Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2004 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2004

Theory of Complex Adaptive Systems and Agile Software Development

Radhika Jain
Georgia State University

Peter Meso
Georgia State University

Theory of Complex Adaptive Systems and Agile Software Development

Radhika Jain
Department of Computer Information Systems
Georgia State University, Atlanta, GA 30303
rjain@cis.gsu.edu

Peter Meso
Department of Computer Information Systems
Georgia State University, Atlanta, GA 30303
pmeso@cis.gsu.edu

ABSTRACT

Recently academicians as well as practitioners have shown growing interest in agile software development methodologies. These methodologies refer to the development processes that are lighter and faster developed in response to uncertain and volatile market environment. Much of the prior literature does not provide any theoretical basis for these methodologies. In this paper, we demonstrate the use of various characteristics exhibited by the complex adaptive systems (CAS) to understand how agile software development organizations function and organize themselves to accomplish the development task.

Keywords

Agile software development, theory of complex adaptive systems, complexity theory

INTRODUCTION

One of the problems of structured software development is the lack of theories that can describe and predict 1) the behavior of information systems developed via structured software development approaches, and 2) the tactical fit of structured software development approaches within the larger organizational context where such systems are deployed and used. As the organizations evolve over time, the information systems that they depend on also need to evolve dynamically. This need for evolution arises due to the complex interactions among internal and external business processes and the stakeholder entities participating in these processes. Not only does the firm structure and its evolution influence its information systems, but vice versa also holds true.

Given the unpredictable, dynamic and emergent business-process requirements that arise due to the now omnipresent uncertain and volatile market environment within which organizations operate, means to effectively evolve their information systems are needed. Recently agile software development methodologies and various practices espoused by them have gained significant momentum for developing software in such environments (Refer to section 3 for detailed discussion). Although these methodologies are primarily being used in the volatile world of Internet and web software development, there appears to be a growing interest in extending their application for large, critical software projects (Glass, 2003; Paulk, 2001; Reifer, Maurer and Erdogmus, 2003). While agile methodologies have proved popular and effective response mechanisms, there remains an absence of theory on how to apply them to improve the tactical fit between innovative software-solution development and strategic software-solution use.

In this paper, we argue that the theory of complex adaptive system (CAS) provides a useful theoretical framework for understanding the dynamic nature of business requirements in the modern day distributed and knowledge-intensive organizations. The present day information systems can be perceived as complex-adaptive dynamic systems that exhibit both unpredictability and underlying order. The paper concludes that such an understanding provides for a more grounded theoretical underpinning for the use of agile software development approaches in developing information systems.

The paper is structured as follows. Next section introduces the theory of CAS and how it is applied to organizations. Section 3 briefly describes various agile methodologies. In Section 4, based upon the review of various articles we catalog the commonly followed agile practices and discuss how various CAS characteristics can help explain these practices. Finally, in section 5 we conclude with the research contributions and the future research avenues.

THEORY OF CAS AND SOFTWARE DEVELOPMENT

The study of CAS is based on the assumption that there is more to a system than the sum of the components and the linkages that compose the system. Systems show properties that are truly emergent, irreducible to explanations that take into account only lower level components' properties. The general characteristics of CAS can be summarized as follows (Cilliers, 2000):

1. Complex adaptive systems consist of many different parts (that in themselves can be simple and are highly interconnected and interactive) which by a process of self-organization become more ordered and informed than the system that operates in approximate thermodynamic equilibrium with their surroundings.
2. These parts interact dynamically by exchanging energy or information. These interactions are rich and nonlinear. Even if specific elements only interact with a few others, the effects of these interactions are propagated throughout the system.
3. The behavior of the system is determined by the nature of these interactions, not by what is contained within the components. Since the interactions are rich, dynamic, nonlinear, and are fed back, the behavior of the system as a whole cannot be predicted from the inspection of its components. The notion of "emergence" is used to describe this aspect.
4. There are many direct and indirect feedback loops.
5. Complex adaptive systems are open systems. They exchange energy or information with their environment and operate at conditions far from equilibrium.

