Agile Software Development Methods: Review and Analysis

Authors: Pekka Abrahamsson, Outi Salo, Jussi Ronkainen and Juhani Warsta

3.3. Crystal family of methodologies

The Crystal family of methodologies includes a number of different methodologies for selecting the most suitable methodology for each individual project. Besides the methodologies, the Crystal approach also includes principles for tailoring the methodologies to fit the varying circumstances of different projects.

Each member of the Crystal family is marked with a color indicating the 'heaviness' of the methodology, i.e. the darker the color the heavier the methodology. Crystal suggests choosing the appropriate color of methodology for a project based on its size and criticality (Figure 6). Larger projects are likely to ask for more coordination and heavier methodologies than smaller ones. The more critical the system being developed the more rigor is needed. The character symbols in the Figure 6 indicate a potential loss caused by a system failure (i.e. the criticality level): **C**omfort (C), **D**iscretionary money (D), **E**ssential money (E) and **L**ife (L) (Cockburn 2002a). In other words, criticality level C indicates that the system crash on defects causes a loss of comfort for the user whereas defects in a life critical system may literally cause loss of life.

The dimensions of criticality and size are represented by a project category symbol described in Figure 6; thus, for example, D6 denotes a project with a maximum of six persons delivering a system of maximum criticality of discretionary money.

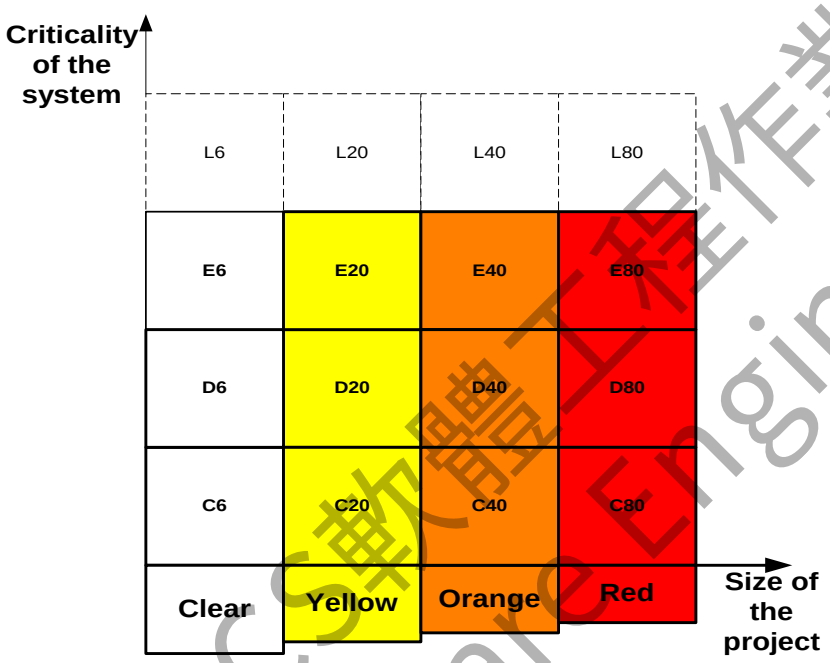

Figure 6. Dimensions of Crystal methodologies (Cockburn 2002a)

There are certain rules, features and values that are common to all the methods in the Crystal family. First of all, the projects always use incremental development cycles with a maximum increment length of four months, but preferably between one and three months (Cockburn 2002a). The emphasis is on communication and cooperation of people. Crystal methodologies do not limit any development practices, tools or work products, while also allowing the adoption of, for example, XP and Scrum practices (Cockburn 2002a). Furthermore, the Crystal approach embodies objectives for reducing intermediate work products and shaping conventions within a methodology for individual projects and developing them as the project evolves.

There are currently three main Crystal methodologies constructed: Crystal Clear, Crystal Orange and Crystal Orange Web (Cockburn 2002a). The first two of these have been experimented in practice and thus also introduced and discussed in this section.

3.3.1. Process

All of the methodologies of the Crystal family provide guidelines of policy standards, work products, "local matters", tools, standards and roles (see 3.3.2) to be followed in the development process. Crystal Clear and Crystal Orange are the two Crystal family members that have been constructed and used (Cockburn 1998; Cockburn 2002a). Crystal Orange (Cockburn 1998) also presents the activities included in the process. In this subsection, these two methodologies are discussed by viewing their contexts, similarities and differences and illustrating the process and activities of Crystal Orange.

