

## **A Systems Model for the Field of Informatics**

**A. J. Cowling**

**Department of Computer Science, University of Sheffield,  
Regent Court, 211, Portobello Street,  
Sheffield, S1 4DP, United Kingdom**

**Email: A.Cowling @ dcs.shef.ac.uk  
Telephone: +44 114 222 1823 Fax: +44 114 222 1810**

### **Abstract**

*Where the different volumes of Computing Curricula 2001 refer to similar material they often adopt different structures for it, and these differences create problems in defining curricula that cross the boundaries between the different disciplines within informatics. After analysing these structures, this paper proposes a new structure for a curriculum model that is intended to cover the whole field of informatics. This model has three dimensions, for the aspects of informatics corresponding to products, processes and people, where each dimension has a lattice structure that is derived from the application of general systems theory to information systems. The principles underlying the model are discussed, and the structure of each of the dimensions is described in detail. The model is evaluated by comparisons with Denning's "Great Principles" model and with the one used in the "Computing Ontology" project, and against the criterion that its structure should enable it to model the relationships between different topics within informatics that are important to the organisation of a curriculum.*

### **Keywords**

*Curriculum models, curriculum principles, computing knowledge, computing topics, systems theory, software systems, software development processes, people in information systems.*

### **1. Introduction**

The construction of Computing Curricula 2001 (CC2001 from now on) has been an enormous intellectual achievement. At the same time, though, this activity has stretched to the limit the fundamental structuring concept on which it has all been based, namely that of the simple hierarchy. This concept was used to divide up the whole field of informatics (or computing, as it is usually termed in the UK and USA) into an overlapping set of disciplines, as documented in the overall volume (ACM/IEEE, 2005), or CC2005 from now on. It was also used in the models for these disciplines, which were each documented in their own volumes: Computer Science (CS from now on) in CS2001 (ACM/IEEE, 2001), Computer Engineering (CE) in CE2004 (ACM/IEEE, 2004b), Software Engineering (SE) in SE2004 (ACM/IEEE, 2004a), Information Systems (IS) in IS2002 (Gorgone *et al*, 2002) and Information Technology (IT) in IT2005 (ACM SIGITE, 2005). Each of these models uses a hierarchy that divides its discipline into knowledge areas, knowledge areas into knowledge units, and knowledge units into a set of topics, which may also have learning outcomes identified for the study of these topics.

The weakness of this hierarchical structure is that it does not explicitly represent relationships between topics that are assigned to different knowledge units, and so the volumes where such relationships are significant have had to document them by cross-referencing between the topics. Hence, university departments or schools that are just operating a single undergraduate degree programme in informatics, which aligns closely with one of the identified disciplines, can

readily apply the CC2001 model to their programme, without these relationships between topics creating significant problems. By contrast, in an institution such as the author's (the Department of Computer Science at the University of Sheffield), which operates programmes in several of these disciplines, the problems of reconciling the different hierarchical structures become much more apparent. This because of the economic requirement to provide courses (the term used in CC2001 to describe what are otherwise often called modules or course units) that can be shared between the different programmes, and that therefore have to satisfy the often conflicting demands that arise from the different hierarchical structures of the relevant discipline models.

One approach to reconciling these different structures (Cowling, 2001) is to define "interfaces" through which the needs of one discipline can be applied to material that belongs to another, but it is not clear that this approach can scale up to situations where ownership of material should be shared between several disciplines. For instance, much core computing material is shared by CS and SE, amongst others, but SE2004 treats as single topics material that in CS2001 forms complete knowledge units, or in one case (Programming Fundamentals) a complete knowledge area. This creates problems for users of these models, who expect that topics should be of roughly the same "size" in terms of the amount of knowledge that is to be covered, but this is not possible when there are such major differences in structure. The models try to address this by documenting the sizes of knowledge units in terms of lecture hours, while also warning that this is an unsatisfactory measure, but the main effect of this is just to emphasise how uneven some of the hierarchical structures are. For instance, in SE2004 two knowledge areas out of the ten account for over half of the material. Hence, even when the recommendations from the different volumes are put together, the user is still left with comparatively little guidance as to how much time should be allocated to some of the topics in these large knowledge areas.

Such issues make it difficult for institutions that only have experience of delivering one discipline to widen the set of programmes that they offer, in order to extend to other disciplines as well. This suggests a need for some alternative model for curriculum structures within informatics as a whole, which should be more flexible than the simple hierarchical one that has been used in CC2001, so that it can also describe naturally the relationships amongst those topics that are shared across several disciplines within informatics. Hence, the purpose of this paper is to propose such a model, by describing the principles of its structure and providing illustrations of how it operates for certain key topics. In doing this the paper does not just add detail to the earlier extended abstract (Cowling, 2006a) and associated conference presentation (Cowling, 2006b), but also develops from it.

The structure of the rest of the paper is therefore as follows. Section 2 describes the fundamental principles that have been adopted in creating this curriculum model, and in particular that it defines a "knowledge space" with three dimensions: products, processes and people. Sections 3 to 5 go on to describe each of these dimensions in turn, showing how concepts from general systems theory are used in defining their structures, while section 6 illustrates applications of the whole model. The model then needs to be evaluated, and so sections 7 and 8 compare it with two external sources, namely Denning's (2003) "Great Principles" model, and the results from the "Computing Ontology" project (Cassel *et al*, 2005) respectively. Section 9 then evaluates the model against internal criteria, and section 10 summarises the conclusions of the paper and discusses possible further work.

## 2. Structuring Principles

Three basic principles have been used in creating the structure for this new curriculum model. The first and most important of these is the principle that the inter-relationships between topics within the whole field of informatics arise essentially from just three different aspects of

informatics, namely the products that are created within it, the processes by which these are created, and the people who are involved (in whatever capacity) with these products and processes. While these three aspects are not completely orthogonal, the basis of the model is that they are sufficiently independent that these inter-relationships can best be understood by treating each of these three aspects as a separate dimension, and then organising the various topics within informatics into the resultant three-dimensional space.

## **2.1 The Three Dimensions**

The importance of these three aspects was identified originally (Cowling, 1994) in the context of trying to define a framework for the SE curriculum, when this was still a relatively new discipline for undergraduate programmes. That paper made some proposals for curriculum structures for each of the three aspects, and the generalisation of treating each aspect as a dimension of an SE knowledge space has developed from these in stages. The first stage was to define some structure for the products dimension, based on the different levels of abstraction at which the components of products could be described (Cowling, 1998). Along with this, some other possible dimensions for the knowledge space were proposed, with the particular aim of trying to describe how SE as a whole related to some of the other disciplines within informatics, viz CS, CE and IS. The second stage was during the development of the SE2004 model, which had taken as one of its principal starting points the structure of the SE Body of Knowledge, or SWEBOK (Bourque & Dupuis, 2004). This structure was defined largely in terms of the dimension concerned with software development processes, and so this stage mainly involved showing how the product and process dimensions fitted together (Cowling, 2005), while the dimension concerned with people received comparatively little attention.

