# Towards a Systemic Approach to Autonomic Systems Engineering

David Bustard[1], Roy Sterritt[1], A. Taleb-Bendiab[2], Andrew Laws[2], Martin Randles[2], Frank Keenan[3]
[1]*Faculty of Engineering, University of Ulster, UK*
[2]*School of Computing & Mathematical Sciences, Liverpool John Moores University, UK*
[3]*Department of Mathematics and Computing, Dundalk Institute of Technology, Ireland*

## Abstract

*An autonomic system is structured as a network of autonomic elements that collaborate to achieve the system's purpose. This paper examines the potential benefit of using well-established systems concepts and techniques in the development of such systems. In particular, it considers the possible role of Checkland's Soft Systems Methodology and Beer's Viable Systems Model in system design. The paper summarizes the relevant aspects of each approach and then assesses both their individual and joint strengths in support of the construction and evaluation of designs. Some practical issues in the use of these approaches are also identified. The discussion is illustrated using aspects of the design of an autonomic operating system.*

## 1. Introduction

Abstractly, a *system* is a collection of interrelated parts [1]. In an *autonomic system* [3, 15, 19, 20], the individual parts are autonomic elements, which are related through their mutual dependence and ability to interact. Systems are often hierarchical, meaning that any one system is typically part of a larger enclosing system. Everything outside an autonomic system relevant to its operation is usually described as its *environment*. This is similar to the machine-domain relationship described by Jackson [21].

Any computing system can be constructed in an autonomic form and, indeed, that seems desirable to ensure its effectiveness. Consider, for example, the basic structure of an operating system, as depicted in Figure 1. The outer box represents a workstation, showing external connections to physical devices and the Internet. Internally, the applications are shown connected to elements of the operating system responsible for managing the facilities and services of the workstation.

If each of the components in the diagram is autonomic then each will have an active role in performing its designated function. For example, the printer element, as well as implementing print requests from the applications can take initiative in reporting problems, such as low toner. It may also be responsible for recognising the connection of a new printer and obtaining the required driver from the Internet. Similarly, each application on the workstation can actively seek out and install its own new releases or updates. Such support is obviously very helpful to users but will also increase their confidence in using the workstation.

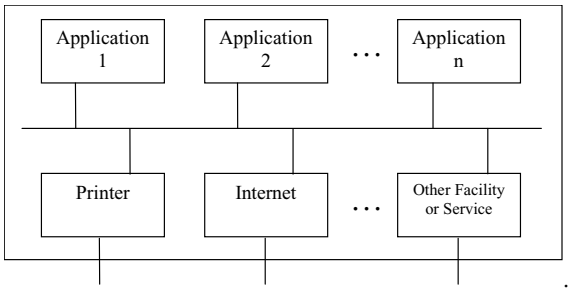



**Figure 1 Basic Operating System Structure**

This paper effectively addresses two particular questions in relation to the design of an autonomic system: (i) how should the elements of the autonomic system be organised internally; and (ii) to what extent should the environment of the autonomic system be understood and modelled, including what communication and control mechanisms are required to facilitate the coordination and co-evolution of a given autonomic system with respect to its environment.

From Figure 1, it is perhaps not obvious that an organisation structure among autonomic elements is necessary in that it appears to represent a collection of cooperating components of equal status. Such simplified diagrams, however, often omit management elements. For example, if applications compete for shared resources, such as workstation memory, some higher level mechanism is needed (another autonomic element) to observe the consequences of this competition and tune the tasks and resources allocation to achieve best performance overall. Also, as each element is self-monitoring (being typically composed of a managed component and an autonomic manager [26]), it will have to inform a 'higher authority' if it detects a problem that it is unable to resolve; this would be another management element in the system.

Knowledge of the environment is also important in designing the autonomic system. Understanding the environment is, of course, a basic part of determining the

Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)
0-7695-2308-0/05 $20.00 © 2005 **IEEE**

requirements of any computing system [21], but in autonomic systems some model of that environment also needs to be developed for inclusion within the autonomic system itself. For example, if toner is low on the printer then the person responsible for maintenance should be informed, which requires knowledge of the context. For a personal computer, reporting low toner might simply be achieved by displaying a message on the user's screen. In other cases, it may be more appropriate to contact a designated technician (perhaps via e-mail or text messaging).

