Software Evolutionary Dynamics Modelled as the Activity of an Actor-Network

P Wernick, T Hall and C L Nehaniv
School of Computer Science
University of Hertfordshire
College Lane, Hatfield, Hertfordshire AL10 9AB, England
tel. +44 1707 286323; fax +44 1707 284303
{p.d.wernick, t.hall, c.l.nehaniv}@herts.ac.uk

Abstract
The pressures which act on a software system over its life from inception to retirement are many and varied. It is an important goal in considering software evolvability to understand, and if possible to manage these influences. Our previous simulations of software evolution processes have concentrated on capturing the human-related aspects of software evolution, whilst effectively treating technical entities as objects which are acted on by humans and their organisations. Latour's actor-network theory (ANT) suggests that the non-human entities – development tools, document, the system itself – are potentially active participants in their own evolution. We describe Latour's theory, and present a model of a software evolution process in the form of a diagram which places technical and human aspects in juxtaposition closer to that which ANT would suggest than previous models. We believe that this approach will result in a more accurate representation of the process, and thus be a step towards dynamic simulation models whose predictive power will help us to better understand and manage software evolution and evolvability.

1. The problem – and a possible solution

For some time, researchers and practitioners have been attempting to understand the processes by which software systems are changed over time, usually with the intention of finding ways to manage and control them. At present there is no way of predicting when the rate of *software evolution*[1] will speed up, slow down or even stop completely, and no theory with predictive power of *why* any particular system might do this.
Lehman has described the systems of people, artefacts and events which control the evolution of industrial software-based systems (including operating systems and limitary systems) as the 'global software process' (see, for example, [5]). They note that this process includes "… the activities of all involved, for example, developers, managers, marketeers, support personnel and users".
Lehman's description of the global software process, supported by our earlier system dynamics simulation models, sees this process as being primarily driven by feedback. This is made explicit in the VIIIth Law of Software Evolution ("feedback system") which states: "E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must, in general, be treated as such to achieve significant process improvement for other than the most primitive processes." [8:125].
In our previous simulation models intended to represent the highest-level causal mechanisms producing observed patterns of commercial software evolution [3, 17, 19], these mechanisms have been highly abstracted, particularly with respect to the actions of individual people.
Previous descriptions of the global software process have also tended to be based on an explicit or implicit division of these agents into 'active' people and 'passive' technical elements. The influence of this mind-set can be seen, for example, in [5], in which their Figure 1 shows a program surrounded by people, who are interacting with it but which are very obviously different sorts of things from the program itself. This mode of thinking is also implicit in our simulation models, in which we have sought to describe this process [3, 16, 17, 19]. By way of contrast, Latour's actor-network theory (ANT) [7], an approach based on a sociological view of technological change, provides us with a viewpoint from which the effects of both human and technical participants on process characteristics and behaviour can be considered on a more equal footing. We believe that looking at the problem for a point of view which takes into greater account the effect of the technological participants will result in a more accurate representation of real-world global software processes. Latour's theories are controversial in the world of

[1] We define software evolution here as the making of changes to a software-based system to support its continued useful employment.

sociology and elsewhere, at least in part because they seem to give to non-human elements of a system some characteristics normally ascribed only to humans.[2] It is, however for this very reason that we believe that by applying them as given they form the basis for an analysis of global software processes which can provide useful insights, as we show in this paper.

Due to their number and complexity (which we demonstrate below), the human/social influences in the global software process may outweigh technical issues in determining the behaviour of the process. In addition, newly-proposed technical solutions need to be assessed on the basis of their effect on the combined human-technical process. It is therefore important to understand the social networks within which the technical work is undertaken, such as the social embedding of the technical process and promotion of interaction between user and developer actors explicit in eXtreme Programming [1].