While these two stages focussed just on SE, the work described here has extended this concept of a three-dimensional knowledge space to the whole field of informatics. This has been underpinned by the second basic principle, which is that informatics as a whole is concerned with studying particular kinds of systems, namely those that manipulate information, and that do so at least in part by using computer technology. Such systems are usually referred to as information systems (not to be confused with the discipline of IS), but people are also important elements in them, which brings in the third dimension.

## **2.2 The Application of Systems Theory**

This centrality of information systems to informatics has been recognised for a long time by the British Computer Society (BCS), which until its recent restructuring in 2004 described itself essentially as the professional body in the UK for information systems engineering, and as such was and still is concerned with professional accreditation for degree programmes in all of the disciplines comprising informatics. For a long time this accreditation process put a particular emphasis on the engineering aspects of programmes, but more recently its focus has broadened, so that it no longer puts such an emphasis on the engineering aspects of programmes. It is this broadening in the application of information systems concepts that has partly motivated the application of these concepts in extending this model of a three-dimensional knowledge space from SE to informatics as a whole.

The specific concepts from information systems engineering that are used here are those that involve the application of general systems theory to information systems, which can be summarised as follows.

- Every system must have a purpose, which will involve the carrying out of some activity. For an information system this activity will essentially consist of the processing and storage of information.

- Every system must have a boundary, and its activity will involve interactions across this boundary. For an information system these will consist of communicating information by some means to and from entities outside the boundary of the system.
- Every system will be structured from sub-systems using some structuring paradigm, where the sub-systems will either be systems in their own right, or components that are so fundamental that they can not usefully be further sub-divided.
- In every system the internal sub-systems must interact in order to achieve the overall purpose of the system, where the interactions will either be with each other or across the system boundary. For an information system these internal interactions will also consist of the sub-systems communicating information amongst themselves by some means.

These concepts can then be applied within each of the three dimensions of the model, although the precise application of them will obviously vary between the different dimensions.

### 2.3 Hierarchical Structuring

As well as using this three-dimensional structure, and applying the concepts of general systems theory within each of the dimensions, the third basic principle used in this model is that of the hierarchical structure. Since the original presentation (Cowling, 2006a) of this model, though, it has become apparent that there are actually two forms of hierarchical structure, which need to be distinguished carefully. These two forms can be described as exclusive and inclusive hierarchies, where the difference between them concerns the relationships that can hold between the elements in layer  $n$  of a hierarchy when there are a number of elements in the layer above (ie layer  $n-1$ , if the layers are numbered so that the root is layer zero). In an exclusive hierarchy, for any elements  $e1$  and  $e2$  in layer  $n-1$ , the direct descendents of  $e1$  (which will be in layer  $n$ ) must all be different from any of the direct descendents of  $e2$  (which will also all be in layer  $n$ ). By contrast, in an inclusive hierarchy sharing of the elements is permitted, so that for some  $e1$  and  $e2$  there may well be elements  $e3$  in layer  $n$  that are direct descendents of both  $e1$  and  $e2$ .

For instance, in any physical system (such as a piece of hardware) the hierarchical structure of its components must be an exclusive hierarchy, since any component in layer  $n$  must be part of exactly one component at layer  $n-1$ , even when its role in this layer  $n-1$  component is as part of the interface to other components. By contrast, in software the whole point of constructions such as subroutines, procedures or methods (ie components in layer  $n$ ) is that they should be shared between different components in the layers above (eg layer  $n-1$ ), and so the hierarchical structure of a software system will nearly always be an inclusive hierarchy. Furthermore, even where the actual components form an exclusive hierarchy, any layer will usually contain multiple occurrences of the same kinds of components, so that the kinds of components will form an inclusive hierarchy, and so too will the knowledge about them and their use.

In terms of mathematical structures, this means that an exclusive hierarchy forms a tree structure, whereas an inclusive hierarchy must be represented instead as a lattice structure. This difference explains some of the problems that have been encountered so far with conventional curriculum models, for they have tried to use tree structures (ie exclusive hierarchies) to represent concepts that actually form inclusive hierarchies. Therefore, the hierarchical structures used in this model are treated as lattices, in order to reflect their inclusive nature.

### 2.4 The Overall Model Structure

The overall structure of the model is therefore not just a three-dimensional knowledge space, because the individual dimensions are not just linear structures. Rather, each dimension is a lattice structure, which reflects the way in which the concepts of general system theory are applied within it. Hence, the model organises the knowledge into three orthogonal lattices, one

for each of the dimensions, rather than into the “three intersecting hierarchies” described in the earlier extended abstract (Cowling, 2006a). Consequently, it is no longer immediately obvious for any dimension which end of the hierarchical structure should correspond to its origin, for while a tree structure has a root that would naturally correspond to the origin, a lattice structure does not necessarily have a single root at either top or bottom. Hence, the orientation of the lattices along the dimensions needs to be discussed separately for each.

This structure is then intended to allow the model to represent naturally the relationships between different topics within informatics, because they will result from similarities in at least one of these three dimensions, through common aspects in the way in which these topics relate to information systems. As one example, programming paradigms (such as, say, the functional and object-oriented ones) have strong similarities, in that each tries to abstract away from machine-oriented issues so as to provide approaches that are problem-oriented. This common purpose and level of abstraction is therefore represented in the model by locating them both in the same level of the hierarchical structure for the products dimension. Further examples will be given below, as each dimension is discussed in turn.

### **3. The Product Dimension**

Within informatics the products are ultimately information systems themselves, in which people (acting as parts of the systems) use or interact with sub-systems that are constructed from computer-based technology, and that therefore consist of software running on some suitable hardware infrastructure. In this model the relationships between people and information systems form a separate dimension, and so this product dimension just deals with those sub-systems that are computer-based. Its inclusive hierarchy therefore consists of the different kinds of systems that can be constructed from software running on suitable hardware, although modern developments mean that the boundary between hardware and software is becoming increasingly blurred. Hence, the underlying hardware is treated as a pervasive concept, so that the system will be referred to simply as software systems, and the layers in the lattice structure will correspond to the layers of abstraction of the various kinds of components within these systems.

