GROUNDING AND MAKING SENSE OF AGILE SOFTWARE DEVELOPMENT

Mark Woodman, Aboubakr A. Moteleb
Middlesex University e-Centre, School of Engineering & Information Sciences,
The Burroughs, Hendon, London NW4 4BT
m.woodman@mdx.ac.uk, a.moteleb@mdx.ac.uk

Keywords: Agile Software Development. Sense-making. Grounded Theory. Information Systems Methods.

Abstract: The paper explores areas of strategic frameworks for sense-making, knowledge management and Grounded Theory methodologies to offer a rationalization of some aspects of agile software development. In a variety of projects where knowledge management form part of the solution we have begun to see activities and principles that closely correspond to many aspects of the wide family of agile development methods. We offer reflection on why as a community we are attracted to agile methods and consider why they work.

1 INTRODUCTION

This is a short exploration of philosophical and methodological underpinnings of Agile Software Development. We offer it as a contribution to deeper understanding of agile approaches. We propose no new method or critique of an existing one. Our examination of what may be the basis of agile approaches has helped us in our work and may help others – or may merely entertain and intrigue.

A purpose of this enquiry is to try to join up previously unconnected concepts, which, for us, have gained our attention by a fortuitous collocation of a set of commercial R & D projects with traditional academic research: at our centre we support academic research by consulting to business. We have been engaged in forms of agile development, while researching fundamental issues in knowledge management, service-oriented architectures and methodologies for such research. During these projects we began to encounter research ideas that appear to make contributions to the understanding of the philosophical underpinnings of agile methods and to why the human-centric, iterative, incremental production and deployment of software has such a profound effect.

Two areas of work to do with knowledge and systems for organizing and utilizing knowledge appear to match ideas in the pragmatic approaches of the various agile methods. The relationship between agile development and lean manufacturing has received considerable comment and discussion, especially in their common goals of providing the client with what they want, in emphasizing actions that result in value, continuous improvement of the product, faster time-to-market, and avoiding waste (e.g. associated with overblown processes). It is usual in our industry and discipline to look for antecedents in other areas; we naturally look for reassurance, validation and fundamental ideas to extend in our own way. No doubt, the comparisons have helped in pragmatic terms. However, little insight has been offered as to why in terms of accepted concepts agile software appears to be applicable and to work. The two areas that we have found contribute to such thinking are Grounded Theory, a long-accepted expression of how people develop an articulation (a "theory", often as a model) of how something is or should be, and a Cynefin, a "sense-making framework", which explains behaviours, decision-making and practices in terms of people's patterns of multiple experiences, personal, cultural and business-based. We believe that these two areas can help us understand what underpins agile development.

We begin by reflecting on what is going on when we develop and deploy new software (or indeed make major changes to an existing system) in terms of software as models and in terms of people experiencing change. Next we describe some of what appeals to us about agile software development – those aspects of XP, Scrum, DSDM, etc. that just seem to be right or just seem to work. In subsequent sections we briefly summarise the main points of

tools from other disciplines: Grounded Theory and the Cynefin sense-making framework. These, or something very like them appear to be underpinning agile methods.

2 CONSTRAINED MODELS OF A POSSIBLE FUTURE

We begin by considering some high-level notions of what is happening when, typically, a client engages a developer to design, implement, deploy and operate (at least initially operate) a system to support a chunk of the client's business, ultimately business with its customers. We make very general remarks here that should not be taken to abstract some specific approach to software development. Also, we will tend to use the term "organization" and "business" without regard to whether there is a financial-profit motivation for such entities.

What is really going on when a business decides it needs a new IT system and commissions a developer to build and deploy the system? First the business will have identified a need. It does not matter how poorly researched and costed or vaguely stated the need is, the business, usually a person with a vision of the future business, has said it must have an IT system, α. Implicitly what is being said is that the business wants to have changed from its current situation, K, to a new situation, P, and to do so it needs an α. So, wanting a new system, wanting α, means wanting to change – wanting a new business based on the current business.

