Databorough

Analysis and Documentation with X-Analysis

» *Analyze Application Graphically*

» *Automatic System Documentation*

Richard Downey
Stuart Milligan

WHITE PAPER

Preface

Developing 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 application analysis and documentation of a RPG applications. Contact info@databorough.com for a copy of the white paper, the Red book and trial software.

Table of Contents

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[1] 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 the X-Analysis carries out analysis and automatic documentation which delivers deep insights of your systems.

[1] *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

As we have seen gathering knowledge about System i applications is not a straightforward task for today's generation of business analysts and developers. To illustrate that point and to fully understand the problem domain we will look at **Why Design Recovery is difficult**[2] by working through the problems that X-Analysis solves in building its repository of design recovery information.

In situations where developers are not familiar with a system or its documentation is inadequate, the system's source code becomes the only reliable source of information. Unfortunately, source code has much more detail than is needed just to understand the system, also it disperses or obscures high-level constructs that would ease the system's understanding. X-Analysis aids system understanding by identifying recurring program features, classifying the system modules based on their purpose and usage patterns, and analyzing dependencies across the modules.

This analysis provides detailed design information about the entire system, accessible to non RPG/COBOL experts, and be easily updated to incorporate ongoing changes in the base system.

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.

[2] If your colleagues or the project sponsors don't appreciate the difficulty of modernization get them to read this section.

Recovering an Application design – [This is beyond the scope of this document so we won't go into detail here.] 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.

Why Design Recovery is difficult

From the point of view of the user of X-Analysis this process of building the cross-reference repository and deriving the models happens automatically! It's just there and happens typically as part of the installation process - though it can be triggered again later on if required. However it is worth taking some time to understand this process and to see what happens, how the model is constructed and the relationships inferred.

If you think for a minute of a typical System i application it is likely to consist of a mix of RPG programs, DDS files and members for display files, database files and logical views, newer systems may have these interspersed with SQL scripts but the sum of knowledge in that system, how it works and interacts amongst its various elements is contained within those source files and compiled objects - the issue is retrieving that knowledge efficiently.

To understand and fully appreciate the problems X-Analysis solves just consider the process you would have to undertake yourself if you wanted to discover how a system operates or make changes to it. As a simple example for part of your application you have a customer details screen with no dedicated place for an email address and mobile phone numbers, the system has adapted itself to the internet age as many System i apps have done by making use of .extra. and .notes. ad-hoc fields. The system has coped but it has been time consuming to retrieve these details when required for marketing purposes. But there is now a budget to correct this and start to look at modernizing the application and making the functionality available to more areas of the business.

You would probably first start by looking at the program and display files that handle the display and maintenance of the customer information, from that you would discover the database tables/files involved.

At this point from a simplistic point of view you have the necessary information to make the changes and they are probably not that difficult - add new fields for email and mobile phone to the database tables or rename the existing ones then modify the program and display files accordingly¡¦ but you're probably thinking what about the rest of the system? What else uses that table? Is the display file used anywhere else? So the change has more aspects than would first appear these are just a few of the questions we have to answer:

- **Scope and impact of the change -** how many programs and tables are effected?
- **Database changes -** do we add new fields or just rename the fields and preserve the status quo? Do we know those fields were only used for email and mobile phone data?
- **Database integrity -** Fields destined for ad-hoc data like 'extra information' and 'notes' are unlikely to have any validation or to be even required so if migrating the existing values to new fields we can't simply copy it over some cleansing will be required.