Thus, the most primitive layer of abstraction will be concerned with analogue circuits, and on top of this one can identify three further layers for hardware concepts, concerned successively with digital circuits, digital components (such as processors and memories), and complete computers. The next two layers of abstraction are concerned with hiding features of the hardware, so that one of them will be concerned with operating system services and the other with fundamental programming concepts. The ordering of these two could be a matter of debate, since most operating systems only have a small kernel that needs to use programming constructions at the assembly language level, but increasingly the libraries provided with programming languages are designed to hide features of the operating system services as well as features of the hardware architecture, and so they are ordered here with the operating system services treated as more primitive than the programming paradigms. Finally, building on these there will be layers that will correspond respectively to basic programming abstractions (such as classes and their components), to applications programming interfaces (built from groups of related classes), and to complete software systems.

These basic layers for this hierarchy were identified originally as forming one of the dimensions for the model of the SE curriculum that was described in (Cowling, 1998), and subsequently that structure was adopted in (ACM/IEEE, 2005) as one of the two dimensions for the space that it used to characterise the different disciplines within informatics. This model includes an additional element, though, which is that general systems theory identifies a number of features

that characterise the components in every layer. These features correspond to the characteristics of systems described in section 2.2, as follows.

- Each level of abstraction corresponds to components with a particular kind of purpose. This purpose is, though, often implicit in the kinds of components themselves, and the consequences of this are discussed further below.
- Each level of abstraction uses a particular structuring paradigm in order to assemble the components in this layer from those in the next more primitive layer, and the nature of the level of abstraction is mostly determined by the characteristics of this structuring paradigm rather than by the characteristics of the components that it produces.
- Because these systems are information systems, each level of abstraction has a particular form for the processing that is carried out by components at this level.
- Similarly, because these systems are information systems, each level of abstraction also involves particular structures for organising and storing the information that the components at this level process.
- For the same reason, each level of abstraction also has specific mechanisms for communicating information at this level, both within the sub-systems defined at this level (ie internal communication) and between these various sub-systems (ie external communication).

Level of Abstraction	Structuring Paradigm	Processing	Storage	Internal Communication	External Communication
Analogue Circuits	Modulation	Gate circuits	Feedback	Analogue signals	Analogue signals
Digital Circuits	Clocking	Combinatoric logic	Sequential logic	Digital signals	Physical layer protocols
Digital Components	Micro-programming	Processors	Memories	Buses	MAC layer protocols
Computers	Assembly language	Interrupt handling, etc	Virtual memory	Device level I/O	Link & network layer protocols
OS Services	Imperative programming	Process management	Filing systems	Buffer management	Transport layer protocols
Programming Concepts	Programming paradigms	Translators, VMs	Data typing	Data streams	Session layer protocols
Programming Abstractions	ADTs	Procedures and methods	Data structures	GUIs, event streams	Application layer protocols
APIs	Software components	Various domain-specific abstractions			
Software Systems	Software architecture				

Table 1: Systems features for the layers of abstraction of products.

These various details for the different levels of abstraction are defined in table 1. At the most primitive levels of the hierarchy (corresponding to analogue and digital circuits) there are only a few kinds of components, fulfilling just one or two purposes, whereas at the levels that are closest to complete systems there is an enormous variety of purposes, corresponding to different application domains for software systems, to the abstractions that they use and the purposes for these abstractions. This indicates that the relationships between the layers are two-directional: in one direction they correspond to the composition of components in one layer from components in the next most primitive layer, while in the opposite direction they correspond to components that are more primitive, but also more universal, being used to produce a variety of more specialised components. Hence, the lattice is actually combining two hierarchies that run

in opposite directions: one for the components of systems and the other for the purposes of these components.

The hierarchy of purposes has a very obvious root, since at the level of analogue circuits there is only one purpose, namely to construct basic logic gates. By contrast, the hierarchy of components does not have any obvious root, since even if one considers actual software systems the way in which these are now so often interconnected means that it may be difficult to determine just what the root of the hierarchy should be for any given system, as exemplified in Meyer's dictum that "*real systems have no top*" (Meyer, 1988, page 47). Furthermore, the huge variety of domains for the systems is matched by a similar variety in the knowledge about those domains. Hence, it is natural to associate the origin of this dimension with the layer corresponding to analogue circuits, so that distance along the dimension is measured in terms of the number of layers of the lattice structure. Then, while this lattice is not a tree structure, the numbers of elements in each layer of it (corresponding to the kinds and purposes of components within that layer) will roughly increase as the distance from the origin increases, particularly once the domain-specific layers are reached.

#### 4. The Process Dimension

For the process dimension there are several possible ways of organising the relevant concepts into hierarchical structures. One, based on the observation that software development processes are a form of software (Osterweil, 1987), is to focus on the abstract structure of methodologies, as these describe how particular classes of development projects may be undertaken. Hence, the key components of this structure are the activities that a methodology requires to be incorporated into the process, and the rules for sequencing these activities within the process. Also, the individual activities involve manipulating models, and these must be expressed in notations (such as UML). This leads to the hierarchical structure shown in figure 1.

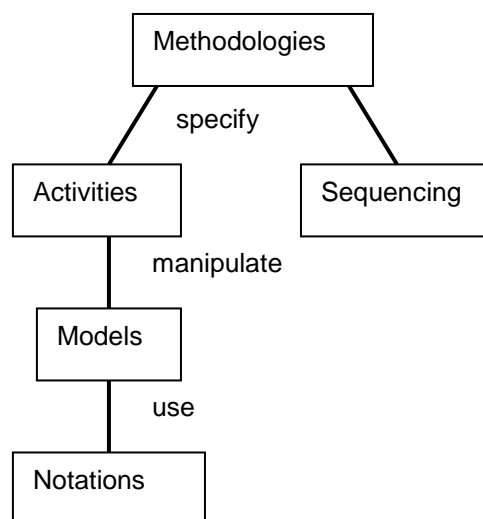


Figure 1: The abstract structure of processing within development processes.

Another way of organising these concepts is to focus on the management of the physical resources that are required for system development, namely people and tools. This has to start from a definition of the objectives for a development project and the constraints on it, and then the derivation from these of the work breakdown structure (WBS) and the schedule for that project, which form the basis for allocating the resources to the various activities within the project. These concrete features of projects form the instantiations of the processes, so leading to the hierarchical structure shown in figure 2.

