2010

Some complex systems engineering principles

Matthew J. Berryman
University of Wollongong, mberryma@uow.edu.au

Peter Campbell
University of Wollongong, pcampbel@uow.edu.au

Publication Details

Berryman, M. J. & Campbell, P. (2010). Some complex systems engineering principles. Australia's Preeminent Systems Engineering and Test & Evaluation Conference

Some complex systems engineering principles

Abstract
Complex adaptive systems can be characterised by many adaptively changing parts, with a large number of interactions, and adaptation at the system level. This body of research can inform the large engineering projects that are characterised by large systems of systems, where interactions to achieve end goals matters, and where designs and the technology of parts in service undergo many adaptive changes. A number of key lessons can be drawn from the complex adaptive systems literature. Firstly, that the use of design patterns, as traditionally applied in software engineering, can also be applied to wider systems engineering and systems integration issues; in particular, the use of the "bow-tie" architectural pattern drives compatibility between different pieces of kit that can then evolve independently. Another key lesson is that focussing on the adaptive learning processes in individuals and groups fosters positive decision-making about complex integration problems.

Keywords
complex, principles, engineering, systems

Disciplines
Engineering | Physical Sciences and Mathematics

Publication Details
Berryman, M. J. & Campbell, P. (2010). Some complex systems engineering principles. Australia's Preeminent Systems Engineering and Test & Evaluation Conference

Some Complex Systems Engineering Principles

Matthew J. Berryman

Peter Campbell

University of South Australia

Defence and Systems Institute, SPRI Building,

Mawson Lakes Campus, Mawson Lakes 5095

Abstract.

Complex adaptive systems can be characterised by many adaptively changing parts, with a large number of interactions, and adaptation at the system level. This body of research can inform the large engineering projects that are characterised by large systems of systems, where interactions to achieve end goals matters, and where designs and the technology of parts in service undergo many adaptive changes.

A number of key lessons can be drawn from the complex adaptive systems literature. Firstly, that the use of design patterns, as traditionally applied in software engineering, can also be applied to wider systems engineering and systems integration issues; in particular, the use of the "bow-tie" architectural pattern drives compatibility between different pieces of kit that can then evolve independently. Another key lesson is that focussing on the adaptive learning processes in individuals and groups fosters positive decision-making about complex integration problems.

INTRODUCTION

Firstly, we will introduce some of the key concepts, then go on to discuss further how the two strands of complex design patterns and adaptive decision-making allow one to get a better handle on systems integration issues. We then present a case study of software integration that further highlights these points, and then we make some concluding remarks.

The nature of complex systems.

A number of different definitions of complex systems are out there, from the whimsical "neat nonlinear nonsense" (Shalizi 2009) through to the more useful ones in terms of a number of elements having diverse, variable behaviours that interact to produce the behaviour of the whole (Bar-Yam 2003). One can also view complex systems in terms of properties that are often true for such systems – namely nonlinearity, chaotic behaviour, many feedback loops, power-law (or at least, scale-free) behaviour (Clauset et al. 2007), and network effects (Barabási 2003). Then complex systems science becomes the usual scientific process applied to complex systems, using tools that are appropriate from the above-mentioned areas of research. A key property of complex systems is that of emergence, where emergence is defined as that which "arises because the collective behaviour is not readily understood from the behaviour of the parts. The collective behaviour is, however, contained in the behaviour of the parts if they are studied in the context in which they are found" (sic) (Bar-Yam 2003). This is in contrast to the more common definitions of emergence, which stress some "unexpected" property larger than that of the parts, as if it just arises through magic. A different but related perspective to Bar-Yam's, is that of (Abbott 2006), who describes emergence as that which arises from viewing the system at a higher level of abstraction. To relate the two definitions, viewing the system at a higher level of abstraction implies that one takes in the context in which an individual part acts.

There is of course a subjective element to both complexity and emergence; however the same is true even in many parts of the "hard" sciences. Complexity (ignoring the definitional debates) is subjective because there are many different ways of viewing the parts, describing interactions, and different ways of measuring complexity. However given some reasonably

objective standards, and measures, then two observers can agree (within statistical error) on how complex a system is. Even ignoring all of the above, if nothing else at least one can take the view that it is important to consider the interactions between parts, and not take a fully reductionist view.