Crystal Clear is designed for very small projects (D6 project category projects), comprising up to six developers. However, with some extension to communication and testing it can be applied also to E8/D10 projects. A team using Crystal Clear should be located in a shared office-space due to the limitations in its communication structure (Cockburn 2002a).

Crystal Orange is designed for medium-sized projects, with a total of 10 to 40 project members (D40 category), and with a project duration of one to two years. Also E50 category projects may be applicable within Crystal Orange with additions to its verification-testing processes. (Cockburn 2002a). In Crystal Orange, the project is split up for several teams with cross-functional groups (see 3.3.2) using the Holistic Diversity strategy (see 3.3.3). Still, this method does not support the distributed development environment. The Crystal Orange emphasizes the importance of the time-to-market issue. The trade-offs between extensive deliverables and rapid change in requirements and design results in a limited number of deliverables enabling to reduce the cost of maintaining them, but still keeping the communication functioning efficiently between the teams (Cockburn 1998).

In the following, the differences and similarities between the Crystal Clear and Orange methods are discussed regarding the different elements provided by the Crystal family for each of its members. The roles are further discussed in sub-section 3.3.2.

Policy standards

Policy standards are the practices that need to be applied during the development process. Both the Crystal Clear as well as Crystal Orange suggest the following policy standards (Cockburn 2002a):

- Incremental delivery in regular basis
- Progress tracking by milestones based on software deliveries and major decisions rather than written documents
- Direct user involvement
- Automated regression testing of functionality
- Two user viewing per release
- Workshops for product- and methodology-tuning at the beginning and in the middle of each increment

The only difference between the policy standards of these two methodologies is that Crystal Clear suggest incremental delivery within a two to three month time frame whereas in Crystal Orange the increments can be extended to four months at the maximum.

The policy standards of these Crystal methodologies are mandatory but can, however, be replaced with equivalent practices of other methodologies such as XP or Scrum (Cockburn 2002a).

Work products

Cockburn's (2002a) requirements for work products of Crystal Clear and Crystal Orange differ to a certain extent. The similar work products of both Crystal Clear and Orange include the following items: release sequence, common object models, user manual, test cases, and migration code.

In addition, Crystal Clear includes annotated use cases/feature descriptions, whereas in Crystal Orange the requirements document is required. In Crystal

Clear, the schedule is documented only on user viewings and deliveries and the comparable work product in Orange that Cockburn lists is (2002a) "schedule", indicating a need for more extensive scheduling of the project. The more lightweight nature of the work products in Crystal Clear emerges also in design documentation: while Orange demands UI design documents and inter-team specifications, in Clear only screen drafts and design sketches and notes are suggested if needed. In addition to these work products, Crystal Orange requires status reports.

Local matters

Local matters are the procedures of Crystal that have to be applied, but their realization is left up to the project itself. The local matters of Crystal Clear and Crystal Orange do not largely differ from each other. Both methodologies suggest that templates for the work products as well as coding, regression testing and user interface standards should be set and maintained by the team itself (Cockburn 2002a). For example, project documentation is required but their content and form remains a local matter. Above this, also the techniques to be used by the individual roles in the project are not defined by Crystal Clear nor Crystal Orange.

Tools

The tools that the Crystal Clear methodology requires are compiler, versioning and configuration-management tool, and printing whiteboards (Cockburn 2002a). Printing whiteboards are used, for example, for replacing the formal design documents and meeting summaries. In other words, they are used for storing and presenting material that otherwise should be typed down in a formal documentation format after a meeting.

The minimum tools required by Crystal Orange are those used for versioning, programming, testing, communication, project tracking, drawing, and performance measuring. In addition, screen drivers are needed for repeatable GUI tests. (Cockburn 1998).

Standards

Crystal Orange proposes selecting the notation standards, design conventions, formatting standards and quality standards (Cockburn 1998) to be used in the project.

Activities

The activities of Crystal Orange are discussed in more detail in section 3.3.3. However, they are included in the Figure 7 as a part of the illustration of one increment of Crystal process.