Our objectives in applying ANT to the global software process are both to understand better the situation in which software evolves, and to explain the behaviour observed as a system evolves, such as the 'regeneration points' in the evolution of software systems when old systems show an increase in the rate of growth in size after a period of progressive decline in that rate [2]. In the final analysis, we are interested in finding out what determines the 'health' of a software-based system at any point, and whether this can be determined by examining the current state of the process evolving that system. We believe that an important step towards achieving this goal is the identification of the elements making up the process, and an understanding of how these elements interconnect and interact to produce the behaviours we observe. Achieving this understanding is a necessary (but not sufficient) condition for controlling the process of software evolution.

2. The actor-network theory viewpoint

The following description of ANT is based on two works by Bruno Latour: a formal description of ANT [7], and a more playful but nevertheless highly illuminating account of the failure of a technological project [6].

Latour claims that his view of how to describe and understand a social situation differs from that of 'traditional' sociologists. He suggests that the latter feel they need to add some intangible thing, the 'social' dimension, to what can be actually seen in a situation, in order to explain it.[3] Latour himself sees no separate 'social' medium in which the people and technologies involved float; instead, a situation comprises human and non-human entities and interconnections between them. He further suggests that social situations are different from other, non-social situations due to the complex interactions between the entities which make them up rather than the presence of some 'social' substance.

In ANT, Latour considers three types of entities, *viz.* actors, mediators and intermediaries. 'Actors' are active in the situation, and are most often people but can also be technological elements or anything else involved in the situation. Actors are called this because not only do they act but also because they are constrained by their situations in the choices they make as an actor on a stage that is constrained by the lines given by the playwright [7:46] and can only be creative within that limited scope. They are likely to combine into identifiable groups, which coalesce or break up according to the pressures on them.

'Mediators' receive and transmit messages like intermediaries, but change the messages they receive in often-unexpected ways before passing them on. Latour gives some examples of mediators, including law, science, religion and economies [7:240].

Finally, 'intermediaries' receive messages from an actor and translate them into a form which can be understood by another actor without changing the content of the messages. An intermediary "… transports meaning or force without translation: defining its inputs is enough to define its outputs. For all practical purposes, an intermediary can be taken … as a black box …" [7:39]. This can be contrasted with the effect of mediators, which "…. transform, translate, distort, and modify the meaning or the elements they are supposed to carry. … [t]heir input is never a good predictor of their output; their specificity has to be taken into account every time." (*ibid.*)

A vital concept in ANT is the 'network', a grouping of actors, intermediaries and mediators linked together by communications channels. Actors may join or leave networks over time, creating a dynamic, ever-changing web of relationships; an actor joining a network may bring their existing network(s) with them.

An actor's level of commitment to the goals of the network may change over time under pressures from the actor's own circumstances, the state of the system and its relationship to the actor, and influences from other elements in the network.

[2] See, for example, the comments of Williams-Jones and Graham [20] when they apply ANT in a different area.

[3] Latour seems to see the 'social' as described by other sociologists as some variety of sociological phlogiston, and to regard this substance as about as useful in promoting understanding as the original.

3. The global software process and actor-network theory: correspondences

3.1. General points

One common feature of our previous accounts of the global software process and ANT as a means of examining social situations is immediately apparent. This is the underlying assumption in ANT of some measure of causality underlying the behaviour of social processes, which sits well with the same assumption made explicit in the use of causal links to develop system dynamics simulation models, including those which we have designed.

Another important observation is that a software system does not of itself 'evolve'; it *is evolved* by the actions of people and other actors on it. At the same time, the system cannot necessarily be seen as a passive recipient of evolutionary actions and pressures; it also causes evolutionary pressures as part of the feedback system within which it is used and evolved. Adopting an ANT-based viewpoint enables us to consider the system as an actor in its own right, participating actively in the processes which lead to its own evolution. ANT provides us with a justification for considering the effect of a system on the world as that of an actor causing as well as undergoing change, despite its not being human.