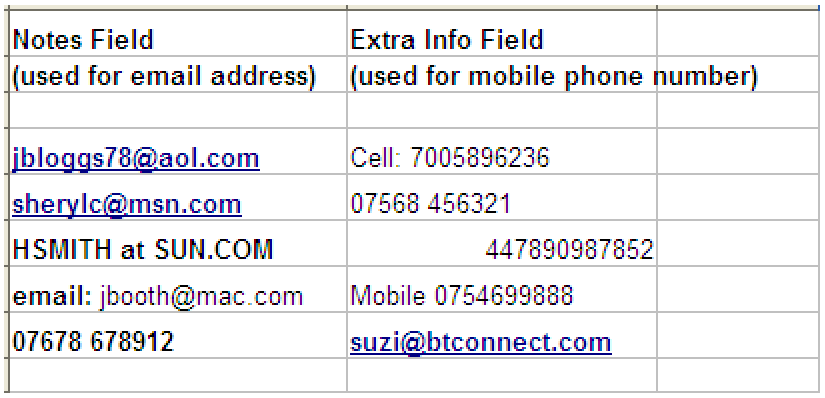
The process of gaining the knowledge to answer these questions may not be all that straightforward, particularly if the systems are complex or the people trying to answer them are new to the application, system or platform.

To assess the scope and impact of the change you need to find out which programs use the files/tables affected , this can be very laborious :

✔ Go through all source files in PDM,
✔ option 25 to searching
✔ then F13 to repeat
✔ press enter ,
✔ type in your search term
✔ review results ...

...and that's just the first enquiry! Depending on the complexity and history of your systems you may have doubts that you were looking at all of the source or the latest version.

Looking into Database integrity may well throw up items like the screen shot below:



Notes Field (used for email address)	Extra Info Field (used for mobile phone number)	
jbloggs78@aol.com	Cell: 7005896236	
sherylc@msn.com	07568 456321	
HSMITH at SUN.COM	447890987852	
email: jbooth@mac.com	Mobile 0754699888	
07678 678912	suzi@btconnect.com	

We have a number of different formats of email address and some extraneous text , similarly on the phone number list there is text and a variety of layouts. Finally we have the inevitable result of using ad-hoc fields with no validation or on screen guidance - transposed data mobile in email and vice-versa.

Hopefully this section raised awareness of the problems around changing and modernizing System i applications, the issues with finding out the necessary information and how seemingly straightforward issues can be time consuming and problematic. X-Analysis is designed and optimized to make the design recovery process as straightforward as possible as the rest of this concepts guide will illustrate.

Analysis, Documentation & Application Subdivision

X-Analysis builds a very detailed repository over an entire application. The repository maintains all information about application objects, their relationships and all necessary information to obtain detailed information from each object across the entire system. 20 years of ongoing development, over thousands of AS/400/iSeries/System-i applications written in all variants of RPGII/400/IV, COBOL, and CL, has produced an unmatched capability to extract everything about an application from object right down to individual variables. The repository is built automatically using a single command, and initially collects all object related information, but then parses every source member in the specified system and every source line mapping the contextual information of each variable in the system. A certain amount of logical abstraction processing then takes place while building the repository to account for some of the idiosyncrasies typical in an RPG application. This includes constructs such as variable program calls, file overrides, prefixing and renaming in RPG. The repository thus represents a map of how the entire application functions right down to individual variables.

Understanding Design & Function More Easily

For efficient familiarization of an application's structure and general function, an abstraction above the source code combined with object-to-object relational information is required. A few simple but rich types of color-coded, graphical diagrams can reveal the data flow and architecture of individual objects or parts of an entire system. This is combined with automatically derived descriptions in the form of Pseudo narratives either in the diagrams or while browsing source code. The drill-down, go-anywhere-from-anywhere, interactive nature of these interfaces in the X-Analysis client provides a unique approach to information assimilation, allowing an analyst to gather information at high level or very detailed in an efficient and intuitive manner. The application abstraction is raised one level above implementation. This instantly removes complexity caused by the idiosyncrasies of different language versions and coding practices, typical in large legacy applications developed over many years.

Here is a brief description of some of these diagrammatic constructs and views:

Structure Chart Diagram - A Structure Chart Diagram (SCD) Display gives a graphic representation of how the control passes from one program to another program within the application. This follows the call structure down the complete stack. The diagram also reveals data input objects and also automatically derives a summarized description of each of the object in the diagram. Color-coding also reveals important functional aspects such as updates, prints, and displays, which help the user to zone in on commonly, sought after details.

