British Computer Society, 8th November 2007 Stuart Jobbins

As an engineer I would like to challenge our thinking by asking a simple question. *Does the evidence suggest that we are sufficiently prepared to realise, that is engineer, Large-Scale Complex IT Systems?*

To try and frame the context of the question and ultimately develop a personal argument that suggests we are currently unprepared, or at least under-prepared, I would like to look at 3 aspects from an embedded system and software development standpoint.

1) Firstly, where does current IT system complexity originate and how capable are we in dealing with systems we currently design
2) Secondly, how well do we judge a viable solution, given an integrity criteria
3) Thirdly, what are the models for future generation of systems of systems, and the factors that increase complexity

So, Looking first at Current system complexity

The overriding trend in 3 decades is unprecedented growth in everything electronic but significantly Information Technology. In fact it is fair to surmise that the solution has always been part of the problem. The ‘capability’ growth of electronics, both from a reliability standpoint, but more significantly from a computation standpoint has enabled engineers to provide solutions that increasingly improve the fidelity of the required operation - a direct factor in increasing system complexity - and allow us to push the boundaries of the system’s operating efficiency.

From a real-time controls perspective, complexity jumped in the move from linear (continuous) systems and their inherent constraints, to the realisation of these systems by digital construction, which allowed us arbitrary and discontinuous relationships. This was the liberating step for flexibility… but the precipice with regards complexity.

Similarly, in communications the need for autonomy and ‘healing’ in networked devices, has driven a significant increase in complexity over statically defined and deployed networks… but a feature we now expect as standard.

Unfortunately, real life is not simple… attempting to real-time model a simple fuel injector, across hydraulic, electro-magnetic and mechanical domains would involve not only the complex interaction of the systems in terms of actuation response, but the environmental factors, temperature, pressure along with the dynamic changes caused to these parameters during actuation, as well as mechanical wear-out, drift, original manufacturing tolerance compensation and lots more.

As an engineer I value simplicity. Why? I can visualise how the product achieves its goals, and more readily understand any errant behaviour (and probably expect to be able to diagnose it, or even effect a repair!). But the champions of simplicity always seem to lose out to the finesse argument – unless it crosses some immediate major cost breakpoint.

As a consumer I value a well-engineered complex product and the diversity, performance or flexibility it enables. Take the mobile phone… or is it a camera, pager, message service, calculator, alarm clock, diary, music player, broadcast radio receiver… and this list is for a simple offering! As consumers we seem to place value on the functionality – even when we don’t use it.

So what continues to drive complexity today…lets just explore a modern automotive power plant – its controlled parameter is essentially generated torque. With the traditional ‘human-in-the-loop’, the variation in torque required to achieve constant speed, yet mitigate varying conditions, is effected by pressing the throttle pedal harder.

Today the throttle mechanism is just a ‘demand signal’ for motive torque… any ‘parasitic’ losses (for example in air-conditioning, power steering, alternator load, gearbox and transmission) have to be discounted and a suitable excess generation has to be summed into the demand…It is not that we consider it unreasonable for a human to ‘finesse’ the controls over a long period, but when you consider automation (e.g. cruise control in an automotive context, or constant propulsive thrust in an aero equivalent) the purity of motive torque or propulsive thrust defines the ideal, simple, control parameter interface.

Of course given that we have real-time control we are able to deal with these changes as asynchronous transient behaviours for seamless, so-called ‘bumpless’, constant delivery by anticipatory control – even though that adds further complexity for transient behaviour.

The reality is that electronic control can deliver much more than ‘control’, it can

1. faithfully reflect limitations (multi-dimensionally, constrained to the design envelope)
2. monitor and diagnose itself (self-fulfilling as due to complexity, errant behaviour is no longer intellectually obvious)
3. enforce regulatory and legislative behaviours (police the user).

From modern electronics, but very definitely from IT, we expect increasing refinement at a lower cost with successive generations. We no longer accept ‘undiagnosed’ failure for anything other than cheap disposable commodities.

But those jumps in complexity have passed, haven’t they? Well here are some metrics on simple system growth - In automotive engine control, complexity has been doubling every 4 years since the advent of electronic control (irrespective of the mitigating actions to simplify). In aero engine control the doubling takes 7 years.

So having looked at current methods and system growth, what of our confidence in judging viable solutions? What limits our view of viable solutions? I suggest it stems from a measure that we define as reliability (judged by availability, robustness, survivability or even freedom from errant behaviour depending on the regulation of your target market).

Electronic component reliability is now almost immeasurably high. The implication being that high reliability (and high –integrity) systems are therefore characterised by the construction, design attributes, the use of the product within its design parameters and design margin… but increasingly characterised by the software solution (a product of the software design philosophy and development process).

Most software un-reliability stems from the interaction of relatively simple components in emergent behaviours – i.e. as the system is exercised through contexts that the designer had not predicted, or foreseen. Simple solutions are characterised by a small number of well-defined interfaces and coherent interface policies. Reliable systems tend to be simple (easy to validate) and the product of a uniform development philosophy (usually stemming from a single corporation, team, or even designer).

