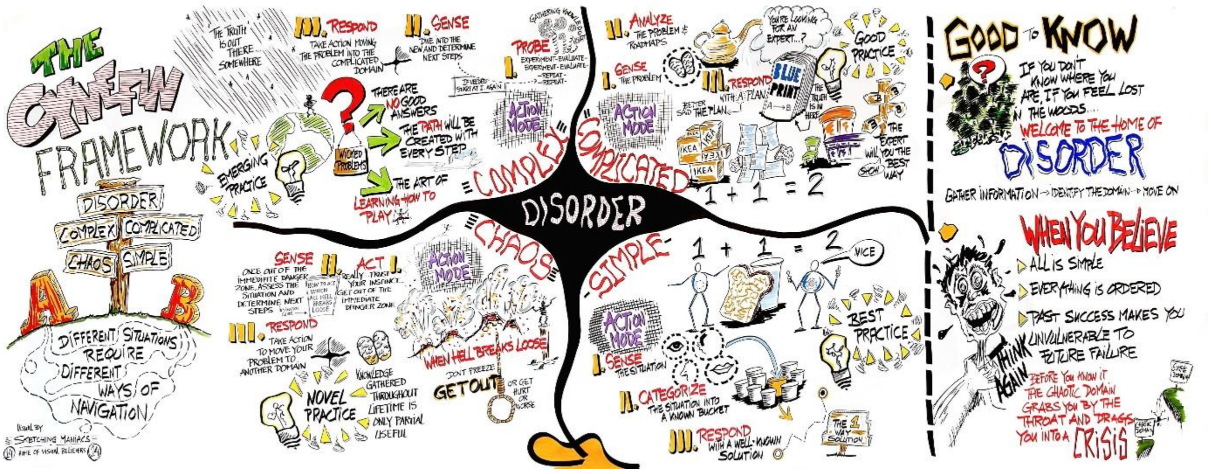
The Rise and Fall of Overcomplicated Systems

Shôn Ellerton, September 12, 2019

Let's take a journey of how overcomplicated systems come into being, why they frequently fail and how to avoid them. It's time to embrace simplicity.



THE CYNEFIN FRAMEWORK
DISORDER
COMPLEX
COMPLICATED
CHAOS
SIMPLE
DIFFERENT SITUATIONS REQUIRE DIFFERENT WAYS OF NAVIGATION
EMERGING PRACTICE
THERE ARE NO GOOD ANSWERS
THE PATH WILL BE CREATED WITH EVERY STEP
THE ART OF LEARNING HOW TO PLAY
PROBE
SENSE
RESPOND
ACTION MODE
ANALYZE
GOOD PRACTICE
BLUE PRINT
1 + 1 = 2
VICE
BEST PRACTICE
CATEGORIZE
RESPOND WITH A WELL-KNOWN SOLUTION
ACT
NOVEL PRACTICE
WHEN HELL BREAKS LOOSE
GET OUT
GOOD TO KNOW
IF YOU DON'T KNOW WHERE YOU ARE, IF YOU FEEL LOST IN THE WOODS... WELCOME TO THE HOME OF DISORDER
GATHER INFORMATION → IDENTIFY THE DOMAIN → MOVE ON
WHEN YOU BELIEVE
ALL IS SIMPLE
EVERYTHING IS ORDERED
PAST SUCCESS MAKES YOU INVULNERABLE TO FUTURE FAILURE
THINK AGAIN
BEFORE YOU KNOW IT THE CHAOTIC DOMAIN GRABS YOU BY THE THROAT AND DRAGS YOU INTO A CRISIS

Many of us advocate that simplicity strives to thrive and complexity tends to die. We unearth ancient buildings of the simplest design while complex structures rot into a state of dangerous disrepair. Old Mercedes 280E cars from the 80s still straggle along in deepest darkest Africa because they're easy to repair using simple and standard parts unlike many of today's modern cars. One can even learn a great deal from our past wars favouring the use of simple over complex weapons and machinery. Longbows vs crossbows. AK-47 vs M16 submachine guns. De Havilland Mosquitoes vs Messerschmidt Me-109 aircraft. As with the examples cited above, IT systems which are simple in design and easy to maintain can survive for a very long time. Moreover, they are cheaper and quicker to build and often meet their intended purposes.

