The Mathematics Behind the Complexity Score

In order to understand the complexity of any environment, it is helpful to quantify it.

Let C stand for complexity, D represent dependencies between functional units and F, functional units. For Information Technology, an example of a functional unit can be a component of software, an operating system, a driver, a patch, a network board, a storage unit and so on or all of the above. The model below provides a means to measure and understand complexity independent of the actual type of the functional units, hence comparing apples to oranges.

If you assume that a 25% increase in functionality leads to twice as much complexity [1] then you can create a simple algorithm of the form:

$$C = F^3 + D^3$$

If you have one functional unit, F, and therefore no dependencies then the simplest level of complexity is C = 1. As our functional units increase, say, a database (1), a middle tier (2), the application tier (3) and they all run on one piece of hardware (4), then you have four functional units and four dependencies just to keep it simple. Complexity can be calculated as $4^3 + 4^3 = 64 + 64$ or 128. As the number of functional units and the dependencies between them increase, you tend towards more and more complexity. While the quantification of functional units and the dependencies between them are subjective in nature, you can see how quickly complexity increases:

FUNCTIONAL UNIT	COMPLEXITY
20	8,000
30	27,000
40	64,000
50	125,000
60	216,000
100	1,000,000

Table 1: Functional units and their corresponding complexity score.

Note in Table 1 how fast complexity or costs increase with moderate increases in functional units. Increase or decrease your complexity and you will increase or decrease your cost.

But where did this formula come from? Once again our governing assumption is:

A 25% increase in functionality corresponds to a doubling of complexity or:

$$2 \times C = 1.25 \times F^x$$

and complexity has an exponential relationship with functional units:

$$C = F^x$$

Let's solve for x.

Take the log of each side of each equation:

$$logC = xlogF$$

$$log2 + logC = xlog1.25 + xlogF$$

Subtracting the two equations leaves us with:

$$log2 = xlog1.25$$

or

$$x = \frac{log2}{log1.25} \approx 3$$

So adding dependencies to our complexity equation (or you could take the product):

$$C = F^3 + D^3$$

But this is only for one system. So if we sum over all systems we now have our complexity algorithm:

$$C = \sum (F^3 + D^3)_n$$

As Julius Caesar wrote, “Fortune… can bring about great changes in a situation through very slight forces”, so too with complexity.

[1] Robert Glass, Fact’s and Fallacy’s about Software Engineering