Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2008 Proceedings

Americas Conference on Information Systems (AMCIS)

2008

System Analysis as Scientific Inquiry

Joerg Evermann
Memorial University of Newfoundland, jevermann@mun.ca

Riddhi Mistry
Victoria University of Wellington, riddhimistry@yahoo.co.in

System Analysis as Scientific Inquiry

Joerg Evermann
Memorial University of Newfoundland
jevermann@mun.ca

Riddhi Mistry
Victoria University of Wellington
riddhimistry@yahoo.co.in

ABSTRACT

Information systems are understood as models or representations of an application domain. System analysis is a mode of inquiry for purposes of understanding the domain and effecting change in it. Scientific inquiry also aims for understanding and description of domain. In contrast to system analysis, scientific inquiry is based on a 4000-year history. Its processes and methods are well accepted and arguably successful. This paper explores the parallels between the two processes and shows implications of viewing system development as a kind of scientific inquiry. A descriptive survey of system development in organizations presents empirical indications as to whether these parallels are in fact observed in system development practice.

INTRODUCTION

Information systems are models of an application domain (Iscoe *et al.*, 1991; Jackson, 1995). For example, a production planning and control system is intended to represent, with its software structures and its data, the production facilities of the business. System development begins with system analysis, an inquiry into and documentation of the application domain, and continues to the implementation and use of the final software system. System development is a relatively young field, and there still exists much debate about its methods and processes. In contrast, there is a rigorous process of inquiry into a domain that is much older, is well accepted in the community practicing it, arguably successful as measured by its results, and well regarded by its wider audience. This is the process of scientific inquiry.

This paper explores the parallels between the processes and artifacts of science and system analysis. As both are concerned with the investigation and description of a domain, we suggest that much can be learned by applying well-accepted and successful scientific methods to the process of system analysis. At the same time, we are interested to what extent scientific practices are already present in system development. We have therefore operationalized the recommendations in the form of a survey instrument. A questionnaire was initially sent out to the CIOs of the 100 largest users of personal computer systems in New Zealand. From these, 15 responses were obtained. The survey and the descriptive results are presented in the following sections of the paper.

The paper begins by examining the development process, and then continues with the role of conceptual models as theories of a domain, modeling languages as paradigms, and social aspects. We conclude the paper with a brief discussion and outlook towards further work.

DEVELOPMENT PROCESS

The first analogy between science and system development concerns the central artifact of each mode of inquiry. In system development, the purpose of the conceptual model is to describe the elements of the domain and their relationships (Mylopoulos, 1992). The conceptual model serves as the basis of understanding and problem solving within the domain. It allows analysts to capture and communicate their understanding of a domain and the problems in the domain. In science, it is the role of theories to describe the constituents of the domain, their relationships and their behavior, and to serve as the means by which problems in a domain are specified and solved. With this central analogy, we turn to the process of inquiry.

Science is defined by its iterative process, rather than its outcome. This distinguishes it from other modes of inquiry and is argued to make it respectable (Thagard, 1998; Casti, 1989). Each iteration proceeds from a problem to a candidate solution or hypothesis, which is then tested and the test results evaluated (Bunge, 1998; Popper, 1968, 1972). The system development process is analogous to the scientific process: Starting with an initial business problem, the analyst forms a theory about the organization in the form of a conceptual model, which expresses the analysts understanding of the domain (Mylopoulos, 1992). This conceptual model then forms the basis for subsequent software design and implementation. The resulting system is understood to be not only a representation of the domain but also a solution to the initial business problem. The system, and

implicitly the model it is based on, is then tested to determine whether it solves the initial problem, similar to the way a scientist might test a hypothesis to evaluate a theory.

As repeated hypothesis formation and testing are a *normal* process in the sciences, system development should similarly be expected and managed as an iterative process. Iterations are necessary for the development process, rather than failures of the process. The iterative nature of IS development has been recognized as useful for example to manage project risk (Boehm, 1988), and is found in evolutionary prototyping approaches (Floyd, 1984; Riddle, 1984) as well as in the more recent Rational Unified Process (Kruchten, 2002). We asked our survey respondents to what extent their organizations used iterative development processes:

- On what percentage of development projects has your organization used prototyping to allow future system users to assess the correctness of the system or software models?
- On what percentage of development projects has your organization used evolutionary prototyping or agile methods?

The responses in Figure 1 show that most organizations employ little prototyping, on less than a quarter of their projects. The reported use of agile methods was somewhat higher, indicating that our respondents recognize these as different techniques. Still, most organizations use such methods on less than half their projects. However, of those organizations that used some form of prototyping, 9 out of 10 use more than two iterations per project, indicating acceptance of prototyping or agile methods. Note that while agile methods demand almost constant refactoring, this would not be seen as an iterative method. Instead the quick release cycles, e.g. every four weeks in XP, are seen as iterations here. Further, while evolutionary prototyping and modern agile methods differ, they both embody a fundamentally iterative approach.