3.2. Applying ANT: initial considerations

The first task in building an ANT model of the global software process is to identify the actors, mediators and intermediaries which enact the process, and the connections between them. Examples of these entities for commercial software development and evolution, derived from general knowledge of software development and use, include the champions of systems in developer and customer organisations, customers, salespeople, the developers of a system and its users, technologies such as Integrated Development Environments (IDEs), analysis, design and programming languages, hardware, and the owners of both developer and user organisations.

In addition it is necessary to consider entities which do not form part of the developer/user complex but which are capable of affecting the behaviour of the software evolution process, such as the general public (included below as 'wider society') as represented by the media and government and its agencies such as tax collectors, police and the security agencies. The effects of major corporations which can by their size or position either directly or indirectly influence the evolution of a software product (for example, Microsoft's influence on the web browser market) will also need to be considered. Overall, it will be necessary to identify and capture the identity and influences of all significant stakeholders, whether direct or indirect.

A recent example of the effect of the last on a software system is the demand for information on internet traffic [12], resulting in a need on the part of internet service provides (ISPs) to retain additional data. This in turn requires actions within the evolution processes by the suppliers of network monitoring and control equipment and software, and potentially the addition to the existing actor-network of another *actor-network* of suppliers of such equipment/software beyond those previously involved in evolving the system.

Finally, the influences of competing developer and user organisations and the products they produce and use must be taken into account.

We will need to determine whether each of these entities is actor, mediator or intermediary in the process. It will also be necessary to determine which groups of actors we can treat as actor-networks, i.e. individual complex entities, and which must be broken down into individual actors.

Some potential influences on individual actors can be identified. For example, what if development staff leave, or if it becomes more difficult over time to hire people with the necessary expertise as languages and tools become obsolescent and thus less popular amongst developers – consider for example the effect of the Year 2000 problem on employability of Cobol programmers. What if users of a system become discontented for a reason unconnected with the system and this ill-feeling results in unhappiness with the system itself? An advantage of the ANT approach is that we can consider such non-technical aspects within the same theoretical framework.

4. Understanding the global software process as an actor-network

4.1. Modelling the network

The work presented here forms a first step towards the development of an executable simulation of the global software process based on ANT principles. For this model we have considered the commercial development of bespoke software; the model as described below would need minor changes to reflect the differences in process for a package software product, and a complete redesign for open-source software evolution processes.

We have identified actors, intermediaries and mediators, and the relationships between them. The human 'actors' which we have identified are roles taken by people or imposed by society, organisations, etc. Some actors themselves represent (and abstract) more complex actor-networks as single actors in the way that Latour allows; examples of these are the system's developers and users.

The elements of our current model have been identified on the basis of the writings of Lehman and his colleagues (e.g. [5]), our real-world experience as software developers, users of software, and in software systems user support, our previous research (e.g. [9, 15, 18], and our impression of general folk knowledge of how software development 'works'. We have been conservative in identifying links between elements to minimise the complexity of the model structure graph.

We have chosen to represent the system as fielded as an actor rather than as a mediator, to reflect the ability of the system to influence its users and its surroundings by way of feedback in the global process; this decision is discussed in more detail in Section 4.3 below.

As an initial approximation, we have identified *intermediaries* in the global software process as represented in our model with those elements of the process which are read and interpreted by machines, such as language compilers, operating systems and file systems. As required by Latour, once the inputs of such elements are described their outputs are deterministically derivable, even if those outputs go on to have differing effects on actors in the process. This may be contrasted with the mediators in the global process, whose outputs are not uniquely defined and which may differ from occasion to occasion. Mediators may therefore be identified with those elements of the process which are both written *and interpreted* by humans; this interpretation may produce understandings in, for example, the readers of a document which differ from those intended by the writers. Candidate mediators in the global software process, not all of which are represented in the current model, include requirements and design documents, the source code of the system (when it has to be read to be modified by actors called 'developers' or similar), and the system as fielded, whose actions and outputs are interpreted by its users in the context both of their models of what is happening inside it and of its interaction with the external world.