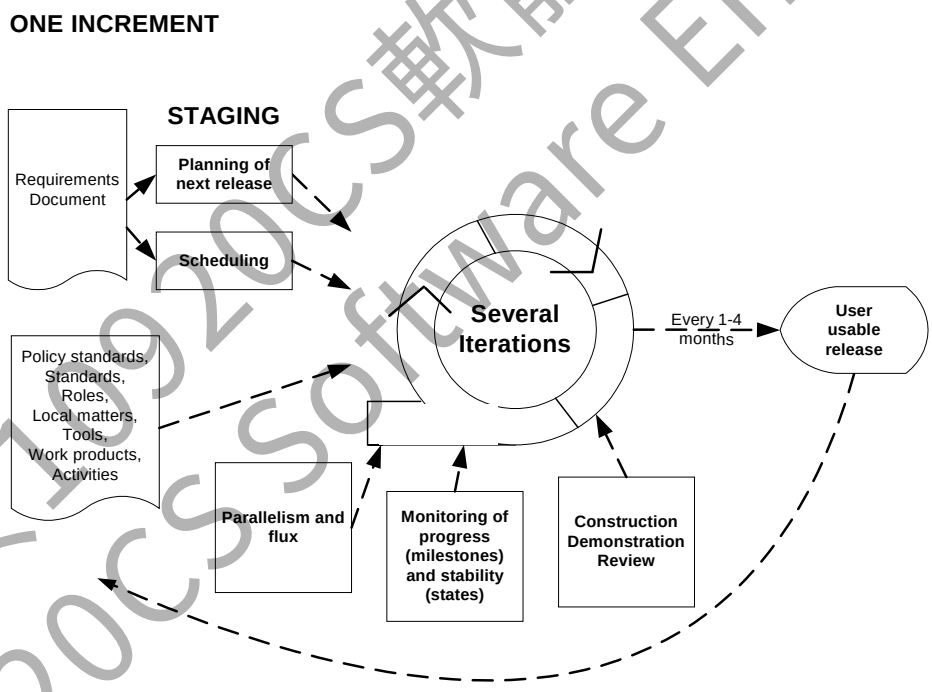

Figure 7. One Crystal Orange Increment

3.3.2. Roles and responsibilities

In this section, the roles of Crystal Clear and Crystal Orange are presented according to Cockburn (2002a). The basic difference between Crystal Clear and Orange is that in the former there is only one team in a project. In Crystal Orange there are multiple teams to follow though the project. In both methodologies, one job assignment may include multiple roles.

In Crystal Clear the main roles requiring separate persons are (Cockburn 2002a): sponsor, senior designer-programmer, designer-programmer and user. These roles embody multiple sub-roles. For example, the designer-programmer consists of business class designer, programmer, software documenter and unit tester (Cockburn 1998). The sub-roles that can be assigned to other roles are coordinator, business expert and requirements gatherer (Cockburn 2002a). The business expert represents a business view in the project, possessing knowledge about the specific business context. He should be able to take care of the business plan, paying attention to what is stable and what is changing (Cockburn 1998).

In addition to the roles introduced in Crystal Clear, Crystal Orange suggest a wide range of main roles needed in the project. The roles are grouped into several teams, such as system planning, project mentoring, architecture, technology, functions, infrastructure and external test teams (Cockburn 2002a). The teams are further divided into cross-functional groups containing different roles. (Cockburn 2002a).

Crystal Orange introduces several new roles (Cockburn 1998; Cockburn 2002a), such as UI designer, database designer, usage expert, technical facilitator, business analyst/designer, architect, design mentor, reuse point, writer, and tester. Crystal Orange (Cockburn 1998) also involves the skills and techniques needed in order to succeed in these roles. One project member can hold multiple roles. For example, reuse point is a part-time role that can be carried out by an architect or a designer-programmer, for instance (Cockburn 1998). This role refers to the identification of reusable software components and acquisition of commercial components. Other roles that need further clarification are the writer and the business analyst-designer. The writer's role includes the construction of external documentation, such as screen specifications and user manuals

(Cockburn 1998). The Business analyst-designer communicates and negotiates with the users in order to specify the requirements and interfaces, and to review the design.

Each group consists of at least a business analyst designer, a UI designer, two to three designer-programmers, a database designer and possibly also a tester. Also technical facilitators may be needed in a group. The reason for having cross-functional groups is to reduce deliverables and to enhance local communication. (Cockburn 2002a)

3.3.3. Practices

All Crystal methodologies involve a number of practices, such as incremental development. In the description of Crystal Orange (Cockburn 1998), the increment includes activities such as staging, monitoring, reviewing, along with parallelism and flux (Figure 7). Also other activities and practices can be identified. These are included in the following descriptions.

