FRESCHE Legacy

IBM i Insights

Future-Proofing Your IBM i

There is an emerging challenge facing enterprises who have a continued reliance on the IBM i.

As the number of active RPG programmers and administrators dwindles due to promotions, retirement and attrition, companies will struggle to retain sufficient development resources to maintain and develop their System i applications.

IT organizations will need to find a way to help new hires quickly and accurately grasp the complexities and subtleties of these oftentimes vast systems, while providing them with the confidence to make changes and extend these systems – even though they have no hands-on development experience with the language or the platform.

In this edition of IBM i Insights, we will expand on this growing challenge in some detail, and discuss technologies and concepts that are available to help IT organizations address the RPG brain drain challenge and support continued development on the IBM i platform.

The Challenge

In an era of vanishing resources, how do corporations ensure the viability of their IBM i applications and help new hires maintain aging systems? By doing the following:

Extracting critical business rules

Understanding process and flows

Uncovering compliancy issues

Identifying obsolete and duplicated code

Performing Impact analyses.

System Information

Information about the system is key. Mission critical applications consist of a great many physical files or tables, and programs. The inter-dependencies of program-to-file and file-to-program alone can easily reach hundreds of thousands.

A typical application on IBM i could be anything from a few thousand to many millions of lines of code, with all of the complexity, design inconsistencies, languages, syntaxes and semantics that go with years of ongoing development. Mission critical applications consist of a great many physical files or tables, and programs. The inter-dependencies of program-to-file and file-to-program alone can easily reach hundreds of thousands. These are not abstracted or esoteric individual pieces of technology, but entire business systems, supporting critical business functions.

As with any successful management system, information about the system is key. The level of detail and availability of this information is another critical factor, which has already been proven in business by the success of ERP and business systems in general. The requirement is not a new one but is becoming more universal as systems continue to grow and mature. A key issue is how to manage the cost and risk of maintaining and modernizing these systems, and continually evolve to align with the needs of the business.

Application mapping – extracting and analyzing a comprehensive database of information about your business application system – is a core solution to the problem.

Technical discussion: Making informed decisions through application mapping

By mapping an entire application, a fundamental base line of information is made available for all sorts of metrics and analysis. Counting objects and source lines is generally the most common practice used for obtaining system-wide metrics. Many companies carry out software project estimations and budgeting using only this type of information. The level of experience and technical knowledge of a manager and his staff might help these numbers to some degree, but more often than not, it's mostly guesswork.

A slightly more advanced approach used with RPG, COBOL or Synon applications is to dig deeper into the application and count design elements in the programs. These elements include:

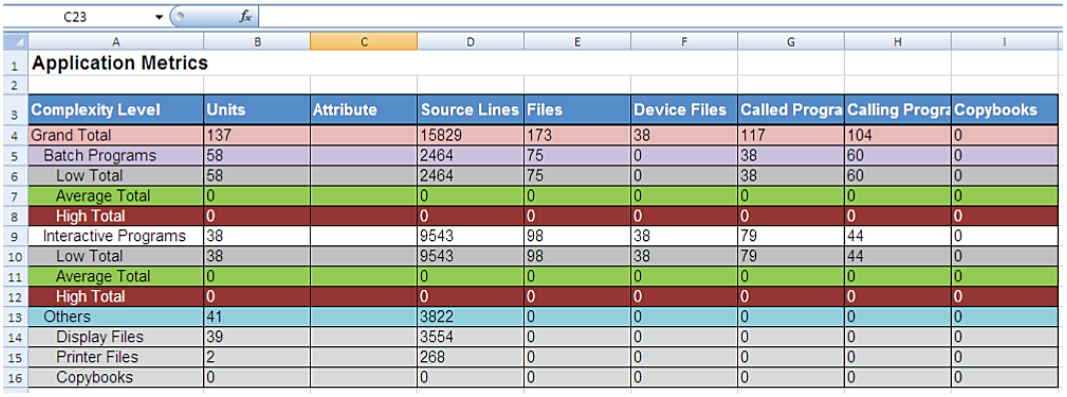
- files
- displays
- sub-files
- source lines
- sub-routines
- called programs
- calling programs

By using a simple formula to allocate significance to the count of an element, programs can be categorized by their respective counts into low, medium and high complexities. This type of matrix-based assessment is still fairly crude, but adds enough detail to make estimations and budgeting much more accurate without too much additional effort.

Application Mapping

A variety of mapping methods let you choose the level of detail you need to help you make accurate estimations and budgets. Calculations that use comprehensive and accurate metrics data for an entire application, such as those available in X-Analysis, will exponentially improve the reliability of time and cost estimation.

A simple Design Complexity Metric displayed in a spreadsheet might look like the following:



	A	B	C	D	E	F	G	H	I
1	**Application Metrics**								
2									
3	**Complexity Level**	**Units**	**Attribute**	**Source Lines**	**Files**	**Device Files**	**Called Progra**	**Calling Progra**	**Copybooks**
4	Grand Total	137		15829	173	38	117	104	0
5	Batch Programs	58		2464	75	0	38	60	0
6	Low Total	58		2464	75	0	38	60	0
7	Average Total	0		0	0	0	0	0	0
8	High Total	0		0	0	0	0	0	0
9	Interactive Programs	38		9543	98	38	79	44	0
10	Low Total	38		9543	98	38	79	44	0
11	Average Total	0		0	0	0	0	0	0
12	High Total	0		0	0	0	0	0	0
13	Others	41		3822	0	0	0	0	0
14	Display Files	39		3554	0	0	0	0	0
15	Printer Files	2		268	0	0	0	0	0
16	Copybooks	0		0	0	0	0	0	0

Figure 1 - Simple Design Complexity Metric in a spreadsheet

Another common practice is to take small representative samples, conduct project estimations, and then extrapolate this information in a simplistic linear way across the entire system or for an entire project. This approach naturally relies upon the assumption that design, style, and syntax for the entire application are consistent with the samples used. The reality is that samples are most often selected based on functionality rather than complexity. However, sometimes the opposite is true whereby the most complex example is selected on the basis of: “if it works for that it’ll work for anything.”

