Systemics, Communication and Knowledge: Shifts of Perspective and the Need for Requirements In Second-Order Science

Thomas J. Marlowe
Seton Hall University
South Orange, NJ 07079, USA
thomas.marlowe@shu.edu

and

Vassilka Kirova
Alcatel-Lucent
Murray Hill, NJ 07974, USA
vassilka.kirova@alcatel-lucent.com

ABSTRACT

The systemic view of second-order science emphasizes the interaction of observer and observed, but tacitly assumes a single observer, or at least a unity of observer perspective. But experience in multiple domains, including software engineering, decision science, health sciences, co-creation and Living Labs, knowledge management, community development and government policy has emphasized the multiplicity of goals and perspectives across stakeholders. We look at the issues that arise when multiple views are incorporated, and propose a toolkit for addressing those issues.

Keywords: systemics, second-order science, requirements, communication, knowledge management

1. INTRODUCTION

While Second-Order Science is based on the interaction of the observer and the observed, there is a tacit picture of a single investigator (scientist, practitioner, social scientist, and so on) or tight community interacting with an identified if not well-understood problem (Figure 1a — see last page). To facilitate discussion, we extend this picture (Figure 1b) to indicate that Observer A is focused on the interaction between herself and the problem B: here Alice (A) is focused on the interaction between herself and B, from its initial statement as a problem (?B) to its possible solution (!B); the outer box indicates that this focus occurs in the environment of Alice's context.

However, modern science is collaborative across larger scales, and applied science and technology is often undertaken by researchers/developers at the behest of clients and for use by consumers who typically lack a commonality of perspective, knowledge or context with those researchers.

In Section 2, we look at the traditional view of Second-Order Science as presented at IIIS Multi-Conferences and elsewhere, and in Section 3 look at the issues introduced by change in observer and focus, using the context of software development of an innovative product line or service. In Section 4, we survey a number of standard tools and approaches that can help resolve issues and narrow gaps, and the modifications that might be needed to apply these more generally. Finally, in Section 5, we briefly present our conclusions.

2. SECOND-ORDER SCIENCE

Second-order science has three related but different meanings in the literature [34]: (1) Web-mediated, cooperative and open collaborative scientific exploration, resulting in a hyperlinked resource in "the internet of things" [36]; (2) meta-science, combining multiple studies and disciplines to attain deeper knowledge [25]; and (3) the expansion of science by means of new concepts and theories, especially through the interaction of observer and observed [18]. We focus on the third of these, which (as its proponents argue) is inescapable in social, decision and health sciences, and frequently fruitful in other sciences.

Second-order science recognizes that science occurs in a context that affects interpretation of the problem, and both guides and constrains progress toward the solution. And while its primary concern is mutual effects from the interaction of observer, observed and context, it recognizes that change may also occur from changes internal to any of the three (reflection and reflexivity), or from external changes.

However, second order science (in the systemic view) concentrates on the interaction of science and the scientist, and thus tacitly assumes a single observer, or at least a unity of observer perspective. But experience in multiple domains and disciplines, including software engineering, decision science, health sciences, co-creation and Living Labs [13, 29], knowledge management, community development, and government policy has underlined the multiplicity of goals and perspectives across stakeholders in the problem, and emphasized the need to integrate these perspectives.

3. MULTIPLE OBSERVERS AND MULTIPLE PERSPECTIVES

We consider a paradigmatic situation of multiple observers: development of an innovative software product line or service. For simplicity and ease of both illustration and understanding, we assume a single client, a single experienced team of developers, and a single, largely homogeneous user community, and no other stakeholders. For convenience, we will think of each as a single individual: Carol, the client; Diane, the developer, and Ursula, the user.

Initially, Carol, the client, begins with a problem to which she would like a solution to be used, applied, or understood by Ursula, the user (Figure 2a). Carol then proposes the problem to Diane, the developer (Figure 2b), after which it becomes Diane's problem (Figure 3a). At this point, the real difficulty occurs: Diane does not have the same context and background as Carol, and may have a different understanding of the problem.

