prescribed in the application code - you just have be able to retrieve them.

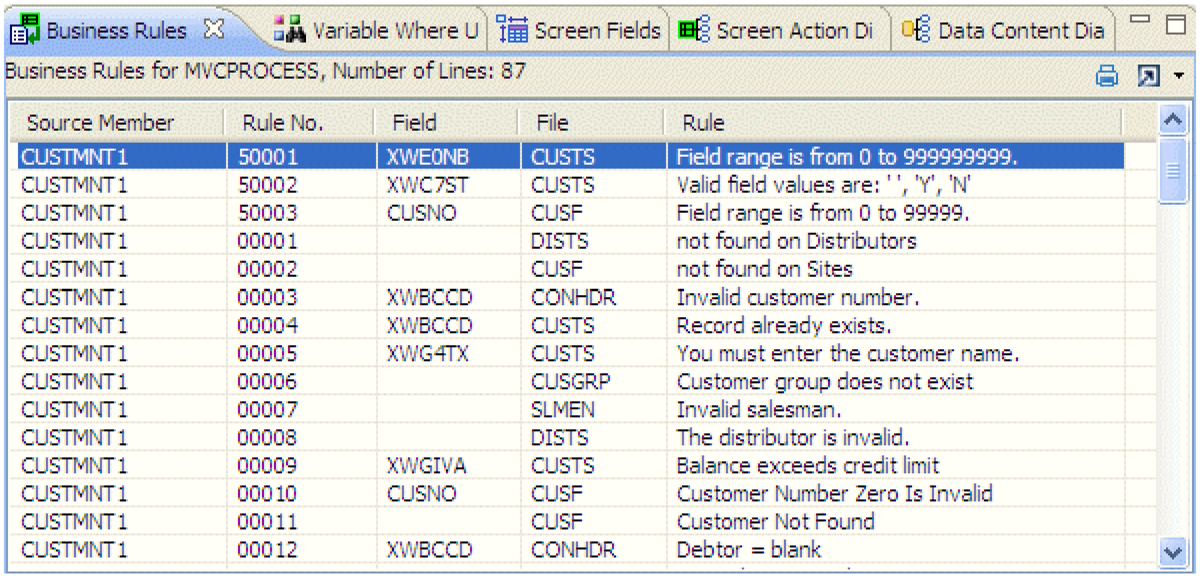
The challenge is that in the vast majority of legacy RPG and COBOL programs, the business rule logic is mixed in with screen handling, database I/O, and flow control. Harvesting the business rules from legacy applications therefore requires knowledge of the application and the language used to implement it, plus a lot of time to identify the rules, define their full scope, document them and organize them. Additionally, these rules need to be narrated and indexed, thus providing critical information for any analysts, architect or developer charged with enhancing or rebuilding a legacy application. The task of harvesting business rules is therefore a highly skilled, labor-intensive, and costly exercise for any organization.

X-Analysis accomplishes this task by automatically scanning RPG and COBOL programs and 2E models programmatically. It then separates out rule code from the body of the application and identifies, indexes, narrates, and stores business rule logic code into a structured, usable repository. In the final part of the process, it supplies appropriate text narratives to describe these harvested rules.

For a given application area, or sub-application, it's this simple:



Once the rules are derived they can be viewed in summary form:

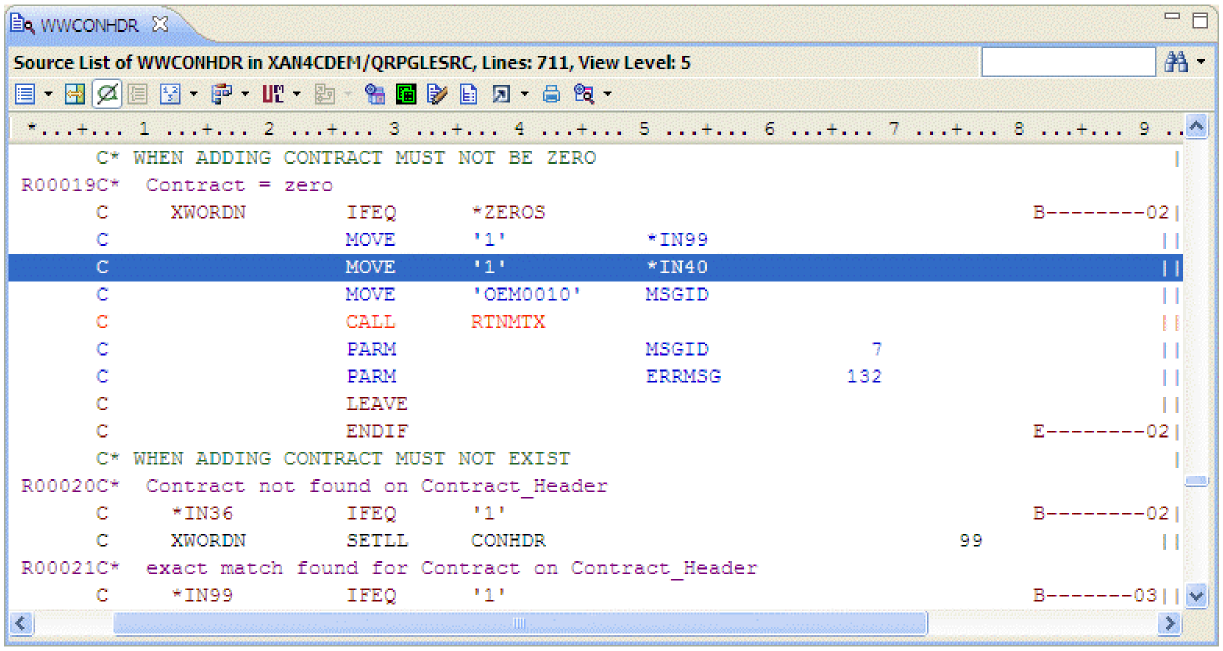


Business Rules for MVCPROCESS, Number of Lines: 87

Source Member	Rule No.	Field	File	Rule
CUSTMNT1	50001	XWE0NB	CUSTS	Field range is from 0 to 999999999.
CUSTMNT1	50002	XWC7ST	CUSTS	Valid field values are: ' ', 'Y', 'N'
CUSTMNT1	50003	CUSNO	CUSF	Field range is from 0 to 99999.
CUSTMNT1	00001		DISTS	not found on Distributors
CUSTMNT1	00002		CUSF	not found on Sites
CUSTMNT1	00003	XWBCCD	CONHDR	Invalid customer number.
CUSTMNT1	00004	XWBCCD	CUSTS	Record already exists.
CUSTMNT1	00005	XWG4TX	CUSTS	You must enter the customer name.
CUSTMNT1	00006		CUSGRP	Customer group does not exist
CUSTMNT1	00007		SLMEN	Invalid salesman.
CUSTMNT1	00008		DISTS	The distributor is invalid.
CUSTMNT1	00009	XWGIVA	CUSTS	Balance exceeds credit limit
CUSTMNT1	00010	CUSNO	CUSF	Customer Number Zero Is Invalid
CUSTMNT1	00011		CUSF	Customer Not Found
CUSTMNT1	00012	XWBCCD	CONHDR	Debtor = blank

In this view the business rules can be viewed in English form, as shown, or as pseudocode.