At the high level of abstraction we have adopted in building our model, we feel that intermediaries need not be modelled explicitly, since their only effect is to change one notation or the equivalent into another. They do not change the values they carry, but distribute information in the network. However, mediators, which may affect the values of their outputs, need to be identified to help understand their effect on the behaviour of the process.

4.2. The model described

Our current ANT model of the global process is shown in the form of a diagram in Figure 1. The diagram was drawn using the Vensim system dynamics modelling tool [14]. This is in part because its System Dynamics notation provides a simple toolset for drawing such directed graphs. In addition its analytical tools allow some conclusions to be drawn concerning the nature and complexity of the interactions between the actors (see Section 5 below).

In the diagram, an actor is represented (arbitrarily) by a hexagon and a mediator by a circle; in each case, the element's name is inside the box. Lines connecting the elements represent the existence of an information flow between the connected elements, with the direction of the flow shown by an arrowhead. Bidirectional flows are shown using a separate arrow in each direction between, for example, the System Development Owners and the System Salespeople.

One matter in the model may need explanation; this is the distinction made between *mutable* and *immutable* tools. The former are those elements of process, tools, and so on (so 'tools' in the widest sense for software engineering) which are at least to some extent under the control of the system's developers, and can therefore be modified if required for a specific development or evolution project. For example, a process whose content is under the control of the project manager may be modified if necessary to meet the needs of a particular project, or a software tool developed for this project by the system's own developers may be modified as required. Such tools can be distinguished from immutable tools, which cannot be modified by those involved in the project, for example an ISO 9000-certified process which senior management have decreed must be followed in this project whatever the consequences, or a bought-in closed IDE whose evolution cannot be influenced, or its toolset or process modified, by this system's developers, however much they need to change its behaviour or functionality.

Another issue which needs to be taken into account is the common situation in which different actors talk about the same project or product by the same name but are actually referring to a different thing [6], or the very process by which a name is given to a process or product [10]. Indeed, the ontological status of "the system" is perceived differently by different actors, on the basis of their own (differing) opinions concerning it and their degree of commitment to its realization and evolutionary change. Reification and/or realization of "the" system emerge in the activity of the actor-network. Does this make the system name itself a mediator within the process? It is only via activity of the actor-network that the system comes into being and is evolved over time. We believe that this viewpoint better reflects the global system process than traditional models of software development and change (cf. [6]).

Some actors or intermediaries which were included in earlier versions of our model have been removed as not being necessary or duplicating others, albeit at some loss of detail in the model. These eliminated elements include:

- the *system specification*; combined in the model into the change input queue mediator, since the specification should be a translation of this. This decision means that the model conflates the translation from real-world language to computing language and the translation from specification to fielded system in the technical software process. However, we currently feel that this loss is outweighed by the gain in simplification in the model, and
- the *system source code* as a separate intermediary from the system as fielded, since the latter is a direct, deterministic translation of the former. Here we lose detail in our model, in that restrictions are placed on the system design and its evolvability due to the programming language adopted, and the style of design and coding can significantly affect the evolvability of a software system. Again, as a simplification we have abstracted this aspect in the current model.

As an additional simplification, all of the technical roles relating to the development and evolution of the software – analysts, designers programmers, testers and so on – have been abstracted to a single actor-network called 'developers'.

4.3. The system: actor or mediator?

One question which has arisen for a number of the elements of our model is whether it should be an actor or a mediator. We consider as an example the case of the 'system as fielded', i.e. the system which is actually used by its users.

It might be argued that the users of the system actually see their translation of its implementers' view of the system. It could therefore be seen as having no life of its own, and that it might therefore be excluded entirely from the model, modelled as an intermediary, or seen as at most a mediator. However, a reading of Latour's *Aramis* gives a very different view of what a system can be; it takes on a life of its own, it participates in the life of the world, it lives, it dies [6:290]. In addition, a software system interacts with its stakeholders, and constrains and shapes their actions.

