



Databorough

# Business Logic Extraction with X-Analysis

- » *Recover Application Design*
- » *Derive Business Logic*

Richard Downey  
Stuart Milligan

WHITE PAPER

## Preface

**D**eveloping tools services for analyzing and re-engineering RPG applications for the last 20 years, has given Databorough a unique view of the very large and complex world of legacy applications running on System i, iSeries and AS/400. In 2005, IBM and Databorough published an IBM Red book “Modernizing and Improving the Maintainability of RPG Applications Using X-Analysis Version 5.6”. This paper expands on the recently released white paper 'Modernizing RPG/COBOL System i applications using X-Analysis 8' incorporating new concepts and methods for design recovery and re-factoring monolithic RPG applications into modern application architecture. Contact [info@databorough.com](mailto:info@databorough.com) for a copy of the white paper, the Red book and trial software.

## Table of Contents

Executive Summary.....	1
Introduction.....	2
Recovering an Application Design.....	3
Recovering the Data Model.....	5
Recovering the User Interface.....	5
Recovering Business Rule Logic.....	7
UML Diagramming.....	8
Design Recovery next steps.....	10
Summary.....	12

## Executive Summary

The knowledge and information contained in an organization's business software is vitally important and very valuable but often this information covering the operation, metrics, and design of the software is tantalizingly out of reach. Without this knowledge, maintenance and changes to the system are not as efficient or effective as they could be, and the risk of failure or problems increases exponentially the larger the enhancement required. This could lead to a paralysis where changes can't be made due to a lack of confidence in the outcome.

Accurate and current information about an entire system can greatly improve the productivity of your IT staff, and reduce maintenance costs by eliminating the need to research, catalog and assemble the information manually for each service request, or modernization project.

Existing System i<sup>1</sup> applications have some fairly consistent and distinct characteristics that mark them as costly and potentially high-risk:

1. Applications tend to be large and complex
2. Little or no documentation
3. Original Designers no longer available
4. They have been developed over many years
5. Monolithic Programming Model
6. Written in obsolete languages

Points 1 through 3 can largely be managed more effectively by investing in a product like X-Analysis to both recover the design of the application, and provide highly productive analysis tooling to compensate for the complexity of the application.

Inconsistent programming standards and designs, significant amounts of redundant and duplicated code, and an increasingly costly demand for globally diminishing legacy development skills, are the results of points 4, 5 and 6.

This paper will illustrate how X-Analysis carries out the Design Recovery process delivering deep insights into to your systems.

---

*<sup>1</sup> Note - for consistency throughout this document we use System i to refer to the family of computers that grew out of IBM's System/38 over the last twenty years namely the AS/400 , iSeries and latterly the System i and IBM i on Power. We have picked System i as it is the current name used by IBM though most readers (and the author) will likely use the earlier terms.*

## Introduction

The concept of reusing existing code or logic is not new. The challenge has always been to identify, isolate, and reuse only those designs that are relevant in the present context. The sheer volume of code, its complexity, and the general lack of resources to understand legacy languages, specifically RPG, represents a tragic potential waste of valuable business assets. In many cases, these expensive and well-established legacy designs have little chance of even having their relevance assessed, let alone being reused. The Design Recovery Toolset addresses this problem by isolating, indexing, and documenting those design elements that could be relevant in a modern version of the application being assessed.

The design elements of an application are valuable assets. Information about the legacy application designs, helps assess the nature and relevance of these assets without the need for expertise in RPG/COBOL or even the legacy application.

Whatever the business needs driving companies to modernize their applications, most want to ensure that the business logic and functional design are preserved to varying degrees as these are core assets.

Design Recovery of an application can be broken down into a few logical steps or stages that represent a generic adaptable approach to any application modernization project:

**Analysis, Documentation, Application Subdivision** – This type of analysis represents the most common use of the X-Analysis tool across the world. On top of very powerful cross-referencing functionality, graphical, narratives or a combination of both, are used to abstract and describe the system in a simple and intuitive way, even for non-RPG/COBOL experts. The legacy application can be completely documented using modern diagramming standards such as UML, Entity Relationship Diagrams, System Flow Diagrams, and Structure Charts etc. Furthermore, the legacy system can be automatically subdivided into application areas so that effective system overview & interface diagrams can be generated. The complete application documentation can then be output to a variety of third party design tools such as Rational, MS Visio, MS Word, etc. – indeed any tool capable of importing XML or DDL is supported.