So far, work on the design of autonomic systems (and self-adaptive systems in general) has tended to concentrate on system architecture, both in terms of design patterns [14, 16-18, 28] and the structuring of lower level components [26]. This paper focuses more on the design process. In particular, it considers the potential benefit of using general systems concepts and related techniques in modelling the environment and refining the autonomic systems design. Two specific approaches considered are Soft Systems Methodology (SSM) [12, 13, 30, 31] and the Viable Systems Model (VSM) [4-7].

The next two sections provide overviews of VSM and SSM, illustrated with respect to the workstation example. A concluding section critically assesses the value of VSM and SSM to the engineering of autonomic systems and identifies some issues that need to be resolved.

## 2. VSM and Autonomic Systems

A viable system is one that is robust against internal malfunction or external disturbance; it has the ability to respond and adapt to unexpected stimuli, allowing it to survive in a changing and unpredictable environment [4-7, 17, 18, 22, 23]. Stafford Beer developed the Viable System Model as a way of describing the essential elements of a viable system, with particular reference to an organisation or organism [6]. His work, which started in the early 1950s, provides a theoretically supported cybernetic model of organisation. The original model identified five necessary and sufficient subsystems that together maintain overall system viability. This was later extended to a sixth subsystem as indicated in the summary in Table 1 [22].

The table provides a brief general description of each subsystem, together with a diagram to indicate how the subsystems are related. Logically, the S1s, S3, S4 and S5 subsystems are structured in a hierarchy, connected by a central 'spine' of communication channels passing from the higher-level systems through to each of the S1 management elements. These provide high priority communication facilities to determine resource requirements, account for allocated resources, raise alerts that a particular plan is failing and re-planning is necessary, and disseminate "legal and corporate requirements" or policies of the system [5, 7]. The VSM is a recursive description, and, in particular, the S1 subsystems are also expected to be 'viable'.

When describing an organisation, subsystems S2-S5 are perceived as management activities, implemented by individuals in the organisation. For organisms, the lower level subsystems S1-S3 are automatic but the others (S3*-S5) can be influenced by higher-level brain functions. Autonomic computing systems aspire to the level of internal management of an organism while being amenable to human monitoring and guidance.

In the VSM, the S1 subsystems are an exact match for autonomic elements as they have an operational part (managed component) overseen by a management part (autonomic manager) [26]. In the autonomic operating system example, the applications and operating system services are all S1 subsystems. Each application, therefore, is expected to have individual management support, responsible for facilitating and monitoring the desired behaviour of the application.

*Coordination* (S2) is needed in the operating system when using shared resources such as disk memory, the user screen, and the processor (or processors) of the workstation. *Control* (S3) is exercised when functions cannot be performed as required because of resource shortage, such as the disk becoming full or main memory exhausted. Control is also required when there is failure in some part of the system, such as the unexpected termination of an application or the loss of a network connection.

The most obvious example of an *audit* (S3*) function on a workstation is the periodic running of a virus checker. Auditing can also be useful as an aid to cleaning up files on disk storage and, at a management level, in ensuring that there are appropriate licences to support the applications on the workstation.

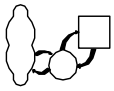
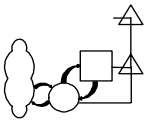
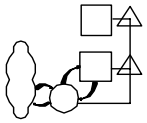
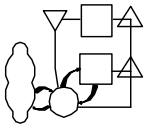
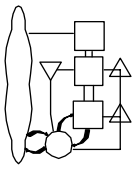
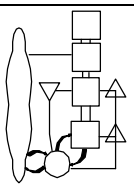
There is also a growing '*intelligence'* (S4) in operating systems in detecting environmental changes and reacting to them. In particular, concerns over workstation security have encouraged the development and use of automatic updates for both the operating system and virus detection software. In principle, these updates could be implemented without human intervention but in practice their installation often has to be initiated and overseen by the user in case problems occur.