We can see that the problems emerge when context and perspective change: handoff (Figure 3a), delivery (Figure 4a) and use (Figure 5b). Further problems and complications, with even more views and versions, arise in debugging, maintenance and evolution, particularly if multiple stakeholders are involved in discovering and resolving the error, flaw, or environmental change. The fundamental if not fully realizable goal will be to involve all stakeholders and unify their perspectives in each step of the process.

Multiple perspectives and multiple stakeholders inherently introduce differences in background and expertise, work environment, problem languages and glossaries, context, knowledge, and expectations, in addition to issues introduced by differences in social culture and mother tongue. (There may also be complications introduced by standards, statutes and regulations, by intellectual property concerns, and by the need for trust.) These issues persist even if the multiple observers are multiple teams of researchers, multiple developers collaborating or otherwise working together, or a service being designed for a consortium of clients. Continuing, Diane solves the problem (Figure 3b) and delivers the solution (Figure 4a), at which point Carol needs to determine if her interpretation (X) of Diane's solution (C) corresponds to Carol's problem (B) (Figure 4b). Finally, the solution/product is deployed for use by Ursula (Figure 5a) — and the same problem arises again (Figure 5b).

4. ADDRESSING THE PROBLEM: A TOOLKIT

The problems identified above can be partially addressed and ameliorated via a combination of well-known approaches, with two goals, partitioning the approaches by the goal each addresses.

The first group aims primarily at establishing concordance between stakeholder views: interoperability (in both narrow and broad senses) to establish a common business and technical infrastructure and opportunities for shared frameworks, and support cooperation and collaboration; requirements elicitation and analysis, to identify stakeholders and their interests, as well as nature and constraints of the problem; and knowledge transfer and knowledge management, to enrich context and understanding of the problem, as well as to create shared context.

The second group is more oriented toward assuring a proper solution process and better solutions of the original problem: verification and validation, to assure both proper translation across contexts and correct steps toward solutions; maintenance and evolution for both the project and the product, to handle changes gracefully while preserving concordance; and a number of engineering approaches common in software engineering, aimed at optimizing the solution process.

Each group has some impact on the other goal, and the activities overlap and interleave. For example, requirements and knowledge management are clearly tightly coupled, and both are needed for good risk analysis and management, as is interoperability. Likewise, validation has to rely on requirements analysis and knowledge management to assist in clarifying stakeholder goals and to explicate constraints, as well as good communication and interaction; conversely, validation or verification failures lead to maintenance problems or evolution of the problem, its requirements, or its solution.

- Interoperability: Interoperability traditionally addresses platform and protocol standardization or alignment for IT and communication (data, objects, software systems, services, communication channels) [27, 35], to assure resources and channels for interaction, with standard meaning and effect (semantics) for shared functionality.
  It has been extended [8, 15, 20] to

  (1) alignment of technical and business processes (rules, process, strategy),
  (2) understanding and integrating culture, language and glossaries (culture, knowledge),
  (3) providing consistency for problem and solution facets (security, risk), especially at interfaces or affecting multiple stakeholders; and
  (4) supporting consistent use of shared frameworks (social networks, cloud) and a consistent project view and understanding of responsibilities (ecosystem),

  that is, to provide a common technical, business and knowledge environment to support communication, coordination and collaboration among stakeholders.

  While interoperability is most important with multiple autonomous developers, these facets matter even in the simple scenario we are considering.

- Requirements: Requirements acquisition, elicitation, and analysis are common engineering and business activities [7, 12, 16]. Short-circuiting or badly performing these processes has been responsible for many engineering and software disasters. Acquisition and elicitation seek to establish domain context, understandings, expectations and constraints; analysis to verify completeness, coherence and clarity. The key activity throughout the process is questioning: identifying stakeholders, considering goals, constraints, difficulties and risks [3]; eliciting stakeholder expectations; and assuring high quality, well-documented results. A standard list of requirements workshop questions can be found in [28]; these may need to be extended to deal with collaboration [22], even in the context of co-creation.