Certain systems may display some of these characteristics more prominently than others. Like complex adaptive systems, organizations have large numbers of independent yet interacting actors. The notion of complexity has been applied to organizations in a number of different ways, and with varying degrees of rigor. Its application to organizational setting (Cilliers, 2000) suggests that

1. Since the nature of a complex organization is determined by the interaction between its members, relationships are fundamental.
2. Unpredictable and novel characteristics may emerge which may or may not be desirable, but by definition they are not an indication of malfunctioning. For example, a totally unexpected loss of interest in a well-established product may emerge. Novel features can, on the other hand, be extremely beneficial. They should not be suppressed because they were not anticipated.
3. Complex organizations cannot thrive when there is too much central control. This certainly does not imply that there should be no control, but rather control should be distributed throughout the system. One should not go overboard with the notions of self-organization and distributed control.
4. Complex organizations work best with shallow structure. This does not mean that they should have no structure.

Thus rather than ever reaching a stable equilibrium, the most adaptive of these organizations keep changing continuously and remain at the edge of chaos that exists between order and disorder. By staying in this intermediate zone, these organizational systems never quite settle into a stable equilibrium but never quite fall apart while simultaneously adapting to continuously occurring changes.

Proponents of complexity theory enthusiastically see signs of it everywhere (Levy, 1994), pointing to the ubiquity of complex dynamic systems in the natural and engineered world. From a theoretical perspective complexity theory is congruous with the postmodern paradigm, which questions deterministic positivism as it acknowledges the complexity and diversity of experience. While postmodernism has had a profound influence in many areas of the social sciences including several business disciplines such as strategy, marketing and finance, it has been neglected by the software developers until very recently (Levy, 1994; Papazoglou, 2001; Sutherland and Heuvel, 2002)

Though not intending to introduce the notion of complexity theory, Meyer and Curley (1991) developed a matrix for taxonomizing knowledge-based systems based on their technological and knowledge complexity (Refer to Figure 1). Perhaps unbeknownst to them, the framework they introduced provided a view into the future evolution of almost all information systems. With the advent of the object model, distributed ubiquitous computing infrastructures such as the internet, and the standardization of component and protocol architectures, most of present-day's information systems are significantly complex to be classified into quadrants II, III and IV of their taxonomy.

A close examination of these systems suggests that the complex nature of these systems is much more than the result of their complexity as indicated by number of modules or lines of code (technical complexity) and extent of expert knowledge they embed (knowledge complexity). A larger degree of their complex nature comes from the dynamic emergent behavior of these systems, which is the result of the non-linear interrelations and interaction of its constituent parts with the uncertain and volatile social-technical business landscape. Complexity theorists refer to this type of behavior as complex-adaptive and the systems that exhibit it as being complex-adaptive systems. While much of system development research has adapted Meyer

and Curley's view of information system complexity, it has ignored the complex-adaptive behavior of such systems. This may explain the lack (until very recently) of association between theory of CAS and study of software artifacts.

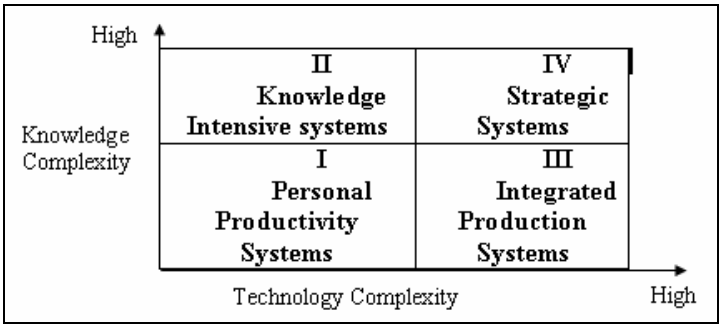

Figure 1: Taxonomy of Information System Complexity (Meyer and Curley 1991)

