

Some Thoughts on Practical Approaches for Complex Systems

Michael Ryan

School of Information Technology and Electrical Engineering

The University of New South Wales

Australian Defence Force Academy

Northcott Drive

CANBERRA ACT 2600

Introduction

The theme of this conference, “Practical Approaches for Complex Systems”, is particularly appropriate because the adjectives focus directly on the two major challenges we face in systems engineering. First, the world is inherently complex [1] and complex problems do not generally have simple solutions. Yet, we cannot specify, develop, manage and operate complex systems by using processes that are complex beyond our overall capacity to understand and manage and where the component parts challenge our intuition [2]. Such complex processes can not, and do not stand the test of time. Our challenge is therefore to be able to find practical approaches to cope with complexity throughout the whole system life cycle. As an aside, this difficulty is not confined to systems engineering, its manifestations are readily obvious in almost any field of endeavour, especially those of project management, business, and public policy.

Now, fortunately, we are standing on shoulders here—the discipline of systems engineering has come a long way and we already know a great deal about processes for managing complexity. By way of example, I would like to refer briefly to just two elements of SE: first to illustrate that systems engineering already has strong practical approaches to solving complex problems; and second, to provide us with common ground to discuss further some practical implementation issues.

SE and Managing Complexity

Current systems engineering practice has a strong focus on top-down design and bottom-up development, which provides a firm base for managing complexity, as is evident in two snapshots of the premier systems engineering standard ANSI/EIA-632: Engineering a System.

Traditional engineering design methods tend to be based on a bottom-up approach in which known components are assembled into subsystems from which the system is constructed and then tested for the desired properties. The design is modified in an iterative manner until the system meets the desired criteria. This approach is valid and extremely useful for relatively straightforward problems that are well defined. It is therefore appropriate for detailed design (in fact, it is largely how we train engineers). Unfortunately, however, complex problems cannot be solved that way, nor can complex systems be adequately developed. Additionally, a top-down focus is essential because systems have emergent properties that may not be discovered by taking a bottom-up approach.

Rather, the complex system must be viewed as a whole, which facilitates the subsequent analysis of the system and its interfaces. Once system-level requirements are understood, the system is then broken down into subsystems, and the subsystems further broken down into components until a complete understanding is achieved of the system from top to bottom. For example, Figure 1 illustrates the ANSI/EIA-632 approach to top-down development [3].

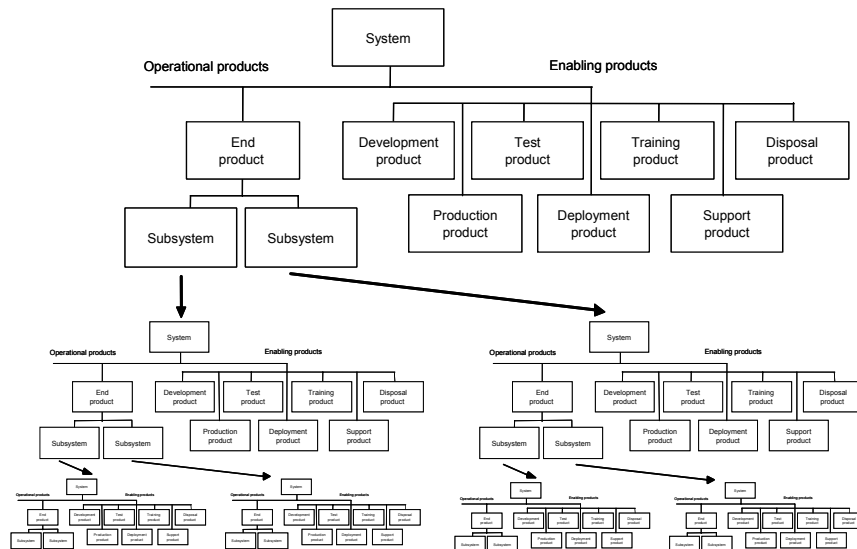


Figure 1. ANSI/EIA-632 building block concept for top-down development.