Operating systems embody some aspects of the *policy* function (S5) as part of their basic design. Where they are currently weak, however, is in making these policies visible and in having a reasonably well developed 'world view' of the environment in which they exist. For example, in the case of reporting low toner on a printer, as mentioned in the introduction, this situation would typically be handled by reporting the problem to the person using the printer rather than the technician responsible for its maintenance, as the concept of local

technical support is not normally part of the design of a personal workstation.

In terms of overall effectiveness, it seems highly desirable to have some means of describing the context in which the workstation is being used so that the operating system can interact appropriately with that environment and support the wider business function. The next section describes Soft Systems Methodology, which appears to have the potential to meet this need.

**Table 1. The Major Subsystems of the Viable System Model**

| System Type | Structure |
|---|---|
| System One (S1): *Operations*. S1s perform the productive operations of the system, with each providing a distinct product or service. An S1 contains an operational element controlled by a management process and is in contact with the operational environment. |  |
| System Two (S2): *Coordination*. S2s are concerned with coordinating the activities of S1 units. They are essentially 'anti-oscillatory' in that they attempt to contain or minimize inter-S1 fluctuations. This is achieved by the provision of stabilizing, coordinating facilities such as scheduling and standardisation information that is disseminated over all S1s, but tailored locally to suit individual S1 needs. |  |
| System Three (S3): *Control*. Each S3 is concerned with the provision of cohesion and synergy to a set of S1 units. The management processes contained within this system will be concerned with short-term, immediate management issues, such as resource provision and strategic plan production, where 'strategic' in this situation refers to planning with existing resources rather than in the normally accepted sense. |  |
| System Three* (read as System Three Star): *Audit*. S3* provides facilities for the intermittent audit of S1 progress and provides direct access to the physical operations of a particular S1 allowing immediate corroboration of that progress. This essentially provides additional data over and above that provided by normal reporting procedures. |  |
| System Four (S4): *Intelligence*. S4 is concerned with planning the way ahead in the light of external environmental changes and internal system capabilities. To this end, S4 'scans' the environment for trends that may be either beneficial or detrimental to the system and constructs developmental organisational plans accordingly. To ensure that such plans are grounded in an accurate appreciation of the current system, the intelligence function contains an up-to-date model of system capability. |  |
| System Five (S5): *Policy*. S5 determines the overall purpose of the system i.e. defines the activities that are performed by S1s. As such, S5 represents the policy-formulation or normative planning function. Policy formulation is informed by a 'world-view' provided by S4 and representing the current beliefs and assumptions held by the system about the environment and models of current system capability, populated by data flowing from the lower level systems in the organisation. |  |

## 3. SSM and Autonomic Systems

Like VSM, Soft Systems Methodology (SSM) [12, 13, 24, 30, 31] has a long-standing, well-respected pedigree in the systems community. It emerged from work that began at the University of Lancaster (in the UK) in the late 1960s and evolved through action research up to 1990, when it reached its current stable form [12, 24]. It is essentially a general systems improvement technique that helps identify opportunities for beneficial change by promoting a better understanding of a 'problem situation' among system stakeholders. This is achieved through the construction of relevant system models. The models and the process through which they are constructed promote discussion and debate about possible improvements and lead to recommendations for change. The approach is applicable to any problem situation but its use in information systems development has received particular attention [27]. In this work, SSM has been used as the first stage of analysis to provide context information for subsequent development through the models it creates. SSM seems particularly relevant to autonomic systems development as its models can serve as their environment description.

Classically, SSM has been described as a seven-stage process [12], as illustrated in Figure 2. There are five stages associated with so-called 'real world thinking': two of them for understanding and finding out about a problem situation (1, 2), and the other three for deriving change recommendations and taking action to improve the problem situation (5-7). There are also two stages (below the dotted line) concerned with 'systems thinking' (3, 4), in which root definitions and conceptual models are developed. Each root definition provides a particular perspective of the system under investigation. A conceptual model defines activities necessary to implement the perspective given in a root definition.