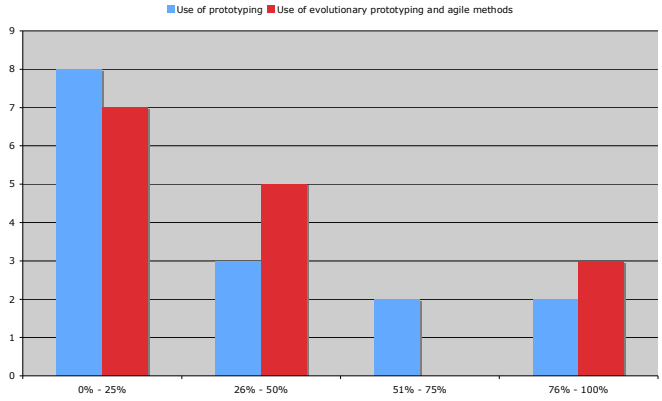

Figure 1: Use of iterative methods and iterations per project

No scientific theory is entirely accurate. It will necessarily be revised and replaced by more accurate, more predictive, or more explanatory theories; the process of scientific discovery never ends. Similarly, no conceptual model or software system should be expected to be completely accurate in representing the domain; it will need repeated adjustment and revision. System analysis should therefore be seen as an ongoing process rather than a one-off endeavor and should be actively managed throughout the period where the need for the system remains. We asked our respondents to what extent they keep re-evaluating systems during the their production life-span, as we wanted to know whether they are developed and forgotten, or whether they are proactively adapted and improved:

- After the implementation and rollout of each application, how frequently is the system re-evaluated for fit with the business?
- Is there a formal process in place for such re-evaluations?

The responses in Figure 2 show that most organizations take a "develop and forget" approach to system development and only revisit a system when specific problems occur. However, about a third of our respondents take a more pro-active approach to re-evaluation and continuous improvements of their systems. This is typically done on a more or less regular schedule of less than once a year. Ten of our 15 respondents reported having a formal process in place for re-evaluations. However, we found no correlation between a pro-active approach to system re-evaluation and a formal process being in place, suggesting that

some respondents have a formal process for ad-hoc re-evaluations. A build-and-forget approach has been advocated especially for the typically short-lived internet-facing applications of a firm (Baskerville et al., 2003) and it may be these types of applications that our respondents had in mind. However, not all application development is of this nature, as the many legacy-systems that are still in use demonstrate.

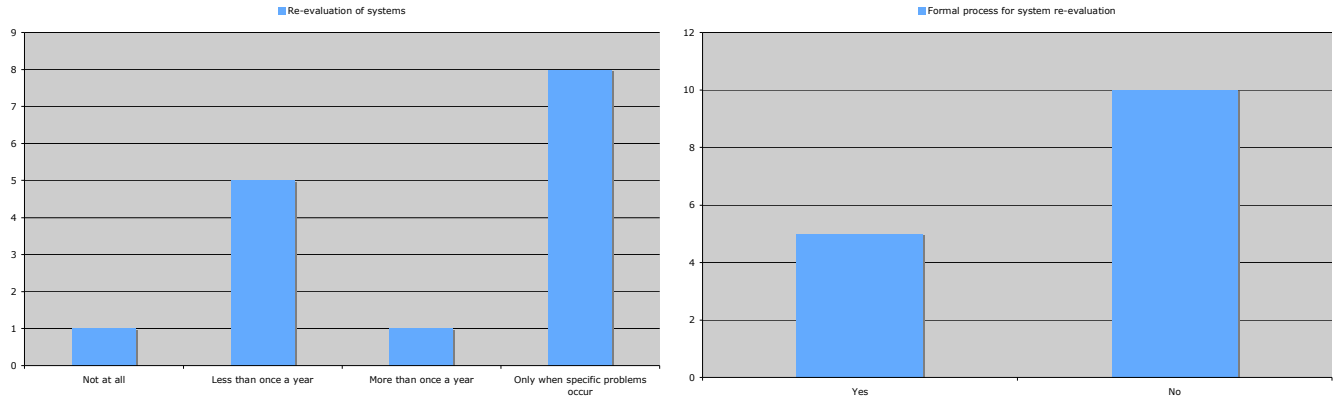

Figure 2: Re-evaluation of existing systems

In science, theories are tested based on a-priori acceptance criteria, which determine whether test results are valid and acceptable. There are criteria for statistical validity, acceptable research design, etc. Similarly, system development projects need to establish acceptance criteria that determine whether a conceptual model or software system is a valid representation of a domain. We asked our respondents whether their organizations had such a-priori criteria in place:

- For each software development project, are there any acceptance criteria defined for the completed system before testing the completed system?

Figure 3 shows that about half of the responding organizations have testing criteria in place before a test commences, similar to the way in which scientists have a-priori criteria in place before testing a theory by means of hypotheses. The remainder of our respondents either does testing without pre-defined criteria, or perhaps uses the test results to establish acceptance criteria post-hoc. Such behavior would not be accepted in the scientific inquiry mode.

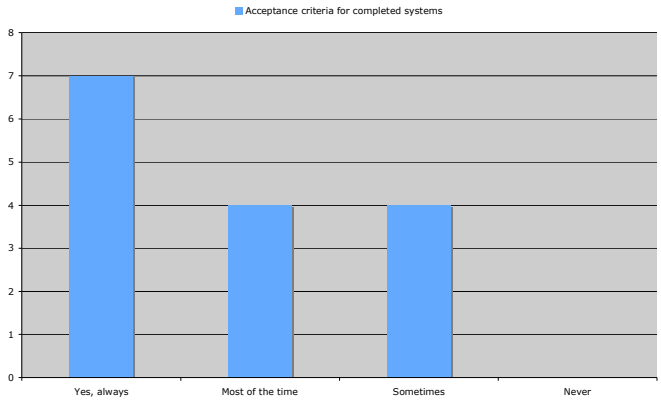

Figure 3: Use of pre-defined test criteria

THEORIES

Scientific theories are more than mere sets of logical statements or a number of axioms. They are structures with a central axiomatic core and surrounding corollaries and derived laws (Lakatos, 1978). In principle, an experiment tests the entire logically connected structure of a theory. In practice, when observation disagrees with prediction, scientists will attempt to find the weakest links in the argument and adapt or discard it (Duhem, 1954; Quine, 1953), which are those that connect the edges of the theoretical structure to empirical reality (Quine, 1953). Re-evaluation of a particular part of a theory may in turn require recursive re-evaluation of other parts until the whole structure is again logically consistent. Drawing the analogy to system development, the implemented software forms the outer fringe of the theory, where theory meets reality. We asked our respondents where they look for faults, and where they fix faults:

