



Databorough

Five Reasons to Measure the Complexity of Your Software

Steve Kilner

Five Reasons to Measure the Complexity of Your Software

How complex is your software? What are the top 1% most complex subroutines in your system? How often do you maintain them? Why?

How do you get answers to these questions? But first, how would you benefit if you knew?

1. You can achieve *more predictability* in managing software projects if you know the level of complexity of the code being maintained
2. You can *lower the risk* of introducing defects into production if you plan for and manage software complexity
3. You can *lower software maintenance costs* if you proactively keep your software from becoming excessively or unnecessarily complex
4. You can *preserve the value* of your software asset and prolong its useful lifetime if you keep it from becoming excessively complex
5. You can estimate *when it is better to rewrite* code than to keep maintaining it

Why is complexity particularly important for legacy systems?

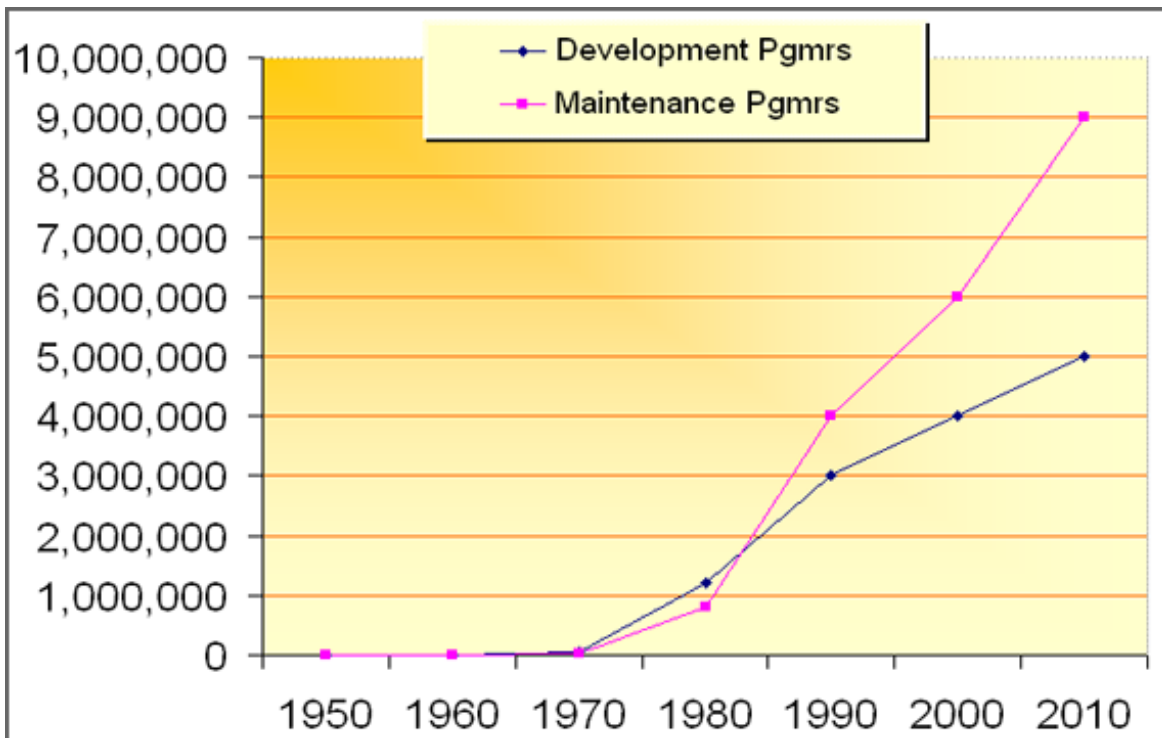
To paraphrase of a point articulated by software maintenance guru, Professor Alain April, "*the act of maintaining software necessarily degrades it.*"

It's a surprising statement when we first hear it - we tend to think that maintenance makes things better. It is surprising, yet we also recognize its reality.

Another professor, Meir Lehman, composed "eight laws of software evolution," the second of which states, "*as a system evolves, its complexity increases unless steps are taken to reduce it.*"

Yes, there are professors who specialize in software maintenance. And why not? Leading software metrics researcher Capers Jones points out that most of the programmers in the world are engaged in software maintenance, rather than new development:

Five Reasons to Measure the Complexity of Your Software



“It’s harder to read code than write it.”

I’m not sure who said that first, but it’s another of those surprising statements that we immediately know to be as true as it is surprising. Maintenance programmers, especially in the System i world, are under tremendous pressure to maintain software that often has experienced 20 years or more of complexity clogging evolution. And frequently they are dealing with code they did not write, which they have to read, understand and modify, both quickly and safely.

Measuring Complexity

“You can’t manage what you can’t measure”, so the first thing to look at it is how to measure software complexity. There have been many methods developed for measuring complexity and many of them are language independent and can be adapted to legacy System i code, such as RPG.

The basis of all these metrics lies in the capacity and limitations of the human mind to engage in symbolic processing. For learning and manipulating software that is to be maintained, cognitive scientists first break down programmers’ mental processes into a couple of very high level tasks: understand a program’s control flow, and understand a program’s data flow.

Can we look at a program’s source code and make a determination of how hard it will be for a programmer to understand its control flow or data flow?

There are indeed some theory based metrics that have been devised for this:

Five Reasons to Measure the Complexity of Your Software

Cyclomatic complexity is a measurement of how much control flow exists in a program. In RPG, control flow is represented by such operation codes as IF, DO, SELECT, etc. We all know that a program with more conditional logic is more difficult to understand, and this metric gives us an assessment of that.

Halstead volume is a measurement that attempts to quantify how much “information” is in the source code, and thus must be learned – this partially relates to data flow. It looks at how many variables are used, and how often they are used. It does the same for functions and operation codes, as those are additional pieces of information of programmer must learn. In short, it is a measurement of information “volume”.

Maintainability index is a measurement that attempts to formulate an overall score of how maintainable a program is. It has a more empirical basis and was developed over a period of years by consultants working with Hewlett-Packard and the practical experience of their software teams. It makes use of the above two measurements as well as number of lines of source and number of lines of comments.

Also useful are more down to earth measurements that relate more specifically to the System i. Examples of these are counts of various elements such as the number source lines, files, displays, copy books, etc. used by a program. These types of insights are useful for identifying special case programs with high numbers of such elements.

Is there anything that can be done about this tendency of software to become more difficult to maintain over time?

Yes: Next article - How to manage application Complexity