All IT systems, especially their software component, represent a model of the business and processes of an organisation. Imagine a range of business models for a firm, each model being represented by a letter of the alphabet. If a mature company operating according to model K, wants to move from K to P by introducing α, the essential elements for making P a reality must be captured in the system α. In other words, α must provide, as best as possible within time and budget constraints, an essential model (McMenamin & Palmer, 1984) of the business P (not of the original business K). So, whatever methods software developers use to work out what model captures the essence of P, whether they use SSADM, OOAD, XP, Scrum, software will be implemented and deployed to represent just some part of the world of the anticipated, future business P, even though it was first envisaged in the world of business K. In other words: as people in our community know, but sometimes don't talk enough about, there is a lot going on when software is being developed. Pretending we know exactly how a business system will turn out is at best being optimistic, and getting from the starting point (K, as a business was) to the end point (P, as the business wants to be) is often very, very tricky.

There are systems, often dominated by the mathematics of science or engineering, whose requirements can be fully stated in advance of design, coding and testing (e.g. various forms of control systems) but IT systems that are concerned with business or complex organizational behaviour are rarely of this sort. So, what frequently happens is that on the way from K to P, the business visionaries typically realise that P, is not where they want to end up at all. At some point it is realised that Q is the place to aim for, then R seems the obvious end point, and so on. Agile software developers know this and act to accommodate the inevitable change. The question is why is it so clear in agile development and so concealed in so-called traditional approaches.

However, the development process is not just about getting from one model of some part of a business to another. The interactions with people are profound and complex. Often these disrupt a business because the change represented in an IT system is unwelcome by many of those affected.

The commissioning, development, deployment, operation and use of software-intensive systems means change to a business – regardless of whether the introduction of a new system, or major modifications to an existing system are to take place. This is again because the IT system is a model representing the business and processes of an organisation, albeit a grossly simplified, possibly distorted model. This can be seen even in the most simple situation. For example, if a successful (non-chain) Main Street retailer introduces a system to monitor and manage inventory levels, it will be because of a reason – maybe the retailer has realised that too much of its capital is tied up in stock. If the situation were acceptable no change would be needed: the need for change gives rise to the IT system. To develop or procure a system to manage inventory will require some model of what the business is going to be (after the introduction of the new system). However, even if those promoting the change understand it, members of an organisation that are affected by it may not really understand the motivation for change.

One explanation of this type of situation characterises work environments in terms of "ordered dimensions" and "unordered dimensions". Change, including that enabled or accelerated by IT, shifts the ordered dimensions of an environment so that they become unordered. In ordered dimensions people function within known and/or predictable environments; in unordered dimensions they function within chaotic/unpredictable environments

(the effects of which can be magnified by personal change). In response to a chaotic/unpredictable environment people seek to identify new patterns of behaviour to follow, make sense of it depending on previous experience and knowledge, and respond to it by finding new order. IT systems tend to make all this harder, because for most people they obfuscate the patterns, prevent recognition of relevant experience and knowledge, and block responses to order. These problems cannot be managed away, in the sense of logistical optimisation, but require the management of the behavioural responses, which means truly involving people.

A complementary view comes from game theory. We can consider the introduction and deployment of a new IT system as a move in a zero-sum game or, for some business situations, as an attempt to move to a new Nash equilibrium point (Nash, 1950). Either case represents a change of business context in which an organisation's workforce can become uncertain of the game they are to be engaged in. A new or revised IT system becomes the embodiment of the business change, and so is resented or rejected by users who still have the previous context. This notion would explain the multiple changes of direction that occur when trying to sort out what a future business is supposed to be and what its IT system is supposed to be.