Staging
Staging includes the planning of the next increment of the system. It should be scheduled to produce a working release in every three or four months (Cockburn 1998) at the maximum. A schedule of one to three months is preferred (Cockburn 2002a). The team selects the requirements to be implemented in the increment and schedules what they feel they are able to deliver (Cockburn 1998).

Revision and review
Each increment includes several iterations. Each iteration includes the following activities: construction, demonstration and review of the objectives of the increment (Cockburn 1998).

Monitoring
The progress is monitored regarding the team deliverables during the development process with respect to their progress and stability (Cockburn 1998). The progress is measured by milestones (start, review 1, review 2, test,

deliver) and stability stages (wildly fluctuating, fluctuating and stable enough to review). Monitoring is required in both Crystal Clear and Crystal Orange.

Parallelism and flux

Once the monitoring of stability gives the result "stable enough to review" for the deliverables the next task can start. In Crystal Orange, this means that the multiple teams can proceed with maximum parallelism successfully. To ensure this, the monitoring and architecture teams review their work plans, stability and synchronization. (Cockburn 1998).

Holistic diversity strategy

Crystal Orange includes a method called holistic diversity strategy for splitting large functional teams into cross-functional groups. The central idea of this is to include multiple specialties in a single team (Cockburn 1998, p.214). The holistic diversity strategy also allows forming small teams with the necessary special know-how, and also considers issues such as locating the teams, communication and documentation and coordination of multiple teams. (Cockburn 1998).

Methodology-tuning technique

The methodology-tuning technique is one of the basic techniques of Crystal Clear and Orange. It uses project interviews and team workshops for working out a specific Crystal methodology for each individual project (Cockburn 2002a). One of the central ideas of incremental development is to allow fixing or improving the development process (Cockburn 1998). In every increment, the project can learn and use the knowledge it has gained for developing the process for the next increment.

User viewings

Two user viewings are suggested for Crystal Clear per one release (Cockburn 2002a). In Crystal Orange, users reviews should be organized three times for each increment (Cockburn 1998).

Reflection workshops

Both Crystal Clear and Orange include a rule that a team should hold pre- and post-increment reflection workshops (with recommendation for also mid-increment reflection workshops) (Cockburn 2002a).

Crystal Clear and Crystal Orange do not define any specific practices or techniques to be used by the project members in their software development tasks. The adoption of practices from other methodologies such as XP and Scrum is allowed in Crystal to replace some of its own practices, such as reflection workshops, for instance.

3.3.4. Adoption and experiences

At least one experience report (Cockburn 1998) can be found of a project adopting, evolving and using practices that currently form the Crystal Orange methodology (Cockburn 2002b). The two-year 'project Winifred' was a medium sized project with 20 to 40 staff members, with an incremental development strategy, an object-oriented approach and a goal of delivering a rewritten legacy mainframe system.

According to Cockburn (1998) the project Winifred run into problems in the first increment of the project. Problems with the communication within and across teams, a long requirements phase that prevented designing any architecture for several months, high turnover of technical leaders and unclear job assignments gave rise to the evolvement of software development process into a form that is now called Crystal Orange. One solution was the adoption of an iterative process for each increment. This provided the teams with a chance to identify their own weaknesses and to reorganize themselves. Also, the iterations included functional viewings with the users for defining the final requirements. This kept the users consistently involved in the project. Furthermore, the lessons learned on the first increment encouraged those involved to focus on issues such as clear job assignments, planning the lines of communication, defined ownership of deliverables and management support.

It is to be noted that in the project Winifred different teams could have different numbers of iterations depending on the their tasks. The communication across the teams, for example in the form of deliverables, was then planned to suit the iterations and vice versa.

In conclusion, the key success factors of the Winifred project were the increments enabling the fixing of the process, certain key individuals, and the organization of the team into a habit of delivering (Cockburn 1998).

3.3.5. Scope of use

At present, the Crystal family of methodologies does not cover life-critical projects Figure 6 (Cockburn 2002a). Another restriction identified by Cockburn (2002a) is that only co-located teams can be addressed by these methodologies.