It must be recognized, however, that while system design is conducted using a top-down approach, the system is implemented using a bottom-up approach. As you know, this is the essence of system engineering, its contribution twofold: a rigorous, reproducible process by which the complex system can be broken into a series of manageable components that can then be designed using the traditional engineering bottom-up approach; and a process by which the components and subsystems can be integrated to achieve the desired system properties.

Importantly, ANSI/EIA-632 applies the top-down approach to itself, taking a higher vantage point than traditional systems engineering (an enterprise-wide viewpoint), looking at the broader picture of *engineering a system*, rather than *systems engineering*. Figure 2 [4] shows the upper-level framework for the standard that defines thirteen processes in five broad categories. Again, complexity is eased by breaking up the task into well-defined processes with well-defined interfaces [5].

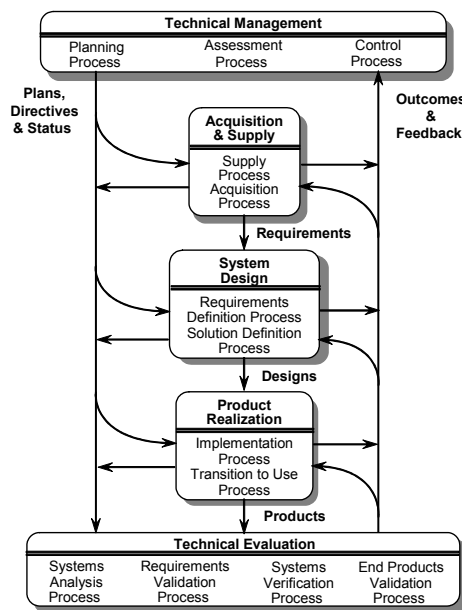


Figure 2. ANSI/EIA-632—thirteen processes for engineering a system.

With that background, I would like to offer some brief observations on practical approaches to the implementation of complex systems.

The Human Element

Figure 2 shows the top-down processes associated with engineering a system. Of course as good systems engineers we immediately recognise that we need more than just the processes: we need the people to develop, implement and monitor them; we need data, documentation, training, hardware, software, facilities, and so on.

First among these, however, is the human element—people. If the catchcry in real estate is “location, location, location!”, the call in solving complex problems must surely be “people, people, people!”.

People have different perspectives of the same problem; different perspectives can even be held by the same individual. Different organisations have different cultures; different parts of the same organisation can even have different cultures. We lose this richness of views if we adopt a single view of the problem. Here we have a strong need for soft-systems approaches to be integrated into the harder systems engineering methodologies [6].

Additionally, while we recognise people and associated human factors when we develop systems to support them, we need to pay much more attention to the people involved in the *development* of the system, as well as the people who are elements of the system (such as users, maintainers, and so on). Unfortunately, however, while the human element is the most-creative, most-intuitive system element it is also the one that we have the poorest processes for managing, and is the least predictable, the least homogeneous, and the most fallible of elements in the whole system.

Stakeholder management [7], for example, is arguably as important as requirements management. You can buy at least a dozen computer programs to assist you with the latter—with the former, you have no such choice. Like a rowing crew, a project team is most productive when all members are rowing in the same direction. Developing complex systems and solving complex problems requires the same commitment to a common purpose. Unfortunately, the project environment (like other complex environments such as a public policy) is often dominated by conflicting interests, hidden agendas, and so on. We must manage these human aspects first, before we can manage the necessary engineering processes.

Even when we are all rowing in the same direction, however, we must recognise that our ability to manage complexity relies principally on humans, each of whom can only cope with a very small number of relatively simple concepts at any one time. In systems-engineering terms, the human element therefore represents one of our greatest constraints in the development of complex systems. People provide for us a great paradox: complex systems do not have simple solutions, yet simple solutions are the ones that we humans most readily understand. The only way we can hope to succeed is if we can develop methodologies by which we can take a complex problem and break it into a series of components that are understandable, manageable and solvable within our human capabilities. This is the great strength of the top-down approach of systems engineering—which is absolutely essential, not only because of the complexity of systems and the need to identify emergent properties, but also because there are people involved.

Because there are people involved, however, the top-down approach is not sufficient in its own right—there are a number of related issues to consider.