We have therefore concluded that the system can itself be an active participant in its own evolution. This is also in accordance with Lehman's VIIIth Law, and parallels thinking behind the system dynamics models which we have previously published. On this basis, we have represented the system as fielded as a fully-fledged actor in our model.

5. Some initial conclusions

A first sight of Figure 1 reveals the complexity of the web of social and technical interactions which controls the software evolution process. As Lehman's VIIIth Law suggests, the model comprises a feedback system of great complexity. This in itself may help explain why it is so difficult either to theorise about the nature of this process or to control and manage it in practice.

The impression of the complexity of the interactions between actors is both exemplified and reinforced by an analysis using the Vensim loop analysis tool, which counts and lists the participants in each complete loop in the directed graph. In this case, the tool showed that there is a total of 2534 feedback loops in the model. The longest loops in the model are 14 elements long.

An example of a loop in the model, in this case of 11 elements, is as follows:

System as fielded
    Users
    Sponsor/owners
    System salespeople
    System change input queue
    Developers
    Project manager
    System development owners
    Mutable tools
    System design/architecture
System as fielded

Such loops can be examined on the basis that each tells a 'story' of one influence on how the actor network operates to evolve the system. Successfully telling the story both provides insight as to how the process might operate, and gives some degree of comfort that the model is reasonable. In this case, the story might be as follows:

- the system as fielded affects its users, perhaps frustrating them or preventing them from doing their job, or alternatively suggesting an opportunity to improve the user process by a system enhancement. The users therefore raise these issues with the system's sponsor/owners.
- The sponsor/owners then demand changes in the system from the system salespeople, which results in new change demands being placed in the system change input queue.
- The developers examine these change requests, and as a result of their analysis they find that they need to ask the project manager to get the system development owners to authorise changes to the mutable tools.
- The modified tools are then used to modify the system design/architecture.

- Finally this architectural change is reflected in changes seen by the users in the system as fielded in its next release.

As suggested earlier, this example show the closeness of interactions between social and technical aspects, reflecting both the technical software process and the social interactions which surround and control it. Such examples, talked through with process experts, also enable us to check the reasonableness of both the model structure and the individual elements comprising it.

6. Next steps: from model to simulation

In this section we describe the steps necessary to turn our current qualitative, descriptive model into an executable simulation of the global software process.

6.1. Refining the model

Our first step in moving from the descriptive, qualitative model described here to an executable simulation must be to select the *focal actor* for our analysis. This will be the actor from whose viewpoint the creation and operation of the network is to be described [13:1663]. Given our reason for modelling the process, i.e. to understand how and why the system as fielded evolves as it does, the focal actor is most likely to be the system as fielded itself, since it is its evolution process that we are looking at. The most important variable in the simulation is therefore some measure of 'system health' as perceived (perhaps differently) by its stakeholders.

We will then need to continue our analysis and identify any additional actors, mediators and intermediaries, and determine the ways in which each affects others. Sources for this information will include reports of practice from both successful and failed projects in software development such as such as Yourdon's *Death March* [21], interviews with practitioners, experience reports from Software Engineering conferences and other literature (e.g. [4]), and other analyses of analogous processes outside software development, including, but not limited to, Latour's own work such as *Aramis* [6].

Another issue yet to be addressed in the model is the status of the data held in a software system. Questions to be considered include whether the data is an active participant in the evolution process as the software system and if so whether it is actor, mediator or intermediary, and whether data and software can be abstracted together into a single actor-network. We currently believe that since the data can be read and interpreted in many ways, like a law code or a religious text, it is more than an intermediary and at least a mediator. Whether the data needs to be treated as an actor is yet to be determined.

6.2. First thoughts on the simulation

Having identified the structure of our model, we will need to be able to represent actors and mediators. These are autonomous and have their own identity and states, and an arbitrary number of links into and out of them. Any usable simulation environment must support these features.