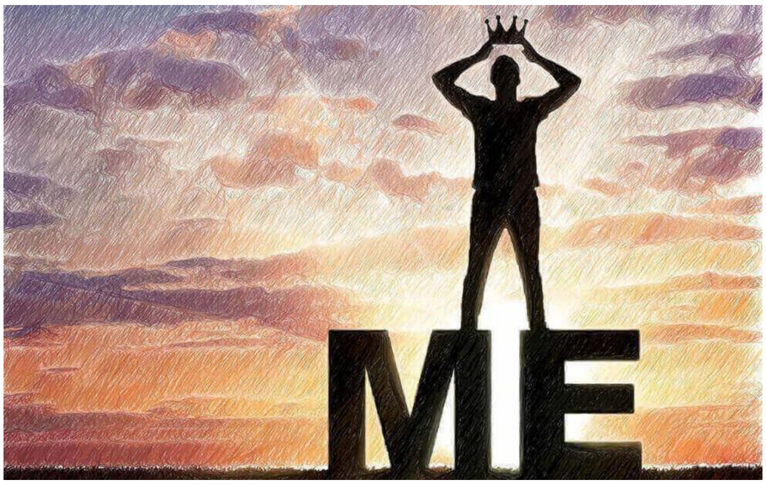
Back in early 2014, a small team comprising me, a lead developer, a C# coder and a reporting analyst put together a system required to service a national mobile base station rollout project in Australia from 2013 to 2018. The system supported several disciplines across the project from financials, town planning, property management, radio planning, logistics and milestone management and needed to cater for up to 700 users. At first, it was a little crude, but from day one, it was a *functioning* product and it worked. Our team was nimble enough to make the changes required to mature the system.

Meanwhile, in the upper echelons of IT management, another project was heralding its way back in 2014 to create an '*all-singing, all-dancing*' work management system which was to service all projects and all clients in the business. During 2017, this system was still under development and nowhere close enough to being used as a working product while racking up significant costs twenty times the amount of what was required to build the simpler system our team worked on.

Complex systems *do* have their place; however, unless there is a compelling reason for not choosing a simpler solution, why do so many IT projects implement overcomplicated solutions?

The ego of the architect

The word, *architect*, conjures divine-like powers of creation. In my civil engineering days, my colleagues and I would often liken the typical architect to be that arrogant individual with a massive ego, an individual who gets all the credit for designing the world's great structures, yet, not having the knowledge required to physically build them. We were, of course, students of civil engineering who ourselves, were at the mercy of being on the butt-end of jokes centred around crassness, simplicity, being classically uneducated along with the inability to form words with more than two syllables!



Enter the world of IT where the *data, enterprise and solution architects* are the designers and creators of information technology frameworks. Coming from an engineering and project-management background, my debut into data architecture was markedly different from the many others who started their careers with backgrounds of having computer science and IT degrees.

I love simplicity. If it works using basic technologies; why design a system which only the elitist few can understand?

This, herein, lies the problem. For many data architects, the very notion of designing a system with bog-basic standard parts which any individual with a modicum of IT skills could dissect is almost unthinkable.

I remember an occasion where we brought in the company's enterprise architect to advise on how we could integrate our project tracking system into the overall corporate system. We also invited the project's delivery-manager, an outspoken highly practical telecommunications engineer originally from Germany who despised anything overcomplicated.

When the enterprise architect arrived, he herded us into a glass-panelled meeting room and immediately commenced with sketching endless flow and data architecture diagrams with a black texter, admonishing us on how we should have originally built the system. He talked incredibly fast seemingly without taking one single breath. None of us managed to get a complete sentence sideways throughout the one-way engagement. After the 'lecture' was given to us and the enterprise architect went on his merry way home (by jet), our delivery manager, dumbfounded, scratched his head and commented in a very dry and amusing way complete with thick German accent, 'Who the f… was that clown?'.