Business rules can also be optionally viewed inline in the source code from which they are derived:



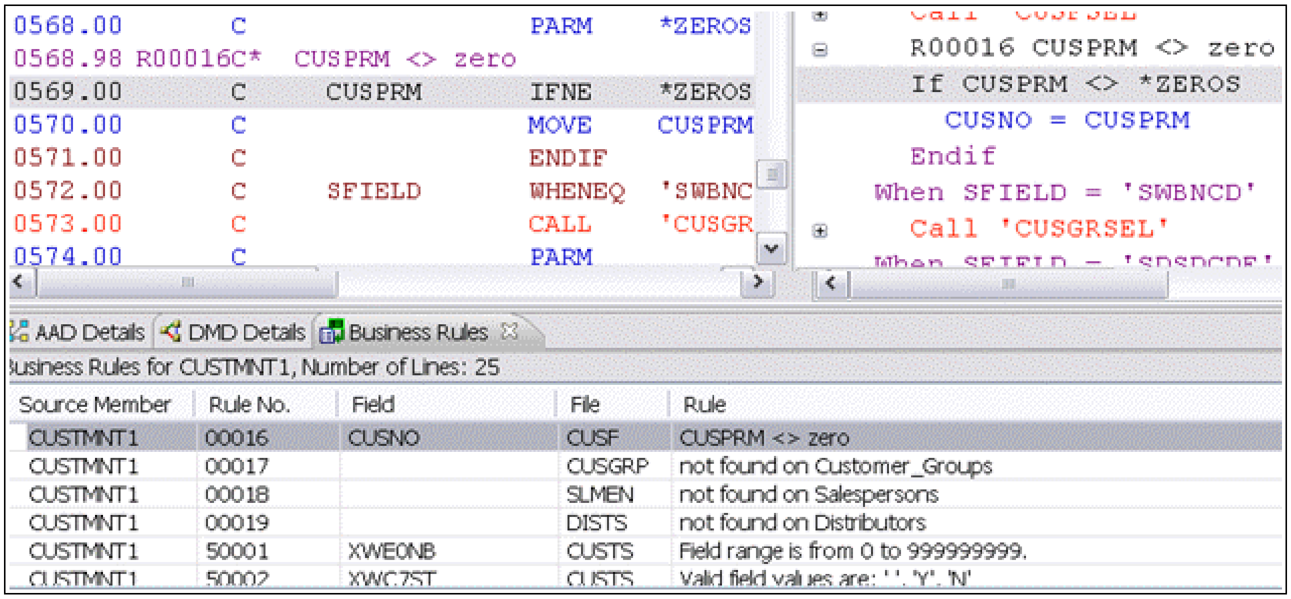
Source List of WWCONHDR in XAN4CDEM/QRPGLESRC, Lines: 711, View Level: 5

```
 *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ..
      C* WHEN ADDING CONTRACT MUST NOT BE ZERO
R00019C*   Contract = zero
      C      XWORDN        IFEQ      *ZEROS                                        B---------02
      C                    MOVE      '1'           *IN99
      C                    MOVE      '1'           *IN40
      C                    MOVE      'OEM0010'     MSGID
      C                    CALL      RTNMTX
      C                    PARM                    MSGID              7
      C                    PARM                    ERRMSG           132
      C                    LEAVE
      C                    ENDIF                                                   E---------02
      C* WHEN ADDING CONTRACT MUST NOT EXIST
R00020C*   Contract not found on Contract_Header
      C      *IN36         IFEQ      '1'                                           B---------02
      C      XWORDN        SETLL     CONHDR                               99
R00021C*   exact match found for Contract on Contract_Header
      C      *IN99         IFEQ      '1'                                           B-------03
```

Note that the description of the business rules is automatically generated it is not lifted from existing comments. All the lines in purple have been derived by X-Analysis in the example above. Also note that these Business rules comments are not added into to the RPG source but retained in the X-Analysis repository and integrated into the view on demand.

The business rule repository can then be used programmatically to generate new code, enhance the built-in documentation and work in combination with cross referencing and annotation capabilities to research the application. It may also be used by developers as the necessary input for specification development, whether for new applications or for modifications to the current system.