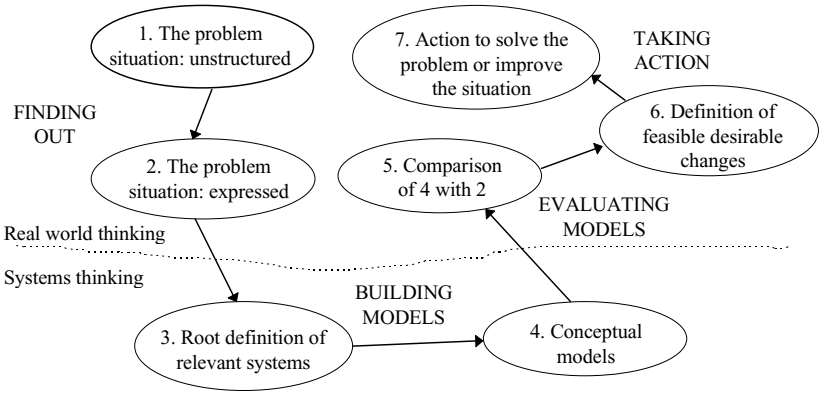



**Figure 2: Seven-Stage Soft Systems Methodology Model**

One broad perspective on workstations is that they exist to facilitate business activities in the workplace, through the provision of suitable hardware and software. Another, more specific perspective, is that workstations are there to facilitate communication with customers and among staff, using e-mail, document production tools, and support for audio visual presentations.

SSM root definitions are structured descriptions of individual perspectives. In general, each root definition identifies or implies six particular pieces of information, as listed in Table 2.

**Table 2: General Components of a Root Definition**

| Components | Meaning |
|---|---|
| *Customers* | The beneficiaries or victims of a system |
| *Actors* | The agents who carry out, or cause to be carried out, the main activities of the system |
| *Transformation* | The process by which defined inputs are transformed into defined outputs |
| *Weltanschauung* | A viewpoint, framework, image or purpose, which makes a particular root definition meaningful |
| *Owner* | Those who own a system (have the power to close it down) |
| *Environment* | Influences external to a system that affect its operation |

The 'Weltanschauung', or world-view, identifies *why* a system exists, and the 'transformation' indicates *what* the system does to achieve its purpose. These are the two most important elements of the root definition but their meaning is perhaps not obvious from the general descriptions shown.

As an illustration of the form of the six components, Table 3 gives possible descriptions for the 'enabling technology' perspective on workstations. This applies to some unnamed host organisation (*owner*), based on the belief that computing technology facilitates business activities (*Weltanschauung*). Technical and management staff (*actors*) are responsible for providing a computing service (*transformation*) to operational staff (*customers*), with the constraints that the technology should function as required and be cost effective (*environment*).

**Table 3: 'Enabling Technology' Root Definition**

| Components | Meaning |
|---|---|
| *Customers* | Operational staff |
| *Actors* | Technical and management staff |
| *Transformation* | Provide a computing service |
| *Weltanschauung* | Computing technology facilitates business activities |
| *Owner* | Host organisation |
| *Environment* | Technology should function as required; technology must be cost effective |

A root definition is usually presented as a single statement combining its six individual components. In this case it might be:

*A [host organisation] owned system, operated by [technical and management staff], to [provide a computing service] to [facilitate business activities] of [operational staff], taking account of the need for the [technology to function as required and be cost effective].*

Further root definitions can be produced in the same way to describe other perspectives, such as the communications viewpoint mentioned above.

Each root definition is expanded into a conceptual model, defining the activities necessary for the system to meet the purpose specified, and indicating relationships among the activities involved. For example, Figure 3 shows a conceptual model based on the enabling technology root definition described in Table 3. The activities have been labelled for convenience.

The model includes the transformation taken directly from the root definition (A1). This is essentially the central activity of the model. Another important activity is A2, which monitors that the defined Weltanschauung (viewpoint) is achieved, taking control action if necessary (TCA), which can affect any other activity in the model. Activities are also added to handle the environmental constraints listed in the root definition (A3-A6) and to cover consequential or implied activities (A7-A10).