Needlessly to say, we didn't take much of anything useful out that session. Certainly, not all solution, enterprise or data architects are like this, but a great many of them are from experience.

> ***Why design a system which only the elitist few can understand?***

Stranded on an island with little water

Let’s dream up a scenario where we have fifty or so people stranded on a little island in the Pacific Ocean. Water and food supplies are running out. 20 miles away is a much larger island with all the food and water needed to sustain everyone for months. On the island which the stranded people are on, there is enough material to make boats or rafts to escape the island and flee to the much larger one.

Now, the elected leaders of the stranded group have three choices to make.

1) Build a large number of small rafts each capable of holding three or four people;
2) Pool all resources together to build a much larger boat to accommodate everyone; or
3) Sit and wait for rescue.

Each option has its own risks.

The first option is straightforward enough. Building a raft to support three or four people is not technically difficult and each one can be made near-identical to each other. Small rafts; however, are very much at the mercy of the sea and there could be a high risk of some of the rafts not making it to the larger island either from being destroyed or those on board getting lost straying from the group. However, the probability of *some* of the rafts making it is quite high.

The second option is technically challenging. In this scenario, the deadline of building the boat is, effectively, running out of water. More than half the time will probably be spent in designing the boat; at which point, morale is probably sinking to an all-time low. Running out of time *before* a single piece of the boat is put together is a real risk. If the group *did* manage to build the boat and build it correctly; certainly, the risk of the sea destroying it is far reduced and the likelihood of all reaching the island in safety is high. When deadlines are critical, building a complicated solution is extremely risky.

The third option is a total gamble. If the group is aware of the larger island being relatively near, it is unlikely that no attempt would be made to make way to the sanctuary of the larger island.

This scenario suggests that as timeframes become increasingly critical, the solutions put forward become increasingly simple.

> ***When deadlines are critical, building a complicated solution is extremely risky***

Wait… I forgot what we're doing this for!

Whether designing a website, a car or a wheelbarrow, if nobody can picture what the final product will look like and what its primary purpose is, then the project is likely to fail usually due to unintended complexity being built into the product to cater for unknown variances in the final product.

Many design offices display posters or drawings of their end-product to remind and reassure those working on the various design elements of what they are aiming to achieve. With complex products, teams are often broken down into sections to achieve completion of a small part of the complete product; however, they must all work together in unison with the same vision. If building a website, for example, the first and foremost task is to create a visual wireframe of the proposed website *and* show it to the entire team.

Not having a clear vision of the purpose and proposed outcome of any project often leads to confusion and chaos. Furthermore, an entirely different product could be spawned! I've seen my 4-year-old boy randomly put bits of Lego together which could turn out to be anything; however, if I ask what he intends to build before laying a single piece, he often has a good crack at building something resembling it.