Though there is little published research on the application of CAS theory to software development methodologies, there is increasing consensus that they inform the shift towards iterative and agile software development methodologies. For example, Eoyang (1996) describes how complexity theory helped unravel the complicated issues that threatened to stymie a customer support system's GUI development at MegaMoney Inc. In providing an argument for a complex adaptive view of software development he states "Our old measures of product quality and process efficiency are always unrealistic and frequently destructive when they are applied to development of complex, adaptive software products… we must change the way we look at the world of software development and business processes." Eoyang (1996) lists the factors that necessitate the use of agile software development methodologies as being evolving and diverse business processes, user-centered design, distributed computing platforms, and intricate yet technologically diverse graphical user interfaces. In another example, Martin Gerber (2002) states "…by incorporating the principles of self-organization called isomorphies, new methodologies… including object-orientation, extreme programming (XP), and lightweight methodologies use some concepts and principles of natural complex systems …that's why they produce better results than conventional methodologies if they are applied in appropriate areas…" Highsmith (1999) in his book on Adaptive Software Development suggests emergent order as an alternative to a belief in and a dependence on imposed order. He introduces three major models: the Adaptive Conceptual model, the Adaptive Development model, and the Adaptive Management model. A central theme of emergent order links all these three adaptive models together.

Demand and development of complex, adaptive software processes is driven by the need for complex software systems that are capable of meeting the evolving needs of its users in highly interactive and high-velocity business environments. Therefore we posit that the theory of complex adaptive systems may be suited to explaining the agile approaches for developing and maintaining information systems.

AGILE SOFTWARE DEVELOPMENT METHODOLOGIES

To bring about some discipline to chaotic process of software development, many development methodologies are proposed and used in the practice. These methodologies impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. Agile methodologies such as Extreme Programming (XP) and traditional methodologies such as waterfall approaches represent two extreme ends in the methodology domain. Agile methodologies promote flexible software development processes that can adapt and react promptly and appropriately to changes and demands imposed by development environment (Krutchen, 2001). The development environments for the most projects adopting agile processes are highly turbulent, uncertain and dynamic environments characterized by rapidly evolving technologies, changing customer requirements, and sweeping regulatory changes (Cusumano and Yoffie, 1999; MacCormack, Verganti and Iansiti, 2001).

In addition to various "named" agile methodologies (such as Extreme Programming and SCRUM), there exist many "homegrown" agile methodologies. Despite the diversity and development practices suggested by them, agile methodologies in general are characterized by the following attributes: incremental (small software releases with rapid development cycles), cooperative (a close customer and developer interaction), straightforward (easy to learn and modify and are sufficiently documented), and adaptive (the ability to make and react to last moment changes) (Abrahamsson, Warsta, Siponen and Ronkainen, 2003). Next we briefly describe leading agile approaches.

Extreme Programming (XP)

XP is a lightweight process targeted at development project that are ill-understood and/or have rapidly changing requirements (Beck, 1999). XP empowers developers to confidently respond to changing customer requirements, even late in the life cycle emphasizing teamwork. Managers, customers, and developers are all part of a team dedicated to delivering quality software. XP can improve a software project development process in four essential ways; communication, simplicity, feedback, and courage.

Adaptive Software Development (ASD)

Rather than the optimizing focus of process improvement techniques (CMM, ISO), ASD emphasizes on producing high-value results based upon the rapid adaptation to both external and internal events. ASD targets development teams where competition creates extreme pressure (both high-speed and high-change) on the delivery process. Highsmith (1997; 1999) suggests that in this environment, "… adaptation is significantly more important than optimization…"

Feature Driven Development (FDD)

With FDD, managers know what to plan and how to establish meaningful milestones. They get the risk reduction that comes from managing a project that delivers frequent, tangible working results. FDD uniquely sees very small blocks of client-valued functionality, called features, organizing them into business-related groupings. It focuses on producing working results every two weeks and facilitating inspections. It provides for detailed planning and measurement guidance, promotes concurrent development within each "design by feature, build by feature" increment (Coad, LeFebvre and Luca, 2000).