- When software applications (or their prototypes) do not satisfy their requirements, where do project teams in your organization tend to look for problems? Please rank in importance.
- When software applications (or their prototypes) do not satisfy the requirements, where do project teams in your organization apply solutions? Please rank in importance.

The rank responses in Figure 4 show that the core system model is most important for problem identification, followed by the software model and then the programming code. Organizations appear to assume that problems have deep rather than surface causes. However, when it comes to making changes to solve problems, the system model typically ranks in third place, with the software model in second place. Our respondents provided ambiguous information about the importance of programming code for problem solving, but it generally ranks higher than it does for problem identification. One respondent added scoping of the system as the primary way of solving problems.

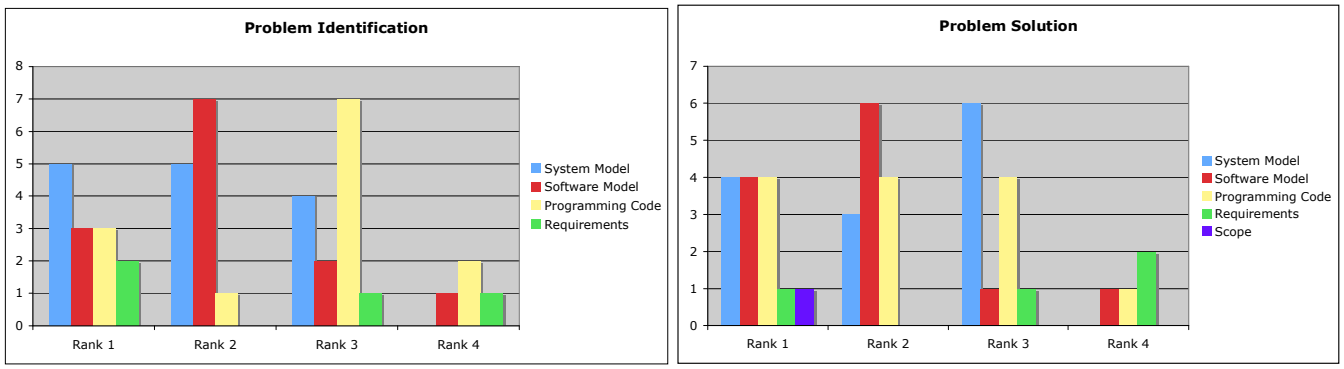

Figure 4: Problem identification and solution

Projects need to establish principles that guide the process of recursive re-evaluation of model elements to maintain internal consistency. We asked our respondents to what extent formal or informal processes are in place to establish and maintain consistency of the models. The responses in Figure 5 show a mix of both formal and informal processes employed to about the same extent. Consistency is not always achieved or maintained, but most respondents strive for internal consistency in some of their models, with only a few reporting that model consistency as not being important. These results indicate that models may not always serve the role of a rigorous, logical theory, but may instead be used for quick communication where consistency may not be necessary, in agreement with earlier findings about the use of UML (Dobing and Parsons, 2006)

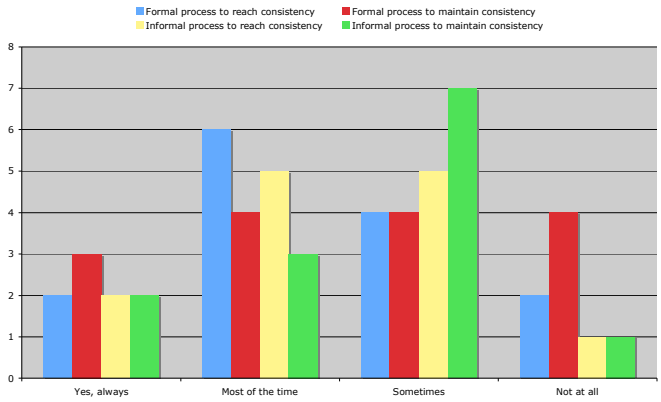

Figure 5: Achieving and maintaining consistency

Scientific theories include statements also about the dynamics of domain elements, which are often neglected in system analysis models that, before a database background, focus on static structures. This is reflected in the responses from our sample when we asked to what extent domain behavior and dynamics are modeled. Only 3 of our 15 respondents always document the domain dynamics, the remainder does this infrequently (Figure 6). This indicates that important aspects of the domain description are neglected in many development projects. Again, this agrees with findings by Dobing and Parsons (2006) who report that UML class diagrams are used much more frequently than activity or collaboration diagrams.