Calculations that use comprehensive and accurate metrics data for an entire application, versus data from a sample, will exponentially improve the reliability of time and cost estimation. Risk is not entirely removed, but plans, estimates and budgets can be more accurately quantified, audited, and even reused to measure performance of a project or process.

In the next section of this article, we’ll discuss application mapping in some detail. With an application map, a number of very useful statistics and metrics can be calculated including detailed testing requirements and a “maintainability index” for entire systems or parts thereof.

Application Knowledge

Maintaining application knowledge is crucial in reducing cost and risk. Native platform functionality provides a certain amount of information about your applications, but detail is limited and the work is tedious for large systems.

Building application maps

The cost of ownership of these large complex IBM i applications increases and maintenance becomes more risky as application knowledge is lost and goes unreplaced.

Some information is available through native functionality, such as the DSP commands. Display Program References (DSPPGMREF shown below in Figure 2) provides information about how a program object relates to other objects in the system.

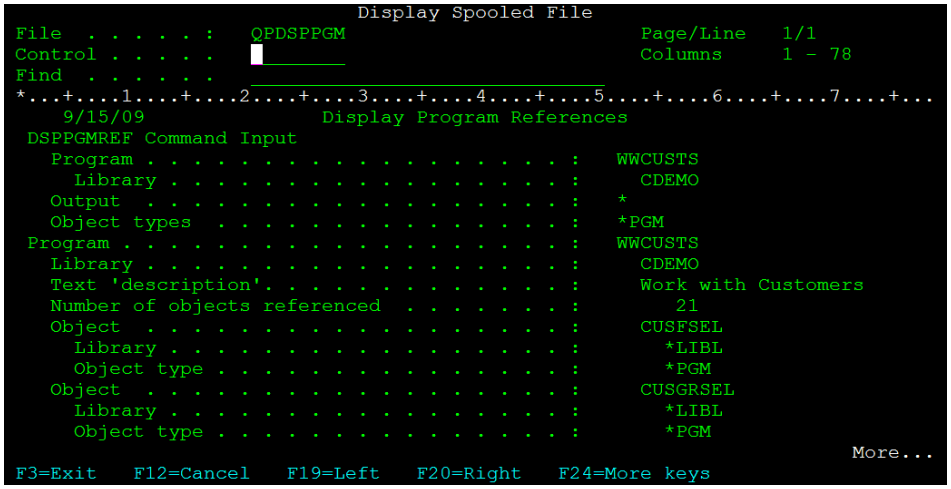
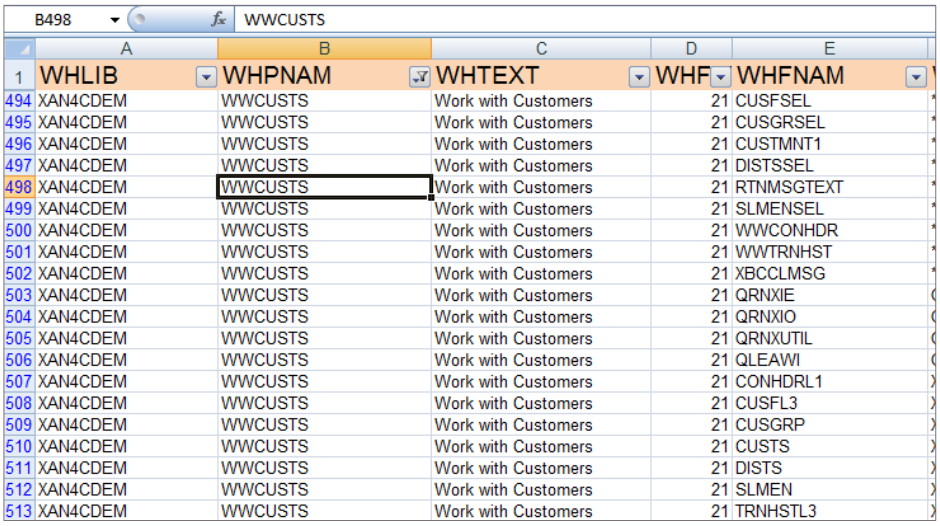


Figure 2 – DSPPGMREF screen output

This information is very useful in determining how a program relates to other objects. It is possible to extract this information and store it in a file, and carry out searches on this file during analysis work, as shown below.



