



**Databorough**

# **Five Secrets About Software Metrics**

**Steve Kilner**

Do you measure the complexity of your software?

There are some surprising, real world benefits from doing so:

1. Develop more quantifiable programming estimates and project plans
2. Quantify the barriers to modernization
3. Clarify and quantify for users the challenges of difficult enhancements
4. Quantify the impact of installing new packaged releases
5. Gauge the quality of outsourced work

### **How do you measure the complexity and maintainability of your software?**

The ISO Software Quality Model defines characteristics of Maintainability as a key part of quality:

- Analyzability – the ability to locate and scope features or faults within the code
- Changeability – the effort required to make changes to the software
- Stability – the likelihood that changes to the software will result in defects
- Testability – the effort required to test changes to the software

These characteristics have a large, but typically unmeasured impact on your team's work.

How do your programmers deal with these challenges? Cognitive scientists who study software maintenance break down the mental processes into areas such as learning control flow, data flow, beacon recognition and statement chunking.

**Curious about your maintainability metrics?** For more information on how to measure the maintainability of your software and integrate it with your plans and operations [click here for our free white paper on management with software metrics.](#)

Let's look at some real world examples of how these metrics are being used in some organizations today.

## Five Ways To Use Software Complexity Metrics

### 1. Develop more quantifiable programming estimates and project plans

Typical practice in midrange shops is for a manager to give some project requirements to a programmer and ask for an estimate. The manager then gets an estimate:

Program To Modify	Hours To Modify
CU150R	40
IT222R	16
OE930R	50
PR111R	12

Is it right? What's it based on? Is this based on something more than gut feel? How about getting something like this instead:

Program To Modify	Hours To Modify	Subr To Modify	Max Depth of IF/DOs	Fields Set in Subr	LOC in Subr	LOC in Largest IF/DO Block	Files Updated
CU150R	40	WRTCUS	1	5	34	3	1
OE930R	50	LODORD	7	23	81	22	0
IT222R	16	CLCINV	11	54	495	217	5
		OUTORD	3	21	99	31	1
PR111R	12	PRTALL	2	17	37	5	0

**Green heading information supplied by metrics system such as Databorough's X-Audit.** (LOC = Lines Of Code)

**Overestimated?** Look at that first program, CU150R. An estimate of 40 hours to modify what looks like a very simple little subroutine which has 34 lines of code and 1 IF block of 3 statements. Depending on the change requirements this may or may not be reasonable, but a manager can quickly assess whether or not this section of code is difficult to change or simple.

**Underestimated?** Conversely, look at the third program, IT222R. An estimate of 16 hours to change what looks like a very complex subroutine – IF's nested to a depth of 11! An IF block of code spanning 217 lines, and 5 files updated. This is challenging code. Perhaps the programmer has an isolated, simple change in mind, but this certainly warrants some confirmation.

## 2. Quantify the barriers to modernization

You want to re-engineer your application to either more modern service oriented RPGLE or even another language. Can you just convert the code? Or is it so complex that it will be even more difficult to maintain? Or, if you want to extract the core logic and rewrite it manually, how difficult will it be to untangle?

How can you make this assessment across an entire system?

Program To Convert	Overall Complexity	Total LOC	LOC of Largest Subr	LOC of Largest IF/DO Block	Greatest Depth of IF/DO	Decision Density	Average Variable Span	Nbr Actual Global Variables
OE333R	100	7,214	542	309	11	100	35	119
PO219R	97	8,667	378	218	9	99	29	95
IT901R	97	8,012	521	244	8	95	30	107
PO317R	96	5,476	198	190	10	91	25	108
CU557R	96	7,122	323	210	6	89	25	94

This listing uses a custom metric, overall complexity, and ranks the programs top to bottom based on that. The other columns show some of the individual metrics that go into the overall metric.

It is also useful to drill down and show these kinds of scores by subroutine. In many cases it is a handful of specific subroutines that have most of the complexity and present the challenges when converting to functions in new languages.

## 3. Clarify and quantify for users the challenges of difficult enhancements

How well do users and management understand the scope of the challenge IT managers face on each project? How can it really be quantified?

Again, the complexity of maintaining and enhancing systems comes from the complexity of the code base. Providing users and management with factual, quantifiable information about the programs being modified puts IT managers on more solid footing when explaining estimates, schedules, defects, setting expectations, etc.