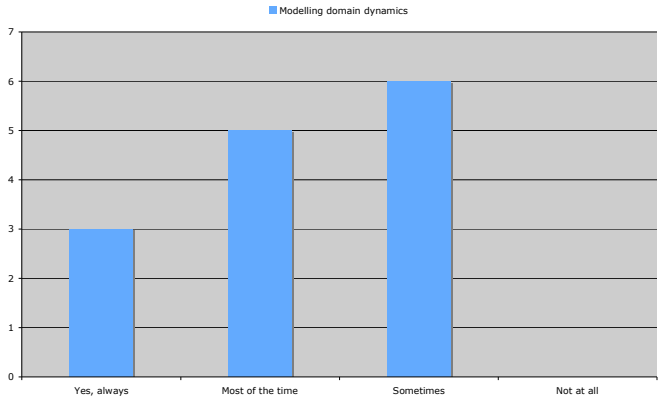

Figure 6: Modeling domain dynamics

PARADIGMS

Theories depend on the scientific paradigm, which determines the ontology, the terms in which we think and speak about the domain (Kuhn, 1996). In system development, the conceptual model as a domain theory relies on the modeling language that determines the terms in which the domain can be described. In science, the failure of "normal science" gives rise to a "scientific revolution" (Kuhn, 1996). The paradigm is discarded and with it the language and any theories. In system analysis, such a revolution would discard the modeling language and all conceptual models. For example, the analyst may realize that the business can be better described in terms of agents and their goals (Yu, 2001) rather than objects and their operations. The challenge for development projects is to recognize and facilitate necessary change, rather than oppose it. We asked our respondents whether "revolutions" in modeling occur in their projects:

- On average per project, how often do project development teams in your organization discard and rebuild the system or software models?

The responses in Figure 7 show that this is not usually the case and organizations typically stick with their initial conceptual models. Given the responses in Figure 4 above showing that system and software models are modified in the process of problem solving, this seems to imply that many models are flexible enough to accommodate even major changes without requiring rebuilding. On the other hand, it might mean that projects are on what Yourdon calls a "death march" where, despite obvious needs for change, the team retains the existing model (Yourdon, 1997).

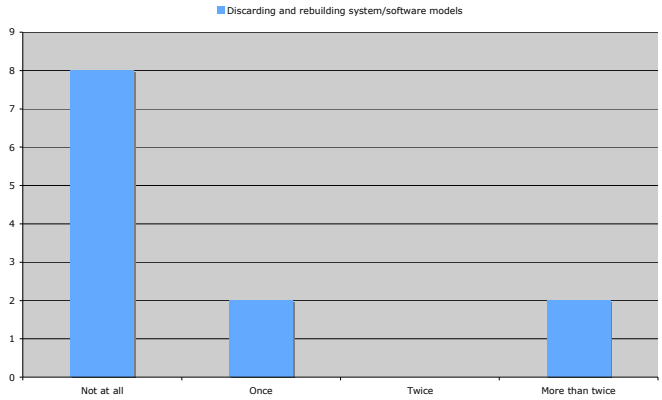

Figure 7: Discarding and rebuilding of system or software models

In the science, revolutions occur when the community of scientists agrees that the existing paradigm is no longer sufficient. We were interested in whether organizations have processes in place to review conceptual models, and to identify the need for change. We also wanted to know whether this was driven by agreement among analysts, as in the scientific mode of inquiry:

- Does your organization have formal or informal processes for review and changes to software or system models?
- Do software developers in your organization seek agreement with other project stakeholders or project members on changes to the software or system models?
- Do project members in your organization usually agree with each other on when and how to change the software and system models?

The responses in Figure 8 show that both formal and informal processes are in place for some projects, though not for all. As in the scientific mode of inquiry, the process of evaluating models and assessing the need for change is typically driven by agreement among analysts. The community of analysts determines when a model as domain theory has outlived its usefulness and where and how it needs to be changed. Project members frequently communicate with other stakeholders to achieve agreement on changes to the models. A related question about the presence of formal review board or panel for changes to software or systems models showed that 8 organizations had a formal review board or panel, and 7 responding that they did not, indicating that reviews are done informally without a set process, as is also the case in science, where not central board or panel exists to determine the goodness of theories.

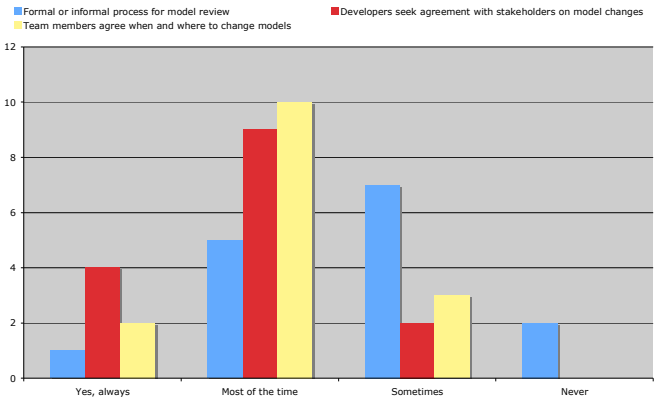

Figure 8: Processes for discarding software or system models, and review of changes

Co-existing Paradigms

Kuhn's scientific paradigms need not be seen as necessarily successive and mutually exclusive, but may be viewed as co-existing on different levels of abstraction. Drawing the analogy to system analysis, multiple modeling languages may be used at the same time, each being applicable in its own domain of abstraction. For example, analyzing an organization on the business unit level can describe functional structures and control relationships. When analyzing departments within business units, analysts may want to describe the dynamics of individual, goal driven actors and their activities (Yu, 2001). The challenge is to clearly delineate the respective domains of the various languages and to maintain consistency between models in different languages. For example, business process descriptions using UML Activity Diagrams need to be consistent with models of goal driven behavior of agents. We asked our respondents to what extent they use multiple modeling languages, and whether they maintain consistency between models. We were also interested in whether they use multiple models at the same time in their projects, as an indication of competing theories or paradigms:

- How frequently do development projects in your organization use more than one modeling language for software development projects?
- When development teams in your organization use multiple modeling languages together, to what extent do they ensure the consistency between the different diagrams?
- To what extent do project teams in your organization originally work with or develop multiple, different software and/or system models?