Extant Processes / People / Technology

The first can be identified by a brief examination of the system life cycle—we use the system phases defined by Blanchard and Fabrycky [8] and illustrated in Figure 3.

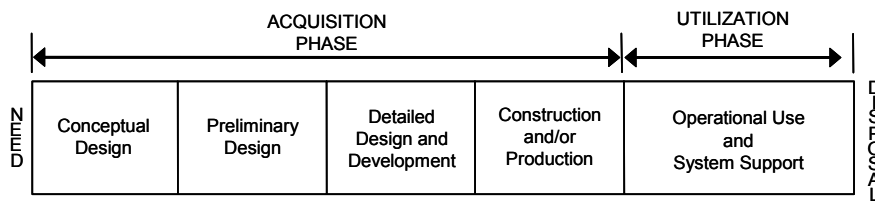


Figure 3. System life cycle.

When we implement top-down design, we are always careful to avoid any unknown quantities in the end products. That is, the purpose of the top-down approach is to arrive at a series of sub-systems, modules and components that can be built using existing techniques, by engineers and tradesmen with existing experience and skill levels, using existing technologies and processes. If these things do not exist at the time of stating the need, they must be attained by the beginning of Detailed Design and Development, or we have lost most of the advantage of our systems-engineering approach. If we have to make significant changes in the relatively short period between the need and Detailed Design, we are introducing significant risk to the project.

One of the main aims of the systems-engineering process is therefore to ensure that, at the preliminary design review, we have defined a physical architecture that is able to be designed and built using extant processes, extant people and extant technology.

It is one thing to embark on a system development with the intention of making use of a new technology; to begin the process with the expectation of embracing new technologies as they come along is an entirely different, and very risky, thing.

Now, my previous comments may seem to fly in the face of the need (perceived, at least) in the modern age to be able to embrace new technologies and react to rapid changes in requirements. Notice, however, that there is no timeframe associated with the system life cycle and the waterfall development model is not assumed. If adoption of new technologies and rapidly changing requirements are necessary then there are better development methodologies (incremental, spiral, evolutionary, rapid-application development, just to name a few). The implementation of any of these methodologies does not obviate my previous comments—the process must still be sequential, albeit in a much shorter timeframe. The requirements for a software module cannot be changed as the programmer is writing or testing the code; the material properties of a new aluminium armour cannot be modified as it is being welded on the production line; 240V cannot suddenly be required from a power supply designed to provide 24V; the project landscape is littered with those who have tried.

Manageable (Small) Steps

Despite what sometimes seems to be overwhelming evidence to the contrary, we also appear to have an unshakeable belief that to be successful we must embark on projects that require us to take huge steps in uncharted directions. This is particularly true in the area of information technology in this Information Age. Surely, if we have never been this way before, we should take small steps until we have shed light on the path and the yet-unknown attendant risks of the journey. We would not countenance diving into a flooded stream without checking for logs, or running headlong into a pitch-black room. We insist that a new drug is rigorously tested before it can gain approval for use in the community, yet we are happy, almost at a

whim it seems, to embrace new management philosophies, to reorganise, and to adopt new technologies without requiring any evaluation of either the old way of doing business or the proposed benefits of the change.

If we have learnt through bitter experience to be cautious as humans in our personal lives, why are we still so adventurous in the project world. We are almost guaranteed to fail if we charge off in uncharted directions with incomplete systems-engineering products; immature, unproven technologies; an insistence on fixed-price contracts; all managed by project teams that change three or four times during the course of the acquisition.

The latter issue leads me to my next observation.

Maintaining Focus

If the human element is one of our main constraints in acquiring complex systems, the development of the project team (particularly the system designers) is critical. If I can continue with the rowing analogy: we must look for skilled team members, train as a team and then compete as a team. We have little chance of success if we suddenly drop a novice in as stroke, or have all team members stand up mid-race and move one place to the stern.

Yet, it is very rare for the same team to participate unchanged in all phases of the system life cycle. The effect of these personnel movements is quite debilitating and there are many examples of projects that have suffered as a result of turmoil in key positions. How can we hope to clear the fog of complexity when our staffing policies make a significant contribution to the dynamic nature of the problem?

