In Proceedings of Extreme Programming and Agile Methods - XP/Agile Universe 2002, pp. 197-207.

Empirical Findings in Agile Methods

Mikael Lindvall[1], Vic Basili[1,4], Barry Boehm[3], Patricia Costa[1], Kathleen Dangle[1], Forrest Shull[1], Roseanne Tesoriero[1], Laurie Williams[2], and Marvin Zelkowitz[1,4]

[1]Fraunhofer Center for Experimental Software Engineering, Maryland
{mlindvall, vbasili, pcosta, kdangle, fshull, rtesoriero, mzelkowitz}@fc-md.umd.edu

[2]North Carolina State University
williams@csc.ncsu.edu

[3]University of Southern California Center for Software Engineering
boehm@sunset.usc.edu

[4]University of Maryland Empirical Software Engineering Group
{basili, mvz}@cs.umd.edu

Abstract. In recent years, the use of, interest in, and controversy about Agile methodologies have realized dramatic growth. Anecdotal evidence is rising regarding the effectiveness of agile methodologies in certain environments and for specified projects. However, collection and analysis of empirical evidence of this effectiveness and classification of appropriate environments for Agile projects has not been conducted. Researchers from four institutions organized an eWorkshop to synchronously and virtually discuss and gather experiences and knowledge from eighteen Agile experts spread across the globe. These experts characterized Agile Methods and communicated experiences using these methods on small to very large teams. They discussed the importance of staffing Agile teams with highly skilled developers. They shared common success factors and identified warning signs of problems in Agile projects. These and other findings and heuristics gathered through this valuable exchange can be useful to researchers and to practitioners as they establish an experience base for better decision making.

1. The rise of Agile Methods

Plan-driven methods are those in which work begins with the elicitation and documentation of a "complete" set of requirements, followed by architectural and high level-design development and inspection. Examples of plan-driven methods include various waterfall and iterative approaches, such as the Personal Software Process (PSP) [1]. Beginning in the mid-1990's, some practitioners found these initial requirements documentation, and architecture and design development steps frustrating and, perhaps, impossible [2]. As Barry Boehm [3] suggests, these plan-driven methods may well start to pose difficulties when change rates are still relatively low. The

industry and the technology move too fast and customers have become increasingly unable to definitively state their needs up front. As a result, several consultants have independently developed methodologies and practices to embrace and respond to the inevitable change they were experiencing. These methodologies and practices are based on iterative enhancement, a technique which was introduced in 1975 [4] and that has been come to be known as Agile Methodologies [2, 5].

Agile Methodologies are gaining popularity in industry although they comprise a mix of accepted and controversial software engineering practices. It is quite likely that the software industry will find that specific project characteristics will determine the prudence of using an agile or a plan-driven methodology – or a hybrid of the two. In recent years, there have been many stories and anecdotes [6-8] of industrial teams experiencing success with Agile methodologies. There is, however, an urgent need to empirically assess the applicability of these methods, in a structured manner, in order to build an experience base for better decision-making. This paper contributes to the experience base and discusses the findings of a synchronous, virtual eWorkshop in which experiences and knowledge were gathered from and shared between Agile experts located throughout the world.

2. An experience base for software engineering

In order to reach their goals, software development teams need to understand and choose the right models and techniques to support their projects. They must consider key questions such as: What is the best life-cycle model to choose for a particular project? What is an appropriate balance of effort between documenting the work and getting the product implemented? When does it pay-off to spend major efforts on planning in advance and avoid change, and when is it more beneficial to plan less rigorously and embrace change?