3 THE APPEAL OF AGILE DEVELOPMENT

Since Kent Beck first revealed the ideas of Extreme Programming (Beck, 2000), as a community we have been reflecting what it means to be agile and what approaches to software development can be blessed with the term "agile". Cockburn (2001), Highsmith (2002) and others have helped make it a broad church and helped emphasise the human situations in which software is developed. Many have enumerated advantages to the agile approach.

The Agile Alliance's Manifesto and Principles set out a range of priorities and beliefs about agile development that have struck a chord in the last few years. They are often articulated by the foremost proponents as a reactions to the failures of "traditional", heavy-process oriented ways of developing software. Without too much hard-selling, the promise of agile methods seems to be recognized as being somehow right: is that because people feature so much, or because of the intention to improve earned value and build functionally correct systems? We believe it is because agile advocates are helping us to make sense of what we know works and keeps us grounded with respect our business priorities.

Take the very name: clearly a software development approach must, to be classed as agile, be flexible. It must impose few barriers to changing what has already been done to something else. It has been accepted for decades (Boehm *et al.*, 1975) that the cost of changing an IT system increases the further away from coding towards requirements capture the change is needed. Because it is difficult or even impossible to say what is needed in modern business environments, and because the business owners may have no facility for understanding textual or graphical representations of needs, constraints and effects, agility is achieved by a commitment to fast coding and testing so that business owners can concretely experience what is being built and make adjustments, as the IT system is being built. An apposite metaphor is that of sailing: the owner of a yacht may want to get from point K to point P by a certain time, but the crew may not be able to go via a route the owner has proposed. Rather than sit down and plan everything in advance regardless of a changing environment, it is better for small adjustments to be made and checked with the owner (and, as we have discussed, the ultimate destination may be Q and not P).

With relatively few exceptions, software releases in agile development are meant to be of business value. To be deemed as such, by definition, means that a representative of the client must be involved enough in the development activities to know whether or not a release is of business value. This is a crucial feature that helps to keep agile projects grounded in the client's business. Agile software developers implicitly promise not to confuse a project or change programme with their own agendas. The reason for a project is the client's business, so the essential model of where that business wants to be belongs to the business, not the developer. Consequently, if after a particular release the client announces that the system is good enough for what the business needs that should be the end of a project. Making this option a reality for clients through review/reflection has the appealing benefit of engendering trust between developer and client.

The close involvement of the client also brings indirect closeness to their business's end-user. Often the client's staff make up the end-user community, but often they must act as a proxy for the end-users. While indirectness is not ideal, it is hugely better than developers guessing how end-users might behave or forcing them to behave in a particular way. Again, client involvement means keeping control with the client, but in a way that does not diminish the skills or responsibilities of developers.

One aspect of introducing IT systems for business advantage (with change) is what might be

termed the ethical dimension – the extent to which negative impact on people by IT systems in changed environments is acceptable. Again, agile approaches to software development tend to be better in that deep client involvement in the small steps on the journey signalled by multiple software releases allows such issues to be recognized and factored into ongoing decisions. Of course, there are considerable benefits to be had by involving people whose whole involvement in an enterprise is at an operational level, e.g. as espoused by lean manufacturing or other forms of worker participation (see Ehn, 1988).

In summary: agile development keeps a project firmly located in the client's world and helps us make sense of complexities.

4 GROUNDED THEORY AND AGILE DEVELOPMENT

In this section we will explore some of the essential philosophy of the Grounded Theory methodologies and where specific methods bring out different aspects that are relevant to our views. For now we stick to the terms from this area so as to be clear later about the differences between agile development and grounded theory.