Although conceptual models are largely informal—in that the meaning of each activity identified is described solely by the text displayed in the diagram, and the linking arrows simply imply relationships between activities with no accompanying labels or explanations—they do provide a good basis for debate about the meaning and implementation of activities within a system. For example, in deciding that computing facilities should be cost effective, this raises the rather difficult issue of assessing the benefit of the computing technology to the operation of the organisation, which needs to be resolved. Conceptual models can also provide a basis for further analysis towards the development of specific implementations of change, such as the creation or enhancement of information systems [24, 26] or simply computing systems [10]. They can also form the basis of other types of model such as dataflow diagrams [8], process models [9] and object models [10, 11, 28].

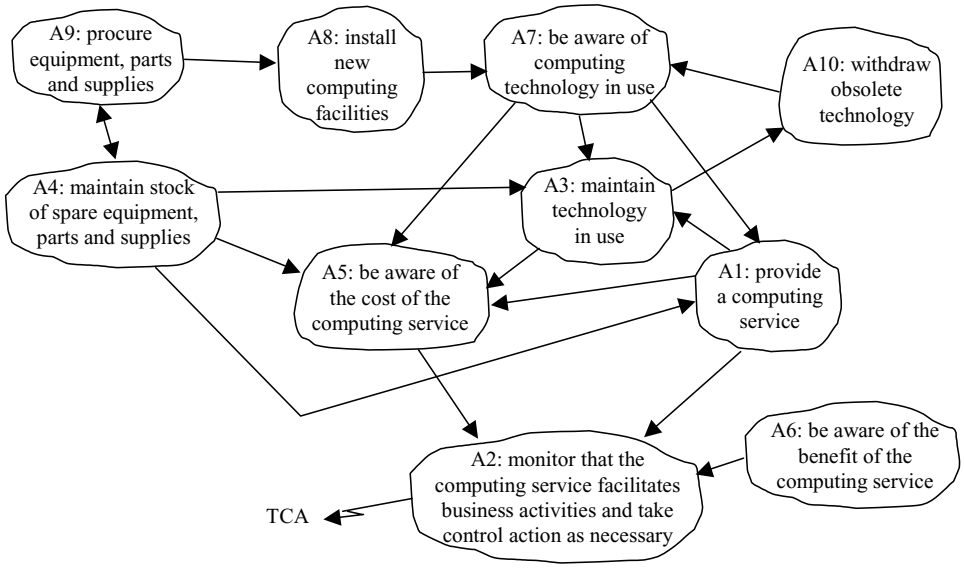



**Figure 3: Conceptual Model for Enabling Technology Perspective of an Organisation**

Conceptual models are hierarchical, allowing for the possibility of each individual activity being expanded into a full conceptual model in its own right. Such expansion seems desirable, for example, to explain A1: *provide a computing service* and A3: *maintain technology in use*.

In relation to supporting the development of autonomic systems, conceptual models offer a number of advantages:

- They clarify the role of the autonomic system in the wider business context and so encourage the development of a design that takes business objectives into account. For example, knowing that the system is meant to be 'cost effective', it would be possible to keep a record of the use of individual applications to compare against the cost of providing those applications.

- Consideration of the wider context helps identify opportunities for additional improvement. For example, if application usage is tracked then it will be possible to identify applications that are not used and so can be withdrawn.
- Analysis of conceptual models reveals the need for computing support systems. For example, in the operating system case, databases will be required to keep track of equipment and spares. These then form part of the environment for the autonomic system and can be consulted in performing some of the higher-level functions of the system. For example, a database of licence agreements can be referenced when auditing software use on a particular workstation; similarly, details of the stock of toner cartridges in the equipment database can be monitored against printer usage to trigger a replacement order or inform a decision to maintain a higher stock level.

## 4. Evaluation

The preceding two sections have outlined the principles of VSM and SSM and indicated how they might be relevant to the design of autonomic systems. In effect, these ideas have been presented as an enhancement to some pre-existing, but unspecified, design process. The implied steps are: (i) design software as usual; (ii) validate and refine the design using VSM; and (iii) extend and refine the design through integration with an environment model developed using SSM (in particular, this supports the implementation of the 'policy' subsystem required by VSM).