**Recovering an Application design** – This advanced level of analysis extracts model information from the existing application. X-Analysis uses its own analysis repository, plus pattern searching algorithms, to derive relational data models, extract business rules, build UML Activity/Use/Case Diagrams, and logical screen flows. Only relevant designs need be used as a base specification for new developers to rewrite the application. The structured, repository-based format of these extracted designs, make it possible, to programmatically reuse them for rebuilding the core of a new application. This can be done with purpose-built tools, with X-Modernize or a combination of both.

**Redeveloping Using a Recovered Application Design** – [This is beyond the scope of this document so we won't go into detail here, if you require more

information please contact Databorough for the modernization white paper.] This starts with database modernization using the recovered data model. The designs for the view, controller, and business rule logic are also extracted and reused in modern frameworks such as Hibernate, and with new JSF/Facelets and Java bean components. This option makes it possible to programmatically re-factor the existing application into modern, consumable assets and artifacts for developers to use for a system rebuild. The objective is to produce clean, well structured, industry standard code rather than messy syntax conversions with un-maintainable code.

### Recovering an Application Design

The concept of reusing existing code or logic is not a new one. 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, specifically RPG, represents a tragic potential waste of valuable business assets. In many cases, these expensive and well-established legacy designs have little chance of even having their relevance assessed, let alone being reused. The Design Recovery Solution Set of X-Analysis addresses this problem more directly, by isolating, indexing, and documenting those design elements that could be relevant in a modern version of the application being assessed.

Modern applications are implemented with distributed architecture. A popular standard used for this architecture is MVC or Model-View Controller. Figure 8 below shows a typical legacy and MVC architecture side by side. MVC allows for independent implementation and development of each layer, and facilitates OO techniques and code re-usability rarely used in legacy applications. All these characteristics of a modern application radically improve the maintainability and agile nature. Legacy applications do have these same elements, but they tend to be embedded in and mixed up in large monolithic programs, with vast amounts of redundancy and duplication throughout. Implementing an RPG application using MVC requires that the business logic be separate from the user interface and controller logic. This can be implemented using 5250 and pure RPG<sup>3</sup>, but it's more likely and common implementation is using a web interface for the view, with the controller logic written in a modern language that supports web interfaces such as Java, EGL or C#. The optimum modernization result is to reduce dependency on legacy languages as much as is possible, if not altogether.

To achieve this recovered design assets are reused as input to redevelop the appropriate layer.

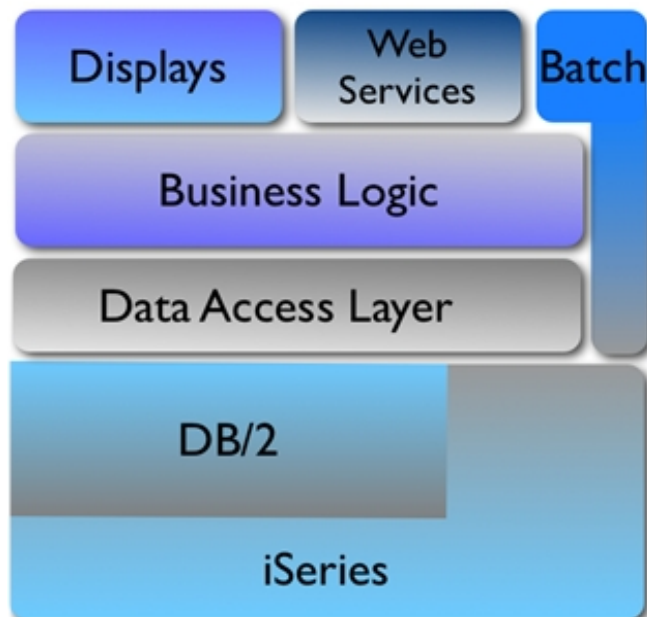
---

<sup>3</sup> Historically some sites have built or modified their systems to separate the presentation layer from the business logic often called n-tier or 3 tier applications these systems worked but delivered very little 'bang for the buck' as most of the improvements were hidden.



*Figure 1: The 'Black Box' that is many iSeries installations*

Because the business logic is mixed in with the display and batch program logic it is very difficult to isolate the different elements of the system and to separate the essence of your systems i.e. the business logic from the more mechanical aspects. With X-Analysis Design recover the picture becomes clearer.