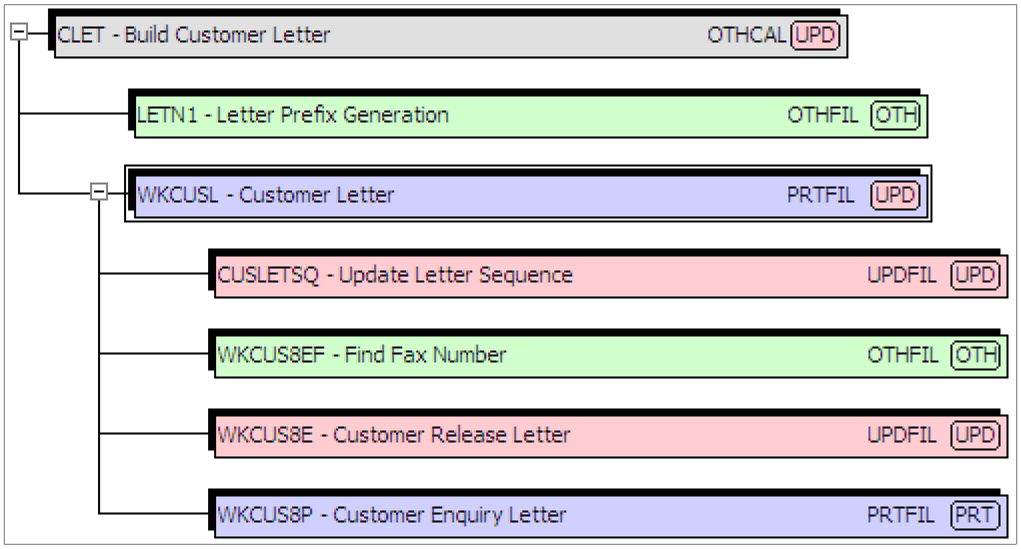

CLET - Build Customer Letter
OTHCAL
UPD
LETN1 - Letter Prefix Generation
OTHFIL
OTH
WKCUSL - Customer Letter
PRTFIL
UPD
CUSLETSQ - Update Letter Sequence
UPDFIL
UPD
WKCUS8EF - Find Fax Number
OTHFIL
OTH
WKCUS8E - Customer Release Letter
UPDFIL
UPD
WKCUS8P - Customer Enquiry Letter
PRTFIL
PRT

Figure 1: Structure Chart Diagram for a Program

Data Flow Diagram - A Data Flow Diagram (DFD) is a graphical representation of a program/object where used, showing the files and programs accessed by the subject object. It is also color-coded and shows both flow of data at a high object level, and contextual information about the specific variables/parameters passed between objects.