Once we have derived business rules we can now view the code in multiple ways: the normal source view, a pseudocode view, and a business rule repository view:



```
0568.00         C                   PARM      *ZEROS
0568.98 R00016C*   CUSPRM <> zero
0569.00         C     CUSPRM        IFNE      *ZEROS
0570.00         C                   MOVE      CUSPRM
0571.00         C                   ENDIF
0572.00         C     SFIELD        WHENEQ    'SWBNC
0573.00         C                   CALL      'CUSGR
0574.00         C                   PARM
```

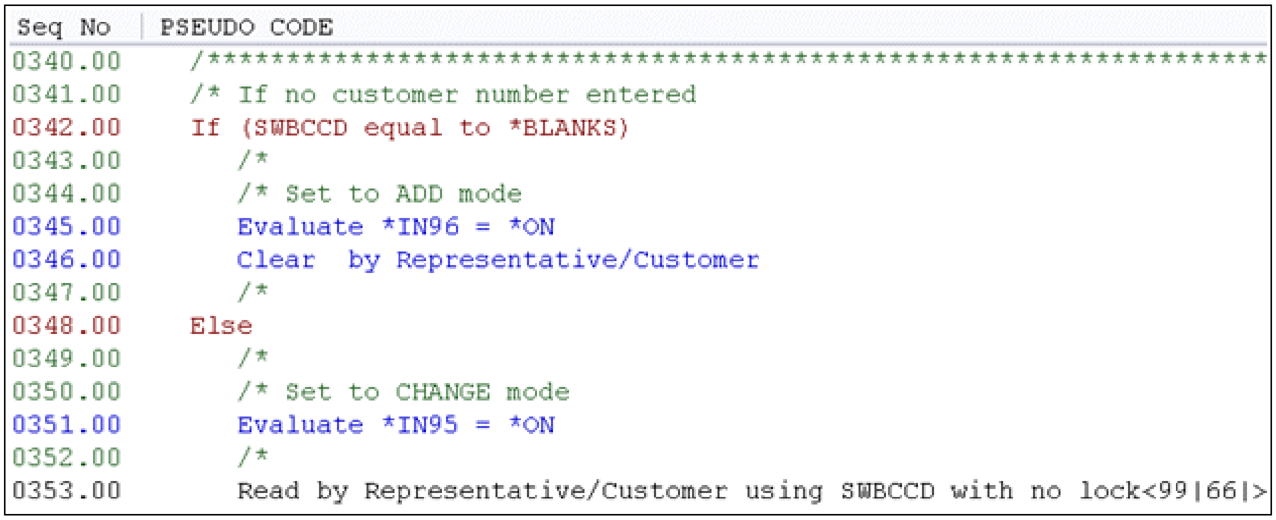
```
       R00016 CUSPRM <> zero
       If CUSPRM <> *ZEROS
          CUSNO = CUSPRM
       Endif
    When SFIELD = 'SWBNCD'
       Call 'CUSGRSEL'
```

AAD Details | DMD Details | Business Rules

usiness Rules for CUSTMNT1, Number of Lines: 25

Source Member	Rule No.	Field	File	Rule
CUSTMNT1	00016	CUSNO	CUSF	CUSPRM <> zero
CUSTMNT1	00017		CUSGRP	not found on Customer_Groups
CUSTMNT1	00018		SLMEN	not found on Salespersons
CUSTMNT1	00019		DISTS	not found on Distributors
CUSTMNT1	50001	XWEONB	CUSTS	Field range is from 0 to 999999999.
CUSTMNT1	50002	XWC7ST	CUSTS	Valid field values are: ' ', 'Y', 'N'

There is also a full panel view of pseudocode which can be used for reengineering or other analysis purposes.