*Figure 2: X-Analysis reveals the structure of your system*

## ***Recovering the Data Model***

The relational model of an enterprise application is an extremely powerful piece of information and potentially valuable asset to the organization. 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, scrambling 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 or COBOL 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.

**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 extract automatically, 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 the 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:



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 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 currently generates a JSF/Facelets UI version. The design meta-data can also naturally be used to generate new interfaces using any technology such as EGL, Ajax, RCP, C#, VB or even RPG.

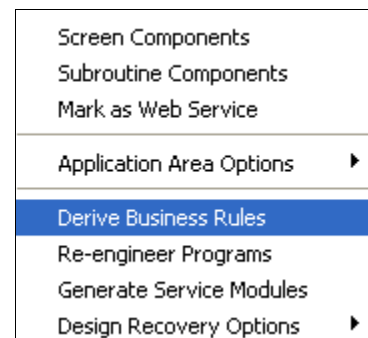
### **Recovering Business Rule Logic**

Once the system UI, data access & data model has been recovered & the application has been rebuilt or rewritten from this design, it is then necessary to extract the logic that gives the application its particular characteristics. The generic term for such logic is Business Rules. The challenge is to extract or "harvest" these rules from the legacy code.

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

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 resources. Once harvested these rules need to be narrated and indexed, thus providing critical information for any analysts, architect or developer charged with 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 the RPG and COBOL programs 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 textual narratives to describe these harvested rules. Once the rules are derived they can be viewed in summary form:



**Figure 4: Derive Business Rule option**

Business Rules for CNTCMAINT, Number of Lines: 6						
Source Member	Narration	Type	Rule No.	Field	File/Program	Rule
CNTCMAINT	If the field ZUSERNM is bla...	V	00001	USERNM	CNTACS	You must enter a contac
CNTCMAINT	If the field ZTELNO is not e...	V	00002	TELNO	CNTACS	The telephone no. is inve
CNTCMAINT	If the field ZFAXNO is not e...	V	00003	FAXNO	CNTACS	The fax. no. is invalid.
CNTCMAINT	If the field ZSINIT is not eq...	L	00004	SINIT	CNTACS	Field logic
CNTCMAINT	Verify the field ZSINIT agai...	V	00005	PERSON	SLMEN	Invalid salesman.
CNTCMAINT	If the field ZSTATUS is not ...	V	00006	STATUS	CNTACS	The status is invalid.

*Figure 5: Business Rules for a program*

Or inline in the code from where they are derived:

```

0160.95  =!BRC*  If the field ZUSERNM is blank then it is invalid , set
0160.96  =!BRC*  to off.If the field ZUSERNM is blank , set the indicato
0160.97  =!BRC*  the field MSGID to 'OEM0020' , set the field VALID to c
0160.98  %!BRC*  Business Rule No. XAN4CDEM/QRPGLESRC/CNTCMAINT/161 Vali
0160.99  @!BRC*  V00001  CNTACS      USERNM      You must enter a contac
0161.00  B>!BRC          if          zusernm = *blanks
0162.00  ->!BRC         eval          *in33 = *on
0163.00  ->!BRC         eval          msgid = 'OEM0020'
0164.00  ->!BRC         callp(e)    rtmsgtext(msgid:errormsg)
0165.00  ->!BRC         eval          valid = *off
0166.00  ->!BRC         leavesr
0167.00  E>!BRC        endif

```

*Figure 6: Embedded Rules displaying inline Business Rules*

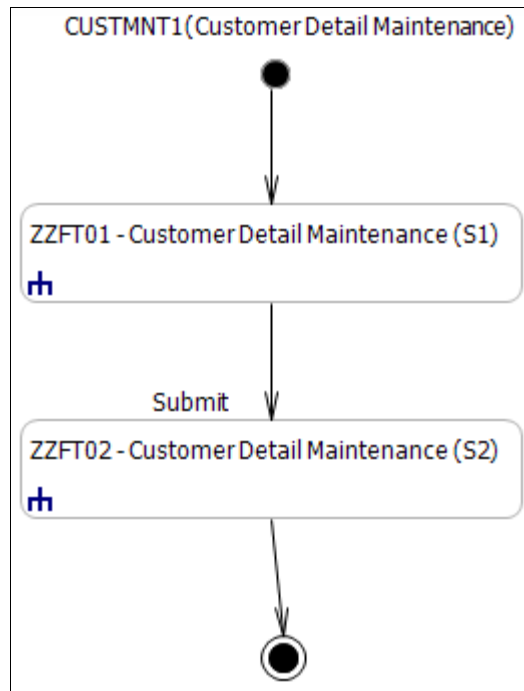
The business rule repository can then either be used programmatically to generate new code, or the built-in documentation, cross referencing where-used and annotation capabilities, may be used by new developers as the necessary input for re-specification exercises, whether for new applications or for modifications to the current system.