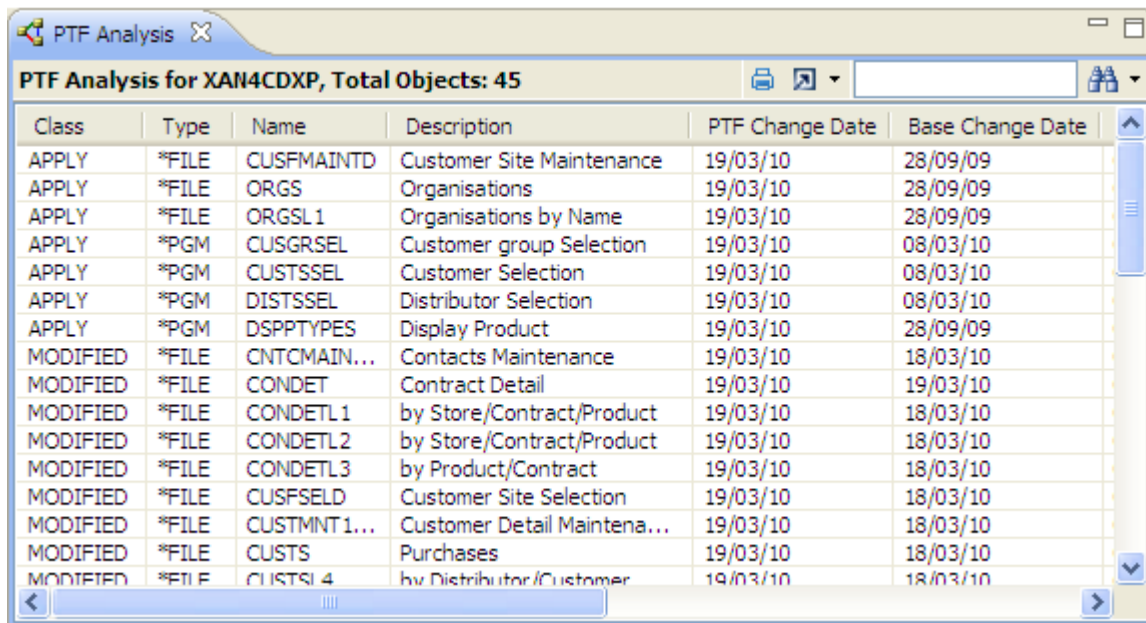
A modification of the first report shown above can help communicate the challenges and risks of a project:

Program To Modify	Hours To Modify	Overall Complexity	Risk Factor	Comments
CU150R	32	27	low	Outsourcing
OE930R	50	32	med	
IT222R	72	85	hi	Need to prepare extra user test plans; assigned top programmer
PR111R	12	12	low	Outsourcing

Some initial education is required to explain what’s behind the Overall Complexity numbers, but eventually users and management will understand the role that existing complexity plays in maintenance programming.

**4. Quantify the impact of installing new packaged releases**

Integrating a new release of a package for which you have made customizations can be a major challenge. Having a tool that automatically tell you exactly which objects need changing or review can be a real time saver and bring much higher quality deployment results.



A tool like Databorough’s X-Audit does exactly that and categorizes every object in every conceivable way. In the following categories “PTF library” refers to the new release of package changes and “customized” library refers to the customizations that have been made over time to the base package.

**Modified** - The object from the PTF library was found in one of the customized libraries. The PTF object will have to be reviewed and changes applied in the customized library must be manually applied to the object in the PTF library.

**New** - The object from the PTF library was not found in the base repository. The PTF object can be placed in the base library.

**Apply** - The object from the PTF library was found in one of the base libraries but not in any of the customized libraries. Therefore the PTF object can overlay the object in the base library.

**Refers** - The object from the PTF library refers to one or more objects in one of the customized libraries. The PTF object will have to be analyzed to make sure all customized objects referred to still meet the requirements of this object.

**Referenced** - The object from the PTF library is referenced by an object in one of the customized libraries. The customized objects will have to be reviewed to make sure the PTF object will still interface properly to the customized objects.

## 5. Gauge the quality of outsourced work

Short of reviewing all the code your outsources create and modify, how can you make quick assessments of their work?

One technique you can use is to monitor changes in code metrics and compare them to the level of change expected for assigned projects.

Project	Program To Modify	Overall Complexity	Total LOC	LOC of Largest Subr	LOC of Largest IF/DO Block	Greatest Depth of IF/DO	Decision Density	Average Variable Span
117-New tax	BL219R	1	29	12	0	0	1	0
	BL230R	0	13	0	0	0	1	1
	IF305R	-5	-59	0	0	-2	-2	-1
119-Change regions	CU290R	0	12	12	0	1	1	0
	SL300R	5	86	23	0	0	3	1

Again, green columns are generated from the Databorough X-Audit metrics system.

**Drop in complexity?** Looking at program IF305R, which was modified as part of Project 117, we see a distinct drop in complexity and lines of code. This does not fit the expectation of adding a modest amount of functionality approximately like the other programs in this project.

**Jump in complexity?** Conversely, looking at program SL300R we see a fairly significant jump of 5 in overall complexity and the addition of 86 lines of code. This project was simply to make entry of a region code optional, so this doesn't fit the expectation and requires investigation.

## Summary

Modern management is all about numbers and metrics. Quantifying how much stock we have, account balances, money owed, items on order, transaction history, policy values, and the list goes on. The investment for many IBM i shops in application software goes into tens of millions, if not hundreds of millions of dollars.

Are you doing everything you can to follow a scientific and quantifiable approach to software management?

[Click here for our free white paper on management with software metrics.](#)

*“You cannot manage what you do not measure.”*

Bill Hewlett, Hewlett-Packard