Although this enhanced design approach has merit, there are some significant costs involved. The use of VSM, however, is not expensive. Its checklist of six activities (subsystems) necessary for viability is both easy to understand and to remember so little effort is required to become familiar with the approach. Using the VSM concepts effectively, however, as in any creative endeavour, still relies on the skill and experience of the designer in being able to look critically at a design and appreciate the requirements and implications of the viable model with respect to that design.

In effect, the VSM is a design pattern [17, 22, 23], which can also influence the way that designs are created as well as being used for validation. With practice, this pattern will shape the thinking of the designer so that he/she is likely to incorporate the viable subsystems without conscious effort.

Using SSM as an add-on to an existing design process is much more problematic. Firstly, significant effort is required to understand the methodology to a level where it can be applied effectively. This will necessarily involve practical experience and not just academic study. It is not difficult to produce models that look plausible but again, like all design, achieving relevant content that captures the situation adequately requires aptitude and practice.

A second difficulty in using SSM at the end of the design process is that there is likely to be a significant mismatch between the system structure implied in the SSM models and that developed by more traditional means. In particular, SSM, through encouraging a consideration of multiple perspectives of a situation—separate from 'real world' constraints—yields models that are often significantly different from the way an organisation is currently structured. In that respect, it provides good support for business process reengineering. Unfortunately, that also means a likely mismatch with any design that reflects the current organisational structure, and a consequential difficulty in linking the SSM derived environment model to that structure.

A third difficulty is that SSM studies can take a significant amount of time to complete. Examples in the literature often suggest an analysis time of months [25] and certainly it seems unlikely that useful models can be produced in anything less than a few weeks even if the analyst already has some understanding of the problem situation.

The only reasonable conclusion that can be drawn from this evaluation is that SSM is not a practical way of adding an environment description to an existing autonomic design. Even if the cost of developing expertise in the methodology is ignored, and time is available to carry out a particular SSM study, the significant refactoring needed to integrate the SSM and separately created design models is unlikely to be acceptable.

Despite this bleak assessment, all is not lost, but to gain the desired benefit from SSM its use must begin earlier in the design process. In particular, it needs to be used as the first stage of systems analysis to avoid any design conflicts. Fortunately, this is not an unreasonable suggestion. Indeed, it is simply a particular implementation of the top-down approach to systems analysis recommended by Ackoff, another highly regarded systems theorist and practitioner. Specifically, Ackoff identifies three stages of analysis, as follows [2]:

- Identify a containing whole (system) of which the thing to be explained is a part
- Explain the behaviour or properties of the containing whole
- Then explain the behaviour or properties of the thing to be explained in terms of its role(s) or function(s) within its containing whole

The "thing to be explained" in this case is the autonomic system and the "containing whole" is its environment. There has been considerable positive experience reported in using SSM as the first stage in the analysis of information and computing systems [10, 27] so SSM appears to provide a good basis for autonomic system development if the top-down strategy is acceptable.

An overall design process using both SSM and VSM is summarised in Figure 4.

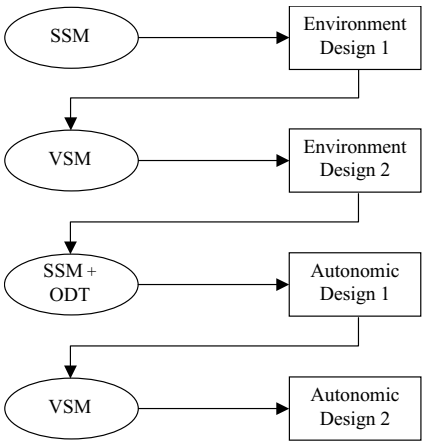



**Figure 4 A Systems-Oriented Autonomic Design Process**

This is a four-stage process. In stage 1 an environment design (system model) is developed using SSM. This is then refined using VSM to ensure that the viable requirements are covered adequately. At this point the context for the autonomic computing system will have been defined. The SSM models can then be taken down to a lower level to describe the autonomic system, coupled with other computing-oriented modelling techniques such as UML (ODT) to produce the autonomic system design. Again this design can be tuned by evaluating it with respect to VSM in the fourth and final stage.