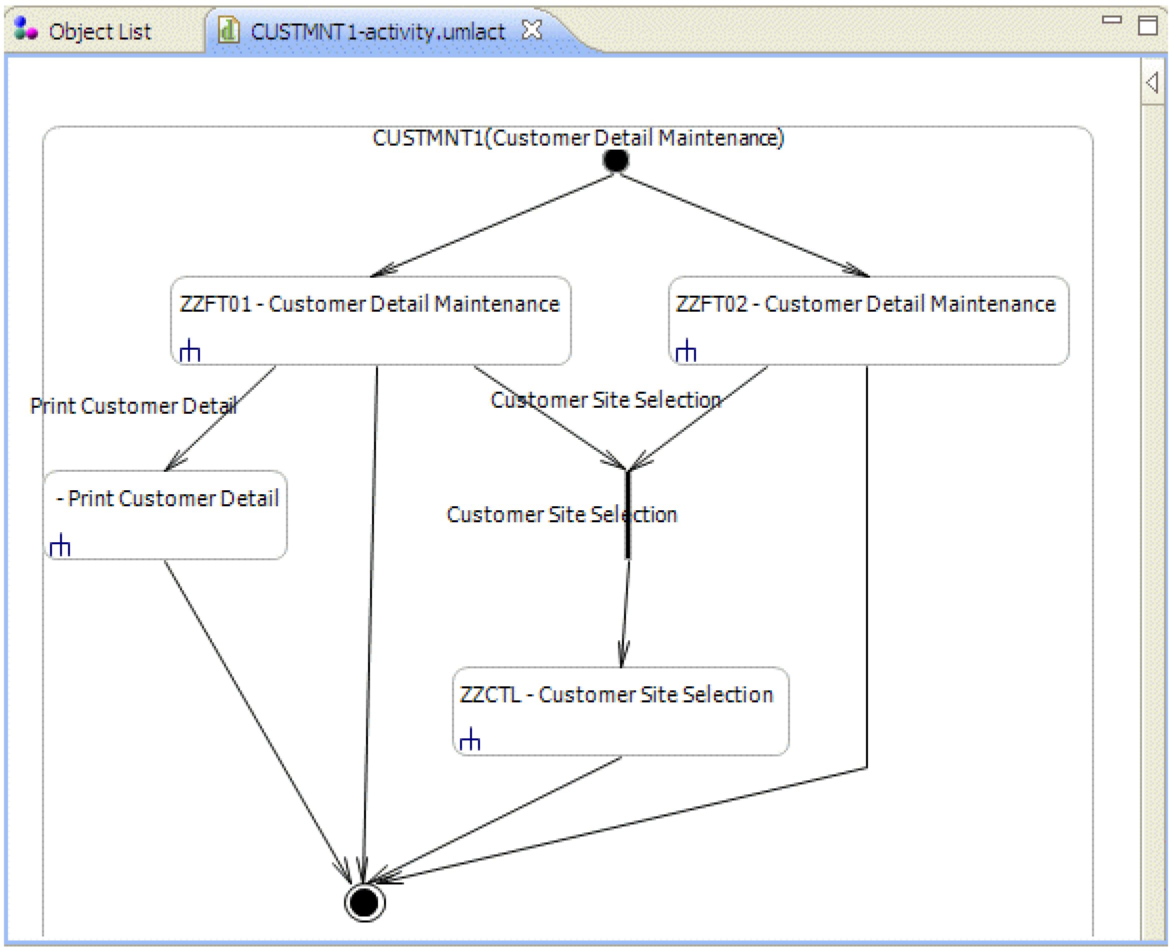


```
Seq No    PSEUDO CODE
0340.00     /********************************************************************
0341.00     /* If no customer number entered
0342.00     If (SWBCCD equal to *BLANKS)
0343.00        /*
0344.00        /* Set to ADD mode
0345.00        Evaluate *IN96 = *ON
0346.00        Clear  by Representative/Customer
0347.00        /*
0348.00     Else
0349.00        /*
0350.00        /* Set to CHANGE mode
0351.00        Evaluate *IN95 = *ON
0352.00        /*
0353.00        Read by Representative/Customer using SWBCCD with no lock<99|66|>
```