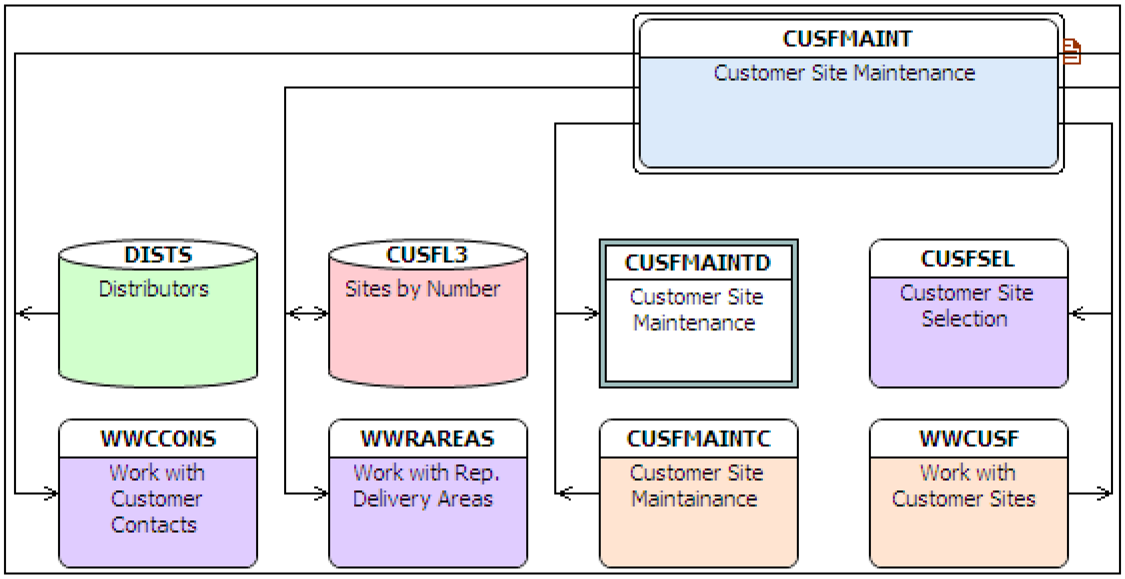


Figure 2: Data Flow Diagram for a Program

Program Structure Chart - A Program Structure Chart graphically displays the sequence of calls in the program. The call could be to execute a Subroutine / Program / Module / Service Program. For details, refer to X-Analysis User Manual.

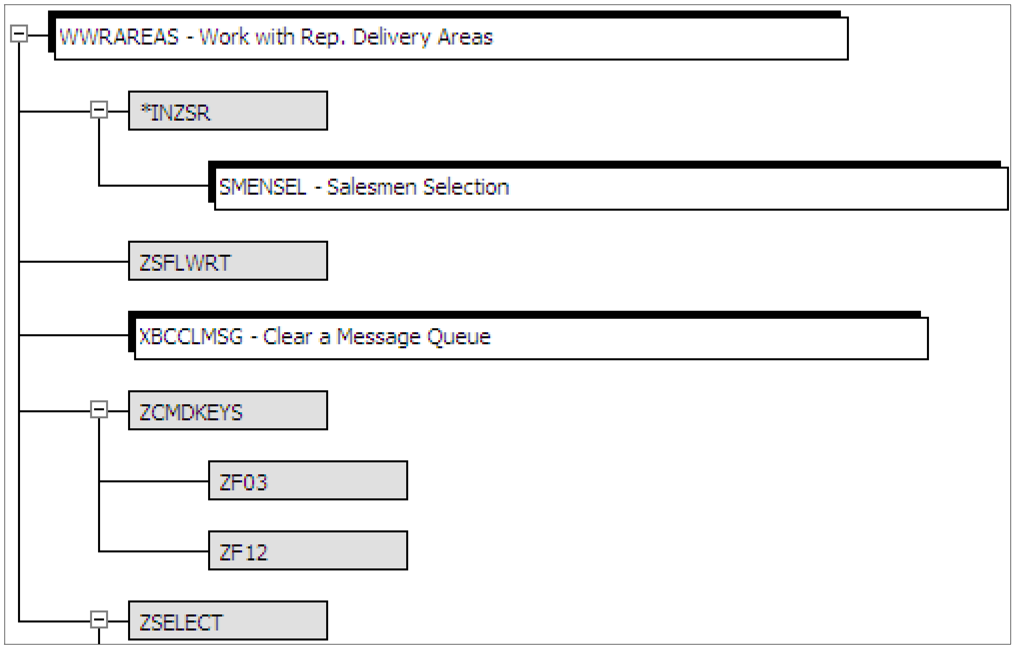

WWRAREAS - Work with Rep. Delivery Areas
*INZSR
SMENSEL - Salesmen Selection
ZSFLWRT
XBCCLMSG - Clear a Message Queue
ZCMDKEYS
ZF03
ZF12
ZSELECT

Figure 3: Program Structure Chart for a Program

Overview Structure Chart - The Overview Structure Chart gives a snapshot of an application. It displays all the entry points to the application, and then the structure chart for each of these entry points.