The goal of the NSF-sponsored Center for Empirically-Based Software Engineering (CeBASE)[1] is to collect, analyze, document, and disseminate knowledge on software engineering gained from experiments, case studies, observations, interviews, expert discussions and real world projects. A central activity toward achieving this goal has been the running of "eWorkshops" (or on-line meetings) that capture expert knowledge to formulate heuristics on a particular software engineering topic. The CeBASE project defined the eWorkshop and has used the technology to collect valuable empirical evidence on defect reduction and COTS. [9]

The rise of Agile Methods provides a fruitful area for such empirical research. This paper discusses the results of the first eWorkshop on Agile Methods sponsored by the Fraunhofer Center Maryland and North Carolina State University using the CeBASE eWorkshop technology. The discussion items are presented along with an encapsulated summary of the expert discussion. The heuristics can be useful both to researchers (for pointing out gaps in the current state of the knowledge requiring further investigation) and to practitioners (for benchmarking or setting expectations about development practices).

[1] http://www.CeBASE.org

3. Collecting expert knowledge on Agile methods

Workshops in which experts discuss their findings and record their discussions are a classic method for creating and disseminating knowledge. Workshops, however, possess limitations: 1) experts are spread all over the world and would have to travel to participate, and 2) workshops are usually oral presentations and discussions, which are generally not captured for further analysis. The eWorkshops are designed to help overcome these problems. The eWorkshop is an on-line meeting that replaces the usual face-to-face workshop. While it uses a Web-based chat-application, the session is structured to accommodate the needs of a workshop without becoming an unconstrained on-line chat discussion. The goal of the eWorkshop is to synthesize new knowledge from a group of experts in an efficient and inexpensive manner in order to populate an experience base. More details about the eWorkshop tool and process can be found in [10].

The goal of the Agile workshop, held in April 2002, was to create a set of heuristics that represent what experts in the field consider to be the current state of understanding about Agile Methods. The participants in this event were experts in Agile Methods. Our lead discussants (the workshop leaders and authors of this paper) formed part of the team that interacted with an international group of invited participant experts. The names of these 18 participants are listed in the acknowledgements at the end of the paper.

4. Seeding the eDiscussion

Participants of eWorkshops prepare for the discussion by reading relevant material, preparing a position statement reacting to proposed discussion points, and reading the position statements of the other discussants. For this eWorkshop, Barry Boehm's January 2002 IEEE Computer article [3] and the Agile Manifesto [11-13] served as background material. The Agile Manifesto documents the priorities of its signers. They value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Many in industry debate the prudence of these values. Steven Rakitin comments [14] that the items on the right are essential, while those on the left only serve as excuses for hackers to keep on irresponsibly throwing code together with no regard for engineering discipline. Another example is Matt Stephens' critical analysis of XP and its applicability[2].

It is important to remember that Agile includes many different methodologies, of which the best known include:

- Extreme Programming (XP) [15-17]
- Scrum [18]

[2] http://www.softwarereality.com/lifecycle/xp/case_against_xp.jsp

- Feature Driven Development (FDD) [19]
- Dynamic Systems Development Method (DSDM) [20]
- Crystal [5]
- Agile modeling [21]

In his article, Boehm brings up a number of different characteristics regarding Agile Methods compared to what he calls "Plan-Driven," the more traditional waterfall, incremental or spiral methods. Boehm contends that Agile, as described by Highsmith and Cockburn, emphasizes several critical people-factors, such as amicability, talent, skill, and communication, at the same time noting that 49.99% of the world's software developers are below average in these areas. While Agile does not require uniformly high-capability people, it relies on tacit knowledge to a higher degree than plan-driven projects that emphasize documentation. Boehm's point is that there is a risk that this situation leads to architectural mistakes that cannot be easily detected by external reviewers due to the lack of documentation.

Boehm also notes that Cockburn and Highsmith conclude that "Agile development is more difficult for larger teams" and that plan-driven organizations scale-up better. At the same time, the bureaucracy created by plan-driven processes does not fit small projects either. This again, ties back to the question of selecting the right practices for the task at hand.

Boehm questions the applicability of the Agile emphasis on simplicity. XP's philosophy of YAGNI (You Aren't Going to Need It) [15] is a symbol of the recommended simplicity that emphasizes getting rid of architectural features that do not support the current version. Boehm feels this approach fits situations where future requirements are unknown. In cases where future requirements are known, the risk is, however, that the lack of architectural support could cause severe architectural problems later. This raises questions like: What is the right balance between creating a grandiose architecture up-front and adding features as they are needed?

