Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2001 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2001

Categorizing Philosophically-based Research into Software Development, Evolution, and Use

P. Wernick
University of Hertfordshire

D. Shearer
University of Hertfordshire

M. Loomes
University of Hertfordshire

CATEGORISING PHILOSOPHICALLY-BASED RESEARCH INTO SOFTWARE DEVELOPMENT, EVOLUTION, AND USE

P. Wernick
Department of Computer Science
University of Hertfordshire
p.d.wernick@herts.ac.uk

D. W. Shearer
Department of Computer Science
University of Hertfordshire
d.w.shearer@herts.ac.uk

M. J. Loomes
Department of Computer Science
University of Hertfordshire
m.j.loomes@herts.ac.uk

Abstract

This paper suggests a taxonomy of research aimed at applying ideas and concepts shared with or derived from philosophy to the field of information systems. It first describes categorisations of the development, use and evolution of information systems, and of the possible areas in which philosophy may be applied to this discipline. These are then combined into a single two-dimensional taxonomy. Papers from AMCIS 2000 are used as data for an initial exercise intended to refine the taxonomy, and areas of research not represented at AMCIS 2000 are identified. Possible future research directions suggested by an examination of the taxonomy are also described.

Keywords: Software, information systems, MIS, philosophy, taxonomy

Rationale for this Work

Much of the published work applying ideas and concepts shared with or derived from philosophy to the development and use of software has sought to start either with a single philosophical principle, or with the work of one or more philosophers. It has then applied this to a particular software process- or product-related problem. We note that current research of this type appears to be fragmented, using work of different philosophers in different areas. The selection of the philosophical basis seems often to be pragmatic, based on a surface similarity between the software issue and the problem addressed by the philosopher in question. For instance, the use of Popper's work forms the basis of much testing-related theorising, and Toulmin's ideas concerning classification have been adopted by the object-oriented community. Since different viewpoints taken from different philosophical schools are applied to different software-related problems, this body of research does not cohere into a single unified vision which can be applied in all software-related areas. Our aim in this paper is to start from the 'other end' of the question, by asking how, and in what specific areas, philosophy can help to improve that discipline concerned with the development and use of software-based systems. To do this, we have attempted to categorise the processes, activities and artefacts involved in software development, evolution and use, and the ways in which philosophy might assist in understanding and/or advancing this work.

In the course of describing the wide range of application of work based on the simulation of software processes, Kellner *et al.* (1999) have devised a framework into which research or practice in this area can be fitted. This framework is based on two axes: the *timescale* examined by the simulation, and the *reason* for performing the simulation. Kellner *et al.*'s intention was to support a description of work in progress in the field, and to provide advice to new entrants on simulation approaches which have been fruitful in specific areas defined by the taxonomy. In this paper, we describe the results of an initial exercise aimed at developing an equivalent taxonomy covering the application of philosophy to the development, evolution and use of software-based systems. We intend our work to be a first step towards developing a useful taxonomy of the ways in which philosophically-based thinking and practice can assist in software development, evolution and use. We have employed this embryonic taxonomy to characterise the papers presented in the Philosophical Foundations of Information Systems mini-track of AMCIS 2000 (Chung, 2000) and examples of our own previous work in this field. We have also used the taxonomy as a tool to consider how philosophy might be applied in this area.

We acknowledge that the taxonomy presented here is at an early stage of development, and expect that it will need to be extended in the light of future discussions and experience of its application. However, we hope that it may prove capable of development to the point at which it provides an agreed basis for describing the problems which the discipline seeks to address. In this way, it may help to define and widen the commonly-agreed core of belief, on the basis of which research and its application may be able to proceed more quickly as Kuhn (1970) claims to be the case for scientific disciplines.

Bounding the Discipline

As a prerequisite to setting out those problems in a discipline with which philosophically-based research can assist, it is necessary to define that discipline, its content and its boundaries. However, attempts to define any discipline are problematical, and are inevitably subject to change as it advances. We have attempted to bound the area covered by our taxonomy in the light of these considerations.

