About a love-hate relationship with complexity

~Erik Philippus

Although I'm currently surrounded by complicated devices and systems with an unprecedented abundance of functionality, I sometimes think back nostalgically to more convenient arrangements of my life. To control my home cinema set with Dolby surround sound en satellite receiver, I have 5 different remote control units at my disposal. I have just counted the total number of controls: 203 buttons, for most of which I have only a vague impression of their function. Only a few buttons are worn-out: apparently the majority of options is lost on me. Our new car has a speech-controlled phone, music centre and navigation system, and especially at night the soft illumination of all buttons around me sometimes gives the impression that I'm navigating a plane instead of a car.

I have to admit straight away that most of the time, I enjoy these achievements of technology. Under the condition that the equipment is operating as expected, it brings me a lot of comfort, and also new ways to express my creativity. But when I have to spent many hours to get my new HDV handycam recognized by my computer, or when I discover that I'm trying to switch on the TV with the DVD remote controller (…), I sometimes wonder where we're heading.

So far I have addressed just a few domestic worries - what about the large, complicated software-intensive industrial systems that we are designing, building and using today? Are we at the edge of becoming outstripped by the complexity of advanced systems?

The creation of these intricate systems is often a multidisciplinary enterprise, requiring an accumulated investment of several hundred years of work. The software at the core of these systems needs to comply to high standards, especially when health or safety issues are at stake. Apart from the perhaps more philosophical question how (and whether) human kind is best served with all these high-tech accomplishments, I like to focus on the way we are dealing with the rapidly increasing complexity in demanding software development projects.

My point of departure is that we already create complex software systems that exceeds the human measure, and that there is an intricate and largely unrecognised interrelationship between the complexity of development environment and delivered products. This essay is an invitation for software/system architects to explore the idea that the struggle to minimize complexity and to maximize predictability as advocated by most development methodologies, is actually a lost battle and actually contra-productive.

Let's start our journey with a closer look at the concept of complexity. In everyday language, complexity refers to the degree to which a system has a design (in particular the user-interface) and/or implementation that is difficult to comprehend and verify. This is not an accurate definition of complexity, however. The difficulty in finding the appropriate buttons on my collection of remote controls, doesn't necessarily make my home cinema system a *complex* system – it is a *complicated* system at the most. In fact it is rather the opposite: most complex systems are governed by apparently simple rules, which can give rise to incomprehensible (and sometimes breathtaking) behaviour. In other words: an intricate (static) appearance on the outside is not a necessary condition for, or even an indication of complexity.

So what is complexity? Of course the term *complex system* is subjective and therefore interpretive. The root of the Latin word 'complexus' is 'to plait or twine', which correctly suggests that a complex software system consist of a large number of mutually interacting and interwoven parts. In the context of this article, I like to define a complex software system as: 'a high dimensional software-intensive system with many strongly coupled degrees of freedom, where local and global phenomena interact in complicated, often nonlinear ways'.

An interesting property of most complex systems is *emergence*, which refers to understanding how collective properties arise from the properties of the parts. Emergence is the process of complex pattern formation from basic constituent parts, and manifests itself as an irreducible property of the relationships between those elements. Hence, for a behaviour to be termed emergent it should be unpredictable from a lower level description. The occurance of emergence is sometimes debatable in case of complex software systems with their intricate underlying processes. Indeed we must guard against misusing the concept of emergence in lieu of a satisfying explanation of uncomprehended phenomena.

Here I like to add that when we say that 'the whole is greater then the sum of its parts', we have the tendency to forget that we assign properties to a system that are really properties of the relationship between a system and its environment. It would be more realistic to use the phrase 'dynamic ecosystems', since the boundaries of a complex software system are in fact arbitrary. Nevertheless, most software development methodologies have a stubborn exclusive preference for the *reductionist perspective*, based on the belief that every system can be understood (and composed) completely by analysing it in terms of its smallest parts, largely ignoring emergence and the often intricate and subtle interactions with its environment. Reductionism is of course an essential element of understanding our world – but the scale shouldn't tip toward the unrealistic desire to predict global system behaviour by observing the behaviour of individual software components. Another, related flaw is the attempt to describe the overall system with rules and principles belonging to a lower level of abstraction. This 'human habit' of trying to find a common denominator for different views on a system is not uncommon, but this process suffers from the tendency to remove useful abstractions and therefore can even add unnecessary complexity.