B498 WWCUSTS

	A	B	C	D	E
1	WHLIB	WHPNAM	WHTEXT	WHF	WHFNAM
494	XAN4CDEM	WWCUSTS	Work with Customers	21	CUSFSEL
495	XAN4CDEM	WWCUSTS	Work with Customers	21	CUSGRSEL
496	XAN4CDEM	WWCUSTS	Work with Customers	21	CUSTMNT1
497	XAN4CDEM	WWCUSTS	Work with Customers	21	DISTSSEL
498	XAN4CDEM	WWCUSTS	Work with Customers	21	RTNMSGTEXT
499	XAN4CDEM	WWCUSTS	Work with Customers	21	SLMENSEL
500	XAN4CDEM	WWCUSTS	Work with Customers	21	WWCONHDR
501	XAN4CDEM	WWCUSTS	Work with Customers	21	WWTRNHST
502	XAN4CDEM	WWCUSTS	Work with Customers	21	XBCCLMSG
503	XAN4CDEM	WWCUSTS	Work with Customers	21	QRNXIE
504	XAN4CDEM	WWCUSTS	Work with Customers	21	QRNXIO
505	XAN4CDEM	WWCUSTS	Work with Customers	21	QRNXUTIL
506	XAN4CDEM	WWCUSTS	Work with Customers	21	QLEAWI
507	XAN4CDEM	WWCUSTS	Work with Customers	21	CONHDRL1
508	XAN4CDEM	WWCUSTS	Work with Customers	21	CUSFL3
509	XAN4CDEM	WWCUSTS	Work with Customers	21	CUSGRP
510	XAN4CDEM	WWCUSTS	Work with Customers	21	CUSTS
511	XAN4CDEM	WWCUSTS	Work with Customers	21	DISTS
512	XAN4CDEM	WWCUSTS	Work with Customers	21	SLMEN
513	XAN4CDEM	WWCUSTS	Work with Customers	21	TRNHSTL3

Figure 3 - DSPPGMREF output to spreadsheet

Graphical Display

Graphical display of application structure offered through X-Analysis provides a much richer understanding of the application and its relation to the greater system. X-Analysis provides graphical information such as the following:

- **Detailed data flow diagrams**
- **Structure Chart diagrams**
- **Application overviews**
- **Impact analyses**
- **Business rules**
- **UML diagrams**
- **Metrics analysis**

A much more efficient way of showing the same information is to display it graphically using X-Analysis. Additional information such as the directional flow of data can easily be included and understood when added to diagrams. Systems design and architecture is best served using diagrams. Color coding within these constructs is also important as it helps assimilate structure and logically significant information more quickly. A good example of this is showing where updates take place using the color red (see figure 4 below).

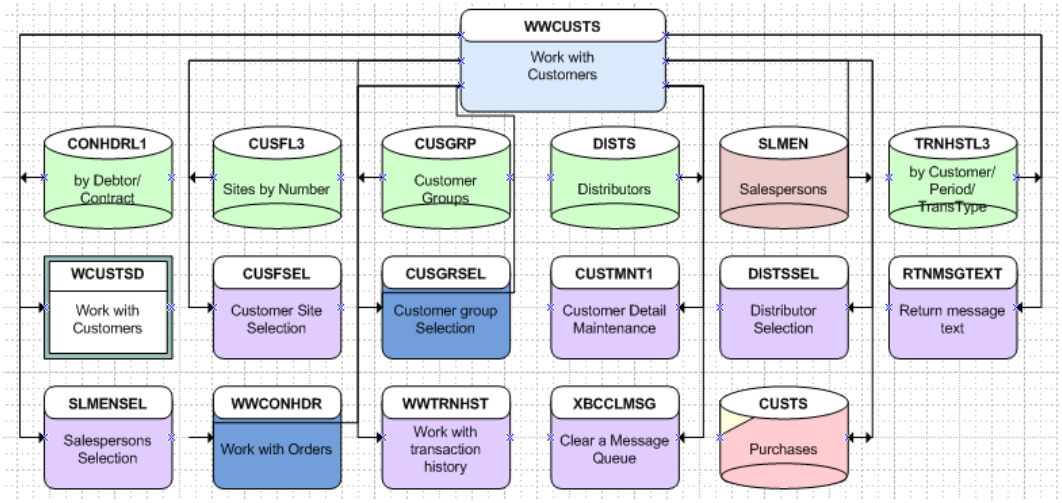


Figure 4 – Visualizing Program References in X-Analysis

Calculating Complexity

Halstead Volume

Halstead complexity metrics were developed by the late Maurice Halstead as a means of determining a quantitative measure of complexity directly from the operators and operands in the module to measure a program module's complexity directly from source code. Among the earliest software metrics, they are strong indicators of code complexity.

Cyclomatic Complexity

Cyclomatic complexity is a software metric (measurement) developed by Thomas McCabe and measures the amount of decision logic in a single software module. It is used for two related purposes. First, it gives the number of recommended tests for software. Second, it is used during all phases of the software lifecycle, beginning with design, to keep software reliable, testable, and manageable.

Detail

Graphical Application mapping information can be extended to simultaneously include details about individual variables associated with each of the referenced objects. The automated approach offered by X-Analysis is a necessity for this level of detail.

Embedding other important textual information such as object texts into diagrams is another way of presenting information effectively and efficiently. In Figure 4 (above) we see how graphical and textual information is combined to provide rich information about the program references with arrows used to show the flow of data between the program and the other objects.

The data flow diagram concept confirms that the flow of data through a system is critical. Application mapping information can be extended (Figure 5) to simultaneously include details about individual variables associated with each of the referenced objects. The method used to extract this level of precise variable detail is to scan the source code of the programs and establish which entry parameters are used in the case of a program-to-program relationship.

In a program-to-file relationship, the job is somewhat more tedious, as you have to look for instances where database fields and corresponding variables are used throughout the entire program. It is also useful to see where individual variables are updated as opposed to being used just as input. The diagram now presents a very rich set of information to the user in a simple and intuitive way. The amount of work to extract and present this level of detail can quickly become prohibitive, and so is better suited to X-Analysis' tools-based approach rather than manual extraction.