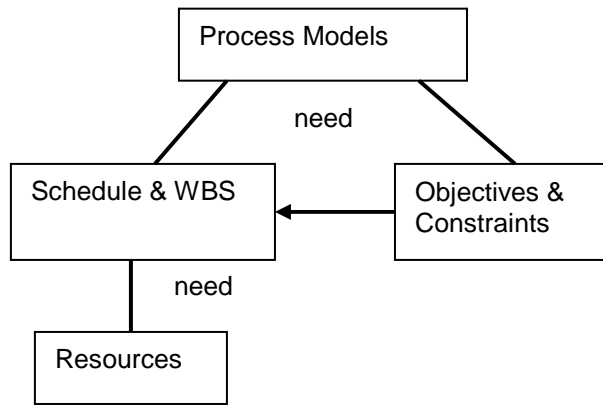


Figure 2: The concrete structure of development projects.

These two structures can then be combined, since the WBS from figure 2 defines a set of the activities from figure 1, so giving the model shown in figure 3, as described in the conference presentation of the earlier version of this material (Cowling, 2006b). This model does, though, have two weaknesses. Firstly, it is placing different concepts from general systems theory at different levels in the hierarchy instead of applying them all across the levels, for instance by showing the information contained in the models that are processed at a lower level than the activities that actually process this information.

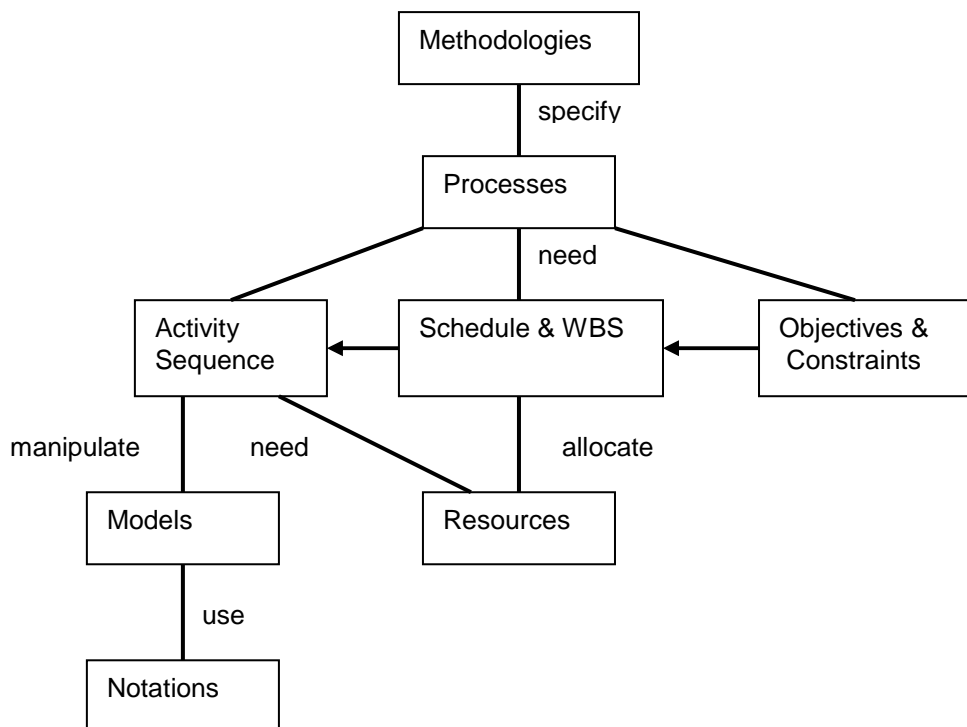


Figure 3: A combined hierarchical structure for development processes.

Secondly, it is not at all clear how the concept of different levels of maturity of processes should be represented within the model. As defined originally in the capability maturity model, or CMM (Herbsleb *et al*, 1997), each level of maturity for a development process is characterised by the incorporation into that process of additional features or activities, which in the CMM are known as key process areas, or KPAs. In terms of general systems theory, this suggests that in

principle each level of maturity should be seen as a system that has as its components a less mature process, plus other sub-systems corresponding to the additional KPAs for that level of maturity. In practice, though, the additional KPAs for some of the levels of maturity within the CMM do not correspond to additional sub-systems, but rather to more complex processing being carried out within the existing sub-systems. Hence, the resultant hierarchical structure, shown in table 2, actually has fewer levels of abstraction than there are levels of maturity in the CMM.

In this structure, the most primitive layer of abstraction corresponds to the basic activities that are carried out on the individual components of a system being developed, as in figures 1 to 3. These components (eg requirements documents, code, etc) will each be structured documents, but there will not necessarily be any unifying concept to their structures, which will vary to reflect the purposes of the components. The activities that are carried out on these structured documents will then consist either of creating and editing them, possibly using information contained in other documents, or of conducting various forms of quality review of them, possibly with automated assistance. As such they will involve either just individual developers or small teams, who internally will communicate amongst themselves or with the tools that they use to perform the activities. Then, the external communication will consist of documents that have been produced in one activity being used subsequently as inputs to another activity, and insofar as there is any co-ordination of the individual activities it will result from the fact that an activity can not start until the prerequisite documents have been produced.

<b>Level of Abstraction</b>	<b>Structuring Paradigm</b>	<b>Processing</b>	<b>Format and Storage</b>	<b>Internal Communication</b>	<b>External Communication</b>
Basic Activity	Depends on the nature of the outputs	Creation or review of documents	Structured documents	Person to person, person to tools, intra-team.	Documents between activities
Basic Process	Sequencing of activities	Following defined sequence	System models in some notation	Inter-team	Sets of documents to clients and managers
Quantitative SE process	Managed activity sequence	Plan, monitor, control	Models plus quantitative plans	Between teams and project managers	Models and quantitative plans to clients
Optimised SE process	Support of managed activities	Optimising of planning, monitoring and control	Repository plus experience factory	Between managed teams, clients and support functions	All stakeholders are part of the process

Table 2: Systems features for the layers of abstraction of processes.

This co-ordination is then formalised in the structuring paradigm at the next level of abstraction, which is where the activities are organised into some process, governed by some kind of WBS. Hence, the processing at this level consists of following the sequence of activities that is defined by this WBS, and the internal communication is therefore between the teams that are carrying out the different activities. Along with this the diverse structures of the different documents will be integrated into some more unified structure, which constitutes a system model.

At the next level of abstraction the additional feature is that processes are planned and managed on some quantitative basis, even if only the use of simple budgets for cost or schedule, or the balancing of these against quality or functionality. Hence, project managers also form part of the system, as do the quantitative plans that they make and the processes of monitoring and controlling the development work to ensure that it conforms to these plans.

In practice, of course, for this planning and controlling to be effective the clients for the projects often need to be brought into the management activities as well, as they are in agile methodologies such as Extreme Programming (Beck & Andres, 2005). Hence, the highest level of abstraction incorporates all of the stakeholders into the system. These and other activities that support the development process, such as the operation of unified project repositories, the mining of the repositories to provide an experience factory, and others (such as staff training) that are not shown in table 2, all contribute to forming a process that is as near to optimal as can be achieved.