The nature of complex engineered systems.

What then of complex systems engineering, and complex systems integration? It is instructive first to look at what we mean by systems engineering and systems integration, and then consider how complex systems engineering and complex systems integration differ as fields of research.

The International Council on Systems Engineering defines Systems Engineering as "an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle", where "a system is a construct or collection of different elements that together produce results not obtainable by the elements alone…" (Fellows of the International Council on Systems Enginering 2006). Systems integration is defined as getting the parts to work together as a whole. These definitions combined, illustrate the nature of the problems with which systems engineering purports to deal.

Complex systems engineering then, is bringing a complex systems perspective. The key point is that the interactions between the parts are not irreducible to simple interfaces, with no inter-subsystem effects. One can still take a reductionist view, but one must also be aware of emergent effects (from interactions between parts). In particular, trying to ascertain as many as possible and as best as possible comprehend what these mean for the (complex) system. A related distinction is drawn by (MITRE 2005), where traditional systems engineering and complex systems engineering are seen as complementary processes. Traditional systems engineering is seen as dealing with the interactions among components, whereas complex systems engineering is concerned with "components of the system for the system as a whole" and the "environment and processes by which the system is going to be created, which is separate from designing the system itself". Part of this focus on the environment is to do with setting up an environment in which designs can be evolved, as starting from scratch is too difficult given the complex nature of the problem. The emphasis is on bottom-up evolving design, rather than top-down design (Doursat and Ulieru 2009). This use of adaptation is also intertwined with results described later on the use of adaptation within a human (in the form of learning) to better manage complex problems. It is with this in mind that we later introduce the topic of complex decision-making, as without an understanding of how humans perceive and understand complex systems, and the limits thereon, we cannot help to improve the systems engineering process to better manage complex systems. It is also worth mentioning that other types of systems perspective, for example critical systems heuristics (Midgley 1997), as these also add value to understanding systems engineering.

On change and adaptation.

The essence of evolution as established by Darwin is that variation, followed by differential replication and selection, leads to a fit between some system (in the case of natural selection an organism) and its environment (Dawkins 1989, Grisogono 2006). Although traditionally engineering is a process that is not purely random (unless an evolutionary algorithm[1] is used), the engineering process can be viewed through the evolutionary lens, both within a design team (competing designs) and across a marketplace of competing products. Engineers also typically build on pre-existing technologies, as nature also re-uses existing features. In both engineering and evolution there are similar processes like exaptation (using existing things in anew ways), exploration of new designs through recombination of genes/ideas, and exploitation (fine-tuning) of existing genes/designs.

Taking a broader view of adaptation, from the meaning 'to fit', one can also consider human learning to be a form of adaptation (Grisogono 2006). There is obviously not variety

[1] Defined broadly to encompass a wide range of algorithms including genetic algorithms, genetic programming, evolutionary programming and evolutionary strategies. Even with an evolutionary algorithm, it is technically not purely random in the sense that the selection mechanism creates a bias.

amongst individuals, in this case, however there is still a variety of different ideas or strategies tried, with feedback on these.

Design patterns.

The use of design patterns originates in architecture, through the work of (Alexander et al. 1977). This has then been extended to the field of software engineering (Gamma et al. 1995), looking at software patterns that recur in solving particular problems or meeting requirements, for example to provide a common graphical user interface across different software packages.

Since most software packages meet the requirements of our earlier definition of complex engineered products, and design patterns help manage this complexity, it makes sense to consider whether design patterns are of more general use in complex engineered (or other) systems. This work has been carried out by (Niarchos et al. 2008), who found that some (but not all) of the software design patterns carry over, in modified form, and that there are other patterns such as the bow-tie and kernel patterns that may also be useful design patterns in complex systems engineering. The use of design patterns to manage complexity in the engineering life-cycle, raises the issue of complex decision making, which is introduced below and discussed later in this paper.