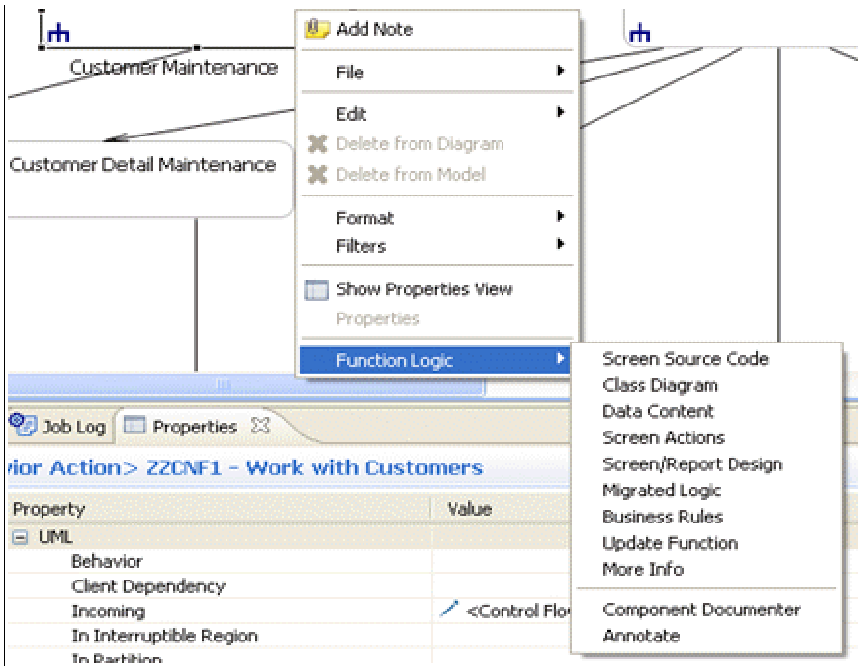
Recovering Business Processes

The objective of recovering business processes is to help sketch application designs and to make such sketches portable and reusable in other IDE's such as Rational, Borland, MyEclipse, etc. The three diagrams automatically generated by X-Analysis are:

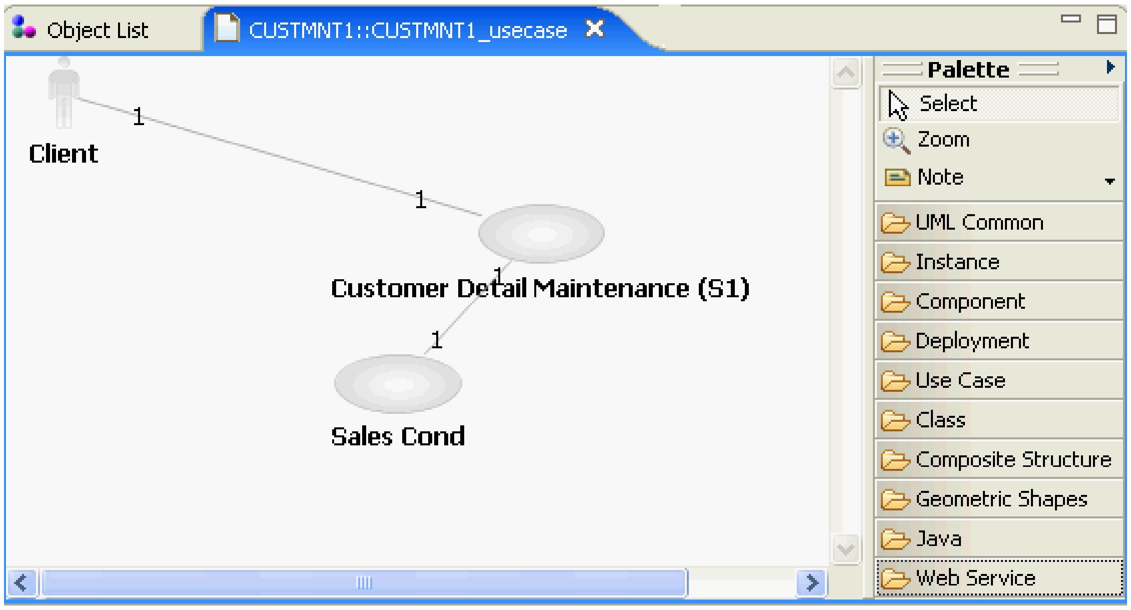
Activity Diagram - Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. X-Analysis produces these automatically either from a single program with multiple screens, or a group of programs. Each activity in the diagram represents a usable screen format in the RPG program. A user can also view the extracted Business Rules, relevant to that particular activity/format directly from within the diagram.