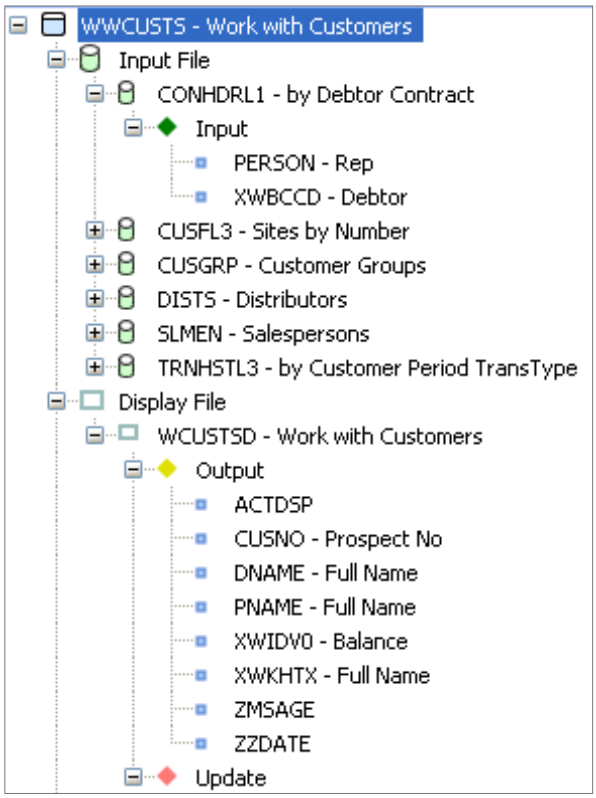


Figure 5 – Variables/Parameter Details

Application Flow

X-Analysis graphical diagrams provide color coding for updates, arrows for data flow, and detailed variables displayed simultaneously. Diagrams help users (and especially new developers) visualize the application and understand how the system is organized.

Figure 6 below shows a program-centric diagram in X-Analysis. The same diagram where the file is the central object being referenced is also very useful in understanding and analyzing complex applications. The same diagrammatic concepts can be used, such as color coding for updates, arrows for data flow, and detailed variables simultaneously being displayed. By using the same diagram types for different types of objects in this way, the same skills and methods can be reused to twice the effectiveness. Figure 5 above shows how additional information such as related logical files (displayed as database shapes) can be added and easily recognized by using different shapes to depict different object types.

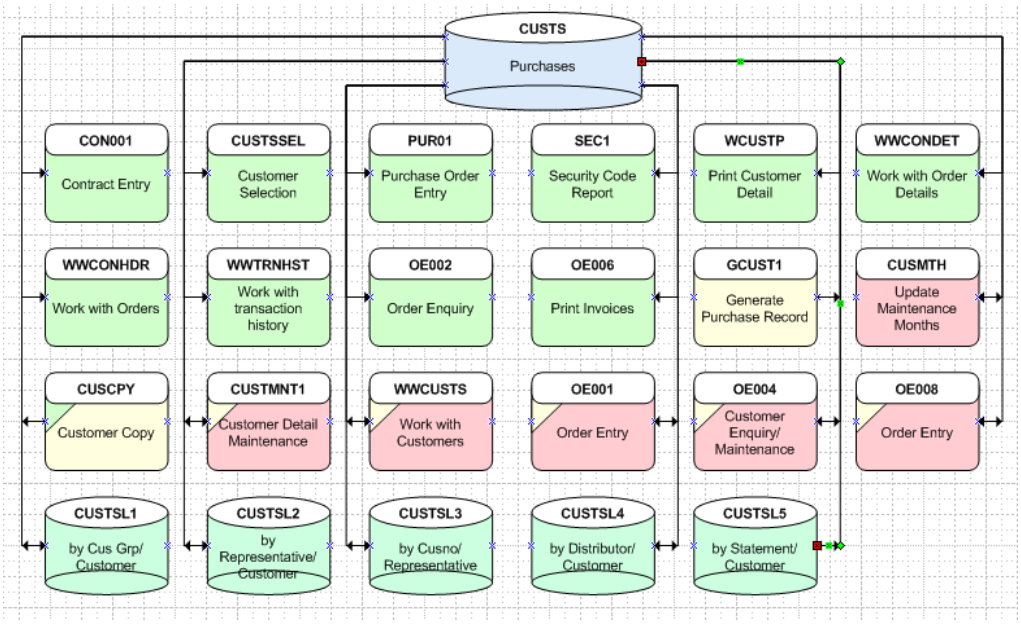


Figure 6 File-Centric Object References in X-Analysis

Functional organization of an application

Single level information about an RPG, SYNON or COBOL program is obviously not enough to understand a business system's design. You need to be able to follow the logical flow through the application.

DSPPGMREF data can be used to do this. If we start at program A and see it calls program B, we can then look at the DSPPGMREF information for program B and so on. Additionally we can deduce precisely in this structure where and how data, print, and display files are being used in the call stack, which is very useful for testing and finding bugs that produce erroneous data.

However, for large, complicated systems, this can be a slow and tedious process if done manually using the display output of the DSPPGMREF. If you extract all programs' DSPPGMREF information out to a single file, you can recursively query this file to follow the calls down successive levels starting at a given program. This can then show the entire call stack or structure chart for all levels starting at a given program or entry point.

Call Stacks

Call stacks may go down as many as 15 levels from a given starting point. Being able to display or hide details becomes an important function. X-Analysis gives you full control over how much detail you need.

But there's a better way to do it. A given program's call stack or call structure can be represented much more effectively diagrammatically in X-Analysis than with any textual description alone. Quite often these call stacks may go down as many as 15 levels from a single starting point.