## UML Diagramming

The objective of UML diagrams in this context is to help sketch application designs and to make such sketches portable and reusable in multiple development environments. The three diagrams automatically generated by the X-Analysis are:

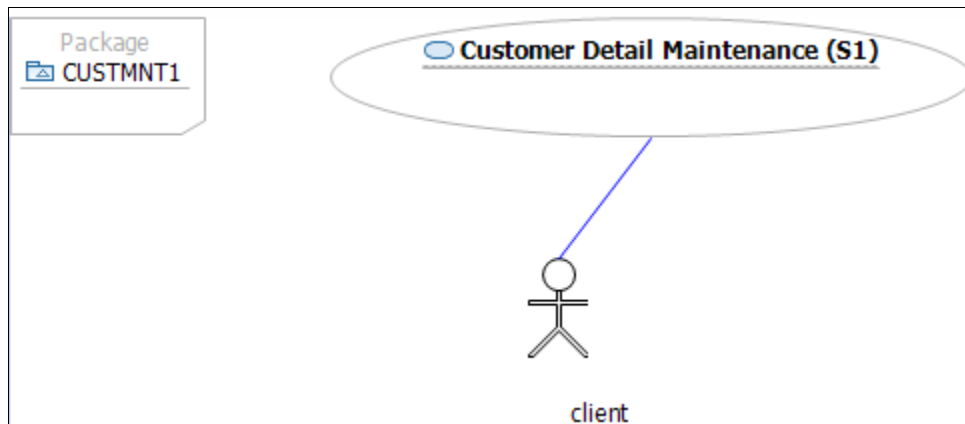
**Activity Diagram** - Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model 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.



*Illustration 1: Activity Diagram*

**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.



*Illustration 2: Use Case Diagram*

**Class Diagram** - Class diagrams are the backbone of all object-oriented methods, including UML. They describe the static structure of a system. Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes. An extracted class in a class diagram corresponds to the individual screen formats and all of the specific attributes of that particular format. X-Analysis deduces the links between these classes using a combination of the derived data model, and call or action information extracted from each program.