The matter is made worse by a desire to solve any problems in developing complex systems by changing procedures and organisations at a rapid rate. You are hard pressed to find too many government or commercial organisations whose structure has not undergone fewer than three or four fundamental changes in the last five years. We can't hope to solve complex dynamic problems with procedures, organisations and staffing arrangements that are themselves complex and dynamic.

Common Language

I have been involved in a large number of projects in many arenas, in Australia and overseas, in defence and other government departments and within industry. While there are many common elements in projects in all those domains (project management philosophies, contracting procedures, quality assurance, and so on), there is a distinct lack of a common view of a systems-engineering framework and a lack of a common language for engineering a complex system. Almost every project has a different term for the same systems-engineering document and activity [9]. Some projects even have the same acronym for different products.

A common language is one of the factors that sets a profession or a discipline apart from others. We must continue to work in systems engineering to develop a common language, or at least a *lingua franca*. Complex problems cannot be solved using vague and conflicting terms that have varying meanings for the different members of the problem-solving team. The players in an orchestra need not only a common sheet of music, but also a common interpretation.

The Big Picture

If we are to be successful in the development of complex systems we must focus on the detail: we must, for example, have a template for the DID for the OCD, we must ensure that we have a good contracting template, and so on. All these things are necessary, but they are not

sufficient. As well as a focus on the detail, we must elevate our view to keep our eyes on the issues that surround the process. Complex problems cannot be solved by simply insisting on form: we must maintain a focus on function.

Conclusion

We have come a long way in systems engineering in the last few years; we are better equipped than we ever have been to solve complex problems and develop complex systems. I do believe, however, that there is a difference between *systems engineering* as we have known it and *engineering a system*. Practical approaches to complex systems are part of engineering a system and require more than a strong focus on systems-engineering processes and products. We must elevate our view of systems engineering and apply the process to itself. That is, we must take a systems view, not only of the system itself, but also of its development.

In particular, we must focus on the human element. In systems engineering, there are always a number of technical problems to solve, but as I am fond of telling my students, a technical problem is defined to be one in which the human element is only 90% of the problem.

This means that our hard-systems, systems-engineering, methodologies must be supplemented with soft-systems approaches so that we can take into account the human element, organisational cultures, and so on. Unless we do our ability to solve complex problems will always be hamstrung by our inability to identify and deal with those project and enterprise constraints that shape our endeavours.

Endnotes

- 1 For a discussion of complexity, see: Kline, S.J., *Conceptual Foundations of Multidisciplinary Thinking*, Stanford University Press, Stanford, California, 1995.
- 2 Intuition is a valid and important part of decision making and we use it frequently in the management of complex problems. See for example: Klein G., *Sources of Power: How People Make Decisions*, MIT Press, 1998; and McLucas A., *Decision Making: Risk Management, Systems Thinking and Situation Awareness*, Argos Pres, Canberra, 2003.
- 3 ANSI/EIA-632-1998, *Processes for Engineering a System*, Washington, D.C.: Electronic Industries Association (EIA), 1999.
- 4 ANSI/EIA-632-1998, *Processes for Engineering a System*, Washington, D.C.: Electronic Industries Association (EIA), 1999.
- 5 Note that this process is highly iterative—iteration is a very important tool in our understanding of problems.
- 6 See for example: Wilson, B., *Soft Systems Methodology: Conceptual Model Building and its Contribution*, John Wiley & Sons, Chichester, UK, 2001; and McLucas A., *Decision Making: Risk Management, Systems Thinking and Situation Awareness*, Argos Pres, Canberra, 2003.
- 7 See for example, Eden, C. and Ackermann, F., *Making Strategy: The Journey of Strategy Management*, Sage, London, 1998; and McLucas A., *Decision Making: Risk Management, Systems Thinking and Situation Awareness*, Argos Pres, Canberra, 2003.
- 8 Blanchard, B. and W. Fabrycky, *Systems Engineering and Analysis*, Upper Saddle River, N.J.: Prentice-Hall, 1998.
- 9 See for example: Sutton, D., “Linguistic Problems with Requirements and Knowledge Elicitation”, *Requirements Engineering*, No. 5, Springer-Verlag, London, pp. 114-124, 2000.