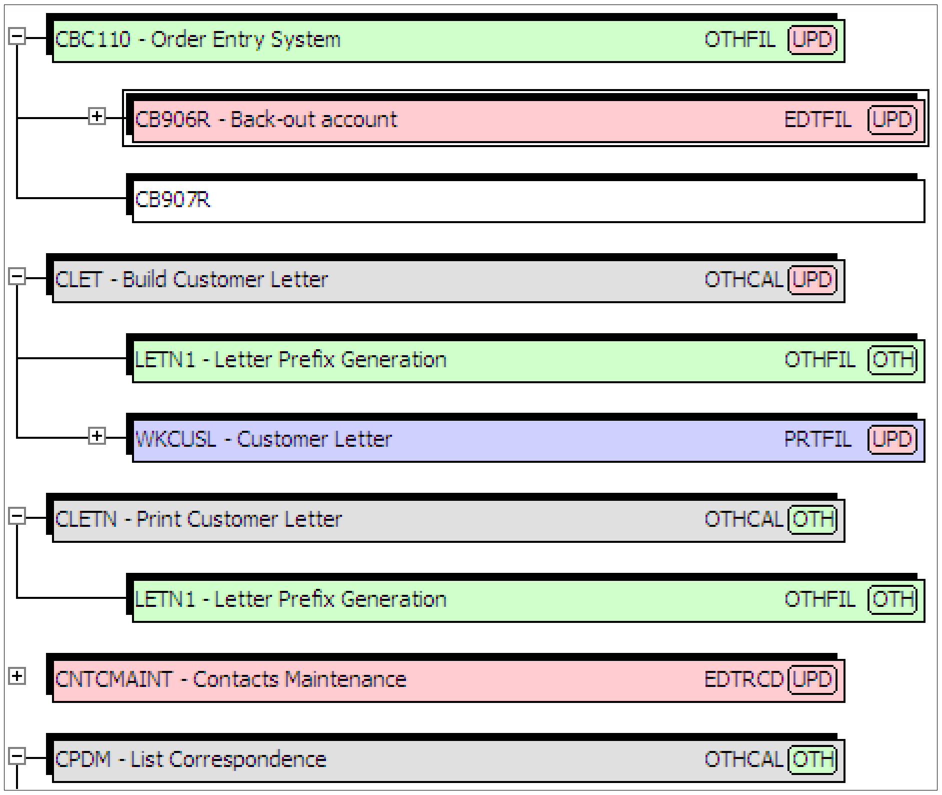
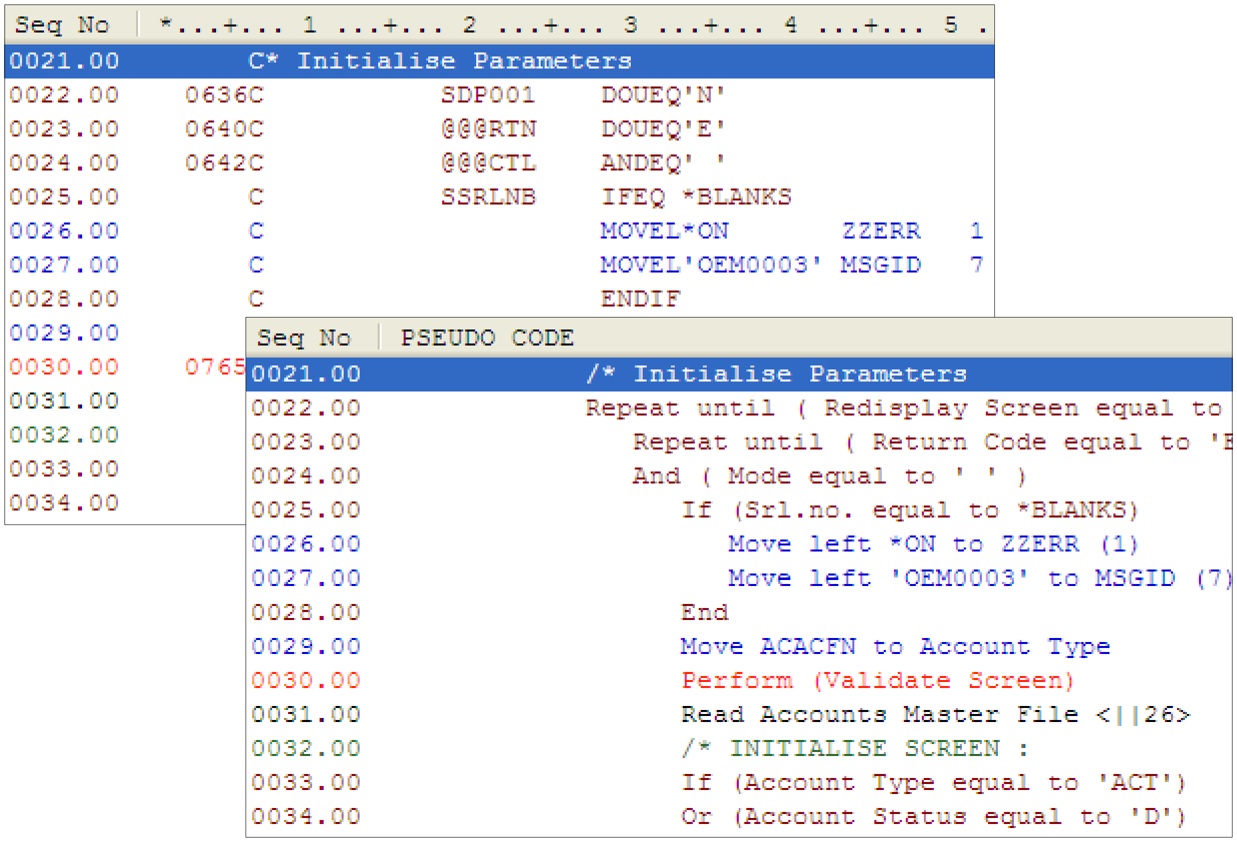