The responses in Figure 9 show that multiple languages are used frequently, but not on every project. When multiple languages are used, most organizations attempt to maintain consistency between the models in these languages. A follow-up question about the different languages in use revealed that many organizations use either multiple UML diagrams, or UML diagrams together with Entity-Relationship models. In contrast, while multiple language use appears to be accepted, the use of multiple or competing models of the same domain is not usually practiced. Only two of our respondents regularly work with multiple domain models at the same time, while the large majority never uses multiple models. These findings are also in agreement with results of Dobing and Parson's (2006) study on UML use, which found most organizations use multiple UML diagram types for each project.

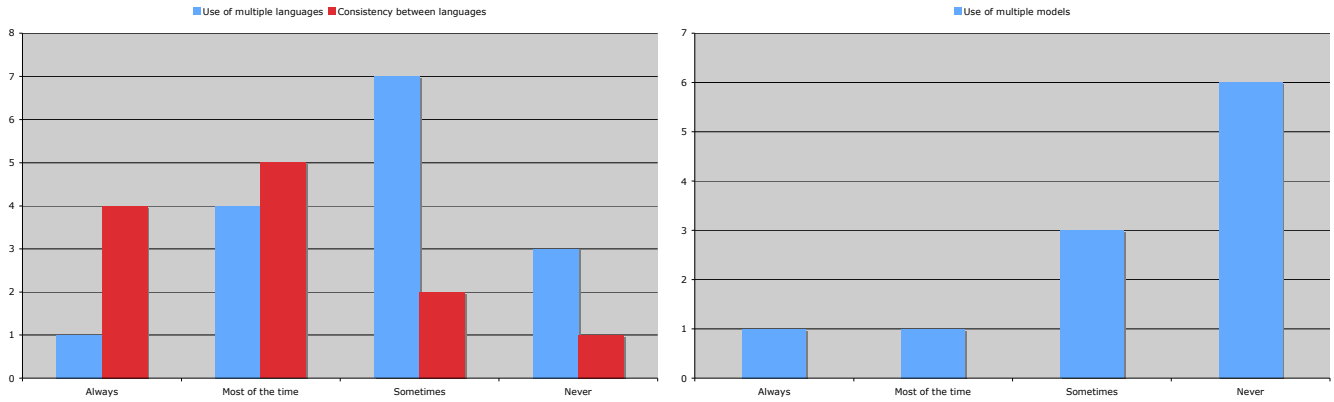

Figure 9: Use of multiple languages and models

Choice of Theory

Scientific theories cannot be proven; they can only be refuted (Popper, 1972). A theory is preferable to another, if it has withstood more attempts at refutation (Popper, 1968; Lakatos, 1978). This implies for system development that the conceptual model should be rigorously tested and evaluated. Statements and predictions of the conceptual model can be tested and should be attempted to be falsified by actively looking for counter-examples within the domain, using techniques such as model walkthroughs. Theory assessment and evaluation in science is a social activity with both formal and informal processes at work (Laudan, 1990). We asked our respondents to what extent this was the case in their organizations:

- Does your organization have a formal evaluation process of software and system models such as model walkthrough?
- Does your organization have an informal evaluation process of software and system models such as a model walkthrough?

The responses in Figure 10 show that many organizations have both formal and informal processes for model evaluation. However, it is also clear that these processes are not always applied. It appears that formal processes, when in place, are applied more consistently than informal processes. As a result, many system or software models may not be evaluated at all and problems will not be known until the final software is built, making problem solutions much more expensive (Boehm, 1988).

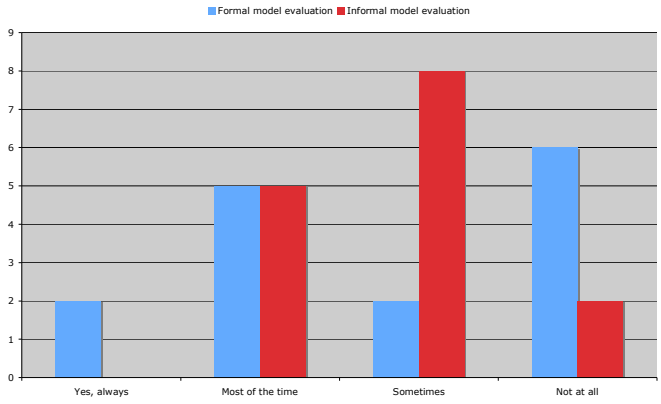

Figure 10: Model evaluations

SOCIAL ASPECTS

Scientists are dependent on frequent and open communication with each other for sharing of ideas, instruments, etc. (Longino, 1990). The community is not hierarchical, but is decentralized, open, and self-governing. Hence, system analysis as scientific inquiry should equally be a team activity with good communication. Mechanisms should be provided to facilitate exchange also of incomplete, initial, and informal ideas, not only of finished deliverables. Analysts should not use strict division of labor and compartmentalized thinking. This exchange of ideas and instruments should extend beyond the scope of a particular project, just as it is in science. Best practice approaches such as reference models (Scheer, 1994), analysis and design patterns (Larman, 2002) and open source software (Raymond, 2001) are examples of IS development approaches embodying these ideas. We were interested in the communication modes of team members in our responding organizations and to what extent the decision-making in system development projects is an open, participatory process:

- How frequently do project team members communicate formally (e.g. meetings, etc.) about system and software models during the software development process?
- In your organizations, how do project team members resolve conflicts of opinions about the software and/or system models?