Cockburn (2002a) has also identified limitations concerning the individual methodologies used in the Crystal family. For example, Crystal Clear has a relatively restricted communication structure and is thus suitable for only for a single team located in a single office space. Furthermore, Crystal Clear lacks system validation elements and is thus not suitable for life-critical systems. Crystal Orange also requires its teams to be located in a single office building and is capable of delivering only a system maximum of discretionary money in the level of criticality. It also lacks in its sub-team structures and, as such, is suitable for projects involving up to 40 persons. It is also not very competent regarding design- and code-verification activities and thus not suitable for life-critical systems.

3.3.6. Current research

While only two of the four proposed Crystal methodologies exist, Alistair Cockburn[9] continues the development of his Crystal family of methodologies to cover different types of projects and problem domains. Beyond this, no research can be identified.

[9] crystalmethodologies.org

References

Abrahamsson, P. (2002). "Commitment nets in software process improvement." Annals of Software Engineering, in press.

Agarwal, R. and J. Prasad (2000). "A field study of the adoption of software process innovations by information systems professionals." IEEE Transactions on Engineering Management 47(3): 295-308.

Alexander, C. (1979). The Timeless Way of Building. New York, Oxford University Press.

Ambler, S. (2002a). Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. New York, John Wiley & Sons, Inc. New York.

Ambler, S. (2002b). "Lessons in Agility from Internet-Based Development." IEEE Software March / April: 66 - 73.

Ambler, S. W. (2002c). "Duking it out." Software Development 10.

Anderson, A., R. Beattie, K. Beck, D. Bryant, M. DeArment, M. Fowler, M. Fronczak, R. Garzaniti, D. Gore, B. Hacker, C. Handrickson, R. Jeffries, D. Joppie, D. Kim, P. Kowalsky, D. Mueller, T. Murasky, R. Nutter, A. Pantea and D. Thomas (1998). "Chrysler Goes to "Extremes". Case Study." Distributed Computing(October): 24-28.

Baskerville, R. (1998). Designing information systems security. Chichester, UK, John Wiley.

Baskerville, R., J. Travis and D. P. Truex (1992). Systems without method: The impact of new technologies on information systems development projects. Transactions on the impact of computer supported technologies in information systems development. K. E. Kendall, K. Lyytinen and J. I. DeGross. Amsterdam, Elsevier Science Publications: 241-260.

Bayer, S. and J. Highsmith (1994). "RADical software development." American Programmer 7(6): 35-42.

Beck, K. (1999a). "Embracing Change With Extreme Programming." IEEE Computer 32(10): 70-77.

Beck, K. (1999b). Extreme programming explained. Reading, Mass., Addison-Wesley.

Beck, K. (2000). Extreme Programming Explained: Embrace Change.

Beck, K., M. Beedle, A. Bennekum van, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas (2001). "Manifesto for Agile Software Development." 2002(22.3.2002) http://AgileManifesto.org.

Bergquist, M. and J. Ljungberg (2001). "The power of gifts: organizing social relationships in open source communities." Information Systems Journal 11(4): 305 - 320.

Boehm, B. (2002). "Get Ready For The Agile Methods, With Care." Computer 35(1): 64-69.

Coad, P., E. LeFebvre and J. De Luca (2000). Java Modeling In Color With UML: Enterprise Components and Process, Prentice Hall.

Cockburn, A. (1998). Surviving Object-Oriented Projects: A Manager's Guide, Addison Wesley Longman.

Cockburn, A. (2000). Writing Effective Use Cases, The Crystal Collection for Software Professionals, Addison-Wesley Professional.

Cockburn, A. (2002a). Agile Software Development. Boston, Addison-Wesley.

Cockburn, A. (2002b). "Agile Software Development Joins the "Would-Be" Crowd." Cutter IT Journal 15(1): 6-12.

Coplien, J. O. (1998). A Generative Development Process Pattern Language. The Patterns Handbook. L. Rising. New York, Cambridge University Press: 243-300.

Coyne, R. (1995). Designing Information Technology in the Postmodern Age. Cambridge, Mass., MIT Press.

Crowston, K. and B. Scozzi (2002). "Open source software projects as virtual organisations: competency rallying for software development." IEE Proceedings - Software 149(1): 3 - 17.

Cunningham, W. (1996). Episodes: A Pattern Language of Competitive Development. Pattern Languages of Program Design 2. J. Vlissides. New York, Addison-Wesley.