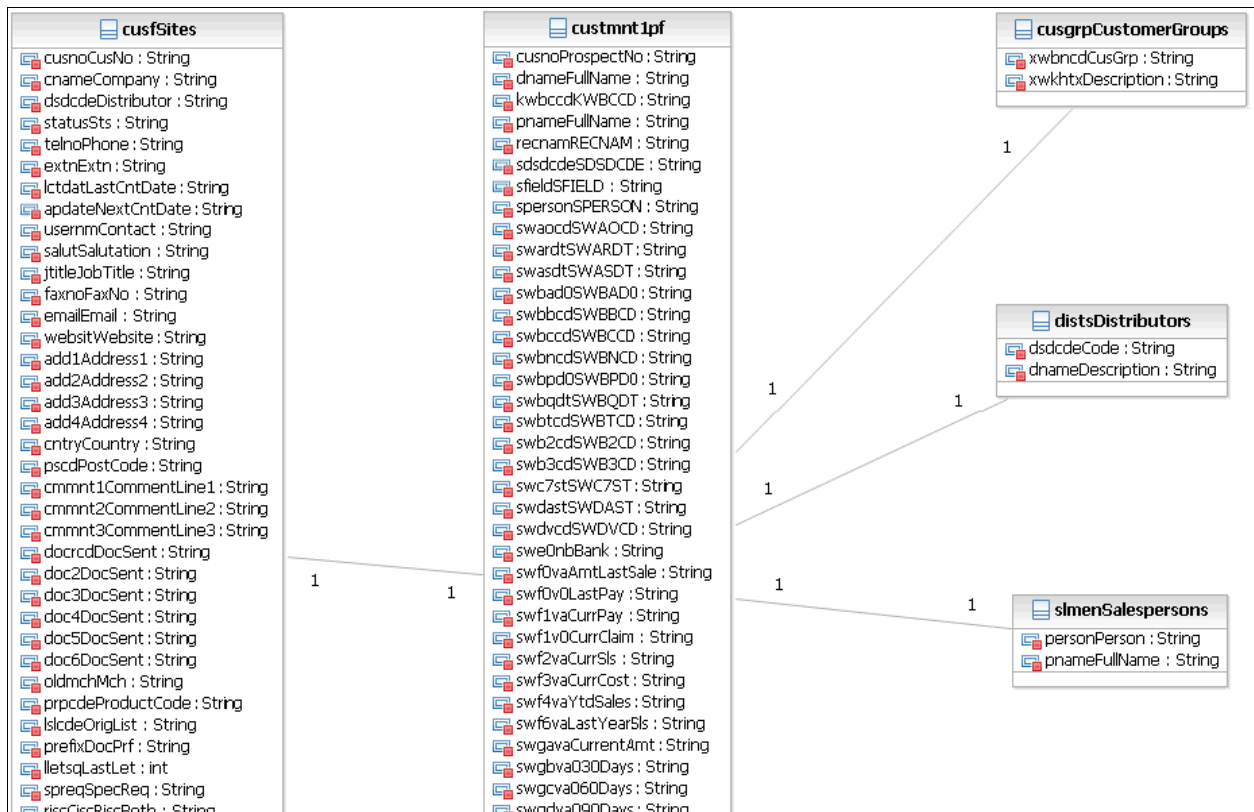


Figure 8: Class Diagram

Producing any of these diagrams from within the X-Analysis is as simple as right-clicking on an object and selecting the appropriate option from the pop-up menu.

## Design Recovery next steps

Once you've used X-Analysis to recover your system design and documented all the processes thoroughly you have a well documented system which allows you move forward in a number of ways all of which can be assisted by Databorough and X-Analysis.

- ✓ Database Modernization
- ✓ Application modernization

## **Database Modernization - using the Data Model assets**

Whilst it has always been possible to access System i data in a relational database like fashion there was originally no way of defining your database in the traditional relational database form with a schema or model. This has meant that most System i applications don't have an explicitly defined relational database schema or model. The data model for a legacy application as deduced by X-Analysis can be used to modernize the database and database access as well as providing valuable information for analysis and documentation. Once you have a modernized database you gain a number of advantages:

Easier access to your data for reporting via Business Intelligence (BI) tools when they use the newly derived Data Model.

- Ability to use modern Object Relational Mapping (ORM) software such as Hibernate for rapid application development in Java and other modern languages.
- Because the database is defined in purely SQL terms rather than in a proprietary file format it becomes portable i.e. it is now an option to consider moving the database to another platform.
- Openness and Standards compliance using Industry standard SQL means that many different tools and applications on multiple platforms can easily access and use your modernized database
- Improved performance as IBM's data retrieval efforts have been concentrated on SQL access rather than file based access for many years now
- Reduced dependency on System i specific skills such as DDS, which may led to cost savings and reduced risk.

## **Application Modernization - building on solid ground**

Many System i shops have a solid base of fast, reliable well-proven apps that their enterprises depend on, but there is inevitably a backlog of enhancements and new development requests to be dealt with and a demand for applications that require less training effort and look more 'modern' - usually but always web browser based.

The X-Analysis design recovery process allows modernization to take place in several different ways ranging from opening up existing processes as web services to building new JEE applications.

In essence X-Analysis allows you to build new applications taking what's best from the old system and leaving behind all that is unnecessary and platform specific. You get an assisted ground-up rebuild with your proven designs and business logic from the existing system without the overhead that code conversion would introduce or the risk inherent in a re-write approach.

## Summary

Comprehensive, accurate, and current documentation of a legacy application improves quality, productivity and reduces risk, for any maintenance, modernization or rebuild IT project. The risk associated with maintaining large complex legacy application, with a rapidly diminishing set of legacy skills, can be largely mitigated by access to such documentation.

Understanding and mapping the relevance of existing designs, and quantifying the scope and metrics of an application provides a solid foundation for either ongoing maintenance and development of systems or modernization projects. Legacy design constructs can be used passively in the form of information they represent, and programmatically to radically accelerate application rebuilds; a requirement for achieving true long-term application modernization. A combination of both allows optimum use of internal and external resources and existing design assets.

X-Analysis delivers against all of these concepts. 20 years of development effort, ensures that virtually any legacy application can be automatically reverse engineered onto a high-level design.

**Richard Downey and Stuart Milligan**  
© Databorough