Unlike the product dimension, in this dimension the hierarchies of purposes and of components both run in the same direction, for the most abstract layer essentially has just a single purpose, whereas at the most primitive layer there is a whole variety of purposes to the different activities. Also the educational requirements are different, for while different disciplines within informatics focus on particular sub-sets of the layers along the product dimension, as illustrated in CC2005, this is not the case for the layers in the process dimension. Rather, in this dimension thorough knowledge of the basic activities defined in the most primitive layer is fundamental to any understanding of the more abstract processes. Indeed, within SE (which is the discipline most concerned with the more abstract layers of this dimension) it has been proposed (Cowling, 2005) that students' knowledge needs to be developed in at least three phases, with phase zero covering programming and basic problem solving, phase 1 consisting of a limited form of SE termed "software development", and phase 2 consisting of a full coverage of SE. Here, a key feature of software development in phase 1 is that it is purely qualitative, with the quantitative aspects being deferred to SE in phase 2. Hence, the ordering of these three phases corresponds exactly to the ordering of the layers of this hierarchy from basic activities to quantitative SE processes. To reflect this, the most primitive layer in the hierarchy is therefore identified as the origin for this dimension, and then distance along the dimension corresponds both to the increasing complexity of the SE methods involved and to the order in which students will need to cover this material within a practical curriculum.

## **5. The People Dimension**

A fundamental element in the notion of an information system is that the processing of information can be done by people as well as by machines, and so people or groups of people can also be regarded as information systems. The simplest such systems will therefore consist of a single person, and more complex systems will consist of progressively larger groups or organisations of people. The key problem in defining the hierarchy of these is that of identifying meaningful distinctions between different sizes of groups, since real organisations come in all sizes, and contain sub-groups of all sizes. In practice, though, there are a number of levels of grouping that can be identified, even though the definitions of them will be very fuzzy.

Thus, for this purpose the next level up is that of what we will call the small group or team, where the fact that it is a team means that there must be at least two people, and small typically means that there are fewer than about ten people. This level is particularly significant when considering the role of people within the software development process, since for this purpose people will often be organised into teams of this sort of size. At the next level up one can identify what we will call business units, which will consist of a number of teams, where again the number could typically be anywhere from about two to ten teams. Thus, in the context of software development such a business unit would correspond to the group required to work on a project that was large enough to need several teams collaborating on it, while in the context of client organisations such business units might correspond to the groups needed to carry out a collection of related business functions.

Above the level of these business units, larger organisations might well require several further levels of hierarchy, but apart from their sizes there is little in the way in which they are structured or managed to distinguish them from business units, and so it is probably not relevant to the purposes of this model to try to identify layers in a general hierarchical structure to correspond to them. Hence, in this model the next level up is that of the complete business organisation, which for commercial or industrial organisations would typically be the level of a company. Of course, company structures can themselves be hierarchical, with groups that consist of a number of subsidiary companies, but again it is not relevant to the purposes of this model to try to identify these as separate layers. Thus, ultimately the top layer in the hierarchy is that of society as a whole, since this level will be relevant to many aspects of how information systems are applied in practice.

Within these layers for the hierarchy of information systems, though, there is little point in trying to document the other aspects that arise from general systems theory, for two reasons. One reason is theoretical, and comes from the development of the soft systems approach (Checkland, 1999), which has shown the limitations of general systems theory when applied to systems composed of people, since these are soft systems rather than the hard systems for which that theory was originally created. The other reason is practical, and is simply the enormous range of possibilities for how these aspects of general systems theory might apply to the systems. Specifically, any system involving people could be structuring their individual contributions in almost any way, and the people in it could be utilising almost any techniques to format and store information, to process it and to communicate it either amongst themselves or with people who they would regard as being part of other systems. Similarly, there is little point in trying to identify the roles that such people or groups of people play within information systems, for again there is an almost infinite variety of purposes that might be recognised for and by the people forming such systems.

This also affects a common distinction between two roles that people have in connection with information systems, namely either as users of the products or as participants in the processes. In this model these two roles result from the relationships between people and either products or processes respectively, and so they are represented by the relationships between the concepts in the people dimension and those in the other two dimensions. Hence, this distinction does not need to be reflected within the structure of the people dimension itself, so that this simply consists of the hierarchy of levels identified above, viz:

- individual people;
- small teams of people;
- business units;
- business organisations;
- society in general.

Given the huge variety of purposes at every level within this structure, one can not identify any hierarchy of purposes that might grow in either direction along it. Also, there are no educational grounds for arguing that knowledge about the roles of people needs to be developed in one direction or the other along the dimension, since technical activities are essentially individual, while it is well established that students also need to learn about the social, ethical and legal implications of applying informatics. Consequently, whatever discipline within informatics such students may be concentrating on, they need to cover at least both ends of this dimension. The selection of the origin of this dimension is therefore based simply on making the model more uniform, by associating it with the layer corresponding to the smallest components of the information systems involved, as is the case for the other two dimensions. Hence, the layer corresponding to individual people is selected as the one that is to be treated as the origin.

## **6. Applications of the Model**

Given this structure for the model, its operation can best be described by giving some examples of how particular topics are fitted into this structure. Four examples will be presented, spanning a variety of topics and disciplines within informatics: programming, database normalisation, software architecture and the performance of real-time transaction processing systems.

Programming is at the heart of CS, and has a major role in all of the other disciplines within informatics. It is concerned both with the structures of programs and the activity of creating them, which in the model is represented by placing it at the intersection of the relevant layers in the product and process dimensions. The activity is a basic one, and so in the process dimension it is part of the processing aspect of the basic activity layer. Also, while the associated technical skills are primarily individual ones, aspects of them involve pairs or teams, and so in the people dimension it is placed in these layers. Within the product dimension, though, programming then needs to be separated into two parts, for it firstly involves different paradigms, each with their own primitive control and data structures, and so the topics related to these are placed in the programming concepts level of the lattice. Secondly, programming also involves the more abstract concepts used in creating programs from these primitives, and the topics related to these are placed in the programming abstractions level of the lattice.

Database normalisation is also a topic that is shared by all of the CC2001 disciplines, although its importance varies significantly across them. In the products dimension of the model it is placed in the layer of the lattice for APIs, since databases are major components that can be used in any system that needs one. Their normalisation is then a basic design activity, and so is located in the processing part of this layer in the process dimension. This activity will primarily involve considering the needs of the business organisation that requires the database and the systems to use it, and so the topic is placed at this layer in the hierarchy of the people dimension.