Grounded theory was first described in 1967 by Barney Glaser and Anselm Strauss. Basically, the term describes methodologies within which a researcher uncovers a "theory" on a matter in hand by repeatedly conversing with those who know about the matter and analysing the result to take it to the next conversation. The intention of grounded theory is "to generate or discover theory, an abstract analytical schema of a phenomenon, that relates to a particular situation" Creswell (1998). In this situation, individuals engage in a process by acting and interacting within a phenomenon. Researchers study this engagement through collecting data, developing and interrelating categories of information, writing theoretical propositions, and validating them against further data collection

The main philosophy behind the original version of grounded theory is the generation of "theory" from data. This can be in the form of a model (which makes it directly relevant to software-intensive systems). A grounded theory is *derived* from data, systematically gathered and analysed through a defined research process (Glaser & Strauss, 1967; Glaser 1978, 1992; Strauss & Corbin 1990, 1998). Grounded theory-based research, in this sense, differs from other research methodologies mainly in that it is concerned with constructing theory rather than testing pre-formulated theory (or testing a hypothesis on which a theory might depend). In addition, this theory is derived from "real data" rather than from a "logico-deductive" speculation (De Vaus, 2002; Glaser & Strauss, 1967) – which clearly fits with understanding what a client wants and needs rather than forcing a solution.

Over the history of grounded theory two distinct camps have emerged behind different philosophies ascribed to the original proponents. Strauss and his later co-worked Corbin were accused of "forcing" theory from data, rather than letting it "emerge" (Glaser, 1992). The emergent philosophy resonates with our experience, so we concentrate on it here.

According to Glaser (1992) "grounded theory is a general methodology of analysis linked with data collection that uses a systematically applied set of methods to generate an inductive theory about a substantive area". More prosaically he also describes it as "a general method to use on any kind or mix of data" (Glaser, 1998). In short, researchers should use a grounded theory approach to scientifically work out what is going on in a particular situation demanding a systematic approach to analysis of data that is mostly based on recorded observations, conversations and interviews. Crucially the understanding of what is going on, the theory, must be grounded in the data and emerge from it: the researcher must not impose their worldview or understanding based on other knowledge or experience. (By the way, we use the term "methodology" to refer to a family of particular methods that adhere to the general principles of grounded theory but differ in detail according to the situation that we are trying to understand.)

Grounded theory methods are iterative (obviously a similarity with agile methods). Researchers using a grounded theory method gather "data" (in the social-sciences sense) through conversations, interviews, etc., then analyse the data in a systematic way, form a putative theory and take that into the next data gathering activity, and thence into the next analysis activity, and so on.

The detail of the method and analysis is not needed for the comparisons we want to make, so we will sketch them very briefly. Since understanding should emerge from the data, the researcher's role is to uncover categories, concepts and properties with the relationships among them. The first stage in an analysis is "open coding". This is essentially about identifying, naming, describing and categorizing what has been found in the data. (Nouns and verbs and instances of categories are explored, much as in an object-oriented analysis.) The properties (attributes in OOA) of categories are also discovered in this stage. Much of this is done informally; you don't have to worry too much about backtracking because subsequent data gathering and analyses will pick up anomalies.

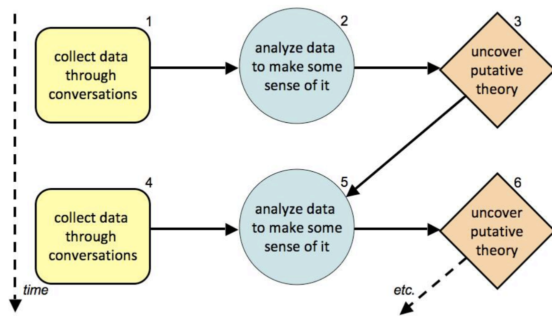

Figure 1: Execution of Grounded Theory Inquiry.

Next comes "axial coding". This activity relates categories and properties to each other. An emphasis is placed on causal relationships and a framework of generic relationships including intervening conditions, action strategies and consequences. Finally in the analysis activities is "selective coding" in which one of the categories is chosen as the core and all others related to it. This provides a single story line for fitting everything else to – in essence, a putative theory of what is going on in the situation being explored.

The process is then repeated, with new data gathering and analysis that takes into account the previously derived putative theory. The cycle stops on "theoretical saturation", i.e. when newly gathered data or newly performed analysis can't add anything to the emergent theory. The general approach is depicted in Figure 1. The cycle is repeated until there is no further benefit to be gained from it.