The responses to these questions in Figure 11 show that many organizations provide formal communication structures and information exchange using these structures is frequent, often more than weekly. Decision-making is very rarely dominated by a single individual and primarily depends on communication and finding agreement. Interestingly, organizations do not employ a simple voting system, but prefer communication and unanimous agreement. We also asked our respondents whether formal communication channels are defined in their teams. Surprisingly, 4 of 15 respondents reported that no formal communication channels are defined, implying that a very open culture of informal communication must exist in order to support efficient collaboration among team members.

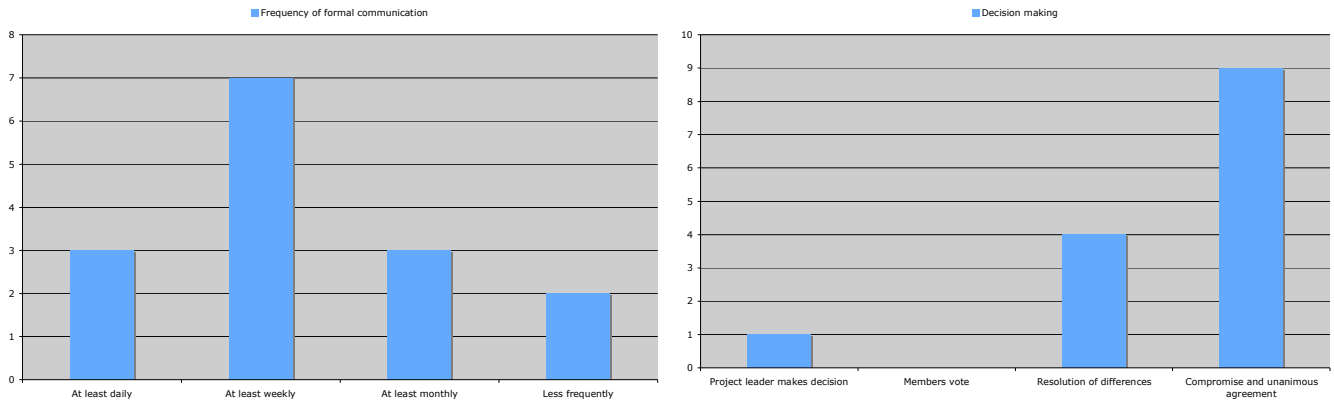

Figure 11: Communication and decision-making

We were also interested to what extent our respondents provide incentives to foster an open and participatory culture, recognizing that science relies primarily on intrinsic rewards, i.e. the pleasure of discovering things (Feynman, 2001):

- To what extent does your organization encourage project members to contribute to the management of the software development, such as providing input at meetings, presenting alternative courses of actions, etc.?
- Does your organization have a formal reward (monetary or otherwise) system for project members' contributions?
- Does your organization an informal recognition for project member's contributions?

The responses in Figure 12 show that while organizations encourage participation, formal reward systems are rare and even informal recognition is only infrequently employed. It appears that organizations rely on intrinsic motivation of team members, as is done in science, where the reward is the work itself, and the satisfaction that a successful project can bring with it. The open source movement has demonstrated that this principle can and does lead to very successful development projects (Raymond, 2001).

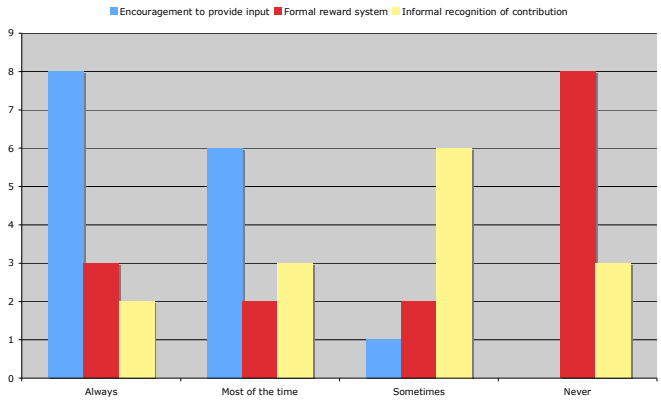

Figure 12: Rewards and incentives

We were also interested in how our respondents judge the structure of their development teams and asked them to rate whether their development projects are predominantly individual or group activities, and whether their development teams are hierarchical or non-hierarchical. Figure 13 shows a bimodal distribution of responses. While many organizations view system development as a group or team activity, similar to scientific discovery, a significant number also see it as primarily an individual activity. Most of our respondents characterize their team structure as open and non-hierarchical with only one respondent organization having strictly hierarchical teams. This agrees with scientific practices, which are generally non-hierarchical, self-governing, and open.