By contrast, software architecture is a topic that belongs primarily to one discipline, namely SE, although some topics in CS (such as operating systems and compilers) involve aspects of it. In the products dimension it is located in the software systems layer of the lattice, since it is primarily concerned with the paradigm used for structuring large components into complete systems, although some aspects of it are shared with the APIs level, as the components at this level will also have internal architectures. Equally important is the placing in the process dimension, as illustrated by the way in which processes such as the Rational Unified Process, or RUP (Kruchten, 2000) are described as being “architecture-centric”, meaning that in these processes architectural design is not just a single basic activity, but affects many of the other activities as well. Hence, software architecture is located in the basic process layer of the lattice, where the system model that it creates provides the storage aspect. As a consequence, decisions about the architecture affect complete development teams, and so in the people dimension software architecture is located primarily in this layer of the lattice.

The performance of real-time transaction processing systems is a topic that is shared between SE and IS. In the development of such a system targets must be set for the performance that it is to achieve, which requires a cost-benefit analysis that has two sides. On one side, determining the costs of achieving some level of performance involves knowledge and skills from SE; while on the other side, determining the benefits to the client organisation of providing that level of performance requires knowledge and skills from IS. In the model this is reflected in the people dimension, where the topic of analysing how the throughput of such a system will affect an organisation’s ability to generate income from it is located in the business organisations layer of the lattice, while the topic of analysing the costs of achieving particular targets will, like other specialised development activities, be located in the individual layer. Within the other

dimensions this topic will be located in the software systems layer of the products dimension, since it is concerned with the specific abstractions involved in defining performance characteristics for such systems. Since these are quantitative properties, the activities involved in managing them will then be located in the quantitative SE layer in the process dimension.

Having thus illustrated how the model operates, the rest of the paper is concerned with evaluating it, and the first stage in this is to compare it with other models that have similar aims to this one. As noted in the introduction there are two of these: one that has already been developed, namely Denning's "Great Principles" model, which is considered next, and one that is still being developed in the "Computing Ontology" project, which is considered in section 8.

## **7. Comparison with the "Great Principles" Model**

The main aim of Denning's "Great Principles" model (GP model from now on) is to provide a basis for structuring the whole field of informatics, but without having to enumerate an ever-expanding list of knowledge areas, which he sees as the major weakness of the approach used in the CC2001 volumes. This GP model is also multi-dimensional, but it has two main dimensions rather than the three in our model, where these two dimensions are termed "principles" and "practices" respectively.

The "principles" dimension has two distinct elements. One is the "mechanics" element, which he describes in terms of five "windows", although these are clearly not intended to be orthogonal. In the order in which he tabulates them, they are: computation, which is concerned with what can be computed; communication; co-ordination of multiple entities; automation, which is concerned with achieving the purpose of using computing technology to perform cognitive tasks; and recollection, which is the term that he uses for the storage and retrieval of information. Hence, these windows correspond very closely to the various general properties of information systems that provide the structure for each layer of the lattices in our model.

Denning calls the other element in the "principles" dimension "design", and it corresponds to two features in our model: firstly the desirable properties of the systems (ie products), and secondly the relationships between the components of these systems that give rise to these properties. Thus, his description of this element begins with a list of concerns (viz: simplicity, performance, reliability, evolvability and security), which are similar to the categories forming the top level of the hierarchical structure for the quality of information systems in ISO standard 9126:1991 (International Standards Organisation, 1991). Then, the rest of his description is concerned with design principles, such as abstraction, information hiding, modules, separate compilation, layering, hierarchy, separation of concerns, reuse, and so on. While some of these might best be described as recurring concepts (a classification that sadly has not been used in CC2001), they essentially cover either techniques that are used in the design activity, or properties that these techniques should try to achieve. Hence, in our model these concepts would either belong primarily to the process dimension, or to the relationships between this and the products dimension.

The GP model's second dimension, the one for "practices", essentially just describes a set of elements that roughly correspond to some of the kinds of activities that form the process dimension in our model, although Denning actually describes these elements in an order that seems to correspond more to the hierarchies of products and people. That is, his first element is programming and his second is the engineering of systems, which are distinguished primarily by the scale of the products as well as by the complexity of the activities and processes involved. His third element in this dimension is modelling and validating, which in its simplest form is part of the basic activities level of our process hierarchy, but on the scale to which Denning is

referring it also involves the co-ordination of various such activities and quantitative modelling, which would be located in the next two layers of this hierarchy. Then, his fourth and fifth elements are innovating and applying respectively, and these are as much social as technical activities, and so in our model they would be related mainly to the people dimension.

Within the two-dimensional space formed by these “principles” and “practises” dimensions, Denning then identifies two further components, which he calls “core technologies” and “applications”, where the difference between these two corresponds precisely to the difference between the various layers of abstraction in the products dimension of our model. Thus, he lists the “core technologies” as including algorithms, databases, architectures, networks and operating systems; which in our model are located within the three layers of abstraction for products that cover operating system services through to programming abstractions. Then, his “applications” component corresponds to the different domains that in our model are reflected in the higher levels of the products dimension, and that arise from the enormous variety of purposes that correspond to systems within the people dimension.

Thus, many elements of the GP model and ours are similar, in that they identify similar sets of concepts, although the two models assemble these into different structures. In particular, some aspects of the GP model run together concepts that in our model are kept separate by using three dimensions rather than two. Also, having defined the structures for each dimension, our model does not then need to further partition the space that they create, whereas in the GP model the concept space needs to be further partitioned into the “core technologies” and “applications” components, in a way that in our model is reflected directly in the structure of the products dimension.

Hence, three conclusions can be drawn from this comparison, of which the first is that the similarities between key features of the two models validate both of them, particularly since these common features have been arrived at by different routes. The second conclusion is that our model is at least as powerful as the GP one, since it covers all of the concepts that are identified in the various components of that model. The third conclusion is that, while the use of three dimensions rather than two makes our model more complex, this additional complexity results in a cleaner structure, since the GP model does have to include additional features that involve some rather arbitrary grouping of concepts, and particularly of one that relate to processes.

## **8. Comparison with the “Computing Ontology” Project**

The aims of the “Computing Ontology” project, and of the model that it is attempting to create (the CO model from now on), are rather broader than those of this model, for it is not just concerned with providing a unified basis for curriculum structures within the whole of informatics, but also with the classification of research work and of the kinds of expertise that might be sought by employers. The CO model is still under development, and the current state of this was presented at the same conference (Cassel, 2006) as the initial version of the material in this paper. Some of the details of it here, and particularly the example of relational data models, are taken from that presentation (Cassel *et al*, 2006).