  Note that even though agile approaches [1, 21] place less emphasis on comprehensive requirements analysis, initial exploration is still required, both to define the problem and to reach an agreement among stakeholders to proceed.

There is also a documentation tradeoff with such approaches. On the one hand, documentation co-evolves with analysis and design in an iterative-incremental process, allowing for and benefitting from improving understanding and knowledge accumulation and refinement (itself requiring steady interactions and shared models of understanding). On the other, there is a risk that evolving implicit understandings may not be reflected in the final documentation. This may in turn affect maintenance and evolution, since solid requirements and design documentation is eventually needed to support collaboration or changes in the development team in the course of long-lived projects [32].

- Knowledge transfer and knowledge management: From the second-order science perspective, the main objective is to assure faithful translation and transfer of problem and context between observers/stakeholders, in tandem with validation and verification activities. With this in mind, important issues for knowledge management [2, 4, 5, 11] will include the following activities and concerns.

  Standardize glossaries, process notations, and concept maps as far as possible, often together with agreeing on infrastructure and tools. (Be careful of overloaded terms such as "security" [electronic? physical? personnel?] or "usability" [availability? ergonomics? disability access? learnability?], and of terms and constructs with social, linguistic, domain-dependent or enterprise-specific meanings or connotations [22].) Determine, classify and articulate knowledge (as far as possible) as explicit, implicit, or tacit knowledge, and (critically) identify assumed shared knowledge, and to the extent possible the kinds of integrated, collaborative or emergent knowledge that may arise [19]. In an enterprise setting, don't forget to consider implicit business rules and policy, and to identify not only formal but informal or implicit advocates/sponsors and flows of information.

- Testing, verification and validation: These three are related but not identical. Testing refers to determining correctness by interaction with the solution; validation to determining functionality with the solution deployed in the client's or users' environment (platform, user community, context, and so on), and verification to formal methods used to demonstrate properties of the solution.

  From the perspective of multiple observers in second-order science, there are two key activities, which may make use of any or all of these approaches. The first, affecting every instance is to check for consistency between the problem and its solution, and fidelity to its specification. The second, in the multiple observer/stakeholder scenario, is to check, to the extent possible, for consistency between understandings of the problem and context at transfer points (as illustrated in Section 3).

  In addition, particularly in the context of software engineering, protocols for interfaces have to be tested, validated and verified, as do fulfillment of stakeholder objectives and satisfaction of critical constraints. Moreover, where appropriate, the solution (or problem monitoring) should be instrumented to facilitate testing and validation. Such testing should be focused on interfaces and points of protected variation (PPV) [17]. These comprise locations at which one or more of the following occur: structural or semantic change is likely, risks are high, there are tricky special cases, critical requirements are addressed, or there is a "dependency knot"—many other components affect or are affected by actions or tests at this location.

- Modern software engineering processes and techniques: These basically fall into three groups: (1) interactions between stakeholders to clarify requirements and validate translations (focus groups, requirements negotiation, use cases and function lists, early prototyping, and ongoing stakeholder interaction, including co-design and co-creation or "Living Labs" [13, 29]); (2) approaches to support incremental and iterative convergence on the correct problem and solution, as well as debugging, maintenance and evolution (including software architectures, adaptive and agile methods [1, 21], design patterns [10, 17], refactoring [9]); and (3) approaches to facilitate collaboration and establish trust and sharing [22].

- Maintenance and evolution as a process: Multiple changes in context, problem and perspective, sometimes orthogonal and sometimes strongly interacting, must be handled, primarily by the developer [30]. In software engineering, maintenance includes preventive (security), adaptive (environmental changes and portability), corrective (fixing faults and bugs) and perfective (evolution) aspects. The latter two are the most significant for multi-perspective second order science.

  The most obvious issues to be addressed are errors in the original problem formulation or failure to properly handle unanticipated exceptions and corner cases. The solution must also evolve to account for changes in client or user needs, the underlying platform and infrastructure, or external issues such as security threats.