CBC110 - Order Entry System OTHFIL UPD
CB906R - Back-out account EDTFIL UPD
CB907R
CLET - Build Customer Letter OTHCAL UPD
LETN1 - Letter Prefix Generation OTHFIL OTH
WKCUSL - Customer Letter PRTFIL UPD
CLETN - Print Customer Letter OTHCAL OTH
LETN1 - Letter Prefix Generation OTHFIL OTH
CNTCMAINT - Contacts Maintenance EDTRCD UPD
CPDM - List Correspondence OTHCAL OTH

Figure 4: Overview Structure Chart for complete application

RPG as Pseudo Code- With a single click, RPG can be viewed as a form of structured English or Pseudo code. Mnemonics. are substituted with file/field/variable texts and constants or literals.



```
Seq No   *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 .
0021.00       C* Initialise Parameters
0022.00    0636C           SDP001    DOUEQ'N'
0023.00    0640C           @@@RTN    DOUEQ'E'
0024.00    0642C           @@@CTL    ANDEQ' '
0025.00       C            SSRLNB    IFEQ *BLANKS
0026.00       C                      MOVEL*ON      ZZERR    1
0027.00       C                      MOVEL'OEM0003' MSGID   7
0028.00       C                      ENDIF
0029.00
0030.00    0765
0031.00
0032.00
0033.00
0034.00
```

```
Seq No   PSEUDO CODE
0021.00                /* Initialise Parameters
0022.00                Repeat until ( Redisplay Screen equal to
0023.00                   Repeat until ( Return Code equal to 'E
0024.00                   And ( Mode equal to ' ' )
0025.00                      If (Srl.no. equal to *BLANKS)
0026.00                         Move left *ON to ZZERR (1)
0027.00                         Move left 'OEM0003' to MSGID (7)
0028.00                      End
0029.00                      Move ACACFN to Account Type
0030.00                      Perform (Validate Screen)
0031.00                      Read Accounts Master File <||26>
0032.00                      /* INITIALISE SCREEN :
0033.00                      If (Account Type equal to 'ACT')
0034.00                      Or (Account Status equal to 'D')
```

Figure 5: RPG to Pseudo code with a single click

Producing Static Documentation Automatically

Interactive analysis via a graphical client is generally the most intuitive manner in which to analyze a system, but there is often a requirement for various types of static information in the form of structured documentation. Examples of this are project documentation, auditing information, testing instructions, and customer support documentation (such as with ISV supplied business software). X-Analysis produces a number of these outputs including:

Data Flow Chart in MS Visio - Any interactive diagram produced by X-Analysis 8 in the client, can be automatically exported instantly to MS Visio with a single click. In addition to this, an RPG or COBOL program can be produced as a data flow chart interactively while browsing the source from within X-Analysis. If the RPG program is in Pseudo Code mode, the Data Flow Chart will use the narratives from the Pseudo code. This enables non-system i technologists and analysts to assimilate information at a detailed level of the application without any dependency on RPG or COBOL experts.