The bow-tie model (pattern) was introduced by (Doyle 2007, Csete and Doyle 2004) to describe the observation that a well-designed, constrained (read in some cases: simplified) interface allows for the evolution of complexity on either side. Consider the TCP/IP (transmission control protocol / Internet protocol) protocol suite, an implementation of the abstract concepts of packets and packet flows. On either side of this interface is great flexibility. One can run TCP/IP over the wireless network layer in a home, over a 3G cellular network, over the Ethernet cable, etc. On top of TCP/IP one can run not just the web (hypertext transfer protocol, HTTP) but also email (simple mail transfer protocol, SMTP), Internet relay chat (IRC), etc. This is illustrated in Figure 1.

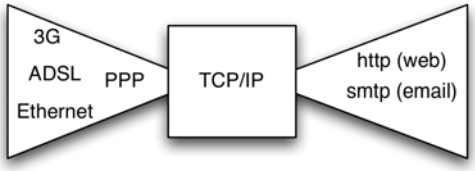

Figure 1. Illustrating the way that TCP/IP acts as a constraint that allows generation of variety on either side of the interface.

These protocols then in turn become interfaces that can then be used to foster evolution, for example consider the web 2.0 transition, with mash-ups, wikis, webmail, robots all running via HTTP. This is discussed in more detail later in the Discussion section.

Another design pattern is the kernel design pattern, which originates in the literature on gene regulatory networks (Davidson 2006). The gene regulatory network shows how the various genes interact to produce the functionality of a cell, and is a little like how the concept of an electronic circuit diagram (and associated mathematical machinery) works. In considering systems integration issues, it is worth studying gene networks, as various genes act as interfaces between complex sub-networks of the overall gene regulatory network. The kernel pattern describes how, over time, sub-components of the overall network become conserved (standardised) and then how these get used in different ways by sub-networks that are undergoing more rapid change. Biological examples include the core metabolic cycles that power the cell, that are very highly conserved across almost all species, and body parts like vertebrae that get used in different patterns—consider the difference between a giraffe and a human neck.

Open architectures.

Open architectures have been proposed as a way of managing software evolution and integration (Oreizy 2000). Open source exposes the code without the architecture also being exposed and open to modification. Only a tight coupling is allowed, which prevents easy distribution and evolution of interacting parts. The essential feature of open architectures is that, in part or in full, the internal architecture is exposed and made malleable. Examples of this can be found in platforms such as Eclipse and Firefox, particularly in Eclipse where the plug-in

framework supports a whole architecture of interacting plugins, which in themselves are open architecture (Fielding 2008).

Complex decision-making.

The work by (Dörner 1997) has shown that most people are not naturally good at managing complex systems. The example is given in his work of people trying to run a small town, with "town mayors" (participants) struggling to comprehend not only non-linear subsystem responses, but the many feedback loops between the different subsystems that comprise the town as a whole. More recent work suggests that even people well-versed in complex systems struggle to manage complex systems (Boschetti et al. 2010). The good news from the work by Grisogono et al. (unpublished), though preliminary, is that even non-complex systems experts can successfully be taught to manage complex systems. Despite this work, very little work has looked specifically at how organisations manage complexity; however there is the work of (Bar-Yam 2004) that looks at what organisational structures may be required, and there are also lessons that can be learned from the organisation psychology literature, particular that on organisation learning (Brown and Duguid 1991). One of the key findings of the complex decision-making work is the need to take an adaptive approach to complex decision making, in an environment where it is safe to learn.

DISCUSSION

Combined use of kernel and bow tie design patterns.

The bow-tie pattern, while guiding one in selection of an interface for providing maximum change and inter-operability on either side of the interface, does not address the need for change of the interface. The kernel design pattern, focussing on incremental change over time, allows one to allow a module to change over time, through change around the edges, that through processes of selection, allows for generation of new features, followed by widespread adoption and standardisation. It does in-and-of-itself address some of the specific requirements of interfaces imposed by the bow-tie model. To link the two together, we must pay attention to where in the system the evolution is occurring. Is it a case where the change is occurring purely in the interface? Or is it a case where the change is also occurring in the parts around the edges? To re-use the example of TCP/IP, change can occur through the change in the interface itself, for example the migration from IPv4 to IPv6, but also in the fact that the surrounding protocols change and over-time become standardised, for example the creation and evolution of (E)SMTP – (Extended) Simple Mail Transfer Protocol. Evolution of an interface is illustrated diagrammatically in Figure 2.