The CO model uses two layers to describe topics, although it is not clear how these layers are linked. In the more abstract layer it identifies four main headings: hardware, software, information and human related aspects, where topics may come under any three of these. It also adds headings for a systems view, an application context, a social context and process-related activities, any of which could apply to some topics. Hence, this layer has obvious similarities to our model, in that its headings of hardware and software form our products dimension, and its

headings of human related aspects and the social context correspond exactly to our people dimension. Also, its headings of information and the system view correspond to our use of general systems theory as applied to information systems in structuring each of the dimensions.

In the more concrete layer, the CO model constructs a network of topics and the relationships between them, which is represented in a form that combines UML class and object diagrams. This network is intended to be at quite a fine level of detail of the topics, and so an important part of the project is the development of tools to maintain the network model. This explicit modelling of relationships between topics has some equivalents in our model, even though it represents the relationships implicitly. For instance, the topic of relational data models is treated as an aggregation of four others: database content, database schemas (both concerned with the formatting and storage of the data), relational manipulation operators (concerned with the processing of the data) and relational database design (concerned with the structuring paradigm for the systems). These relationships therefore correspond to our identification of the aspects of systems theory that structure each of the layers in the lattices. Also, the topic of normalisation is treated as an aggregation of four others: normal forms, anomalies, multi-valued dependencies and joint dependencies. This corresponds to the relationship in our model between the activity of normalisation, which is located in the process dimension, and the relevant features of databases, which are located in the product dimension, so that this relationship is represented implicitly by the placing of this topic at this point in the three-dimensional space.

Similarly, the CO model uses explicit relationships to model the usage of one topic within another. For example, the topic of abstract data types uses the topic of programming paradigms, since both the algorithms and the data structures that are used in implementing an abstract data type must be expressed in terms of some programming paradigm, and how this is done depends on which paradigm is adopted. Such usage relationships correspond in our model to the implicit relationships between adjacent layers in the lattices, since in each dimension topics in each layer will implicitly depend on ones in the next more primitive layer (ie the one nearer to the origin).

Where the CO model goes beyond ours is in breaking down topics into a finer level of detail, and then also modelling the relationships between these. For instance, the topic of relational manipulation operators is treated as having two special cases, namely relational calculus and relational algebra, and the various normal forms (eg 1NF, 2NF, etc) are treated as instances of the general concept of a normal form. Such relationships have no equivalent in our model, since they are effectively defining an internal sub-structure for what might be regarded as the “cells” that are located at the points in our knowledge space.

This comparison also leads to three conclusions, where again the first is that the similarities between key features of the two models validate both of them, since again these common features have been arrived at by different routes. The second conclusion is that our model has a much clearer overall structure than the CO model, because of the use of common structures for the three dimensions and the representation of the relationships in terms of the dimensions. The third conclusion is that the CO model is much more detailed than ours, because it both defines sub-structures for each of the elements of our knowledge space, and it explicitly representing what the relationships are between topics, whereas our model simply represents implicitly the fact that they exist.

## **9. Evaluation of the Model**

As well as comparing our model with these other two, it must also be evaluated against the aims set for it originally. One of these aims was to explore whether its key concepts, namely the three-dimensional structure and the application of general systems theory, could be extended to

the whole of informatics from their original application within SE. To claim complete success here one would have to map the whole of CC2001 into this model, which would be a huge task. However, the set of examples considered above is intended to be sufficiently comprehensive and realistic to give a reasonable basis for claiming that this model is at least as successful as the CO and GP ones, and hence that these principles do extend to the whole field of informatics.

An important aspect related to this aim is whether a different number of dimensions would produce a better model. The work on developing it originally in the context of SE certainly concluded that all three dimensions were needed, and so a smaller number would not be adequate: but that leaves open the question as to whether adding extra dimensions would be useful. In particular, the author in (Cowling, 1998) proposed a dimension for theory versus practice, and this was adopted in CC2005 as the other dimension for the two-dimensional space used to illustrate the relationships between the different disciplines within informatics (the first of these two dimensions being the one that has been used in this model for products). This therefore raises the question as to whether such a dimension is needed in this model, too.

In the present form of the model purely theoretical topics, such as ones in discrete mathematics, are not really accommodated, since it is only the applications of them that are located in the various dimensions (mainly the products one). On the other hand, while a dimension with theory and its applications as its end points is useful for describing a complete curriculum, at the level of an individual topic there are no meaningful intermediate points. Rather, any topic is concerned either with some part of a theory itself, or with some application of a theory, but can not lie between these. Hence, in this model a dimension for theory versus practice could only classify topics as one or the other, but not describe any other useful aspects.

Thus, to represent the theory that is applied within informatics the model would need to have some separate structure added to represent the relevant subsets of a curriculum model for mathematics, and possibly also for other branches of theory, such as psychology or economics. How such subsets might need to be structured is beyond the scope of this paper, although an obvious point is that general systems theory is unlikely to play much part in this. What is more significant for this model is how such a separate structure might be attached to it, to represent the relationships between the theory and the topics where it is applied.

Indeed, this issue of representing the relationships between the topics, as well as just the topics themselves, was the other main focus of this model. Thus, the aim was that a curriculum designer, in trying to combine elements from different disciplines within informatics, could use the model to analyse whether a proposed set of elements did actually form a coherent whole. The model has largely achieved this aim, since the structure of its knowledge space means that topics that are adjacent to each other, in any of the dimensions, will be more strongly related than ones that are further away. Hence, to be coherent a curriculum should correspond to a subset of this knowledge space that has no significant holes in it, in the sense of not omitting any topics that are positioned in any of the dimensions between others that are included.

In achieving this aim, though, the model has been limited by the key feature that it has inherited from the hierarchies used in the CC2001 models, namely that the relationships between topics are represented implicitly by the way in which the structure groups them together. Consequently, while this model represents which topics are related, it does not specify what the relationships are, unlike the explicit representations used in the CO model. Hence, while the model can be used to analyse how coherent a proposed set of topics is, it provides little support for making decisions about which topics on the edge of this set might justify being included or excluded.

## 10. Summary and Conclusions

In summary, the curriculum for the whole field of informatics can be modelled as a knowledge space with three dimensions, for products, processes and people, each structured as a multi-level inclusive hierarchy. The layers of these hierarchies reflect the application of general systems theory, and this theory also identifies the set of aspects that is important in every layer: purpose, structuring paradigm, and the mechanisms used to store and format information, to process information and to communicate information both internally and externally.

A goal of this model was to represent implicitly where relationships exist between topics, and this has been achieved, since the structures of the dimensions define “cells” in which topics are located, and provide an informal notion of the distance between cells, such that the closer cells are together, the stronger will be the relationships between the topics within them. Hence, the model can be used as a basis for determining how coherent a set of topics located in this space would be as a curriculum for a degree programme within informatics.