We started by taking definitions of the term 'software engineering' from undergraduate textbooks. Whilst this is not a perfect approach, it provides at least one vision of a 'generally accepted' definition of the discipline and its scope. We have deliberately ignored the semantic issues arising from the use of the term 'engineering' in this context (for a critique of this usage, see Wernick (1996, p.31 *et seq.*)). As examples of these general definitions, Sommerville states (2001, p.v) that "Software engineering is an engineering discipline which is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use." Pressman refers to "a process, a set of methods, and an array of tools that we call *software engineering*." (2000, p.5) Other authors writing books on more specific techniques or approaches define 'software engineering' in such a way as to promote their views of what it should be. For example, Fenton and Pfleeger, in a book on software metrics, define the activity of 'software engineering' as "… implementing … software in a controlled and scientific way." (1997, p.9)

Here, we take the definitions typified by Sommerville and Pressman as a starting point. Our definition therefore includes the philosophical underpinnings and implications of such necessary activities and approaches as model-building, the design and use of notational and enactable languages, and the philosophical implications of abstraction. Our scoping of the discipline also extends to areas outside the technical software production process, to include the necessary interactions of the processes and products of software development and evolution with their external environments. For example, we include in our areas of concern the use of software-based systems, and the effects that software-based systems have on their users' beliefs regarding the world. The latter include users' definitions of 'truth' and how to establish it in the context of using a software-based system and its outputs. We therefore define the discipline whose philosophically-related research we are considering as being:

> *the development, evolution and use of software-based systems, including the development of tools and processes employed in these activities.*

Categorising the Axes

Taking as the two axes of our analysis the discipline defined above and how philosophy might be applied to it, the next task is to subdivide each axis into areas, in a fashion which is both meaningful and appropriate to our taxonomising task. The lists presented below are the result of our initial analysis, modified by a practical exercise described in Section 4.1 below.

Our aim in defining these areas is not to segregate research or argument into a number of predefined categories with immutable, impermeable boundaries. We regard the definitions of the areas and the boundaries between them as being fluid, and we feel strongly that they should not be seen as a barrier to investigations into issues concerning more than one area. It is also inevitable that, in addition to specific research activities being deliberately performed in more that one area simultaneously, research focused on one area will achieve results or have implications which extend beyond the boundary of that area into others.

We believe that our categorisation may help researchers to identify connections between areas which any new research should take into account. It may additionally assist in the identification of branches of existing or proposed philosophical work, and of individual philosophers or schools, which would be useful in an area, by bounding that area. We thus see the taxonomy as a useful lens to focus thought, rather than as a rigid framework. We believe that its continued usefulness will require on-going re-evaluation of its dimensions and the definitions of their elements.

Categorising Areas of Software Development, Evolution and Use

We have categorised the target discipline, as defined above, into the areas set out below. This division has been derived from a consideration of the discipline seen as a hierarchy. This hierarchy extends from consideration of what software and the notations used to develop and evolve it actually *are*, through the development of one software product release and its use in the real world, to the long-term co-evolution of that product and its context over a number of releases from initial release to eventual retirement. We present examples of philosophically-related issues and aspects for each area.
The areas are:

- *the nature of software*; seeing the software artefact itself as an object of philosophical study. Here would be considered aspects such as the use of alternative logics, or of non-logically-based mechanisms, as a basis for defining or describing software system behaviour, and support for (and limits to) the validity of statements which can be made by software-based systems;
- *the discipline of software development, evolution and use*; looking at this discipline as a whole, considering those aspects which are common to all activities involved in or resulting from software development, evolution and use, and noting that the activity of using software systems may itself be a part of this discipline and that the users of these systems are more than just passive recipients of a system once it has been completed;
- *the individual software system (or type of system) in its external context, and the context in which systems are used*, including all aspects of relating a single system as installed in the real world to its environment, for example the implications for a complete worksystem incorporating a software-based system of setting the system/environment and computer system/human system boundaries in any specific place;
- *data and other information held within the system or modified by it*, including consideration not only of data held in the software-based system as representations of information about the world outside it, but also of the information stored within the system to support the software-based mechanisms themselves, and asking questions relating, for example, to the nature of data as a body of 'fact', and to epistemological mechanisms employed by different participants in the processes of systems development, evolution and use;
- *the development and evolution of individual software systems*, relating to all aspects of the system's life from the recognition that there is an issue or problem which could profitably be addressed by the use of some 'system', whether or not it incorporates a software-based entity or element, to the final decommissioning of the last example of the system still in use, and also incorporating the relationship between the system and the socio-technical processes which develop and evolve it;
- *the single development and implementation cycle*, in which a single release of a software product is produced and put to some real-world use, and in which software processes may be seen as rationalistically- or relativistically-based processes or as combinations of these;
- *the languages, tools, notations and process models used in software development and evolution* , including the design and use of graphical notations and computer programming languages, the development of models or representations in these languages, and the processes of abstraction and formalisation by which complex problems or data are simplified and re-presented in forms suitable for programmers or computers to use as inputs. Investigations in this area should also consider assumptions made in the design of languages, tools, notations and process models about the nature of the world within which the system is to be operated and the elements of which that world is comprised; and
- *the notational element*, considering, for example, how the individual elements comprising the languages used in software design and implementation can be related to aspects of real or possible worlds, including the meaning of individual notational elements, implications of the design of individual graphical elements, and mechanisms by which these elements become connected to specific elements of the computational models.