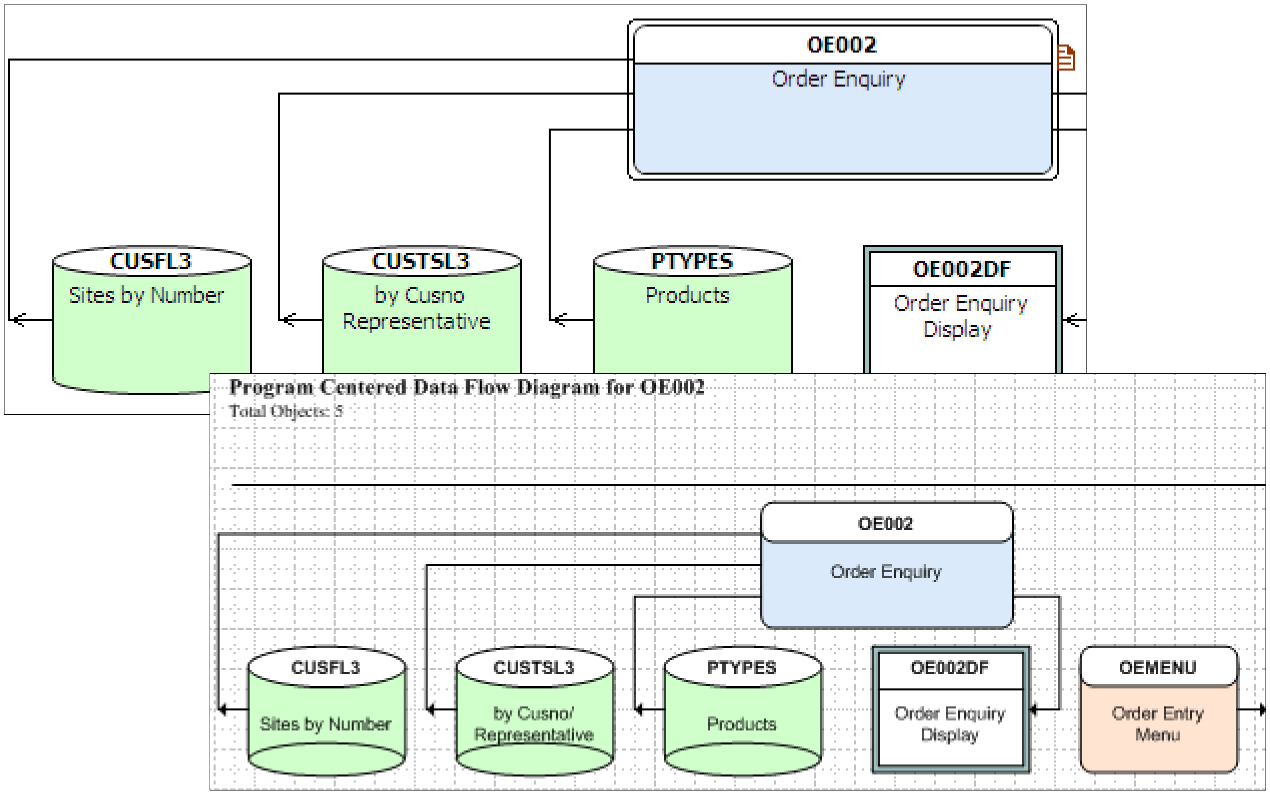


Figure 6: DFD exported to MS Visio

Lists and Results sets – Any source, object, or impact-analysis result list can be directly exported to formatted MS Excel or Word with a single click while using the client.

MS Word Project Documentation Wizard – With the use of a simple wizard, documents that might take weeks to produce manually, allow the user to select any of the graphical diagrams, lists, flowcharts, annotation and business rules summaries generated interactively by the client interface, can be collated into a single document with contents and index. This can be done for a single object, an application area (explained below), a list of objects, or an entire system. Any of these documents can then be edited and distributed as required.