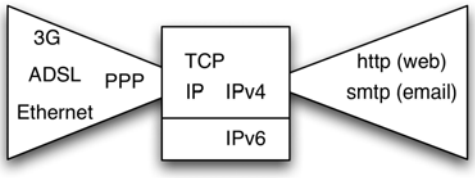

Figure 2. Bow-tie model showing evolution of an interface.

It is worth noting how slow the adoption to IPv6 has been. In part, this reflects a lack of pressure to adopt IPv6 (through hardware and software changes) due to IPv4 address space still being available, however it also reflects some of the difficulties in designing a new IP layer that will preserve much of the existing design, and also run on-top of the same network layers, and underneath the same application protocols (Hovav et al. 2004, Lawton 2001).

In the case where change is occurring in the interface, the bow-tie pattern indicates that maximum evolvability comes from having a well-defined, standard core interface. How then to evolve it? The kernel pattern then suggests that the core interface then be extended, while keeping critical functionality conserved (standard). Doing this is non-trivial, however, and depends on the exact details of the interface. In future work we will consider in detail how these concepts and architectural patterns have played out in the transition from IPv4 to IPv6.

In the case where change is occurring around an interface, then one can envisage new bow-ties arising from the conservation of features that have arisen through evolution around an existing bow-tie. To help clarify what we mean, we will use the example of the Ajax (asynchronous JavaScript and XML) web technologies (Garrett 2005). Here, HTTP acts as a constraint, around which a number of different web technologies (examples include Adobe Flash, Microsoft

Silverlight) can be built. One set of technologies is the Ajax standard. These have independently evolved, and are now part of a core set on which web applications and web APIs are built. The process is one then of conservation of interfaces, which can be considered as a "pinching" in of the middle of the bow-tie as more features become part of an enlarged interface, with then a stretching out of the edges as more evolution (variety and change) occurs. This is illustrated in Figure 3. Once these become part of the middle of the bow-tie, then other software depends on these interfaces, forcing them to remain relatively static, as we saw earlier with the slow and minimalistic upgrade of IPv4 to IPv6. The Ajax APIs, providing a way for Ajax programs to be easily written, then become more and more conserved and standardised over time, thus adding to the conserved core (kernel) that is part of the bow-tie.

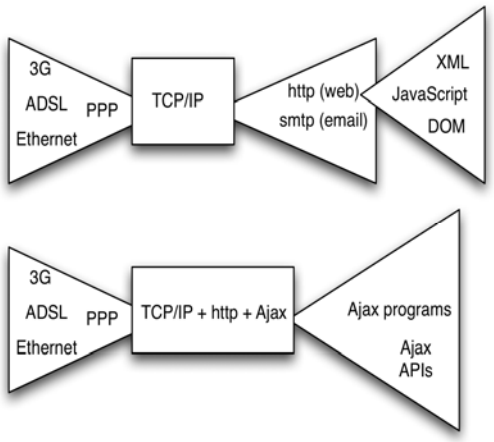

Figure 3. The combination of a bow-tie pattern with a kernel pattern, showing features moving into an expanded core interface (from the top subfigure to the bottom).

Complex decision making.

What does the work on complex decision making suggest for complex systems engineering, and in particular for complex systems integration? Firstly, at a very high-level of organisation structure, the work of Bar-Yam indicates the need to look closely at whether the organisational structure is a good fit for the types of problem and competitive environment an engineering organisation finds itself in. The work looking at human decision making in the face of complexity, suggests a number of things:

- That people be trained in managing complexity. The type of training will in part need to be tailored to the role, however there are many generic skills that need to be addressed.
- A key skill is that of learning. There are a number of key factors that can lead to people not learning, both personal, such as confirmation bias and emotional factors, as well as environmental factors, such as the risk of lawsuits if a failure is reported.
- In communication between individuals, or between groups, attention needs to be paid to the limits and biases of human cognition with respect to managing complexity, but also importantly to any emotional factors that may hinder (Dörner 1997) or actually help with managing complexity (Lehrer 2009).
- That strong emphasis needs to be placed on how organisations as a whole learn, particularly with respect to complex systems.