DeMarco, T. and T. Lister (1999). Peopleware. New York, Dorset House.

Dempsey, B., D. Weiss, P. Jones and J. Greenberg (2002). "Who Is and Open Source Software Developer." CACM 45(2): 67 - 72.

Dhillon, G. (1997). Managing information systems security. London, UK, MacMillan Press.

Dobrica, L. and E. Niemelä (2002). "A survey on software architecture analysis methods." IEEE Transactions on Software Engineering 28(7): 638-653.

DSDMConsortium (1997). Dynamic Systems Development Method, version 3. Ashford, Eng., DSDM Consortium.

eWorkshop (2002). "Summary of the second eworkshop on agile methods." 2002(August 8) http://fc-md.umd.edu/projects/Agile/SecondeWorkshop/summary2ndeWorksh.htm.

Favaro, J. (2002). "Managing Requirements for Business Value." IEEE Software March/April: 15 - 17.

Feller, J. and B. Fitzgerald (2000). A Framework Analysis of the Open Source Software Development Paradigm. 21st Annual International Conference on Information Systems, Brisbane, Australia.

Gallivan, M. (2001). "Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies." Information Systems Journal 11(4): 273 - 276.

Ghosh, R. and V. Prakash (2000). "The Orbiten Free Software Survey." 2002(03052002) orbiten.org/ofss/01.html.

Gilb, T. (1988). Principles of Software Engineering Management. Wokingham, UK, Addison-Wesley.

Glass, R. L. (1999). "The realities of software technology payoffs." Communications of the ACM 42(2): 74-79.

Glass, R. L. (2001). "Agile Versus Traditional: Make Love, Not War!" Cutter IT Journal 14(12): 12-18.

Grenning, J. (2001). "Launching XP at a Process-Intensive Company." IEEE Software 18: 3-9.

Haungs, J. (2001). "Pair programming on the C3 project." Computer 34(2): 118-119.

Hawrysh, S. and J. Ruprecht (2000). Light Methodologies: It's Like Déjà Vu All Over Again. Cutter IT Journal. 13: 4 - 12.

Highsmith, J. (2002). Agile software development ecosystems. Boston, MA., Pearson Education.

Highsmith, J. and A. Cockburn (2001). "Agile Software Development: The Business of Innovation." Computer 34(9): 120-122.

Highsmith, J. A. (2000). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York, NY, Dorset House Publishing.

Hightower, R. and N. Lesiecki (2002). Java Tools for Extreme Programming. New York, Wiley Computer Publishing.

Humphrey, W. S. (1995). A discipline for software engineering, Addison Wesley Longman, Inc.

Hunt, A., Thomas, D. (2000). The Pragmatic Programmer, Addison Wesley.

Jacobsen, I. (1994). Object-Oriented Software Engineering. New York, Addison-Wesley.

Jacobsen, I., M. Christerson, P. Jonsson and G. Overgaard (1994). Object-Oriented Software Engineering: A Use-Case-Driven Approach. Reading, MA, Addison-Wesley.

Jeffries, R., A. Anderson and H. C. (2001). Extreme Programming Installed. Upper Saddle River, NJ, Addison-Wesley.

Kruchten, P. (1996). "A Rational Development Process." Crosstalk 9(7): 11-16.

Kruchten, P. (2000). The Rational Unified Process: an Introduction, Addison-Wesley.

Krueger, C. (2002). "Eliminating the adoption barrier." IEEE Software 19(4): 29-31.

Lakoff, G. and M. Johnson (1998). Philosophy in the Flesh. New York, Basic Books.

Lerner, J. and J. Tirole (2001). "The Simple Economics of Open Source." 2002(20.052002) http://www.people.hbs.edu/jlerner/publications.html.

Martin, R. C. (1998). The Process. Object Oriented Analysis and Design with Applications, Addison Wesley: 93-108.

Maurer, F. and S. Martel (2002a). "Extreme programming: Rapid development for Web-based applications." IEEE Internet Computing 6(1): 86-90.

Maurer, F. and S. Martel (2002b). "On the Productivity of Agile Software Practices: An Industrial Case Study." (page accessed August 16, 2002) http://sern.ucalgary.ca/~milos/papers/2002/MaurerMartel2002c.pdf.

McCauley, R. (2001). "Agile Development Methods Poised to Upset Status Quo." SIGCSE Bulletin 33(4): 14 - 15.