Dividing Systems into Application Areas

Entire legacy applications are often too large to effectively comprehend or effect wholesale change. 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 too. 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 function or even generic name. X-Analysis then uses the 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 areas filters can then be used through the X-Analysis Solution Sets to view, document or re-engineer as opposed to individual objects.

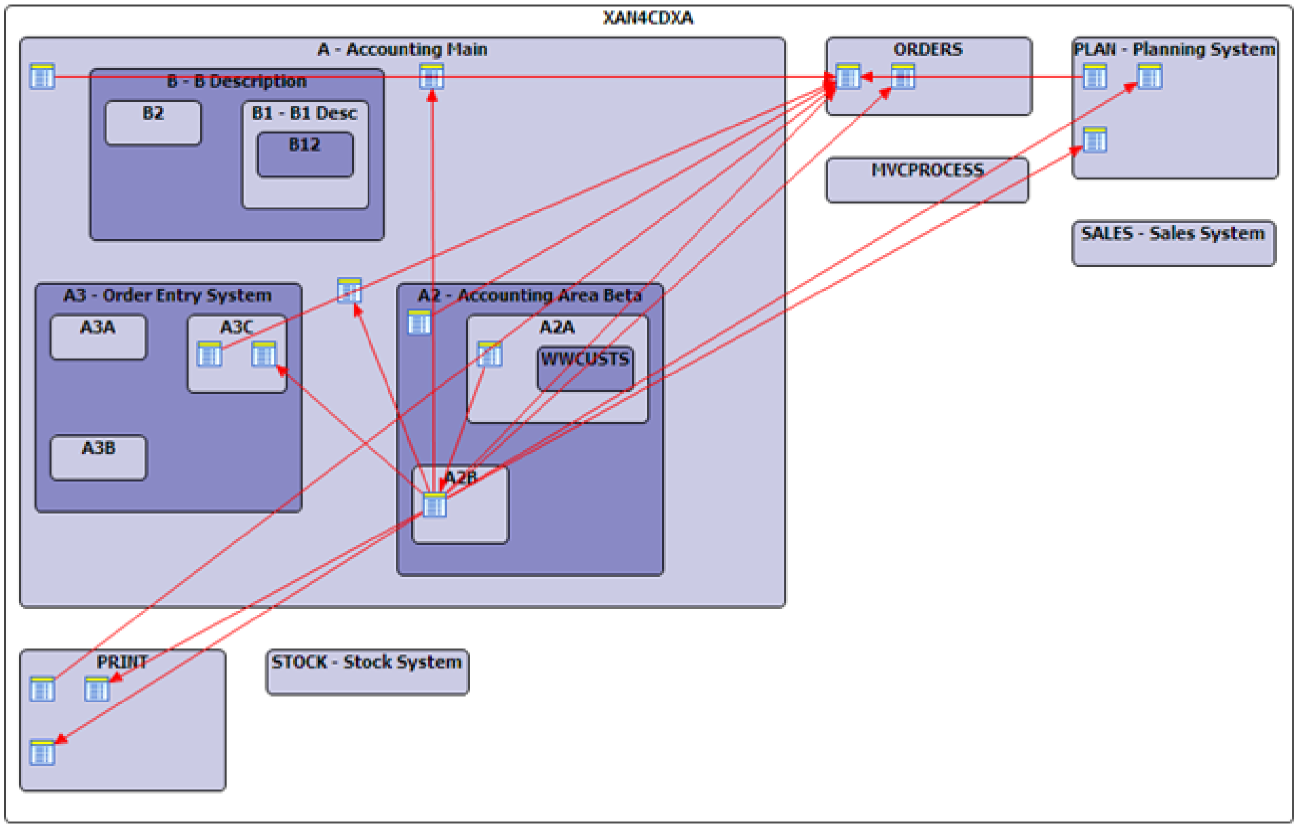


Figure 7: Application Area diagram for System Overview

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.

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.

X-Analysis provides quality analysis and documentation feature. 20 years of development effort, ensures that virtually any legacy application can be analyzed and automatically documented.

Richard Downey and Stuart Milligan