It is therefore an important requirement to be able to display or hide details according to the information required at the time, along with search facilities built into the diagrams.

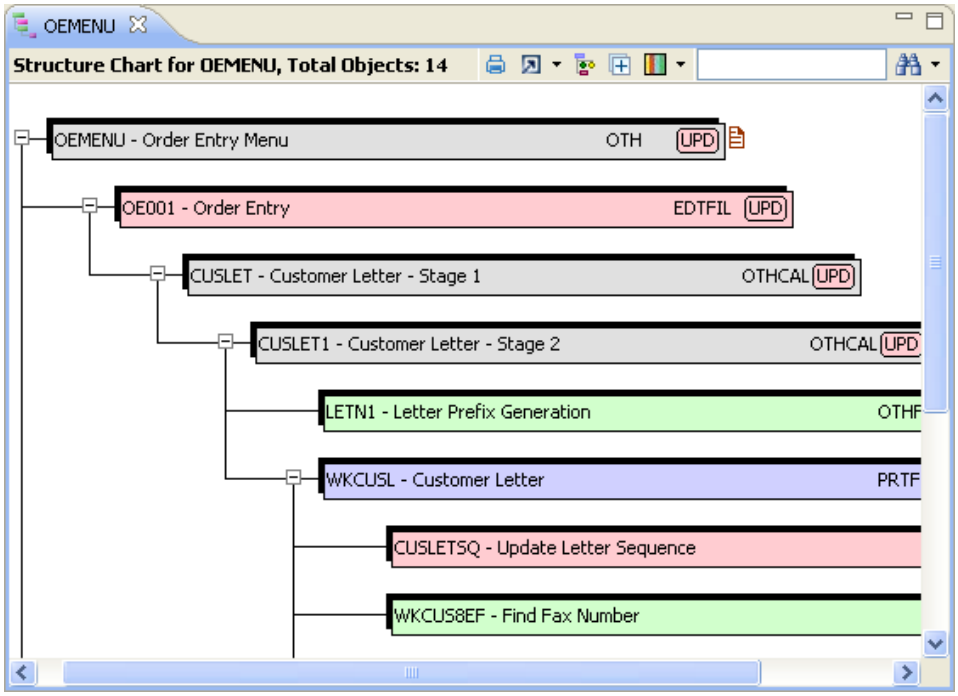


Figure 7 – Diagram showing program call structure in X-Analysis

As with other diagrams, color coding plays an important role in classifying objects in the stack by their general use, such as update, display, input only and so on.

Figure 7 above shows the structure of a program graphically. Additional information such as the data files, displays and data areas used by each object can be added to enrich the provided information.

However, this diagram alone does not give us the complete picture. For example, it doesn't tell us where we are in relation to the overall hierarchal structure of the application. We do not know if the program is an entry point into the system, or is buried in the lower levels of the application. The next page of this article describes how functional areas help the developer.

Functional Area

In X-Analysis, a functional area can be mapped by selecting an entry point (or a group of them) and then using the underlying application map to include all objects (everything including programs, files, displays) in the call stack.

To better understand an entire system, objects need to be organized into functional groups or areas. This can be achieved by using naming conventions, provided that they exist and that they are consistent across the application. In addition, the entry points into the application need to be established. Sometimes a user menu system is useful for this, but is not necessarily complete or concise enough. One way to establish what programs are potential entry points is to determine each program's call index. If a program is not called anywhere, but does call other programs, it can essentially be classed as an entry point into the system. If a program is called and in turn if it calls other programs itself, it is not an entry point.

A functional area can be mapped by selecting an entry point (or a group of them) and then using the underlying application map to include all objects (everything including programs, files, displays) in the call stack. Figure 8 shows an X-Analysis diagram of a series of entry points and their relative call stacks grouped as a functional area.