One systems integration "case study", where these points would have helped, is in the design of the water and electrical systems of the Boeing 747. It is quite reasonable to design these relatively independently, however if closer attention had been paid to the complexity of integrating the two, then perhaps the problem with water seeping through cracks in the barrier between the systems (Australian Transport Safety Bureau 2003) would have been avoided.

CASE STUDY

Here we consider how some of this may play out for a complex problem—that of integrating a general circulation software model of climate change (Intergovernmental Panel on Climate Change 2007), with an agent-based economic software model (Miller and Page 2007, Batten 2000, Tesfatsion 2003). To add to the complexity, we also consider implementing this on a parallel computer. There is a requirement to enable a wide range of parameter values to be explored in a reasonable period of time. The approach to parallelising the model will constrain the conceptual model - this is not particularly surprising, but something that will be a big issue if we are aiming for speed. This relates to the other point, which is somewhat

surprising, in that the approach will also determine some of the implementation of an open architecture bridge between the climate model and the economic model. Part of this bridge is a regional climate model, which translates from the global general circulation model into a model of the regional effects, for example runoff, growing season, etc.

Firstly, some high-level requirements on the system:

- Feedback is crucially important between different parts of the model, because we are dealing with a complex adaptive system. These feedback loops, such as the effect of the economy on the climate, and the climate back on the economy, have different, typically long time scales on the order of years and decades.
- We want to plug and play different conceptual models, with different implementations of the conceptual models (for good science), and different versions (of conceptual models and/or implementations). Disparate groups, spread out across Australia, will develop these models.
- We are designing for speed, because we want to explore widely in the parameter space.
- General Circulation models work well in a traditional vector super computer.
- We want to have the economic model in different resolutions for different parts of the world, namely a high level of detail for Australia, and the best approach to this seems to be a predominantly agent-based one.

Approach one.

This takes the approach that the best way to parallelise the model is to use the natural fit of the models - a "Single Model-run, Multiple Regions" (SMMR) climate model on a vector supercomputer (as per the assumption above), with a "Multiple Model-run, Single Region" (MMSR) economic model that can be run on a cloud.

Approach two.

Split up the economic model into economic/political regions, and run these fragments on the corresponding node doing a parallelised (by economic/political segmentation of) using appropriate regional climate models.

Both approaches have pros and cons, however regardless of the approach, there is a need to allow for:

- Integration of the economic model with the climate model, including possible different software components (sub-models) of the economic model, for example a transportation model and a model of household electricity use.
- Various different implementations of the above, along with evolution of these implementations.

The bow-tie pattern helps us manage the above integration by focussing on the need for a well-defined, constrained interface, to allow for message passing between the different models (including any sub-models). By building the right amount of complexity into the interface (as simple as possible to allow for most messages) we provide for a loose coupling between models, that is required for the evolution of different implementations. Suppose we wish to add more sub-models that require different information to be passed? The kernel pattern method then suggests two different strategies for addition of any sub-models to the economic model, either

- constructing the interface protocol such that it has a variable message length, with the extra segments being used by any sub-models, but ignored by other.; or
- adding another type of communication between the core of the economic model and any new sub-models.

Although both involve relatively minimal changes to the existing software (the key implication of the kernel pattern), the first is a closer fit with the kernel pattern, as well as being a cleaner approach, though more difficult to design up-front.

Both design approaches require an open-architecture design, based around a well-designed bow-tie interface, in part consisting of a regional climate model. The actual implementation at lower levels will vary—in approach one we are transferring things between a supercomputer and a cloud, in approach two, between software components on a node. The actual usage of the interface will vary, that is, not all data types would be required in approach one if we take the minimal feedback route to speeding it up. Further, the models themselves

require an open architecture, particularly if we are to try out more than one of the above approaches or their variants we have mentioned. A shift from approach two to approach one (with the conceptual constraint) would require more of the human actions to be built into the climate model side of things, so we would need to have the software components relatively easily interoperable.