  In the knowledge transfer view, this process is both simpler and more complex than reconciling viewpoints in the development phase. On the one hand, a common context and common glossary has been established, although there may be problems arising there through higher-order contextual or linguistic differences. On the other hand, there are new difficulties in identifying and addressing discordance between developer solutions and client or user expectations, and in interpreting user reports of errors or difficulties.

  In addition, the process itself needs to evolve [31], using feedback from validation, quality assurance and maintenance activities plus project retrospectives [6, 14] to address perceived shortcomings or opportunities for optimization—an echo of the standard second-order science view.

Note that these issues have already been examined and partially addressed, not only in software engineering, but also in traditional engineering ([26]; also compare [23, 33]), and to some extent in management/decision science [12], community development [24, 29], and even in applied and pure science. But note also that the problems are made more complicated and new issues introduced by multiple clients, developers, user communities, and additional stakeholders, including governmental and standards agencies, as well as by intellectual property, privacy and security concerns [22].

5. CONCLUSIONS

The plurality of observers, contexts and perspectives common in the scientific enterprise calls not so much for a revision of second-order science and the systemic approach, as for an acknowledgment that the observer+observed view is a simplification. A full discussion of the systemic view of second-order science must take account of the issues of collaboration, translation, and interaction much as first-order science is in the process of doing.

In this paper, we have identified critical instants even in a simplified project, and suggested a number of tools and approaches borrowed from decision science and software engineering to partially mitigate the problems.

