Reliability vs. redundancy: Complexity is the enemy

Spiceworks, Inc.

10-12 minutes

This is the 164th article in the Spotlight on IT series. If you'd be interested in writing an article on the subject of backup, security, storage, virtualization, mobile, networking, wireless, DNS, or MSPs for the series PM Eric to get started.

It is very easy when working in IT to become focused on big, complex solutions. It seems that this is where the good solutions must lie — big solutions, lots of software, all the latest gadgets. What we do is exciting and it is very easy to get caught up in the momentum. It's fun to do challenging, big projects. Hearing what other IT pros are doing, how other companies solve challenges and talking to vendors with large systems to sell to us all adds to the excitement and it is very easy to lose a sense of scope and goal and it is so common to see big, over the top solutions to simple problems that it seems like this must just be how IT is.

But it need not be. Complexity is the enemy of both reliability and security.

Unnecessarily complex solutions increase cost both in acquisition and in implementation as well as in maintenance, while being generally slower, more fragile and possess a large attack surface that is harder to comprehend and protect. Simple, or more appropriately, elegant solutions are the best approach. This does not mean that all designs will be simple, not at all. Complex designs are often required. IT is hardly a field that has any lack of complexity. In fact, it is often believed that software development may be the most complex of all human endeavors, at least of those partaken of on any scale. A typical IT installation includes millions of lines of codes, hundreds or thousands of protocols, large numbers of interconnected systems, layers of unique software configurations, more settings than any team could possibly know and only then do we add in the complexity of hundreds or thousands or hundreds of thousands of unpredictable, irrational humans trying to use these systems, each in a unique way. IT is, without a doubt, complex.

What is important is to recognize that IT is complex — this cannot be avoided completely — but to focus on designing and engineering solutions to be as simple, as graceful... as elegant as possible. This design idea comes from, at least in my mind, software engineering where complex code is seen as a mistake and simple, beautiful code that is easy to read and understand is considered successful. One of the highest accolades that can be bestowed upon a software engineer is for her code to be deemed elegant. How apropos that it is attributed to Blaise Pascal, after whom one of the most popular programming languages of the 1970s and 1980s was named is this famous quote (translated poorly from French): "*I am sorry I have had to write you such a long letter, but I did not have time to write you a short one.*"

It is often far easier to design complex, convoluted solutions than it is to determine what simple approach would suffice. Whether we are in a hurry or don't know where to begin an investigation, elegance is always a challenge. The industry momentum is to promote the more difficult path. It is in the interest of vendors to sell more gear not only to make the initial sale but they know that with more equipment comes more support dollars and if enough new, complex equipment is sold the support needs stop increasing linearly and begin to increase geometrically as additional support is needed not just for the equipment or software itself but also for the configuration and support of system interactions or additional customization The financial influences behind complexity are great, and they do not stop with vendors. IT professionals gain much job security, or the illusion of it, by managing large sets of hardware and software that are difficult to seamlessly transition to another IT professional.

Often complexity is so assumed, so expected, that the process of selecting a solution begins with great complexity as a foregone conclusion without any consideration for the possibility that a less complex solution might suffice, or even be superior outside of the question of complexity and cost itself. Complexity is sometimes even completely tied to certain concepts to a degree where I have actually faced incredulity at the notion that a simple solution might outperform in price, performance and reliability a complex one.

Rhetoric is easy, but what is a real world example?

The best examples I see today are mostly related to virtualization whether vis-à-vis storage or a cloud management layer or software or just virtualization itself. I see quite frequently that a conversation involving just virtualization for one person brings an instant connotation of requiring networked, shared block storage, expensive virtualization management software, many redundant virtualization nodes and complex high availability software — none of which are intrinsic to virtualization and most of which are rarely for the purpose of supporting or really, even in the interest of the business for whom they will be implemented. Rather than working from business requirements, these concepts arise predominantly from technology preconceptions. It is simple to point to complexity and appear to be solving a problem — complexity creates a sense of comfort.

Filter many arguments down and you'll hear "How can it not work? It's complex." Complexity provides an illusion of completeness, or having solved a problem, but this can commonly hide the fact that a solution may not actually be complete or even functional — but the degree of complexity makes this difficult to see. After this, our minds will not easily accept a simpler approach being more complete when a complex one is not. It feels counter-intuitive.

A great example of this is that we resort to discussing redundancy rather than reliability. Reliability is difficult to measure; redundancy is simple to quantify.

A brick is highly reliable, even when singular. It does not take redundancy for a brick to be stable and robust. Its design is simple. You could make a supporting structure out of many redundant sticks that would not be nearly as reliable as a single brick. If you talk in reliability — the chance that the structure will not fail — it is clear that the brick is a superior choice to several sticks. If you say, "There is no redundancy. The brick could fail and there is nothing to take its place," you sound silly. But when talking about computers and computer systems we find systems that are so complex that rarely do people see when they have a brick or a stick and so, since they cannot determine reliability, which matters, they focus on the easily to quantify redundancy, which doesn't. The entire system is too complex, but seeking the simple solution — the one that directly addresses the crux of the problem to solve — can reduce complexity and provide us a far better answer in the end.

This can even be seen in RAID. Mirrored RAID is simple — just one disk or set of disks being an exact copy of another set. It's so simple. Parity RAID is complex with calculations on a variable stripe across many devices that must be encoded when written and decoded should a device fail. Mirrored RAID lacks this complexity and solves the problem of disk reliability through simple, elegant copy operations that are highly reliable and very well understood.

Parity RAID is unnecessarily complex making it fragile. Yet in doing so, and by undermining its own ability to solve the problem for which it was designed, it also, simultaneously, becomes seemingly more reliable based on no factor other than its own complexity. The human mind immediately jumps to, "It's complex, therefore it is more advanced; therefore it is more reliable." But neither progression is a logical one. Complexity does not suggest that it is more advanced and being advanced does not suggest that it is reliable, but the human mind itself is complex and easily mislead.

There is no simple answer for finding simplicity. Knowing that complexity is bad by its nature but unavoidable at times teaches us to be mindful. However, it does not teach us when to suspect over-complexity. We must be vigilant, always seeking to determine if a more elegant answer exists and not accept complexity as the correct answer simply because it is complex. We need to question proposed solutions and question ourselves: "Is this solution really as simple as it should be? Is this complexity necessary? Does this require the complexity that I had assumed?"

In most system design recommendations that I give, the first technical determination step that I normally take, after the step of inquiring as to the business need, is to question complexity. If complexity cannot be defended, it is probably unnecessary and actively defeating the purpose for which it was chosen.

"Is it really necessary to split those drives into many separate arrays? If so, what is the technical justification for doing so?"

"Is shared storage really necessary for the task that you are proposing it for?"

"Does the business really justify the use of distributed high-availability technologies?"

"Why are we replacing a simple system that was adequate yesterday with a dramatically more complex system tomorrow?

What has changed that makes a major improvement, while remaining simple, not more than enough but requires orders of magnitude more complexity and more spending that wasn't justified previously?"

These are just common examples. Complexity exists in every aspect of our industry. Look for simplicity. Strive for elegance. Do not accept complexity without rigorously vetting it. Put it through the proverbial ringer. Do not allow complexity to creep in where it is not warranted. Do not err on the side of complexity. When in doubt, fail simply. Oversimplifying a solution typically results in a minor failure while making it overly complex allows for a far greater degree of failure. The safer bet is with the simpler solution. And if a simple solution is chosen and proven inadequate it is far easier to add complexity than it is to remove it.

Share your thoughts in the comments below!