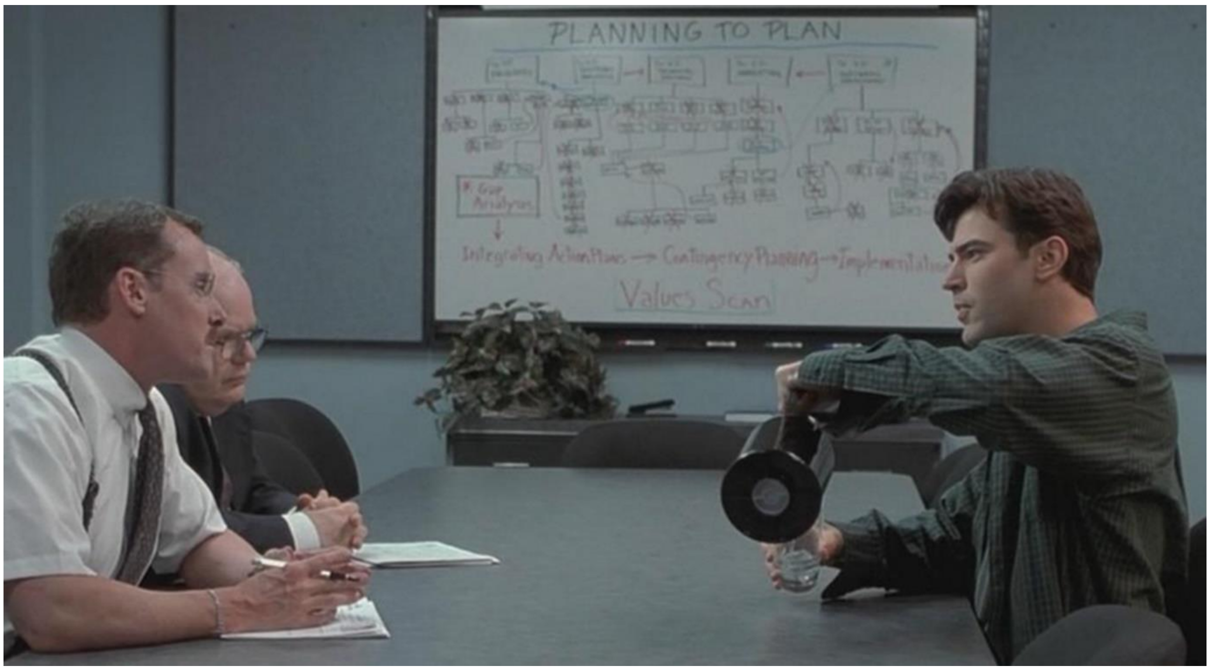
The danger of not being clear on the final product is to design the uncertainty out by creating systems more complex and complicated than they may need to be to cope with rapidly changing or unclear requirements.

> ***The clearer a project’s purpose and outcome, the easier it is to keep the system focused and simple.***

Keep it simple, stupid

The KISS principle is associated with Kelly Johnson, an aircraft engineer who was a proponent of building machinery of which repairs could be undertaken by an average mechanic in combat situations.

This principle is one of the soundest principles that can be applied to just about anything in life. And this, of course, includes IT systems.


PLANNING TO PLAN
Integrating Action Plans → Contingency Planning → Implementation
Values Scan

During mid-2019, I had a short-term contract with an organisation servicing physically and mentally disabled patients. My job was to integrate data from other agencies and institutions into the organisation’s own system. The previous incumbent, another data architect, had to leave at very short notice and I was called on to ‘fill the gap’.

Thankfully, he developed and documented a simple solution which most data professionals would have no problem in understanding. Moreover, he used bog-standard well-known products in his solution. The solution may not have been the most elegant of all solutions, but it worked admirably, and I did not have to spend an inordinate amount of time to unravel some custom funky bit of software riddled with obscure and arcane software technologies.

Toys, toys, and more toys!

Every year that passes by, a bucketload of new software and technologies battle with each other to earn a spot in the IT marketplace offering ease of use and a panacea to fix or remedy problem systems. Many of these products command incredibly high prices.

For example, Alteryx, an advanced BI analytics tool, has some of the highest pricing I've come across with four to five-figure annual licences *per user*. Many data professionals laud the tool as being a saviour in simplifying complex data analysis and ETL processes.

Snowflake, a cloud-based service aspires to really simplify data warehousing and at an attractive price. I bumped into an acquaintance of mine who stated that they are proposing to migrate their Amazon Redshift services to Snowflake.

BI reporting tools have proliferated to the extent that project teams often chew up huge chunks of time on a project deciding which one to go for. The two most well-known heavyweights, Power BI and Tableau, seem to get the lion's share of attention, but there are so many others vying for attention in this most-saturated of marketplaces.

All these products intend to make life as simple as possible for the end user. If chosen wisely, taking into costs, scalability and serviceability into account, all is good. However, mixing and matching a plethora of different products and never quite deciding which product to use when the project is already at its mature stage of its development, can be very costly. ‘One-stop shops’ are great in principle, but in many cases, complex ‘Franken-systems’ are spawned.

Just like Christmas time, when kids get spoiled for choice and get inundated with presents, they often don’t know what to do with many of them.