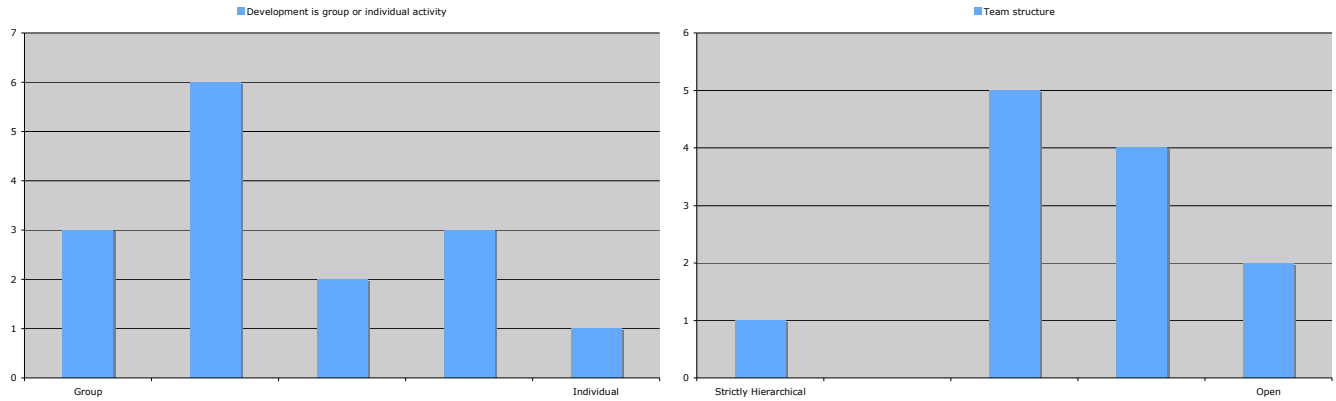

Figure 13: Team structure

In scientific studies, non-scientists may be sources of information but are generally not involved in theory construction. Similarly, in system analysis, users may be sources of information, e.g. by being part of user focus groups, or subjects of inquiry, e.g. in interface development testing. However, participation in domain analysis and conceptual modeling should be limited to trained analysts. We asked to what extent end-user participation occurs in projects and whether end-users are trained to participate in development projects:

- To what extent does your organization employ end-user participants in the development of the system and/or software model?
- To what extent does your organization train end-user participants in IS development techniques before their participation on projects?

The results in Figure 14 show that while end-user participation in development projects is common, with only one organization responding never to employ end-users, the training of end-users is not. Only a few organizations realize that end-users must be trained for them to provide effective collaboration with the main development team and to be effective in participating in system analysis. The lack of end-user training may indicate that end-users are not employed as co-analysts but merely as sources of information, in effect playing the role of scientific subjects. However, Dobing and Parsons (2006) found

that man organizations use UML extensively for user-analyst communication, which would require at least some user training.

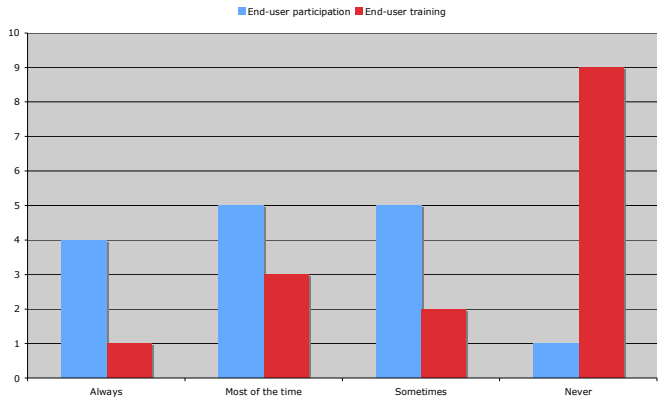

Figure 14: End-user participation and training

DISCUSSION AND CONCLUSION

The previous sections have shown many parallels between scientific inquiry into a domain and system development, specifically system analysis using conceptual models. Table 1 contains a brief list of the identified analogies.

Scientific Inquiry	IS Development
Theory	Conceptual (Domain) Model, Software model
Instrument	Software system, Prototype
Paradigm	Organizational framework
Ontology	Modeling language
Theory building	System analysis
Instrument building	Software engineering
Theory testing	Prototyping and evaluation
Scientists	System analysts
Engineers	Software engineers

Table 1: Parallels between scientific inquiry and IS development

While there are many commonalities, there are also differences between the scientific mode of inquiry and system analysis as inquiry into a domain. System development responds to a particular need, rather than conducting inquiry for the sake of inquiry, as is done in science. Furthermore, unlike science, which operates in a relatively stable environment (laws of nature don't change), system analysis is an inquiry into often constantly changing domains. However, rapid change may be an indication that the models and theories about the business are not on a suitably abstract level. For example, while the specific business processes may change, the structure of the business and the behavior of its agents may be stable.

Our survey of system development practices in organizations shows mixed results. While some aspects of scientific practice are present, primarily the open, non-hierarchical, participatory culture that values intrinsic rewards, other elements are not. Organizations use an iterative process that improves on prior models and theories, but terminate the process when the system is delivered, instead of realizing that analysis should be ongoing, as no model or software will ever be a "true" representation

of the domain. Also, in the process of improving the conceptual models and software solutions, the use of pre-defined test criteria is not nearly as prevalent as it is in science, where this is commonly accepted and required.