Scrum

Scrum is a team-based approach for controlling the chaos of conflicting interests and needs to iteratively, incrementally develop systems and products when requirements are rapidly changing. It can improve communications and maximize co-operation. Scrum is scalable from small single projects to entire organizations (Rising and Janoff, 2000).

MOST OFTEN FOLLOWED AGILE PRACTICES AND THEORY OF CAS

We reviewed several articles that reported on the usage of various agile methodologies in various organizations. (Reifer, 2002) also reports a list of various practices in his survey of various organizations. However, it is not clear whether these are the most frequently observed practices. Below we report various agile practices that were used most often, based on the review of various articles that reported the usage of these methodologies. Following the brief explanation of each practice, we discuss how theory of CAS can help explain these practices.

Frequent Releases and Continuous Integration

Larman and Basili (2003) suggest that for a complex system development project to be successful it should be implemented in small steps each with a clear measure of successful achievement as well as option of rollback to a previous successful step upon failure. All the development teams followed this practice with some variation. For example, Grenning (2001) reports that when the process-intensive company they were studying launched XP, development team carried out monthly releases and continuously integrated the new features with the evolving product. Cao et al. (2004) also observed that the development team focused primarily on getting the production code that supports functionalities end-to-end rather than developing heavily integrated modules. These practices were also observed at two software giants, Netscape and Microsoft (MacCormack et al., 2001). It helped the development teams to get timely feedback on various aspects of the system. It was seen as a way to increase customer support and satisfaction (Beedle, Bennekum, Cockburn, Cunningham, Fowler, Highsmith, Hunt, Jeffries, Kern, Marick, Martin, Schwaber, Sutherland and Thomas, 2001) while simultaneously boosting the confidence of the development team and relieving a lot of pressure from the development team (Grenning, 2001).

Theory of CAS stipulates that in order to build systems in "complex" environments, one need to take a different approach. Rather than planning the entire system first and then building it, they need to be built by "grow and evolve" principle to stay at the edge of chaos. Conducting frequent releases was crucial in order to accommodate the changing business requirements, the feedback from various stakeholders as well as to demonstrate the features supported.

Need for Frequent Feedback

All the developments teams that embraced agile methodologies in some form sought extensive feedback from various stakeholders. For example, developers at Netscape and Microsoft constantly sought feedback from their external and internal

users of the products. They developed mechanisms to respond to this new stream of information being brought into the development process (MacCormack et al., 2001). The need for such feedback was considered crucial in these projects. In one case, when it was not possible to seek feedback directly from the customers, the development team had their project managers and business analysts (who had direct contacts with customers) act as the surrogate customers (Cao et al., 2004). Team sought feedback on various technical decisions, requirements, and management constraints. They also used this opportunity to demonstrate various features and functions to stakeholders who were involved in some aspect of the system being developed. The primary goal was to correct possible design errors before throwing in all resources intended for the system (Williams and Cockburn, 2003).

Constructive interactions across a boundary in a CAS are called transforming feedback loops. Change in one part of a system is transmitted via its boundaries to other parts of the system, causing these to change or react in some way. These secondary changes (reactions) are transmitted back to the originating part, causing it to change again. When a development team establishes transforming feedback loops across the boundaries that divide it from users, other development agents, and stakeholders, a complex adaptive development environment is created. The complex interdependencies allow emergent, adaptive structures to be generated by the team as a whole. The creative tension in the group generates solutions in the process and product design that are greater than the sum of the parts. The emergent patterns adapt and adjust to both the technical and business requirements of users in a complex environment. Seeking feedback from various stakeholders was considered crucial to the fate of the project. Netscape and Microsoft had heavy emphasis on the user feedback and carried out alpha and beta testing for getting feedback from their products' potential customers. To accommodate the feedback from the various stakeholders, these development teams had various mechanisms. One such mechanism was to prioritize the elements of the received feedback and incorporate them into next iterations based on their criticality.