Miller, D. and J. Lee (2001). "The people make the process: commitment to employees, decision making, and performance." Journal of Management 27: 163-189.

Miller, G. G. (2001). The Characteristics of Agile Software Processes. The 39th International Conference of Object-Oriented Languages and Systems (TOOLS 39), Santa Barbara, CA.

Mockus, A., R. Fielding and J. Herbsleb (2000). A Case Study of Open Source Software Development: The Apache Server. 22nd International Conference on Software Engineering, ICSE 2000, Limerick, Ireland.

Moore, G. A. (1995). Inside the tornado. New York, HarperBusiness.

Nandhakumar, J. and J. Avison (1999). "The fiction of methodological development: a field study of information systems development." Information Technology & People 12(2): 176-191.

Naur, P. (1993). "Understanding Turing's universal machine: Personal style in program description." The Computer Journal 36(4): 351-372.

O'Reilly, T. (1999). "Lessons from Open Source Software Development." Communications of the ACM Vol. 42(No. 4): 32-37.

Palmer, S. R. and J. M. Felsing (2002). A Practical Guide to Feature-Driven Development.

Parnas, D. L. and P. C. Clements (1986). "A rational design process: How and why to fake it." IEEE Transactions on Software Engineering 12(2): 251-257.

Pfeffer, J. (1998). The human equation : building profits by putting people first. Boston, MA, Harvard Business School Press, cop.

Pöyhönen, J. (2000). Uusi varallisuusoikeus. (In Finnish). Helsinki, Lakimiesliiton Kustannus.

Rising, L. and N. S. Janoff (2000). "The Scrum software development process for small teams." IEEE Software 17(4): 26-32.

Schuh, P. (2001). "Recovery, Redemption, and Extreme Programming." IEEE Software 18(6): 34-41.

Schwaber, K. (1995). Scrum Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation, Springer-Verlag.

Schwaber, K. and M. Beedle (2002). Agile Software Development With Scrum. Upper Saddle River, NJ, Prentice-Hall.

Sharma, S., V. Sugumaran and B. Rajagopalan (2002). "A framework for creating hybrid-open source software communities." Information Systems Journal 12(1): 7 - 25.

Smith, J. (2001). A Comparison of RUP and XP, Rational Software White Paper: http://www.rational.com/media/whitepapers/TP167.pdf.

Sol, H. G. (1983). A feature analysis of information systems design methodologies: Methodological considerations. Information systems design methodologies: A feature analysis. T. W. Olle, H. G. Sol and C. J. Tully. Amsterdam, Elsevier: 1-8.

Sommerville, I. (1996). Software engineering. New York, Addison-Wesley.

Song, X. and L. J. Osterweil (1991). Comparing design methodologies through process modeling. 1st International Conference on Software Process, Los Alamitos, Calif., IEEE CS Press.

Song, X. and L. J. Osterweil (1992). "Toward objective, systematic design-method comparisons." IEEE Software 9(3): 43-53.

Stapleton, J. (1997). Dynamic systems development method - The method in practice, Addison Wesley.

Succi, G. and M. Marchesi (2000). Extreme Programming Examined: Selected Papers from the XP 2000 Conference. XP 2000 Conference, Cagliari, Italy, Addison-Wesley.

Sultan, F. and L. Chan (2000). "The adoption of new technology: The case of object-oriented computing in software companies." IEEE Transactions on Engineering Management 47(1): 106-126.

Takeuchi, H. and I. Nonaka (1986). "The New Product Development Game." Harvard Business Review Jan./Feb.: 137-146.

Truex, D. P., R. Baskerville and J. Travis (2000). "Amethodical systems development: The deferred meaning of systems development methods." Accounting, Management and Information Technology 10: 53-79.

Wall, D. (2001). "using Open Source for a Profitable Startup." Computer December: 158 - 160.

Warsta, J. (2001). Contracting in Software Business: Analysis of evolving contract processes and relationships. Department of Information Processing Science. Oulu, University of Oulu, Finland: 262.

Wiegers, K. E. (1998). "Read my lips: No new models." IEEE Software 15(5): 10-13.

Williams, L., R. R. Kessler, W. Cunningham and R. Jeffries (2000). "Strengthening the Case for Pair Programming." IEEE Software 17(4): 19-25.