As can be imagined, in a grounded theory method, fragments of the whole, emerging theory are constantly being moved around in relation to each other. As the core is being sorted out and possibly revised, the various parts of the theory are put next to each other or made distant as the relationships between the parts are sorted out.

So, what is the possible relationship between grounded theory and agile software development? First, there is an obvious parallel between the iteration in both and the intention of being "grounded" in a potentially complex situation. But the relationship goes much deeper and touches on the issues raised earlier.

Let's look at the iteration aspect. Although iteration and feedback/feed-forward are part of what both involve, the iteration could be seen as merely a practical means to uncover a model. For an emergent model to be uncovered is clearly the priority – the ultimate "theory" from a grounded theory method, or the delivered system in an agile development. But, the iteration is much more than a mechanism for convergence; the discrete stages allow a reconsideration of direction (in the case of software development for business advantage) or consideration of new data (in the case of grounded theory). As in the sailing metaphor, a change of tack can help get to a prescribed goal despite uncontrollable conditions, or may use changing conditions to proceed to an unanticipated end-point.

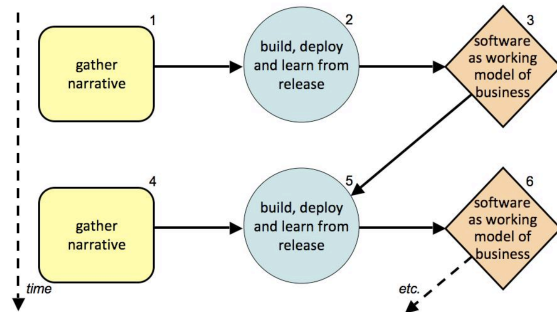

Figure 2: A simplified view of agile development.

Another similarity is to do with the development of a model (a.k.a. theory) in both cases via *construction* rather than via *design*. In a grounded theory method, data from observations (from conversations, etc.) is initially obtained and analysed to construct a putative theory; subsequently new data is obtained and analysed with the knowledge of the previous version of the theory. A system developed in an agile fashion could be similarly described. The gathering of data is replaced by story telling (as in extreme programming, XP) or by other informally articulated narratives that describe business situations. The analysis (theory construction) is replaced by the building of a software release, and knowledge of the release is available in the next part of the cycle. A grossly simplified representation is given in Figure 2.

A grounded theory cycle is repeated until there is no further benefit to be gained from another iteration. It should be the case in agile methods that the production of software releases stops when there is no further business value to be gained – completion of an agile development should be exactly analogous to theoretical saturation. Hence, we have taken the feedback/forward aspect of agile development comfortably into grounded theory, devising and beneficially using a variant that included the released software as part of the emergent theories. This supports our view that grounded theory may be seen as underpinning agile development.

5 THE CYNEFIN FRAMEWORK AND AGILE DEVELOPMENT

We now explore "sense-making". It is a methodology that was first developed in the early 1970s, and so has a long pedigree. It is fundamentally about communication between

humans. Based on considerable evidence its proponents have concluded that the dominant models of communication and information systems do not work.

Sense-making is a "methodology disciplining the cacophony of diversity and complexity without homogenizing it" (Dervin, 1998). According to Dervin, there are three main assumptions:

1. That it is possible to design and implement communication systems and practices that are responsive to human needs.
2. That it is possible for humans to enlarge their communication repertoires to pursue this vision.
3. That achieving these outcomes requires the development of communication-based methodological approaches.

Like grounded theory, sense-making generates "theory". However, concepts emerge from ambiguous interactions and communication within a situation, rather than from more distinct steps like in grounded theory. Sense-making identifies patterns within a complex (unordered) system.