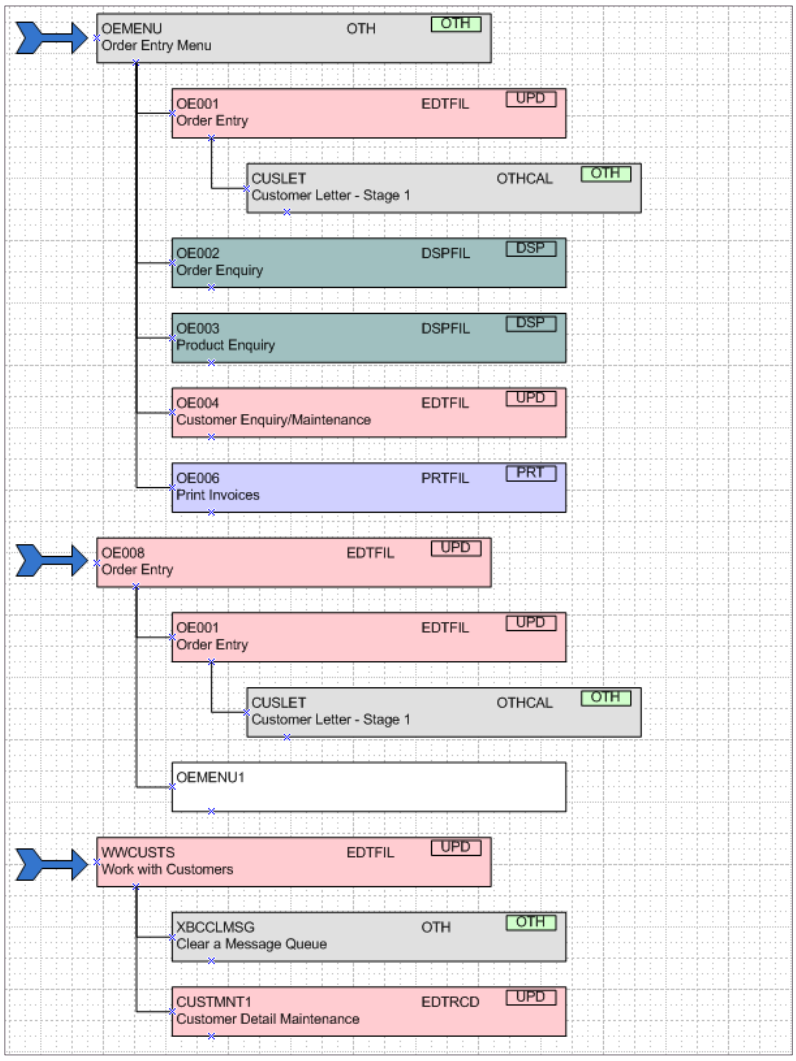


Figure 8 – Functional Application Area shown in X-Analysis

To more accurately describe an entire system's architecture, functional application areas might need to be grouped into other functional application areas. These hierarchal application areas can then be diagrammed, showing how they interrelate with each other. This interrelation can be hierarchal but also programmatic, because some objects might be found in more than one application area simultaneously.

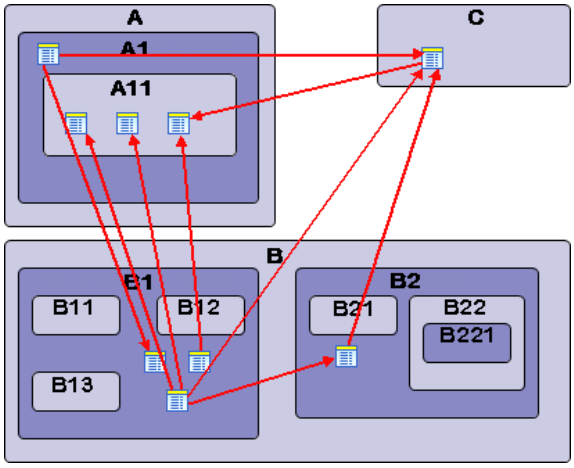


Figure 9 – High Level Functional Relationships in X-Analysis

Figure 9 shows how application areas interrelate. For the sake of clarity, only those programmatic interrelations from entry level objects have been included, showing how the accounting Main application area has other application areas embedded in it. The red lines show the programmatic links between objects within the application. In this example, this level of interrelation has been limited to programmatic links between entry point programs and programs they call in other application areas. This is a very good way of mapping business functional areas to application architecture in a simple diagram.

Logical subdivisions of an entire application are also employed in other areas of application management. Some of these include:

- Clear and concise allocation of responsibility for maintenance/support of a set of objects.
- Integration with source code control/change management tools for check-in and check-out processes during development.
- Production of user documentation for support, training and testing staff.

Database mapping

An IBM i business application is primarily an application written over a relational database. Therefore, no map of an enterprise application would be complete without the database architecture explicitly specified. Not just the physical specifications and attributes, but the logical or relational constraints too.

Database Mapping

Mapping database architecture is a critical part of an application mapping exercise. Since most RPG and COBOL applications running on IBM i have no explicitly-defined relational data model, you need to read millions of lines of code to create one manually. Alternatively, X-Analysis creates a comprehensive, centralized data dictionary that developers, managers and analysts can use.

With the possible exception of CA:2E systems, virtually all RPG or COBOL applications running on IBM i have no explicit relational data model or schema defined. This means that millions of lines of RPG or COBOL code must actually be read in order to recover an explicit version of the relational model, searching for keys that constitute links or relationships between physical files or tables in the database.

In database mapping, the first step is to produce a key-map of all the primary keys and fields for all physical files, tables, logical files, access paths and views in the database. By using a simple algorithm and looking at the DDS or DDL, one can often determine if foreign key relationships exists between files. Figure 10 shows a diagram of this simple algorithm using the database definitions themselves.

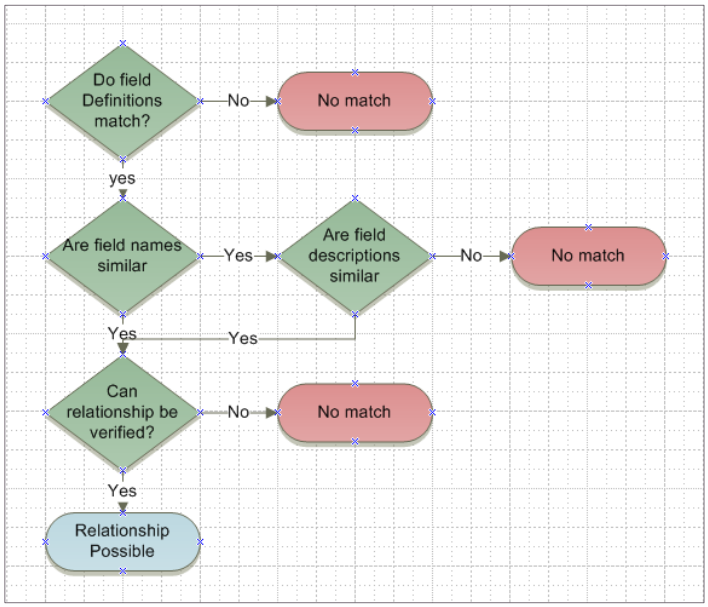


Figure 10 – Establishing foreign key relationships

A more advanced and comprehensive approach for determining foreign key relationships is to analyze the program source code for the system. If we look at the source code of a program and see that more than one file/table is used, there is a possibility that these files are related by foreign key constraints. By finding instances in the program where one of the files is accessed for any reason, and determining the keys used to do so, we can then trace these variables back through the code to keys in another file in the program. If at least one of the key fields match in attribute and size with the other file, and is part of the unique identifier of the file, then we have a very strong likelihood that there is a relationship between these two files. By then looking at the data using these key matches we can test for the truth of the relationship. By cycling through all the files in the system one by one and testing for these matches with each and every other file, we can establish all the relationships.