Modifying An Existing Categorisation for Applying Philosophy

The second axis of our taxonomy describes the ways in which philosophy can be used to support research and practice in software development. We start with an existing list of such possible uses. Courtney and Porra (2000), summarising Hodges (1995), describe five ways in which philosophy can inform information systems research. These are:

- as a reference discipline *per se* in the sense that it would contribute a subject and a methodology;
- by helping to lay conceptual foundations on which to build management information systems (MIS) theory;
- epistemologically informed studies [of issues in MIS research and practice];
- [informing MIS research and practice by the application of] hermeneutics and phenomenology; and
- [research] based on existentialism.

We suggest that the two latter ways are better categorised as examples of the application of specific philosophical directions to software-related issues. They therefore differ from the first three, which map out areas of information systems research and practice which can be supported by reference to philosophy. The third, the conduct of epistemologically informed studies, also

demands the use of only one area of philosophical study. We have therefore only included unchanged in our categorisation the first two of Hodges' categories. We have added to these Hodges' third category which we broaden to encompass the application of *any* relevant philosophically-based input to the design and conduct of empirical studies into software, its development, evolution and use, and to the analysis of the results of such studies.

Our taxonomy for the ways in which philosophy can be used to support information systems research and practice can now be summarised as:

- providing a *reference discipline*, employing lessons and work directly from philosophy by, for example, identifying alternatives to the scientific method such as the Hegelian dialectic to make explicit assumptions implicit in mechanisms and situations, providing mechanisms for and critiques of ethical and systems theory-based aspects, and considering different epistemologies underlying expert systems and artificial intelligence;
- helping to lay *conceptual foundations* on which to build information systems theory by combining philosophy and other disciplines. Hodges suggests the use of scientifically-based mechanisms to design investigations intended to examine underlying conceptual problems of the discipline; an instance of this is provided by the theory presented in Wernick (1996); and
- supporting the design and conduct of *empirical studies* in IS research and the analysis of the results of these studies, using both epistemological understanding, as suggested by Hodges, but also any other relevant theories or concepts from philosophy.

Applying the Taxonomy

We are now in a position to combine the two categorisations into a single, two-dimensional taxonomy, and to apply it to sample data. To determine its usefulness, we also need to identify and consider any insights gained from this application.

Exercising the Taxonomy

In an exercise intended to test the concept of a taxonomy such as that presented here, and to examine the ability of our initial taxonomy to categorise a heterogeneous collection of relevant data, we assigned a body of existing research to our two-dimensional taxonomy. As data for this exercise, we employed the 20 papers published in the Philosophical Foundations mini-track of AMCIS 2000 (Chung, 2000). The results of this analysis are summarised in Table 1. Since space does not allow us to present our analysis completely, we observe here that we have, for example, placed

- a paper describing the application of a generalised mechanism derived from Hegelian dialectic to software development (Wernick *et al.*, 2000) on the software axis under 'single development and implementation cycle' and 'reference discipline' on the philosophical axis, and
- a paper seeking to examine the meaning of 'knowledge' systems in a social context (Butler, 2000) under 'data and information held within the system' using philosophy to examine 'conceptual foundations'.

We have also considered the taxonomy in the context of some of our own previous and current work in the field. We consider the work presented in this paper to lie within the taxonomy by comprising a step in *laying conceptual foundations* for *the discipline of software development, evolution and use*.