As well as an open-architecture approach, to provide for interoperability, we must pay attention to the human processes in communicating these architectures around so that the various software components can be built. One strategy to minimise the complexity that any one developer needs to work with is to separate out the complexity as much as possible into the different parts, so each individual component is as simple as possible, given the constraint that the conceptual model must be of requisite complexity to model the real-world phenomena of interest. Architectural frameworks (AFs), expressed in modelling languages like UML (Unified Modelling Language) or SysML (Systems Modelling Language, an extended "dialect" of UML), are essential for communicating effectively the complexity of the system, and also as an enabler of faster learning. Faster learning is a critical requirement to deal with a rapidly-changing, complex environment (de Rosa et al. 2008, Robinson and Graham 2010), like that we are seeking to model with the combined climate and economic models.

The limitation placed on us by the distances over which the groups are operating means that in electronic forms of communication, we must pay attention to written cues about the workload and stresses that we are under, and how these may affect understanding of the design, as well as module implementation. We need to foster an innovative environment, and not be too ready to constrain options.

CONCLUSIONS

This paper has shown that many of the systems with which systems engineering purports to deal with require a complex systems engineering perspective. This not only requires an understanding of complex systems, but also understanding the psychological aspects (emotion and cognition) that affect understanding, so that when we try to understand complex systems, and communicate about them with others, we minimise the effect of our human limitations on engineering complex systems. This obviously extends to complex systems integration, where parts that were not necessarily designed to operate together are subsequently required to, and understanding their interactions (sometimes, even just the parts themselves) proves problematic. Future research will look at group understanding of complex systems management, with obvious implications for complex systems engineering and integration.

The kernel and bow-tie design patterns are two key design patterns that interact to provide for evolution on both sides of an interface used to integrate a variety of different systems together. By considering where in the system the change is likely to occur, we may adopt different strategies. Future research will study in depth how this has played out in the evolution of TCP/IP and surrounding protocols in the link and application layers.

A recurring theme amongst both themes is the key role that adaptation plays, and the need to foster an environment in which learning and creativity are key.

REFERENCES

Abbott, R. (2006) Emergence explained: Abstractions: Getting epiphenomena to do real work. *Complexity,* 12**,** 13-26.

Alexander, C., Ishikawa, S. & Silverstein, M. (1977) *A pattern language: towns, buildings, construction*, Oxford University Press, USA.

Australian Transport Safety Bureau (2003) Electrical system event - Bangkok, Thailand. Canberra, Australian Transport Safety Bureau.

Bar-Yam, Y. (2003) *Dynamics of Complex Systems*, Westview Press.

Bar-Yam, Y. (2004) Multiscale Variety in Complex Systems. *Complexity,* 9**,** 37-45.

Barabási, A.-L. (2003) *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*, Plume Books.

Batten, D. (2000) *Discovering Artificial*

Economics: How Agents Learn and Economies Evolve, {Westview Press}.

Boschetti, F., Yves-Hardy, P., Grigg, N. & Horwitz, P. (2010) Can we learn how complex systems work? *Emergence: Complexity and Organization,* (in review).

Brown, J. & Duguid, P. (1991) Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation. *Organization Science,* 2**,** 40-57.

Clauset, A., Shalizi, C. & Newman, M. E. J. (2007) Power-law distributions in empirical data.

Csete, M. & Doyle, J. (2004) Bow ties, metabolism and disease. *Trends in Biotechnology,* 22**,** 446-450.

Davidson, E. (2006) *The regulatory genome: gene regulatory networks in development and evolution*, Academic Press.

Dawkins, R. (1989) *The Selfish Gene*, Oxford University Press.

De Rosa, J., Grisogono, A., Ryan, A. & Norman, D. (2008) A Research Agenda for Complex Systems Engineering. *IEEE Systems Conference.* Montreal, Canada.

Dörner, D. (1997) *The Logic of Failure: Recognizing and Avoiding Error in Complex Situations*, Perseus Books Group.