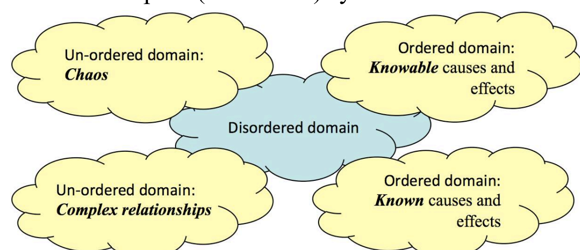

Figure 3: Cynefin "domains".

We have been struck by the useful analysis of Kurtz & Snowden (2003) who have proposed the Cynefin sense-making framework to help make sense of complex systems. The word is Welsh, which means "habitat", but more richly includes notions of the multiple experiences that people have in aspects of their lives. These experiences are a complex mixture of, the personal, the wider cultural, and the business-based or workplace-based. Cynefin is based on the notion that "humans use patterns to order the world and make sense of things in complex situations". Cynefin originated in the practice of knowledge management with the aim of helping managers to "break out of old ways of thinking and to consider intractable problems in new ways".

One of the most striking aspects of Kurtz and Snowden's ideas is that they divide situations into what they call "domains" as in Figure 3. The nature if these domains is very different: the right-hand domains are those of order, i.e. *known* and *knowable* cause and effects, whereas the left-hand domains are those of un-order, i.e. *complex relationships* and *chaos*, and in the centre is the domain of disorder. These domains help with understanding different possible situations in development, as follows.

Known causes and effects: Kurtz and Snowden argue that in this ordered domain repeatability allows for predictive models to be created, because cause-and-effect relationships are "generally linear, empirical in nature, and not open to dispute". Their model in this system state is based on (a) sensing incoming data, (b) categorising that data, and then (c) responding in accordance with predetermined practice. In this domain, knowledge is explicit and can be captured and embedded in structured processes to ensure consistency, through artefacts such as field manuals and operational procedures.

This is where IT systems that depend heavily on known physical properties and behaviours are based. Medium/heavy-weight processes with an emphasis for documentation may be appropriate here.

Knowable causes and effects: according to the Cynefin framework in this ordered domain "entrained patterns" allow for structured models based on assumptions, because cause-and-effect relationships may not be fully known, or may be known only by a limited group of people. Everything in this domain is capable of movement to the known domain depending on affordances. Their model in this state of the system is based on (a) sensing incoming data, (b) analysing that data, and then (c) responding in accordance with expert advice or interpretation of that analysis. In this domain, knowledge is tacit, yet to be "externalised" from experts, or people who possesses it. Kurtz and Snowden argue that this is the domain of "systems thinking, the learning organization, and the adaptive enterprise, all of which are too often confused with complexity theory".

Our experience and use of Cynefin-like thinking makes us believe that agile methods are of huge use here. A characteristic behaviour of this domain is sense-analyze-respond, which corresponds closely to grounded theory and to agile development with its attendant short-cycle releases.

Complex relationships: in this un-ordered domain "emergent patterns" can be perceived but not predicted, because while cause-and-effect relationships exist between "agents", "both the number of agents and the number of relationships defy categorization or analytic techniques". Their model in this state of the system is based on (a) probing to make the patterns or potential patterns more visible, (b) sensing those patterns, and then (c) responding by stabilizing those patterns that we find desirable, by destabilizing those we do not want, and by seeding the space so that patterns we want are more likely to emerge. In this domain, knowledge is embedded in multiple perspectives of the system. Different narrative techniques such as story telling are proposed to capture these perspectives.

The dominant pattern in this domain is probe-sense-respond. The lack of explicit analysis makes it an unsuitable comparator for agile development. However, incremental, value-enhancing releases could be taken to be probes and analysis an implicit part of response. To be crystal clear about the use of agile development in these domains would take further exploration of this sense-making framework, possibly leading to a variation of it. Our gut feeling is that at least near the boundary of the knowable, ordered domain, agile approached will work.