The model could, though, be developed further, particularly in three ways. Firstly, it would benefit from having some structure to represent the domain of theoretical concepts from outside informatics (eg from mathematics), and the relationships between these concepts and the topics within informatics that apply them. Secondly, it would benefit from having some explicit representation of what the relationships are between particular topics (as in the Computing Ontology model), as well as the implicit representation of which topics are related. Thirdly, it would benefit from having some representation of the topics and relationships within each cell of the knowledge space, to allow a finer grained description of the material than at present.

Thus, while this model has achieved its aims, there is scope for developing it further, but such developments must be left as future work.

## 11. References

ACM/IEEE, The Joint Task Force for Computing Curricula 2005, (2005), Computing Curricula 2005: The Overview Report, ACM/IEEE. Also at <[http://www.acm.org/education/curric\\_vols/CC2005-March06Final.pdf](http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf)>.

ACM/IEEE, The Joint Task Force on Computing Curricula, (2004a), Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, ACM/IEEE. Also at <<http://sites.computer.org/ccse/SE2004Volume.pdf>>.

ACM/IEEE, The Joint Task Force on Computing Curricula, (2004b), Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering, ACM/IEEE. Also at <<http://www.acm.org/education/CE-Final%20Report.pdf>>.

ACM/IEEE, The Joint Task Force on Computing Curricula, (2001), Computing Curricula 2001 Computer Science, ACM/IEEE. Also at <[http://acm.org/education/curric\\_vols/cc2001.pdf](http://acm.org/education/curric_vols/cc2001.pdf)>.

ACM SIGITE, Curriculum Committee, (2005), Computing Curricula: Information Technology Volume (Draft dated October 20, 2005), ACM. Also at <[http://www.acm.org/education/curric\\_vols/IT\\_October\\_2005.pdf](http://www.acm.org/education/curric_vols/IT_October_2005.pdf)>.

K. Beck with C. Andres, (2005), Extreme Programming Explained: Embrace Change (2nd edition), Addison-Wesley, Boston, Mass.

Bourque, P. & Dupuis R. (eds), (2004), SWEBOK: Guide to the Software Engineering Body of Knowledge, IEEE Computer Society, 2004. Also at <<http://www.swebok.org/index.html>>.

Cassel, L. N. (2006), Understanding the Entirety of Modern Informatics, In Programme for Informatics Education Europe, Montpellier, The Higher Education Academy, Information and Computer Sciences. At <[http://www.ics.heacademy.ac.uk/education\\_europe/keynotes/Professor Lillian \(Boots\) Cassel.doc](http://www.ics.heacademy.ac.uk/education_europe/keynotes/Professor Lillian (Boots) Cassel.doc)>.

Cassel, L. N., Davies, G., LeBlanc, R., McGettrick, A. & Topi, H., (2006), The Ontology Project (Powerpoint Presentation), In Programme for Informatics Education Europe, Montpellier, The Higher Education Academy, Information and Computer Sciences. At <[http://www.ics.heacademy.ac.uk/education\\_europe/keynotes/Boots.ppt](http://www.ics.heacademy.ac.uk/education_europe/keynotes/Boots.ppt)>.

Cassel, L. N., Davies, G., Finley, G. T., Hacquebard, A., LeBlanc, R., Mann, S., McGettrick, A., Riedesel, C., Sloan, R. H. & Varol, Y. L., (2005), A Synthesis of Computing Concepts, In Proceedings of ITiCSE 2005, Lisboa, Portugal, ACM, pp 162 – 172.

Checkland, P., (1999), *Soft Systems Methodology: a 30-year retrospective; and, Systems Thinking, Systems Practice*, Wiley.

Cowling, A. J., (2006a), A Systems Model for the Field of Informatics (Extended Abstract), In Programme for Informatics Education Europe, Montpellier, The Higher Education Academy, Information and Computer Sciences. At <[http://www.ics.heacademy.ac.uk/education\\_europe/Session\\_3/Anthhony\\_Cowling.pdf](http://www.ics.heacademy.ac.uk/education_europe/Session_3/Anthhony_Cowling.pdf)>.

Cowling, A. J., (2006b), A Systems Model for the Field of Informatics (Powerpoint Presentation), In Programme for Informatics Education Europe, Montpellier, The Higher Education Academy, Information and Computer Sciences. At <[http://www.ics.heacademy.ac.uk/education\\_europe/Montpellier\\_presentations/Session%204/Cowling\\_A.pps](http://www.ics.heacademy.ac.uk/education_europe/Montpellier_presentations/Session%204/Cowling_A.pps)>.

Cowling, A. J., (2005), The Role of Modelling in the Software Engineering Curriculum, *Journal of Systems and Software* 75 (1), pp 41 - 53.

Cowling, A. J., (2001), Teaching Data Structures and Algorithms in a Software Engineering Degree: Some Experience with Java, In Proceedings of the 14th Conference on Software Engineering Education & Training, Charlotte, USA, IEEE Computer Society Press, pp 247 – 257.

Cowling, A. J., (1998), A Multi-Dimensional Model of the SE Curriculum, In Proceedings of the 11th Conference on Software Engineering Education & Training, Atlanta, USA, pp 44 – 55.

Cowling, A. J., (1994), A Framework for Developing the SE Curriculum, In Proceedings of the International Workshop on SE Education, Sorrento, Italy, pp 111 – 118. Also at <<http://www.dcs.shef.ac.uk/~ajc/seteach/isee94.html>>.

Denning, P. J., (2003), Great Principles of Computing, *Communications of the ACM*, 46(11), pp 15 – 20.

Gorgone, J. T., Davis, G. B., Valacich, J. S., Topi, H., Feinstein, D. L. & Longenecker, Jr., H. E., (2002), IS 2002: Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, ACM/AIS/AITP. Also at <<http://www.acm.org/education/is2002.pdf>>.

Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W. & Paulk, M., (1997), Software Quality and the Capability Maturity Model, *Communications of the ACM*, 40(6), pp 30 – 40.

International Standards Organisation, (1991), *Information Technology – Software Product Evaluation: Quality Characteristics and Guide Lines for their Use*, ISO/IEC IS 9126, Geneva, Switzerland.

Kruchten, P., (2000), *The Rational Unified Process, An Introduction* (2nd edition), Addison-Wesley Longman.

Meyer, B., (1988), *Object-Oriented Software Construction*, Prentice Hall.

Osterweil, L., (1987), Software Processes Are Software Too, In Proceedings of the 9th International Conference on Software Engineering, IEEE Computer Society, Monterey, California, pp 2 – 13.