That we have been able to complete this task suggests that the basis of our approach, and the taxonomy itself, are good starting points for this work. Our analysis did however result in the identification of a number of changes to our initial categorisation of the software axis, which have been reflected in the categories described above.

Considering the Results

Having applied the taxonomy to an existing body of literature, we are can now examine the potential usefulness of the taxonomy as a lens for considering research in this area. We have done this by drawing some initial, tentative conclusions based on that previous application.

We first observe an absence at AMCIS 2000 of reports of work into aspects of the use of notational elements. These elements are of importance to the development of information systems since they form the building blocks of the languages used to describe analysis results, designs and implementations. Work on laying the conceptual foundations of the discipline by reference to philosophy was also concentrated in the higher levels of our taxonomy. No papers were presented on the foundations of lower-

level aspects such as single development cycles, and the design and use of languages and notations. We also note the small number of papers using philosophical ideas as the basis for designing or conducting empirical studies.

We conclude that it is possible to use the taxonomy presented here to consider a body of research, and to identify areas which have not yet been exploited.

Extending and Employing this Work

We have extended our analysis of the data provided by the AMCIS 2000 proceedings to identify a number of areas in which the work presented here could be developed and expanded. These include:

- having others repeat our initial exercise, to improve the definitions in the taxonomy's dimensions by considering any differences between our results and theirs;
- identifying and considering interactions between the areas defined above within and across the two categorisations, as made explicit in the intersections of the cells forming Table 2;
- identifying or devising a suitable taxonomy of the discipline of philosophy which could be used as an additional dimension in further analysis;
- re-interpreting existing research in the field in the context of our existing analysis, with the aim of identifying new inter-relationships between what are currently apparently unconnected or loosely-connected strands of existing research;
- considering the taxonomy itself and its constituent axes and categories from a philosophical standpoint; and
- identifying new discipline problems or clarifying the terms of existing discipline problems, with the intention of formalising these into a set of agreed problems which could be attacked by researchers in the field in a concerted fashion. This might form the precursor of a disciplinary matrix or paradigm for the application of philosophy to software and its development, evolution and use. As Kuhn (1970) notes, the emergence of such an agreed belief system allows a discipline to progress more rapidly than is the case in its absence, and its identification is therefore to be seen as a worthwhile goal of discipline research.

In addition to facilitating the analysis of existing work, we hope that the taxonomy we have described may allow or encourage interesting discipline questions to be posed. We set out in Table 2 below some (speculative) examples of possible philosophy-fuelled investigations into each of the intersections defined by our software-related areas and categories of philosophical application. Again, we do not see these examples as a complete manifesto for the application of philosophy to software, but hope that they will assist in understanding the areas and how they inter-relate, and as possible spurs to research into these aspects.

Acknowledgements

We are grateful to colleagues in the Department of Computer Science, University of Hertfordshire, particularly Bruce Christianson and Vito Veneziano, for discussions during the work presented here, and to the anonymous referees of AMCIS 2001 for their valuable comments.

References

Butler T. *Making Sense of Knowledge: A Constructivist Viewpoint*, *in* Chung (2000), 2000, pp.1462–1466.

Chung, H.M. *Proc AMCIS 2000*, Long Beach, California, AIS, 10–13 August 2000.

Checkland, P. *Systems Thinking, Systems Practice*, revised edition, Chichester: Wiley, 1984.

Courtney, J.F., and Porra, J. "Mini-track on the Philosophical Foundations of Information Systems", *in* Chung (2000), 2000, pp.1458–1461.Fenton, N.E., and Pfleeger, S.L. *Software Metrics*, London: International Thomson Computer Press, 1997.

Hodges, W.S. "Five Roles that Philosophy Can Play in MIS Research", *Proceedings of the 1995 Meeting of the Americas Conference on Information Systems,* Pittsburgh, PA., 1995; *cited by* Courtney and Porra (2000).

Kellner, M.I., Madachy, R.J. and Raffo, D.M. "Software process simulation modeling", *J. Systems and Software*, 46, 1999, pp.91–105.

Kuhn, T.S. *The Structure of Scientific Revolutions*, second edition, Chicago: University of Chicago Press, 1970.

Pressman, R.S. *Software Engineering*, 5th edition, European adaptation; London: McGraw Hill, 2000.