Chaos: in this un-ordered domain there are no perceivable relations, because the system is turbulent. Their model in this state of the system is based on (a) acting quickly and decisively, to reduce the turbulence; and (b) sensing immediately the reaction to that intervention, then (c) responding accordingly. In this domain, knowledge cannot be captured or perceived until the system moves to one of the previously mentioned domains. However, according to the Cynefin framework chaos is a domain for innovation, thus we can intentionally enter it to create the conditions for innovation.

The domain of disorder: Kurtz and Snowden state that "the central domain of disorder is critical to understanding conflict among decision makers looking at the same situation from different points of view". People tend to pull "disorder" towards the domain where they feel most empowered by their own capabilities and perspectives. In the Cynefin way of thinking "the reduction in size of the domain of disorder as a consensual act of collaboration among decision makers is a significant step toward the achievement of consensus as to the nature of the situation and the most appropriate response".

In these last two domains other approaches are needed. We believe that a method (we have used grounded theory in this respect) can help an ordered part of a chaotic situation to emerge with which agile works well. Having used Cynefin to choose what type of software development approach to use with clients, we see powerful commonalities between it and practices in agile development.

4 CONCLUSIONS

Those of us who work in uncertain, unpredictable business situations, either know or are prepared to believe that agile methods of software development work. Usually such knowledge is enough. Fast-changing situations demand pragmatic action rather than leisurely, scholarly reflection. However, to understand is to be capable of improving or adapting. This is why we have conjectured as we have – that the basically human-centric practices of agile software development are remarkably similar to those of both grounded theory and the Cynefin sense-making framework.

The relationship between agile development and grounded theory is fundamentally to do with being firmly situated in the problem being dealt with – e.g. a business IT systems for agile development. Both approaches construct models for use and to further deepen the understanding of a problem. Both are iterative and aim to converge, and work by committed stakeholder involvement.

The relationship between agile development and the Cynefin sense-making framework is fundamentally to do with how the framework provides a language and thinking tools for determining where agile development is appropriate and when it is not. We believe agile methods have little use in ordered, known domains, in unordered, chaotic domains and in disordered domains. They are probably highly effective in ordered, knowable domains, but greatest benefit may be along the border between the knowable and chaotic, where successful entrepreneurial businesses may operate.

Whatever the accuracy of our conjectures, we believe that our comparisons deepen the understanding of the efficacy of agile development.

REFERENCES

Boehm, B.W., McClean, R.K. & Urfrig, D. B. 1975. Some Experience with Automtaed Aids to the Design of Large-Scale Reliable Systems. *IEEE Trans. on Software Engineering*, vol. 1, no. 1, pp. 125–133.

Cockburn, A. 2001 *Agile Software Development,* Addison-Wesley, Reading, Massachusetts.

Creswell, J. W. 1998. *Qualitative inquiry and research design: choosing among five traditions.* Sage.

Dervin, B. 1999. On studying information seeking methodologically: the implications of connecting metatheory to method. *Information Processing and Management*, vol. 35, no. 6, pp. 727–750.

Ehn, P. 1988. *Work-Oriented Design of Computer Artifacts*. Stockholm, Sweden: Arbetslivscentrum.

Glaser, B. G. & Strauss, A.L. 1967. *The discovery of grounded theory: strategies for qualitative research*. Aldine, Chicago.

Glaser, B. G. 1978. *Theoretical sensitivity: advances in the methodology of grounded theory*. Sociology Press.

Highsmith, J. 2002. *Agile Software Development Ecosystems*. Addison-Wesley, San Francisco.

Kurtz, C. & Snowden, D. 2003. The new dynamics of strategy: Sense-making in a complex and complicated world, *IBM Systems J.*, vol. 42 no. 3, pp. 462–483.

McMenamin, S. M. & Palmer J. F. 1984. *Essential Systems Analysis*. Yourdon Press.

Nash, J. 1950. "Equilibrium points in n-person games". *Procs. of National Academy of USA* 36(1). pp. 48–9.