## 5. Conclusions

This paper has examined the potential benefit of using well-established general systems ideas in the design process for autonomic systems. It first considered the possibility of simply extending existing design approaches with further systems analysis to enhance the structure of designs produced traditionally.

One conclusion is that Beer's Viable Systems Model (VSM) seems to be a useful aid to ensuring that there are adequate management controls within a design. Specifically, it can be used to assess autonomic designs to ensure that they have elements for ensuring system viability from broad policy-setting down to the monitoring and control of individual activities. It is relatively easy to appreciate the essential concepts involved in VSM and to use them as a checklist in validating designs; thus, the cost-benefit balance seems satisfactory. If VSM is used in this way it is expected that designers will progressively absorb the implied design pattern for viable systems and use it naturally (subconsciously) in their normal design process.

Similarly, Checkland's Soft Systems Methodology (SSM) was considered as a way of adding an environmental model to an autonomic system design. It was concluded that while it was possible to develop the required environment description this way the cost of integrating it with an existing system design is unacceptable. Instead, it was proposed that SSM be used as the first stage of design and so influence the complete design process. In the past, this has been recommended for the design of computing system, in general [10], and corresponds to a general top-down approach to systems analysis. It will be important to find ways of minimising the cost of performing this wider analysis to ensure general acceptability of the approach for autonomic systems.

The conclusions drawn seem plausible in that the individual techniques described have been used successfully in general systems areas. Nevertheless, the authors recognize the importance of looking specifically at autonomic computing systems to bring out any specific issues involved and to add confidence to the conclusions drawn. It is therefore planned to run a number of experiments of different sizes to further explore the systems-oriented design process proposed.

As recognised by others working in the autonomic field [32], the role of design patterns is likely to be of growing importance. Patterns are relevant at the environment level as well as within the autonomic system. Herring and Kaplan have proposed a viable systems architecture based on VSM, which may provide the best fit for the systems-oriented process proposed here.

In general, further work is needed to harness and improve the usability of VSM and SSM to validate a design but, overall, the general systems ideas seem to have substantial potential for providing a framework for autonomic system design and development.

## Acknowledgements

This work was undertaken through the Centre for Software Process Technologies, which is supported by the EU Programme for Peace and Reconciliation in Northern Ireland and the Border Region of Ireland (PEACE II). Further support for the work has been provided by the EPSRC project "Towards a Disciplined

Approach to Integrating Decision-Support Systems for Breast Cancer Care Activities" (GR/R86782/01).

## References


[1] Ackoff, R.L., *Towards a System of Systems Concepts*, in *Systems Analysis Techniques*, J.D. Couger and R.W. Knapp, Editors, John Wiley & Sons, Chichester. pp. 27-38, 1974.

[2] Ackoff, R.L., *Creating the Corporate Future*, John Wiley & Sons, Chichester, 1981.

[3] Bantz, D.F., *et al.*, *Autonomic personal computing,* IBM Systems Journal, Vol. 42, No. 1, pp. 165-176, 2003.

[4] Beer, S., *The Heart of the Enterprise*, John Wiley & Sons, Chichester, 1979.

[5] Beer, S., *Brain of the Firm*, 2nd ed, John Wiley & Sons, Chichester, 1981.

[6] Beer, S., The Viable System Model: Its Provenance, Development, Methodology and Pathology, Journal of the Operational Research Society, Vol. 35, pp. 7-26, 1984.

[7] Beer, S., *Diagnosing the System for Organizations*, John Wiley & Sons, Chichester, 1985.

[8] Bustard, D.W., Oakes, R. and E. Heslin, Support for the Integrated Use of Conceptual and Dataflow Models in Requirements Specification, Colloquium on Requirements for Software Intensive Systems, pp. 37-44, DRA Malvern, 1993.

[9] Bustard, D.W. and P.J. Lundy, Enhancing Soft Systems Analysis with Formal Modelling, IEEE Requirements Engineering Symposium (RE'95), York, pp. 164-171, 1995.

[10] Bustard, D.W., Dobbin T.J. and B. Carey, Integrating Soft Systems and Object Oriented Analysis, IEEE International Conference on Requirements Engineering, Colorado Springs, USA, pp. 52-59, 1996.