Proactive Handling of Changes to the Project Requirements

One of the major highlights of these methodologies is they embrace requirement changes even in later phases. This was observed uniformly across all the development teams. Williams and Cockburn (2003) suggest that it is highly unlikely that any set of predefined steps will lead to a desirable, predictable outcome since the requirements and technology change constantly while people are added and taken off the team. To gain the competitive advantage in such environment, it is important that any changes to the project requirements are handled proactively (Beedle et al., 2001). At Netscape and Microsoft, systemic changes in a project's definition and basic direction were managed proactively.

Theory of CAS suggests that unpredictable and novel requirements may emerge which may or may not be desirable. These they are not an indication of malfunctioning and they should not be suppressed because they were unanticipated. A totally unexpected loss of interest in a well-established product may emerge. For example, Microsoft's decision in April 1994 to add a browser to Windows'95 and Netscape's decision to cancel a project to rewrite Communicator in Java modules (Cusumano et al., 1999). The property of emergent order exhibited by complex adaptive systems also helps explain this practice.

Loosely Controlled Development Environment

Most of the development environments for these projects were not too tightly controlled. This was seen necessary in order to be flexible and adaptable to the frequent changes demanded by the turbulent environment. Cusumano and Yoffie (1999) observed that managers at Microsoft and Netscape tried not to control the development process too rigidly in order to be responsive. Grenning (2001) also observed that by letting senior people monitor the team's progress at a monthly design-as-built review meetings accelerated development process while at the same time lowering the number of bugs. Any bugs found were made part of the next iteration. Project managers at FinApp (Cao et al., 2004) believed that deep hierarchical organizational structure could lead to unresponsive environments with high inertia. Flexible work setting, focus on results rather than micro-management, empowering the people who are actually doing the work were seen as key factors affecting developer morale and motivation (Cao et al., 2004; Cusumano et al., 1999). These observations might lead one to think about the role that project managers should be playing in agile teams. Fowler (2002) and Blotner (2003) suggest that a manager can and should help the team to be more productive by offering some insight into the ways in which things can be done in an agile environment.

Theory of CAS suggests that these complex organizations work best with shallow structures. Although most companies did have some structure, it was minimal. Theory of CAS suggests that conditions of low uncertainty are best adapted to bureaucratic or mechanistic organizations while in conditions of high uncertainty more flexible and adaptive organizations are appropriate. Furthermore, organizations cannot perform well in the turbulent and high-velocity environment if there is too much centralized control. They can flourish well when distributed decision making and control mechanisms are utilized. Leadership and decision making should be decentralized or localized. This certainly does not imply that there should be no

control, but rather control should be distributed throughout the system. For example in Microsoft, developers had veto power to exercise on technical ground. Netscape senior developers along with marketing department could assign higher priorities to their favorite features. Product managers, engineers, and developers could negotiate the terms with their clients (marketing department, for example) when things didn't go quite as per plan.

Some Planning Is Necessary Though It Is Minimal

Although agile methodologies appear to follow unplanned and undisciplined hacking development practices, these methodologies emphasize a fair amount of planning. For example in XP, iteration planning is implemented while FDD advocates detailed planning and measurement guidance within each "design by feature, build by feature" increment (Coad et al., 2000). Rather than planning, analyzing, and designing for the far-flung future (which is highly likely to change), XP exploits the reduction in the cost of changing software to do all of these activities a little at a time, throughout software development (Beck, 1999). The scope of the planning process is kept primarily to the current iteration. Only after executing and evaluating the outcome of current plan, the next set of plans is generated (Martin, 2000). Given the volatile and unstable market environment, it's impossible to specify everything that should go into a new product or a new release before coding. Managers at Netscape and Microsoft clearly believed that some planning was necessary to build the complex products (Cusumano et al., 1999).

Complex adaptive systems lay significant emphasis on the emergent order, thus accounting for minimal planning practice to accommodate various unforeseen requirement changes. Awareness of critical differences and boundaries facilitates knowledgeable evolution of the system to foster unforeseen and unpredictable volatility in the environmental demands on the organization and its information systems. Boundaries are the focal point of turbulence and fronts for emergent and possibly novel occurrences. These boundaries need not be barriers to productive work, but rather the points of constructive interaction, adaptation and learning. The principle of grow and evolve espoused by theory of CAS further helps explain existence of this practice.