Database-file Relationships

X-Analysis builds a graphic depiction of the relational model, handling the structured analysis automatically so that the task is completed in a few hours rather than a few months. This automation lets you keep the relational model current with little overhead.

This task is complicated by the fact that the same field in different files will most often have a different mnemonic name. So when analyzing the program source, we have to deal with data structures, renames, prefixes and multiple variables. By having the program variable mapping information at your fingertips beforehand, the analysis process will be a lot quicker. The majority of this repetitive but structured analysis can be handled programmatically in X-Analysis, and thus enable the task to be completed in a few hours rather than several months. Such automation allows for keeping the relational model current at all times without a huge resource overhead.

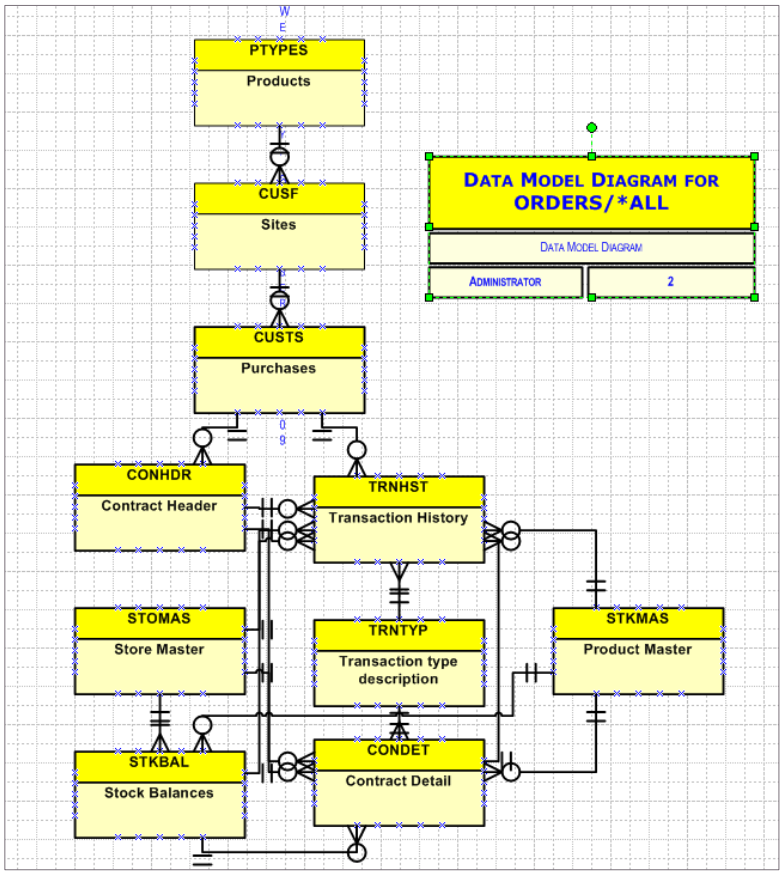


Figure 11 – Mapping database file relationships in X-Analysis

Once explicitly defined, the relational model or architecture of the database can be reused in a number of scenarios including:

- Understanding application architecture
- Data quality referential integrity testing
- Test data extraction
- Test data scrambling and aging
- Building BI applications & data warehouses
- Data mapping for system migrations
- Building object relational maps for modernization

SQL Statements

Database design must be clearly understood to write SQL statements on the IBM i. The IBM i legacy database is unlike other databases. If you try to apply normal standards, you may have problems. Performance and scalability are the obvious impacts but in addition, there are further inefficiencies with debugging, and adding infrastructure to compensate. X-Analysis provides detailed database design information and entity relationships that help developers avoid common pitfalls.

Database access in all modern languages today is primarily driven by embedded SQL. IBM i legacy databases are typified by transaction-based table design with many columns and foreign key joins. This makes the task of writing SQL statements much more difficult and error prone unless the design of the database is clearly understood. It also creates an environment where it is relatively easy for inexperienced developers or users to write I/O routines or reports that have an extremely negative performance impact. One way to combat this challenge is to provide detailed design information about the database being accessed. Figure 11 shows a typical entity relationship diagram in X-Analysis, and this can be accompanied with the underlying foreign keys details as displayed in a spreadsheet in Figure 12.

	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 12 – Foreign key details

Another more generic approach to ensuring integrity of the database, protecting the productivity of modern technology developers, and limiting negative I/O performance impacts, is to build a framework of I/O modules as stored procedures. The explicitly defined data model is a key source of information, which will greatly simplify building of such a framework, and can even be used to automate the generation of the framework itself.

It is also worth mentioning that products like IBM's DB2 Web Query can become exponentially more useful and productive if the meta-data layer is properly implemented. The derived data model can be used to build this data instantly for the entire system.