Simplicity thrives, complexity dies

It never ceases to amaze me how infrequent simplicity is encouraged on IT projects. I’ve been very fortunate in my career that I’ve been instrumental in making decisions to develop systems to service projects which are simple to use *and* simple in design. And here’s the kicker. Many of them *still* work to this day. They were built on standard technologies, with standard principles, modular enough to be taken apart at ease, fully documented and easily serviced by low to medium-experienced IT professionals.

On the polar extreme, I was once asked to ‘lift and shift’ somebody else’s piece of software that was used in another organisation and then implement it into another project. Due to its complexity along with poor documentation, I advised that, perhaps, a more simple and manual approach might be a better option considering the aggressive timescales involved. I further reinforced this when I discovered that this software, as designed and used for the original company, was being dismantled after the brainchild that created it departed. I can only presume that the reason that the company dismantled it was on grounds of risk of not being able to be maintained by their own IT staff.

For those of us who build databases on a frequent basis, we are accustomed to making the decision of simplicity vs complexity. I once got caught in this trap in my earlier years by creating a database that was over-normalised, over-generic and overly dynamic. Man, did I get into a heap of trouble when the client made a major change in requirements! I practically had to dismember it and start from scratch; but this time, with a more balanced pragmatic approach in its design.

> ***As with nature, the laws of entropy apply to systems design. They gradually decline into disorder if not maintained.***

Perfection is the enemy of good

Nothing runs to perfection unless it’s the punctuality of a Japanese bullet train or if one is running a hospital with no patients in it as so comically portrayed in one of the episodes of *Yes Minister*, a popular British political comedy series from the 1980s. Perfection simply *does not* exist in any project.

> ***A system which might have started crudely but has evolved gracefully to adapt to changing conditions will be far more useful than one which is destined to perfection but still on the drawing board when it is needed.***

All too often, so many in the IT industry attempt to design systems of perfection that *anybody* can use with little or no training; systems that are fully automated, dynamic and self-healing; and systems that require virtually no human interaction to maintain them. There has not been one project that I have worked on that has had a system with all or any of these qualities.

Striving for perfection in any system can easily lead to overcomplicating *and* compromising it.

Something doesn't seem right…

One of the most insidious aspects of complex systems is the sheer difficulty of verifying results from first principles. In the case of reporting systems, this is a real danger. Although every effort may have been made to preserve data lineage in any given reporting system, it becomes atrociously difficult to dissect the process of how the data was derived in complex systems. Vast amounts of time and resources can easily get used up to verify and identify problems within complex systems. Moreover, if the results are wrong, the first question most anybody would ask is for how long? And if the results are wrong, *where* did it go wrong? Can it be fixed easily? A system comprising of simple parts can often be easily fixed whereas a complex system, often monolithic in design, comprising an assembly of complicated code with an array of bespoke or obscure software is often not.

It is incredibly easy to rely on data generated from complex systems as verbatim and true. The most alarming example of such an occurrence was during my civil engineering graduate years designing bridges. I used software to undergo a series of finite element calculations. The numerical results were spat out of a dot-matrix printer. I showed them to my boss who then said to me that I should sketch out by hand a basic moment diagram using the results. The bending moment diagram I sketched looked anything but healthy. Clearly, the error was in the input rather than the output. In this case, verifying the validity of the answer was made through common sense, basic knowledge of the principles and the ability to reverse-engineer the process. Glad that bridge was never built!

Conclusion

In 2017, I published an article, Do You Really Need That Expensive Online Work Management System, in which I weigh up the pros and cons of implementing corporate-wide work management systems. This article neatly follows on from what I wrote about in that article and highlights the case to keep systems as simple as possible.

If further complexity must be built into the system to meet end requirements, then this may be the case; however, it pays to be prudent to explore peripheral options, procedures or working practices beforehand. For example, weighing up the benefits of implementing a complex fully automated system requiring little or no human interaction versus implementing a far cheaper, simpler and less automated system which may require more regular human interaction needs careful examination. If something breaks in the new system, can it be fixed by those in the IT team with basic all-round knowledge, or would a specialist need to be hired?

Sometimes, it’s worth remembering. Less is more!