Relevant actor state variables might include, for example, technical and managerial competence for a project manager, the actor's degree of commitment to the on-going development of the system, and some measure of relevance/goodness of fit of the actor's current state to the system and its current direction of evolution (such as the relationship between the facilities provided by the programming language in use and changing demands of the system's developers as they evolve it).

Although more work is needed before we can make a final decision, it seems that it might be possible to model the networks of interacting actors, mediators and intermediaries using an active agent-based simulation environment such as Repast [11]. We now consider some of the challenges which building an ANT-based simulation will pose to its designers.

One problem which emerges in building a simulation of an actor-network is Latour's claim that the outputs of a mediator cannot be derived deterministically from an examination of its inputs [7:58–9]. Latour refers here to the complexity which arises in considering a situation in which mediators do not act deterministically but add richness of their own to the social process. He describes how puppeteers interact with their puppets in a far richer way than merely pulling the strings; the puppets themselves suggest actions to their (alleged) controllers.

How can a situation like this be simulated? Any programmed solution cannot be as rich as the real world can be, in that the simulated mediators cannot be programmed with all of the possible outcomes from all possible inputs. However, we can take advantage of domain knowledge to recognise that in this social situation the actions of any element, be it actor or mediator, are severely circumscribed by both social and technical aspects. For example, the actions of the human actors who are evolving the system are limited by the mindset engendered by their education and training, and by the norms of the discipline within which they work [15].

Similarly, the actions of technology-based mediators are limited to what that technology can actually do for (and to) those who use it, although Latour's analogy of the puppeteer suggest that there can more feedback in this than might initially be expected.

Therefore we propose that the simulated possible actions of technical mediators be limited to a small number of effects, and that these be selected randomly with probabilities depending in part on the current situation in which the mediators currently find themselves; the possible actions and parameter values will need to be determined in discussion with domain experts. As a result of this degree of randomness in the simulation, any particular model may need to be run a number of times on a Monte Carlo basis, in the expectation that some pattern of results will emerge from a large set of repeated runs.

Having identified the attributes of the actors and mediators in the model, if we wish to develop a quantified model with predictive power we will then need to quantify both their state in the form of these attributes and the information transferred between them. Some of the variables and information may be ‘soft’, opinion/emotion-based and difficult to quantify, rather than being directly measurable. For example, the ‘strength of commitment’ of an actor to the continued existence and evolution of the system may need to be represented as a real value ranging over some range which should represent the actor’s current level of support for (or opposition to) that continued existence, as well as being able to represent maximum and minimum values.

Whilst this quantification and the reduction of human emotional states to deterministic or probabilistic processes might be seen as problematical, system dynamicists commonly represent ‘soft’ values in quantified form, often in the form of non-linear numeric scales. As to how to obtain the relevant values, it will be necessary again to refer to experts’ views as to how a specific process (or the process in general) works and reduce their opinions to quantified values. We have previously calibrated values for software processes based on expert opinion during the development of successful simulation models [3, 19].

Finally, a mechanism for connecting the elements of the model into a network will need to be developed. This might be achieved by having each element-representation hold a list of those elements whose current values influence its behaviour. Each element in the list would have its input value *weighted* to represent its current – and possibly changing over time – relative impact on the actor. To save programming effort, it may be best to make all possible connections when setting up the model, then set the weighting of a specific input to zero to reflect a link which is currently not present.

We intend to develop our current model into a simulation with the properties we have described, and expect that this simulation, when calibrated to values representing real-world activities and actions, will be able to replicate behaviours observed in real-world software evolution processes. Such a calibrated model will undoubtedly assist in improving the understanding of the global software process and its behaviours.

7. Acknowledgements

We are grateful to the referees of Software Evolvability 06 and to Martin Loomes and David Bowes for their valuable comments.