6. REFERENCES

1. Agile Consortium, *Agile and Business*. Accessed on August 10, 2012 at http://agileconsortium.blogspot.fr/2007/12/nokia-test.html Archived at http://www.webcitation.org/6Ij7bTlDe
2. W. Agresti, **Knowledge Management**, Advances in Computers, Vol. 53, 2000.
3. H. Barki, S. Rivard, J.Talbot, "An Integrative Contingency Model of Software Project Risk Management", **Journal of Management Information Systems,** 17 (4), Spring 2001, pp. 37-69.
4. K. Bondar, B.R. Katzy, "An Emergent Perspective Based on shared understanding in knowledge-based organizations", 19th ICE & IEEE-ITMC International Conference, The Hague, Netherlands, June 2013.
5. A. Bounfour, A. Leslie, F. Lettice, M. Neumann, N. Jastroch: *Creation of Innovation through Knowledge Management*, IST-2001-34442 CIKM project report, March 2004. Accessed on September 2011 at http://www.metconsult.com/html/english/CIKM%20project%20report.pdf
6. E. Derby, D. Larsen, K. Schwaber, **Agile Retrospectives: Making Good Teams Great**, The Pragmatic Programmer LLC, 2006.
7. Essential Strategies, *Requirements Analysis*. Accessed August 7, 2013 at http://www.essentialstrategies.com/services/analysis.htm Archived at http://www.webcitation.org/6Ij8GEo2v
8. European Commission: *Enterprise Interoperability Science Base.* Accessed on November 10, 2011 at http://cordis.europa.eu/fp7/ict/enet/fines-eisb_en.html
9. M. Fowler, *Information about Refactoring*, http://refactoring.com/sources.html, accessed June 8, 2012.
10. E. Gamma, R. Helm, R. Johnson, J. Vlissides, **Design Patterns: Elements of Reusable Object-Oriented Software**, Addison-Wesley, 1995.
11. W.P. Hall, S. Nousala, B. Kilpatrick, "One Company – Two Outcomes: Knowledge Integration vs. Corporate Disintegration in the Absence of Knowledge Management", **Vine**, 39 (3), 2009, pp. 242-258.
12. D. Hay**, Requirements Analysis: From Business Views to Architecture**, Prentice Hall PTR, 2003.
13. L. Heinze, I. Mulder, P.J. Stappers, "Understanding Networked Collaboration: Fields and Patches of Interaction", 19th ICE & IEEE-ITMC International Conference, The Hague, Netherlands, June 2013.
14. N.L. Kerth, **Project Retrospectives: A Handbook for Team Reviews**, Dorset House Publishing, 2001.
15. F. Koussouris, F. Lampathaki, S. Mouzakitis, Y. Charalabidis, J. Psarras, "Digging Into Real-Life Enterprise Interoperability Areas - Definition and Overview of the Main Research Areas", Collaborative Enterprise (CENT) 2011 Symposium, Proceedings of the 15th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2011), Orlando FL, July 2011.
16. P. Laplante, **Requirements Engineering for Software and Systems**, CRC Press, Redmond, WA, 2009.
17. C. Larman, **Applying UML and Design Patterns,** 3rd ed., Prentice Hall, Pub. 2004.
18. W. Lutterer, "Systemics: The Social Aspects of Cybernetics", **Kybernetes: The International Journal of Systems & Cybernetics**, 34 (3), pp 497-507, 2005. Accessed on-line on August 7, 2013 at http://www.lutterer.de/Lutterer%20-%20Systemics.pdf
19. T.J. Marlowe, N. Jastroch, V. Kirova, M. Mohtashami, "A Classification of Collaborative Knowledge," **Journal of Systemics, Cybernetics, and Informatics**, Vol. 9 (7), 2011.
20. T.J. Marlowe, N. Jastroch, S. Nousala, V. Kirova, "Collaboration, Knowledge and Interoperability—Implications for Software Engineering", Collaborative Enterprise (CENT) Symposium 2012, Proceedings of the 16th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2012), Orlando FL, July 2012.
21. R.C. Martin, **Agile Software Development: Principles, Patterns and Practices**, Prentice Hall, 2002.
22. M. Mohtashami, T. Marlowe, V. Kirova, F. Deek, "Risk-Driven Management Contingency Policies in Collaborative Software Development," **International Journal of Information Technology and Management**, 10 (2-4), 2011, pp. 247-271.
23. NASA, *Assurance Process for Complex Engineering*, accessed on August 7, 2013 at http://www.hq.nasa.gov/office/codeq/software/ComplexElectronics/
24. S. Nousala, A. Miles, B, Kilpatrick, W.P. Hall, "Building Knowledge Sharing Communities Using Team Expertise Maps", **International Journal of Business and Systems Research**, 3 (3), 2009.
25. S. Nousala, A. Moulet, B. Hall, A. Morris, "A poly-disciplinary approach: A creative commons for social complex adaptive systems", Book of Abstracts, European Conference on Complexity Systems (ECCS 2012), p 79, 2012.
26. J. O'Grady, **Systems Requirements Engineering**, Academic Press, 2010.
27. J. Park, S. Ram, "Information Systems Interoperability: What Lies Beneath?", ACM Transactions on Information Systems, 22 (4), October 2004.
28. R. S. Pressman, **Software Engineering: A Practitioner's Approach**, 6th ed., McGraw-Hill, 2005.
29. R. Santoro, A. Braccini, P. Santoro, "KBS Principles Applied to Co-creation of Smart Conference and Communities: A Case Study", 19th ICE & IEEE-ITMC International Conference, The Hague, Netherlands, June 2013.
30. S.R. Schach, A. Tomer, *Development/Maintenance/Reuse: Software Evolution in Product Lines*, Accessed on August 6, 2013 at https://www.research.ibm.com/haifa/info/ple/papers/evolution.pdf
31. L. Solow, B. Fake, **What Works for GE May Not Work for You: Using Human Systems Dynamics to Build a Culture of Process Improvement**, CRC Press, 2010.
32. C.J. Stettina, E. Kroon, "Is There an Agile Handover? An Empirical Study of Documentation and Project Handover Practices Across Agile Software Teams", 19th ICE & IEEE-ITMC International Conference, The Hague, Netherlands, June 2013.
33. Technik, *Requirements Analysis & Engineering Management*, Accessed August 7, 2013 at http://www.technikinc.com/services/requirements Archived at http://www.webcitation.org/6Ij8vARbo
34. S. Umpleby, "Cybernetics as a Language for Interdisciplinary Communication", Plenary Keynote Address, 10th World Multi-conference on Systemics, Cybernetics and Informatics (WMSCI 2006), accessed on August 6, 2013 at http://www.bing.com/search?q=+second+order+science&src=IE-TopResult&FORM=IE10TR
35. Wikipedia, *Interoperability*, accessed on April 27, 2012 at http://en.wikipedia.org/wiki/Interoperability April 2012.
36. Wikipedia, *Science 2.0*, accessed on August 7, 2013 at http://en.wikipedia.org/wiki/Science_2.0