In principle, the processes of design and test that we use today characterise the residual error rates… evolutions of these processes and strict compliance allow us a modest percentage increase in integrity versus cost. High-integrity developers, typical of the security, aerospace or mass transport industries are using the strongest available mechanisms to assure themselves that they exceed their design goals.

Even then, the absolute number of errors are a function of size, but with complexity, size is rising fast, probably beyond the capability of our process improvement to mitigate it. Given this potentially weakening position, at what point do we cease to be confident in the evidence supplied for certification events such as airworthiness?

The ‘product line’ techniques give us a capability to instantiate many applications from a fixed number of basic components by skilful crafting of those components and careful configuration of variation. This reduces the high-risk (in error terms) custom proportion of the development thus reducing cost and increasing confidence in the overall product based on the provenance of the re-usable components.

So what can we predict of the future?

Can we architect systems of systems? Or do we fool ourselves into believing that we have sufficient definition to assure well-behaved interaction.

As architects defer the instantiation of their problem as long as possible, the abstract definition of components also leaves us the flexibility to deploy components of a system with little regard to the physical geography of functions, other than resolving the communication performance. But in dealing with this ‘distribution’ we have to deal with the physical environment and any loss or error introduction that is now a part of the geographic disposition, i.e. the error environment, loss of a node or its communications and how the system counters those losses or errors.

Can we make revolutionary jumps in process for increasing ‘right first time’ design, or improve the level of correctness by construction, perhaps by increasing our bias towards analytical proof, rather than post-development test, whose coverage is limited by manpower (which equates to cost and timescale) and does not readily scale.

Brute force compute approaches… synthetically trying to replicate the physical world will always be constrained by the technology available. Even if we could sufficiently synthesize the behaviour (without introducing error) we would still have to live with the approximation errors, rounding etc of the underlying machines.

So can we look to other examples? Are there more mature industries that have already embraced this burgeoning complexity?

Consider a single building in the construction industry as a system, we see that there is use of well-defined basic materials from many different suppliers, in well defined use-cases, with gated reviews (inspections) to legislated quality standards (building controls) both local and national, depending on local environmental conditions (earthquakes, hurricanes etc). We see specialist trades working on integrating solutions which are defined in detail by architects and underwritten by civil engineers to significant design margins (The twin towers were designed to withstand the impact of a jet airliner... but as conceived at the date of their design!).

If we expand this idea to systems of systems i.e. towns or cities, we see continuous re-generation, because of wear-out or obsolescence. Do we really architect the fundamental services (water, sanitation, power) for a design horizon to span centuries? The reality is that we compromise on the efficiency of individual systems and the total solution to maintain the integration (London streets being a prime example). Brunel for instance is rightly revered, but shouldn't it be for his foresight, as well as his engineering skill?…. This is the trait of a true System Architect.

Larger scale systems imply more concurrent activity, more suppliers and different implementations. The effect of commercial competition and collaboration is probably easy for us to recognise in the construction domain – system solution being a trade of price, performance and materials. Would this really be any different in significant IT solutions?

Finally, what can nature teach us? Nature builds some of the most complex organisms we know. Most are customised to a particular environment; they are highly 'evolved', built on countless iterations, ruthlessly discarding failures, continuously mutating to find the best adaptation to its environment. The natural process takes many 'generations' with very few species actually ever truly stabilising. Prototypes are allowed to evolve in parallel to select optimal solutions, or all may be discarded if the mutation's flaws outweigh its advantages at the time of assessment!

Layered on top of the basic organism, nature teaches us that some behavioural attributes are trained, largely by repetition, without detailed understanding of the individual component interactions – just ask any golfer about his grip, stance, swing etc. A golfer is unlikely to comprehend the scientific 'trajectory' problem, even if he had the formal mathematics with which to compute a solution, would it be reasonable to believe he could calculate such a solution and deploy the actuators (hands, arms, legs) sufficiently fast enough to resolve the problem, especially when aggravated by a poorly sensed and inconsistent environmental variability (wind, humidity)? The breadth we call experience or instinct.

Given that we can generate potential solutions, replicate, mutate and iterate far faster than natures programmes, would it not seem reasonable that 'goal seeking' solutions hold some value? Is stability a un-natural goal? Nature, it would appear, is way ahead of us.

So, in summary, why do I feel we are unprepared? I suggest the empirical evidence is that the complexity of current systems is at the limit of, or even accelerating beyond, our 'comfort zone' from classic education and experience. Our ability to be confident in the solutions we generate, is not keeping pace with complexity growth. Larger IT systems suggest large collaborative, dispersed, commercially differentiated development teams, with an attendant jump in the factors that, in our current thinking, traditionally skew development comprehension, compounding risk.

We need to find some revolutionary, novel jumps and stretch the thinking of our engineers.

Thankyou.