8. References

[1] K. Beck, *Extreme Programming Explained*, Addison Wesley, Boston, MA, 2000.

[2] A. Capiluppi, M. Morisio and J. Fernandez-Ramil, “The Evolution of Source Folder Structure in actively evolved Open Source Systems”, *Proc. Metrics 04*, 2004.

[3] B.W. Chatters, M.M. Lehman, J.F. Ramil and P. Wernick, “Modelling A Software Evolution Process”, *Software Process: Improvement and Practice*, **5** (2–3), 2000, pp.91–102.

[4] P.J. Dunning-Lewis and C.J.W. Townson, “Using Actor Network Theory Ideas In Information Systems Research: A Case Study Of Action Research”, Working Paper 2004/025, Lancaster University Management School, 2004.

[5] G. Kahen, M.M. Lehman and J.F. Ramil, “Empirical Studies of the Global Software Process – The Impact of Feedback”, *Proc. Workshop on Empirical Studies of Software Maintenance (WESS’99)*, Keble College, Oxford, UK, Sept. 3–4, 1999.

[6] B. Latour, *Aramis, or the Love of Technology*, trans. C. Porter, Harvard, 1996.

[7] B. Latour, *Reassembling the Social*, Oxford, 2005.

[8] M.M. Lehman and J.F. Ramil JF, “The impact of feedback in the global software process”, *Journal of Systems and Software*, **46** (2), 1999, pp.123–134.

[9] M.J. Loomes and C.L. Nehaniv, “Fact and Artifact: Reification and Drift in the History and Growth of Interactive Software Systems”, *Proc. Fourth International Conference on Cognitive Technology: Instruments of Mind*, Lecture Notes in Computer Science 2117, Springer, 2001, pp. 25–39.

[10] M.J. Loomes, C.L. Nehaniv and P. Wernick, “The Naming of Systems and Software Evolvability”, *Proc. Software Evolvability 05*, Budapest, 26 September 2005, IEEE Computer Society Press.

[11] M.J. North, N.T. Collier, and J.R. Vos, "Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit," *ACM Transactions on Modeling and Computer Simulation*, **16** (1), ACM, January 2006, pp.1–25.

[12] Privacy International, “European Commission Begins Consultation on Data Retention”, http://www.privacyinternational.org/article.shtml?cmd%5B347%5D=x-347-64804, 5 August 2004, referenced 30 June 2006.

[13] A. Sidorova and S. Sarker, “Unearthing Some Causes of BPR Failure: An Actor-Network Theory Perspective”, *Proc. Americas Conference on Information Systems 2000*, Long Beach, CA, August 10–13, 2000, Association for Information Systems.

[14] Ventana, Vensim Software, http://www.vensim.com/software.html, referenced 30 June 2006.

[15] P. Wernick "A Belief System Model for Software Development: a framework by analogy", PhD thesis, University College London, London, 1996.

[16] P. Wernick and T. Hall, "Simulating Global Software Evolution Processes by Combining Simple Models: An Initial Study", *Software Process: Improvement and Practice*, **7**, 2002, pp.113–126.

[17] P. Wernick and T. Hall, "The Impact of Using Pair Programming on System Evolution: a Simulation-based Study", *Proc. IEEE International Conference on Software Maintenance 2004*, IEEE Computer Society Press.