Object List
CUSTMNT1-activity.umlact
CUSTMNT1(Customer Detail Maintenance)
ZZFT01 - Customer Detail Maintenance
ZZFT02 - Customer Detail Maintenance
Print Customer Detail
Customer Site Selection
- Print Customer Detail
Customer Site Selection
ZZCTL - Customer Site Selection

Each activity in a diagram has a great deal of information behind it which can be drilled down into by right clicking:

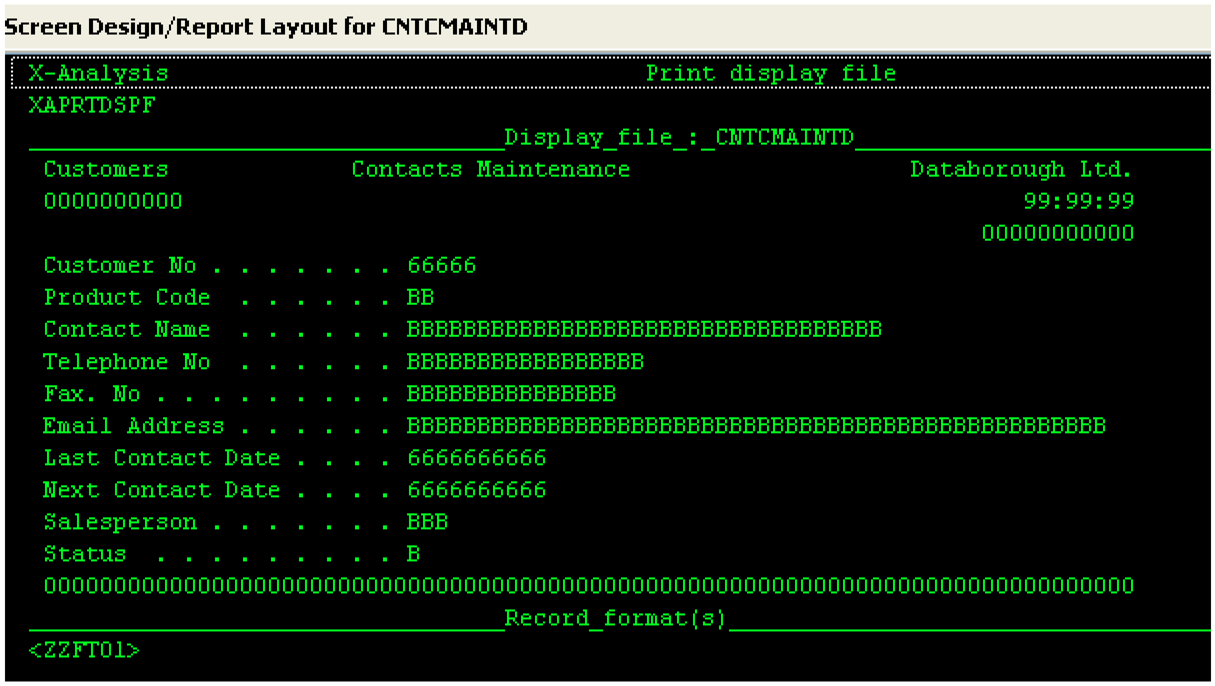


Use Case Diagram - Use Case Diagrams model the functionality of system using actors and use cases. Use cases are services or functions provided by the system to its users. Auto-generated from X-Analysis, this can be used as an alternative view to the Activity Diagram, and also has drill-down capabilities for viewing extracted Business Rules.