Many organizations realize the importance of models as theories that can be used to explain and predict the behavior of software that is built based on them and consequently use them as tools for problem identification. However, when it comes to system modifications, the picture is unclear. An equal number of our respondents rank the system model, the software model, and the programming code as most important for modifications. This agrees with scientific practice where problems may sometimes be a result of the instrument, and sometimes be a result of the theory.

One area where system analysis practice departs from scientific practice is in discarding and rebuilding conceptual models, which is not generally done. In contrast, scientists are typically prepared to admit that their theories and models may be wrong and to discard them to begin anew. Another area of departure is the use of multiple models or theories. Scientists are quite willing to work with multiple, competing models, and to develop them independently to the point where serious differences arise. However, our respondents have indicated that the same is not usually done for conceptual models in their organizations. Only 5 of 15 organizations employ multiple models, and then only for some of their projects.

Our original goal of the study was to not only examine the extent of scientific practice in system analysis, which showed mixed results, but also to investigate whether their use had any impact on development project success. To this extent we asked our respondents to indicate their project failure rate. We defined project failure as projects that are either over budget, over time, or lacking in functionality. However, the responses lacked sufficient variability for further analysis, as all respondents indicated that their failure rate was "0%-25%", which is surprising, given that industry surveys put the average failure rate at about 30%, rising to about 50% when "challenged" systems (which are included in our definition of failed) are included (Standish Group, 2001).

Finally, we acknowledge that our response rate was lower than we had hoped for. We see the present study as a work-in-progress and are continuing data collection to build up a more accurate picture and description of industry practices in system analysis. This might also yield more variability on the project success measure, so that future studies might examine the impact of scientific practices on project success.

REFERENCES

1. Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J. and Slaughter, S. (2003) Is Internet-Speed Software Development Different? *IEEE Software*, **20**(6), 70-77.
2. Boehm, B. (1988). Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, **14**(10), 1462–1477.
3. Bunge, M. A. (1998). *Social Science under Debate: A Philosophical Perspective*. University of Toronto Press, Toronto.
4. Casti, J. (1989). *Paradigms Lost*. Avon Books, New York, NY.
5. Dobing, B. and Parsons, J. (2006). How UML is used. *Communications of the ACM*, **49**(5), 109-113.
6. Duhem, P. (1954). *The Aim and Structure of Physical Theory*. Princeton University Press, Princeton. Translated by P. Wiener.
7. Feynman, R. (2001). *The Please of Finding Things Out*. Penguin Books. London, England.
8. Floyd, C. (1984). A systematic look at prototyping. In R. Budde, K. Kuhlenkamp, L. Mathiasssen, and H. Zullighoven, editors, *Approaches to Prototyping*. Springer Verlag, Berlin.
9. Iscoe, N., Williams, G. B., and Arango, G. (1991). Domain modeling for software engineering. In Proceedings of the 13th International Conference on Software Engineering ICSE 91, Austin, TX, pages 340–343.
10. Jackson, M. (1995). The world and the machine. In *Proceedings of the 17th International Conference on Software Engineering ICSE 95, Seattle, WA*, pages 283–292.
11. Kruchten, P. (2002). *The Rational Unified Process: An Introduction*. Addison-Wesley, Reading, MA.
12. Kuhn, T. (1996). *The Structure of Scientific Revolutions*. The University of Chicago Press, Chicago, third edition.
13. Lakatos, I. (1978). *Philosophical papers*. Cambridge University Press, Cambridge, NY.
14. Larman, C. (2002). *Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process*. Prentice-Hall, Upper Saddle River, NJ.
15. Laudan, L. (1990). Demystifying underdetermination. In C. W. Savage (editor), *Scientific Theories*, Vol. 14. University of Minnesota Press, Minneapolis, MN.

16. Longino, H. E. (1990). *Science as Social Knowledge: Values and Objectivity in Scientifi Inquiry*. Princeton University Press, Princeton, NJ.
17. Mylopoulos, J. (1992). Conceptual modeling and Telos. In P. Locuopoulos and R. Zicari, editors, *Conceptual Modeling, Databases and Cases*. John Wiley & Sons, Inc, New York et. al.
18. Popper, K. (1968). *The Logic of Scientific Discovery*. Harper & Row, New York, NY.
19. Popper, K. (1972). *Objective Knowledge*. Clarendon Press, Oxford.
20. Quine, W. v. O. (1953). Two dogmas of empiricism. In *From a Logical Point of View*. Harvard University Press, Cambridge, MA.
21. Raymond, E. S. (2001). Cathedral and the Bazaar. O'Reilly, Sebastopol, CA.
22. Riddle, W. (1984). Advancing the state of the art in software system prototyping. In R. Budde, K. Kuhlenkamp, L. Mathiasssen, and H. Zullighoven, editors, *Approaches to Prototyping*. Springer Verlag, Berlin.
23. Scheer, A.-W. (1994). Business Process Engineering. Reference Models for Industrial Enterprises. Springer Verlag, Berlin, 2nd edition.
24. Standish Group (2001). *Extreme Chaos*. The Standish Group, Inc. 1-12.
25. Thagard, P. R. (1998). Why astrology is a pseudoscience. In M. Curd and J. Cover, editors, *Philosophy of Science – The central issues*. W.W. Norton and Company, New York, NY.
26. Yourdon, E. (1997). *Death March*. Prentice Hall Publishing Company. Upper Saddle River, NJ.
27. Yu, E. (2001). Agent orientation as a modelling paradigm. *Wirtschaftsinformatik*, 43(2), 123–132.