[11] Bustard, D.W., He, Z., and F.G. Wilkie, Linking Soft Systems and Use-Case Modelling Through Scenarios, Interacting with Computers, 13, pp. 97-110, 2000.

[12] Checkland, P., *Systems Thinking, Systems Practice (with 30-year retrospective)*, John Wiley & Sons, Chichester, 1999.

[13] Checkland, P. and G. Scholes, *Soft Systems Methodology in Action*, John Wiley & Sons, Chichester, 1990.

[14] Cheng, S-W, *et al.*, Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure, IEEE International Conference on Autonomic Computing (ICAC'04), New York, pp. 276-277, 2004.

[15] Ganek, A.G. and T.A. Corbi, *The dawning of the autonomic computing era,* IBM Systems Journal, Vol. 42, No. 1, pp. 5-18, 2003.

[16] Gracanin, D., Bohner, S.A. and M. Hinchey, Towards a Model-Driven Architecture for Autonomic Systems, IEEE ECBS 2004, pp. 500-505, 2004

[17] Herring, C. and S. Kaplan, The Viable Systems Architecture, HICSS, Hawaii, 2001.

[18] Herring, C., Viable Software: The Intelligent Control Paradigm for Adaptable and Adaptive Architecture, PhD Thesis, University of Queensland, Brisbane, Australia, 2002

[19] Horn, P., Autonomic computing: IBM perspective on the state of information technology, in AGENDA 2001, Scotsdale, AR., 2001.

[20] IBM, *alphaworks Autonomic Computing site,* http://www.alphaworks.ibm.com/autonomic,

[21] Jackson, M., Software Requirements & Specifications: A Lexicon of Software Practice, Principles and Prejudices, Addison-Wesley, 1995.

[22] Laws, A.G., et al., From Wetware to Software: A Cybernetic Perspective of Self-Adaptive Software, in Self-Adaptive Software: Applications, Second International Workshop on Self-Adaptive Software, R. Laddaga, P. Robertson, and H. Shrobe, Editors, Springer-Verlag: Berlin, 2003.

[23] Laws, A.G., A. Taleb-Bendiab, and S.J. Wade, Towards a Viable Reference Architecture for Multi-Agent Supported Holonic Manufacturing Systems, Journal of Applied Systems Studies, Vol. 2, No. 1, 2001.

[24] Mingers, J., An Idea Ahead of its Time: The History and Development of Soft Systems Methodology Systemist, Vol. 24, No. 2, pp. 113-139, 2002.

[25] Mingers, J. and S. Taylor, The Use of Soft Systems methodology in Practice, Journal of Operational Research, Vol. 43, No. 4, pp.321-332, 1992.

[26] Sterritt, R. and D.W. Bustard, Towards an Autonomic Computing Environment, in 1st International Workshop on Autonomic Computing Systems at 14th International Conference on Database and Expert Systems Applications (DEXA'2003), Prague, Czech Republic, 2003.

[27] Stowell, F.A. (ed.), *Information Systems Provision: The Contributions of SSM*, McGraw-Hill, London, 1995.

[28] Guo, M., Wu, Z. and F.A. Stowell, *Information Systems Specifications Within the Framework of Client-Led Design*, in *Systems Modelling for Business Process Improvement*, Bustard, D.W., Kawalek, P., and M.T. Norris, Editors, Artech House, pp. 199-212, 2000.

[29] White, S.R., An Architectural Approach to Autonomic Computing, IEEE International Conference on Autonomic Computing (ICAC'04), New York, pp. 2-9, 2004.

[30] Wilson, B., *Systems: Concepts, Methodologies and Applications*, 2nd ed, John Wiley & Sons, Chichester, 1990.

[31] Wilson, B., *Soft Systems Methodology: Conceptual Model Building and Its Contribution,* John Wiley & Sons, Chichester, 2001.

[32] Shackleton, M. Saffre, F., et al. "Autonomic Computing for Pervasive ICT – a whole-system perspective", BT Technology Journal, Vol. 22, No. 3, July 2004.