Figure 1: Two Views of Second-Order Science

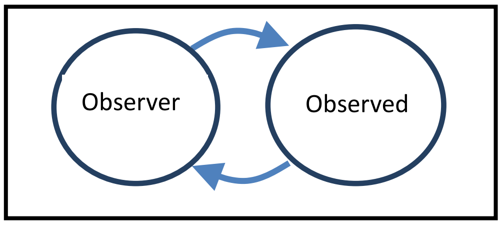

Figure 1a: Standard View

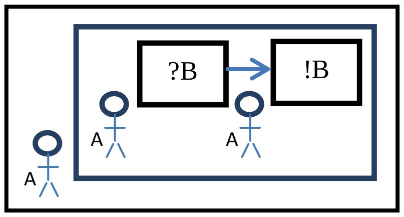

Figure 1b: Focus on actor and context

Figure 2. The Client and the Developer: Assignment

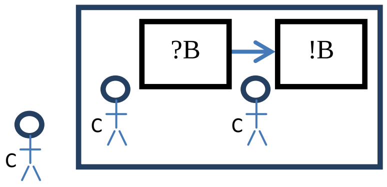

Figure 2a. Client definition

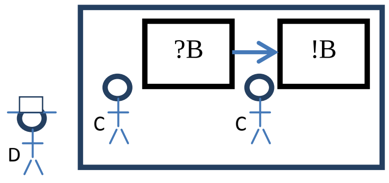

Figure 2b. Request for Proposals

Figure 3. The Client and the Developer: Solution

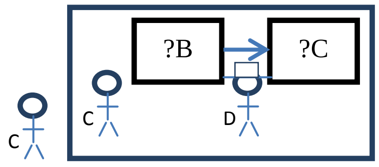

Figure 3a. Handoff to Developer

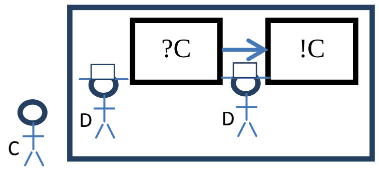

Figure 3b. Developer Solution

Figure 4: The Client and the Developer: Delivery

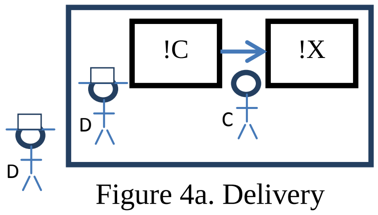

Figure 4a. Delivery

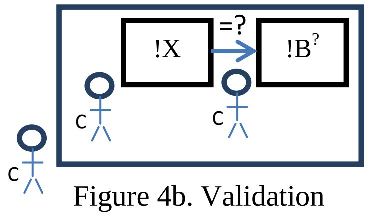

Figure 4b. Validation

Figure 5: The User: Deploying the Application

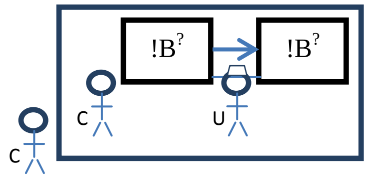

Figure 5a. Deployment

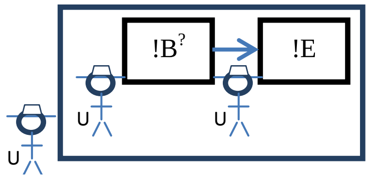

Figure 5b. Use