Sommerville, I. *Software Engineering*; 6th Edition, Harlow: Addison-Wesley, 2001.

Wernick, P. *A Belief System Model for Software Development: A Framework by Analogy*, PhD thesis, Department of Computer Science, University College London, 1996.

Wernick P., Christianson B., Loomes M. J. and Shearer D.W. "A Dialectical Basis for Software Development Tool Building"; in Chung (2000), 2000, pp.1548–1553.

Table 1. Numbers of AMCIS 2000 Papers Relating to Each Area of Software Engineering

	Reference discipline	*Laying conceptual foundations*	*Philosophically-informed empirical studies*
nature of software	1	1	
discipline of software development, evolution and use		2	2
individual software system (or type of system) in its external context	3	5	
data and information held within system		1	
development and evolution of a single software system	1		1
single development/implementation cycle	1		
languages, tools, notations and process models	2		
notational elements used in modelling/implementation languages			

Table 2. Examples of Philosophical Issues Raised in Each Area of Software Engineering

	Reference discipline	*Laying conceptual foundations*	*Philosophically-informed empirical studies*
nature of software	• logic as the basis of software and its behaviour	• consideration of the status of statements which can be made by software-based systems	• ethics as the basis for studies into the inter-relationship between software and society
discipline of software development, evolution and use	• the progress of philosophy as the basis for the progress of the discipline of software development • discipline-wide ethical issues	• philosophically-based theories of the discipline of software development (e.g. Wernick, 1996) • meaning and mechanisms of 'process improvement' for software development and evolution • alternatives to the life-cycle software development model • setting the scope of 'software engineering' • assisting in identifying other relevant disciplines and their interactions	• empirical examinations into underlying discipline belief systems (e.g. Wernick, 1996)
individual software system (or type of system) in its external context, and context with which systems are used	• research underlying the Soft Systems Methodology (Checkland, 1984)	• implications for users of the system of setting the boundary of a software system and its surrounding worksystem • interactions of the boundaries set for a software system and for its surrounding worksystem • ascribing meaning in specific contexts of use to outputs of software-based systems • aesthetic considerations in developing software-based systems suitable for real-world use • changes in 'truth' in an organisation when a software-based system is installed	• examination of effects of placing software systems in social contexts
data and information held within system	• statements concerning status of data with respect to 'truth' and validity • ascribing meaning to data • possibility of interpreting data outside a given structure (specifically that of the system)	• does data held within a computer-based system differ in any fundamental way from data held in any other form? • epistemological statements concerning status of data as interpreted by different users • relationship between structure of data as enforced by the system, and interpretation of (and ascribing meaning to) that data	• inquiring systems used as basis of experiments into beliefs concerning data and how to interpret it

	Reference discipline	***Laying conceptual foundations***	***Philosophically-informed empirical studies***
development and evolution of a single software system	• inquiring systems used directly as basis of techniques used to develop internal mechanisms of software systems • aesthetic considerations in devising process models, and in designing models during the development process, such as the prevalent values of parsimony	• process programming • constructivist and objectivist epistemologies	• inquiring systems used as basis of experiments to compare techniques used to develop internal mechanisms of software systems
single development/ implementation cycle (including both process- and product-related aspects)	• falsification as the logical basis of software test planning • software processes, tools or techniques derived directly from philosophical mechanisms, such as the application of specific inquiring systems • aesthetic principles in designing models	• software processes as 'truth'-seeking mechanisms	• falsification-based studies into the usefulness (however defined) of software processes at the cycle level
languages, tools, notations and process models used in software development and evolution	• formal methods derived from logical principles • aesthetic considerations in devising modelling languages and notations • philosophy of language in combining elements of language • effect of notations used in descriptions of information systems on perceptions of information system by technical and non-technical people, and its implementation.	• theoretical examinations of implications of abstraction, and of model-making as a concept • ascribing meaning to models • models as theories, or theories necessarily underlying models; • models as speech acts, and types as behaviours, logically defined	• empirical examinations of implications of abstraction, and of model-making as a concept
notational elements used in modelling for software development and in implementation languages	• analogies between how software developers and philosophers define terms and use symbols: semantic levels of semiotics, Peirceian views on the philosophy of language in selecting the elements of language	• ascribing meaning to software development discipline elements of notations	• ascribing meaning to user domain symbols used in models