Doursat, R. & Ulieru, M. (2009) Emergent engineering: A radical paradigm shift. *International Journal of Autonomous and Adaptive Communications Systems,* To appear.

Doyle, J. (2007) Complexity and Robustness. *Symposium on Complex Systems Engineering.* RAND Corporation, Santa Monica, CA, USA.

Fellows of the International Council on Systems Enginering (2006) A Consensus of the INCOSE Fellows. International Council on Systems Engineering.

Fielding, R. (2008) Open Architecture. *O'Reilly Open Source Convention.*

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995) *Design patterns*, Addison-Wesley Reading, MA.

Garrett, J. (2005) Ajax: A new approach to web applications.

Grisogono, A.-M. (2006) Success and Failure in Adaptation. *International Conference on Complex Systems.* Boston, MA, USA.

Hovav, A., Patnayakuni, R. & Schuff, D. (2004) A model of Internet standards adoption: the case of IPv6. *Information Systems Journal,* 14**,** 265-294.

Intergovernmental Panel on Climate Change (2007) Climate Models and their Evaluation. *Working Group I: The Physical Science Basis of Climate Change.* New York, United Nations.

Lawton, G. (2001) Is IPv6 finally gaining ground? *Computer,* 34**,** 11-15.

Lehrer, J. (2009) The Decisive Moment: How the Brain makes up its Mind. Edinburgh Canongate Books.

Midgley, G. (1997) Dealing with coercion: critical systems heuristics and beyond. *Systemic Practice and Action Research,* 10**,** 37-57.

Miller, J. & Page, S. (2007) *Complex Adaptive Systems: An Introduction to Computational Models of Social Life (Princeton Studies in Complexity)*, {Princeton University Press}.

Mitre (2005) Perspectives on Complex-Systems Engineering. *Collaborations,* 3**,** 1-4.

Niarchos, T., Campbell, P. & Scott, W. (2008) Complex Systems Engineering Analysis. University of South Australia.

Oreizy, P. (2000) Open Architecture Software: A Flexible Approach to Decentralized Software Evolution. University of California, Irvine.

Robinson, K. & Graham, D. (2010) An improved methodology for analysis of complex capability. *SETE 2010.* Adelaide, Australia.

Shalizi, C. (2009) Complexity.

Tesfatsion, L. (2003) Agent-based computational economics: modeling economies

as complex adaptive systems. *Information Sciences,* 149**,** 263-269.

BIOGRAPHIES

Matthew J. Berryman

Matthew is currently a lecturer with the Defence and Systems Institute, at the University of South Australia. His current research is related to systems thinking—developing new perspectives and philosophies on systems, developing new tools for analysing systems, developing new tools for modelling systems, and developing models of systems. Past research has included research into complex adaptive systems at a conceptual level, as well as human decision-making and systems biology. His primary focus is on agent-based modelling, a method for representing individuals and groups of individuals in software, and on applying these models to interesting problems in economics, health, environmental, and defence systems.

He holds degrees in Computer Systems Engineering, Maths and Computer Science, and Complex Systems from the University of Adelaide, and is currently studying for a degree in Higher Education from the University of South Australia.

Peter Campbell

Peter is a Professor of Systems Modelling and Simulation and Research, at the University of South Australia, and founding member of the Centre of Excellence for Defence and Industry Systems Capability (CEDISC), both of which have a focus on up-skilling government and defence industry in complex systems engineering and systems integration. He is currently design lead for the simulation component of the Defence Systems Integration Technical Advisory funded Microcosm program and program director for three other DSTO funded complex system simulation projects. Through to mid 2007, Peter was a consultant to CSIRO Complex Systems Science Centre to introduce complex system simulation tools to support economic planning of agricultural landscapes.

Prior to his position at the University of South Australia, he was the director of Advanced Computer Applications Center, Argonne National Laboratory, with long term focus on simulation based decision support tools for application in logistics, battlefield environmental representation, hydrological planning, healthcare, and meteorological forecasting applications, including new technical developments for complex adaptive system applications involving human interactions with systems and organizations.