Developers Skills

Agile methodologies put a heavy emphasis on the people and their talent, skill, and knowledge suggesting that for agile development to be effective team members must be responsive, competent and collaborative (Boehm, 2002; Cockburn and Highsmith, 2001). For example in XP, since the developers with different level of experience and skills are paired together they learn from each other (Cao et al., 2004; Grenning, 2001). In order to facilitate this kind of learning, communication capabilities of the developers become crucial (Cao et al., 2004).

Not much of substance can be achieved in isolation since complex adaptive systems thrive on relationships. Different units of the development team must interact with other units. If one unit seeks to exert control over the others, then the system breaks down. In most cases development teams were co-located with various stakeholders. They had heavy interactions among themselves and with various stakeholders. For these interactions to be meaningful, developers need to be competent and collaborative emphasizing skill set demanded of developers.

Emphasis on Working Software Product

Another practice that became quite apparent was the minimal emphasis on the documentation. It was motivated by two main reasons: to make the team more effective in responding to change and reduce the cost of moving information between people (Cockburn et al., 2001). Martin (2000) suggests that teams create all detailed plans that are meaningful only during the iteration and very few artifacts are created including project plans, analysis models, design models, etc. Grenning (2001) argues that a good way to protect future software maintainers is to provide them with clean and simple source code, rather than binders full of out-of-date paper and keep the documentation at the high-level so that usual maintenance changes and bug fixes don't affect it. Cusumano and Yoffie (1999) found that managers at Microsoft and Netscape tolerated incomplete documentation since they put a premium on creating working product.

A path of least effort characteristic suggests that the work of calculating a path of least effort must be included in the total work of taking a path of the least effort. Nothing is gained by calculating a particular path of least effort to a greater degree of precision, when the added work of so calculating it is not more than offset by the work that is saved by using the more precisely calculated path (Zipf, 1949). This helps us understand the practice of the minimal emphasis on the documentation while putting heavy emphasis on building the working product.

CONCLUSION

The above discussion shades some light on how theory of complex adaptive systems can provide useful insights for understanding how agile software development organizations function and organize themselves to accomplish the development task. Above discussion suggests for further research into applicability of CAS theory to software development methodologies. Some of the future research areas that we plan to investigate include development of design principles that are also rooted in the practice based on the CAS theory, empirical evaluation of these principles to establish their effectiveness and rigor, and the identification of various metrics to assess product quality and process efficiency based on CAS theory. The ultimate goal of this research is the development of a design theory that is applicable to not only agile software development but also in general to any software development efforts. As Highsmith (1999) points out succinctly, without the clear understanding of underlying theoretical principles behind software development approaches, organizations are at the high risk of being non-adaptive. Clearly the first step towards these is the appropriate mapping of various agile practices and the CAS characteristics along various dimensions. Currently we are working towards developing such a mapping.