An even more relevant characteristic of complex systems is the phenomenon of *sensitive dependence on initial conditions*. Sometimes this is referred to as 'the butterfly effect', which means that an ostensible miniscule deviation of a critical parameter can eventually cause unexpected (and unsafe) system behaviour. Therefore, the key to predictability of complex system behaviour is accurate knowledge of these critical parameters. Especially for embedded software systems which are driven by the (never 100% accurate) measurement of physical parameters, this symptom can have significant influence on stability and reliability. Confronted with unexpected and unintelligible system behaviour, engineers should be on their guard for this phenomenon of extreme sensitivity on initial conditions, instead of directly blaming the design model, the methodology and/or the utilized development tools.

Apart from this 'initial value problem' and inherent emergent behaviour, complex systems theory offers more challenges for system architects. A closer look at complexity will reveal that it always sits on a thin line between order and chaos. And it appears that most complex physical, biological, economic and social systems operate in a region where the complexity is maximal. This preference for complexity has a very pregnant reason:

"the edge of chaos is the one place where a complex system
can be spontaneous, adaptive, and alive".

So most dynamic complex systems thrive at the edge of chaos, and there is no fundamental reason to believe that complex software systems are an exception to that rule. This notion has some far-reaching consequences. First of all, it provides a revolutionary outlook on the role of complexity in system architecting. The current paradigm of software development holds the belief that complexity will inevitably (and unfortunately) increase during the software lifecycle, and that a variety of measures is needed to keep complexity under one's thumb in order to keep a viable software product. Underlying assumption is that only a perfectly neat and orderly system will get away unhurt when confronted with changes during its lifetime. Within the context of the evolution of complex systems, this assertion is not tenable. Complexity is a *condition* for, and not a *consequence* of evolution. Only a system at the edge of chaos will have superior power to adapt to environmental changes, when an over-organised system only breeds habit and immutable patterns. This is comparable with the persistent pursuit in some organizations to have 'all noses pointing in the same direction', which in the real world will only happen when all employees are lying on their back.

Of course I don't argue in favour of building software system at the edge of chaos as a purpose in itself. Any intelligent fool can make systems more complex then necessary. My point is that it can be counterproductive to aim at simplicity only out of unjustified fear for complexity – which is a natural state anyway. A 'rehabilitation' of the concept of complexity is at least a position to consider in our engineering domain, where a lack of adaptability and maintainability are still responsible for failing projects, huge delays and overspending, in spite of the introduction of many promising development methods and tools the last decade.

The building of complex software systems requires more than a new methodology. It requires a new way of thinking and sense-making, in which the 'human measure' plays a prominent role. In the new paradigm, the real challenge for the development team is to reside in this region of optimum complexity, of course without ending up to chaos. Instead of putting up a frenetic defence to complexity, one should accept complexity (and associated system behaviours) an innate property of large software systems and their development environments in order to find the appropriate coping strategy.
With its roots in a mechanistic worldview, the majority of the current software development methodologies is at the end of the road. Underlying a variety of reasons for the shortcomings of the prevailing paradigm on the strategic, tactical and operational level in development organizations, is the missing capability of *adaptivity*. This is a badly needed, but currently underrated property of the deployed software systems as well as development teams (and their project leaders …).

As a conclusion, I like to break a lance for *'Agile Architecting'*. Instead of aiming for precise prediction and rigid control strategies, the system architect should acknowledge the reality of uncertainty and change as a natural state in development projects. Instead of the ongoing quest for sound cause-and-effect rules in a changing environment, there should be encouragement of a culture of emergent order rather than imposed order. The enforcement of a disciplined, process-driven methodology should be exchanged for the application of a people-oriented development process.

Erik Philippus, **I**mprovemen**T**
Erik.Philippus@improvement-services.nl
www.improvement-services.nl