Context

What is really needed is context. For example, mapping the files and displays specified in a program helps IT organizations understand the impact of change at an object level. Manual effort can only take this labor-intensive mapping so far. X-Analysis can "learn" your application knowledge and make it reusable throughout the organization.

Hard-coding application knowledge

As we've seen, the output of DSPPGMREF is a good starting point for the type of mapping described so far, but it doesn't go far enough.

From a design perspective, application software is made up of discrete layers or levels of detail. In an IBM i application, for example, libraries contain programs, physical files, logical files, data areas, commands and many more object types. Programs might contain file specs, variables, sub-routines, procedures, display definitions, arrays and various other language constructs. Data files have fields and text descriptions and keys and other attributes. Having an inventory of all these elements is useful, but only in a very limited way from a management perspective. What is really needed is context. For example, mapping the files and displays specified in a program helps IT organizations understand the impact of change at an object level. This rudimentary mapping provided by most program comprehension tools is limited in its usefulness as it still only provides information at a single level.

Mapping all levels of detail and interrelationships with all other elements at all levels is the ultimate objective. The only way to achieve this is to use an automated tool like X-Analysis to read the source code line-by-line and infer all relationships implicit in each statement or specification. The X-Analysis mapping process allows for variants of RPG, COBOL and CL going back 20 years to be useful for the vast number of companies who have code written 20 years ago. Relatively few humans have such knowledge or skill and could never keep up with the work load required for even the most modest of IBM i applications. However, computer programs can be "taught" such knowledge and retain it permanently. This hard coded application knowledge can be reused as often as necessary to keep abreast of any code changes that take place.

Google docs

cusfowu

File Edit View Insert Format Tools Form Help

Table by Google

Define groupings and calculations

(All)	(All)	(All)	(All)
Object	Text	(All)	Library
CUSCPY	Customer Copy	File Read By Program	XAN4CDEM
CUSFL1	Sites by Name	File Read/Written To By Program	XAN4CDEM
CUSFL2	Sites by Status	File Updated By Program	XAN4CDEM
CUSFL3	Sites by Number	File Updated/Written To By Program	XAN4CDEM
CUSFL5	Sites by Dist.& Status	Logical File	XAN4CDEM
CUSFL6	Sites By Dist.& Name	Usage	XAN4CDEM
CUSFL7	Sites by Last Cnt.Date	Logical File	XAN4CDEM
CUSFL8	Sites by Next Cnt.Date	Logical File	XAN4CDEM
CUSFL9	Sites by Fax No.	Logical File	XAN4CDEM
CUSFLA	Sites by Product - renamed from cusfla for testing	Logical File	XAN4CDEM
CUSFLB	Sites by Orig.List	Logical File	XAN4CDEM
CUSFLC	Sites by Salesperson	Logical File	XAN4CDEM
CUSFLD	Sites by Validator	Logical File	XAN4CDEM
CUSFLE	Sites by Organisation	Logical File	XAN4CDEM
CUSFMAINT	Customer Site Maintenance	File Updated By Program	XAN4CDEM
CUSFMOLD	Customer Site Maintenance	File Updated By Program	XAN4CDEM

Figure 13 – Pre-Building DSPPGMREF

Pre-building the application map and storing it in an open and accessible format such as a spreadsheet in Google Docs is also a useful tactic. Figure 13 above shows the filtered output of a DSPPGMREF uploaded into a Google Docs spreadsheet. Having the map available provides for any number of complex, system-wide abstractions or inquiries at acceptable speeds.

For a complete and accurate application map, one has to follow the trail of inferred references described in the programs themselves. This is obviously a labor-intensive task made all the more difficult by common coding practices such as:

- Overriding the database field name in a CL program
- Prefixing fields from a file being used in an RPG program
- Moving values from database fields into program variables before passing them as parameters to called programs
- Changing key field names between different database files
- Passing the name of the program to be called as a parameter to a generic calling program rather than making a direct call.

An application map pre-built with X-Analysis includes all of these inferred logical references, so measurement of impact can be complete, and more importantly, instant. It also means that higher-level analysis of rules and model type designs is made easier by virtue of the easy availability of variable and object level mapping.

In summary

Using X-Analysis, application mapping provides a new way to manage and modernize complex business applications. It is also a way to facilitate collaboration between modern and legacy developers. It makes many new things possible every day. Application mapping provides a very strong platform for a number of additional benefits and technologies that will continue to evolve for many years.

In summary, application mapping with X-Analysis provides you with all of the following:

- Graphical documentation of your entire application eco-system, including detailed data flow, structure chart and functional area diagrams
- Extracted relational data models for RPG/COBOL/SYNON applications
- Design, Quality and Complexity metrics
- UML diagrams
- Extracted business rules
- Powerful impact analysis

For more information about application mapping or X-Analysis, go to www.databorough.com, or e-mail us: info@freschelegacy.com.

About Fresche Legacy

As a leading expert in legacy management and modernization, Fresche Legacy helps enterprise organizations transform their business to improve financial performance, increase market competitiveness, remove risk and add business value. Our team of experts has successfully completed hundreds of transformation projects within the most complex enterprise environments, helping organizations future-proof their business by modernizing their business processes, technologies, infrastructure, and methodologies. Committed to 100 percent customer satisfaction, Fresche Legacy's services and solutions span the complete legacy modernization spectrum from concept to maintenance, and include Discovery Services, Modernization Solutions, and Application Management Services & Transformation. For more information about our company, visit us on the Web at www.freschelegacy.com

Are you an enterprise organization seeking solutions for your legacy environment? Drop us a line at info@freschelegacy.com or call us at 1-800-361-6782