REFERENCES

1. Abrahamsson, P., Warsta, J., Siponen, M.T. and Ronkainen, J. (2003) "New Directions on Agile Methods: A Comparative Analysis," 25th International Conference on Software Engineering, Portland, Oregon, 244-254.
2. Beck, K. (1999) "Embracing Change with Extreme Programming," IEEE Computer, 32, 10, 70-77.
3. Beedle, M., Bennekum, A.v., Cockburn, A., Cunningham, W., Fowler, M., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Schwaber, K., Sutherland, J. and Thomas, D. (2001) "Principles behind the Agile Manifesto," http://agilemanifesto.org/principles.html
4. Blotner, J. (2003) "It's More than Just Toys and Food: Leading Agile Development in an Enterprise-Class Start-Up," Agile Development Conference (ADC 2003), 81.
5. Boehm, B. (2002) "Get Ready for Agile Methods, with Care," IEEE Computer, 35, 1, 64-69.
6. Cao, L., Mohan, K., Xu, P. and Ramesh, B. (2004) "How Extreme Does Extreme Programming Have to Be? Adapting XP Practices to Large-Scale Projects," 37th Annual Hawaii International Conference on System Sciences (HICSS'04), IEEE Computer Society, Big Island, Hawaii, 30083c.
7. Cilliers, P. (2000) "What Can We Learn From a Theory of Complexity?" Emergence, 2, 1, 23-33.
8. Coad, P., LeFebvre, E. and Luca, J.D. (2000) Java Modeling in Color with UML: Enterprise Components and Process, Prentice Hall.
9. Cockburn, A. and Highsmith, J. (2001) "Agile Software Development: The People Factor," IEEE Computer, 34, 11, 131-133.
10. Cusumano, M. and Yoffie, D. (1999) "Software Development on Internet Time," IEEE Computer, 32, 10, 60-69.
11. Eoyang, G. (1996) "Complex? Yes! Adaptive? Well, maybe…," Interactions, 3, 1, 31-36.
12. Fowler, M. (2002) "Agile Development: What, Who, How, and Whether," http://www.fawcette.com/resources/managingdev/interviews/fowler/
13. Gerber, M. (2002) "Keynote Speech: Lightweight Methods and Their Foundations In Chaos Theory," 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002), Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland, http://edoc.dstc.edu.au/edoc-2002/keynotes.html#gerber.
14. Glass, R. (2003) "Questioning the Software Engineering Unquestionables," IEEE Software, 20, 3, 119-120.
15. Grenning, J. (2001) "Launching Extreme Programming at a Process-Intensive Company," IEEE Software, 18, 6, 27-33.
16. Highsmith, J. (1997) "Messy, Exciting, and Anxiety-Ridden: Adaptive Software Development," American Programmer, X, 1.
17. Highsmith, J. (1999) Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House Publishing, New York, NY.
18. Krutchen, P. (2001) "Agility with the RUP," Cutter IT Journal, 14, 12, 27-33.
19. Larman, C. and Basili, V.R. (2003) "Iterative and Incremental Development: A Brief History," IEEE Computer, 36, 6, 47-56.
20. Levy, D. (1994) "Chaos theory and Strategy: Theory Application, and Managerial Implications," Strategic Management Journal, 15, 5, 167-178.
21. MacCormack, A., Verganti, R. and Iansiti, M. (2001) "Developing Products on "Internet Time": The Anatomy of a Flexible Development Process," Management Science, 47, 1, 133-150.
22. Martin, R. (2000) "eXtreme Programming Development through Dialog," IEEE Software, 17, 4, 12-13.
23. Meyer, M. and Curley, K. (1991) "An Applied Framework for Classifying the Complexity of Knowledge-Based Systems," MIS Quarterly, 15, 4, 455-472.
24. Papazoglou, M. (2001) "Agent-oriented technology in support of e-business," Communications of the ACM, 44, 4, 71-

77.
25. Paulk, M. (2001) "Extreme Programming from a CMM Perspective," IEEE Software, 18, 6, 19-26.
26. Reifer, D. (2002) "How Good are Agile Methods?" IEEE Software, 19, 2, 16-18.
27. Reifer, D., Maurer, F. and Erdogmus, H. (2003) "Scaling Agile Methods," IEEE Software, 20, 4, 12-14.
28. Rising, L. and Janoff, N. (2000) "The Scrum Software Development Process for Small Teams," IEEE Software, 17, 4, 26-32.
29. Sutherland, J. and Heuvel, W.-J.v.d. (2002) "Enterprise Application Integration and Complex Adaptive Systems," Communications of the ACM, 45, 10, 59-64.
30. Williams, L. and Cockburn, A. (2003) "Guest Editors' Introduction: Agile Software Development," IEEE Computer, 36, 6, 39-43.
31. Zipf, G. (1949) Human Behavior and the Principle of Least Effort, Hafner Publishing Company, New York.