Recovering User Interface

The screens of a legacy application are a classic example where the design is useful in a modernization context, and the code is not. All modern IDE's provide powerful UI development tools. Modern UI standards and preferences for style and technology also vary from project to project. The sheer number of screens in a legacy application presents a logistical problem in recreating them manually, even with the cleverest developers and best tooling. X-Analysis lets you see what the legacy screen looked like without having to run the application which is a great time saver for people who haven't been involved with the original application:



Screen Design/Report Layout for CNTCMAINTD

```
X-Analysis                                          Print display file
XAPRTDSPF
______________________________________Display_file_:_CNTCMAINTD_____________________________
 Customers                Contacts Maintenance                         Databorough Ltd.
 0000000000                                                                    99:99:99
                                                                            00000000000
 Customer No . . . . . . . 66666
 Product Code  . . . . . . BB
 Contact Name  . . . . . . BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
 Telephone No  . . . . . . BBBBBBBBBBBBBBBBB
 Fax. No . . . . . . . . . BBBBBBBBBBBBBBB
 Email Address . . . . . . BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
 Last Contact Date . . . . 6666666666
 Next Contact Date . . . . 6666666666
 Salesperson . . . . . . . BBB
 Status  . . . . . . . . . B
 000000000000000000000000000000000000000000000000000000000000000000000000000000
______________________________________Record_format(s)______________________________________
<ZZFT01>
```

Screen designs of legacy applications are not just about look and feel, there are attributes, and logic embedded which from a design point of view is relevant, no matter what technology being used to implement them. These are:

Formats/Layouts - Some screens may benefit from amalgamation or redesign, but table edits, and non-transaction type screens will largely remain the same, if not identical in layout.
Actions - whether from sub-file options, command keys, or default enter actions, these often represent an important part of the usefulness of an application design. The mechanisms used to offer or invokes these calls may change, but where they go logically and what parameters they pass will largely remain consistent.

Fields/Files/Attributes - What fields are on what screens, and where the data comes from is a requirement in any system development. Attributes of a field can also help determine what type of controls might be used in a modern UI. For example, a Boolean type might be implemented with a check box, a date with a date prompt. Again, these are simple enough to edit in modern IDE's, but the volume associated with any large legacy application modernization can make this work prohibitive.

Data Model Mapping - Validations and prompting mechanisms that ensure referential integrity in legacy applications can also be vital to extract. This is both to implement referential integrity and to provide design information for building modern prompt or selection controls such as drop downs or lists.

Naturally, it will be desirable to redesign some UI's completely. For those programs and screens where this is not the case, the design, and mapping information can be used directly in the new version of the application, even though the UI code has been discarded.

X-Analysis extracts User Interface design information as described above and stores it as meta-data in the X-Analysis repository. This is can be used as reference documentation for rebuilding UI's manually, or for programmatically regenerating new View and Controller artifacts in the chosen new technology. X-Analysis also has reengineering features which can generate a JSF/Facelets UI. The design meta-data can also easily be used to generate new interfaces using any technology such as EGL, Ajax, RCP, C#, VB or even RPG.

Summary

Design recovery and reverse engineering provide great productivity benefits for numerous tasks relating to legacy applications:

- Better organizational understanding and communication of system capabilities
- Improved enhancement and support of the legacy system
- Improved processes for modernization, restructuring or reengineering
- Better management of all the above work activities

Databorough's *X-Analysis Application Discovery* product assists IT organizations with these tasks by providing proven, rich information recovery of all fundamental aspects of reverse engineering to aid in the overall design recovery process:

- Subdividing the software into application areas
- Recovering the data model
- Recovering the business rules
- Recovering business processes

"You cannot fully use, improve or replace what you have, until you know what you have."

Steve Kilner