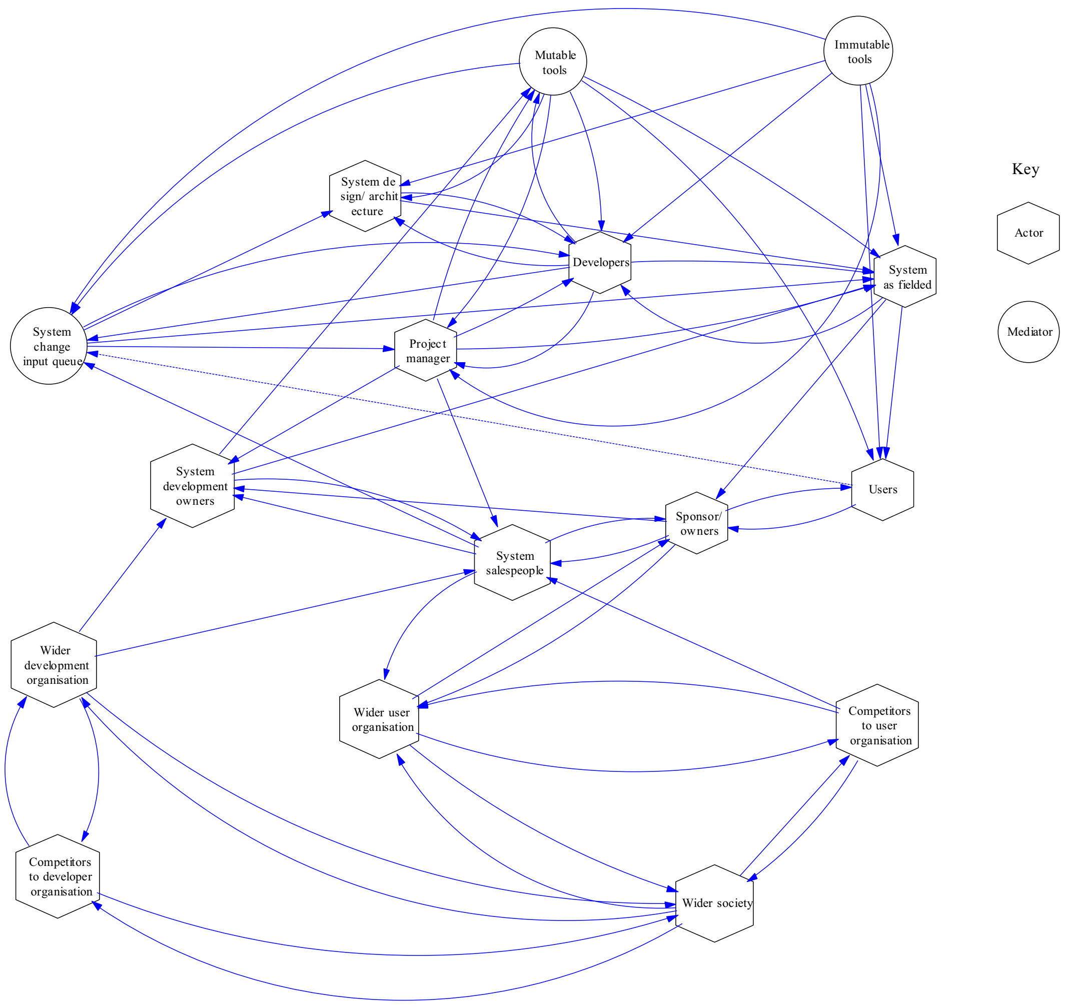
[18] P. Wernick and T. Hall, "Can Thomas Kuhn's paradigms help us understand software engineering?", *European Journal of Information Systems*, **13** (3), 2004, pp.235–243.

[19] P. Wernick and M.M. Lehman, "Software Process Dynamic Modelling for FEAST/1", *Journal of Systems and Software*, **46**, 1999, pp.193–201.

[20] B. Williams-Jones and J.E. Graham, "Actor-Network Theory – a tool to support ethical analysis of commercial genetic testing", New Genetics and Society, **22** (3), December 2003.

[21] E. Yourdon, *Death March*, Prentice Hall, 2003.

Figure 1: an ANT-based model of the global software process


Mutable tools
Immutable tools
Key
System de sign/ archit ecture
Actor
Developers
System as fielded
Mediator
System change input queue
Project manager
System development owners
Users
Sponsor/ owners
System salespeople
Wider development organisation
Wider user organisation
Competitors to user organisation
Competitors to developer organisation
Wider society