Boehm contends that plan-driven processes are most needed in high-assurance software. Traditional goals of plan-driven processes such as predictability, repeatability, and optimization, are often characteristics of reliable safety critical software development. Knowing for what kind of applications different practices traditional or agile are most beneficial is crucial, especially for safety critical applications where human lives can be at stake if the software fails.

The eWorkshop organizers planned to discuss each of these issues (people, team size, design simplicity, applicability for high assurance systems) outlined in Boehm's article in relation to the Agile Manifesto. However, the discussion took on its own shape based on the interests and desires of the discussants. Ultimately, the following issues were discussed:

1. The definition of agile
2. Selecting projects suitable for agile
    a. Size requirements (and scale-up strategies)
    b. Personnel requirements
    c. Use with critical, reliable, safe systems
3. Introducing the methodology
    a. Ideas for training
4. Managing the project
    a. Success factors

b. Warning signs
c. Refactoring
d. Documentation

Each of these will be discussed in the following section.

5. Findings

During the eWorkshop on Agile Methods, participants contributed their own data and experiences on various topics. Excerpts of the discussions are presented below, along with the resulting statements about Agile Methods. The full discussion summary can be found on the FC-MD web site.[3]

5.1 Definition

The eWorkshop began with a discussion regarding the definition of Agile and its characteristics, resulting in the following working definition.

Agile Methods are:

- Iterative (Delivers a full system at the very beginning and then changes the functionality of each subsystem with each new release)
- Incremental (The system as specified in the requirements is partitioned into small subsystems by functionality. New functionality is added with each new release)
- Self-organizing (The team has the autonomy to organize itself to best complete the work items.)
- Emergent (Technology and requirements are "allowed" to emerge through the product development cycle.)

All Agile methods follow the four values and 12 principles of the Agile Manifesto.[4]

5.2 Selecting Projects Suitable for Agile Methods

5.2.1 Project Size

The most important factor that determines when Agile is applicable is probably project size. The goal of the first topic was to collect experience regarding the size of projects that have been using Agile in order to determine when it is applicable. From the discussion it became clear that there is:

- Plenty of experience of teams of up to 12 people
- Some descriptions of teams around size 25
- A few data points of size teams of up to 100 people, e.g. 45 & 90-person teams, described in *Agile Software Development* [5]

[3] http://fc-md.umd.edu/projects/Agile/main.htm
[4] http://www.agilemanifesto.org/

- Isolated descriptions of teams larger than 100 people. (e.g. teams of 150 and 800 people were mentioned and documented in [2]).

Many participants felt that any team could be agile, regardless of the team size. Alistair Cockburn argued that size is an issue. As size grows, coordinating interfaces becomes a dominant issue. Agile with face-to-face communication breaks down and becomes more difficult and complex past 20-40 people. Most participants agreed, but think that this statement is true for any development process. Past 20-40 people, some kind of scale-up strategies must be applied.

One scale-up strategy that was mentioned was the organization of large projects into teams of teams. In one occasion, an 800-person team was, for example, organized using "scrums of scrums" [18]. Each team was staffed with members from multiple product lines in order to create a widespread understanding of the project as a whole. Regular, but short, meetings of cross-project sub-teams (senior people or common technical areas) were held regularly to coordinate the project and its many teams of teams. It was pointed out that a core team responsible for architecture and standards (also referred to as glue) is needed in order for this configuration to work. These people work actively with the sub-teams and coordinate the work.

Effective ways of coordinating multiple teams include yearly conferences to align interfaces, rotation of people between teams in 3-month internships, and shared test case results. Examples of strategies for coping with larger teams are documented in Jim Highsmith's Agile Software Development Ecosystems [2], in which the 800-person team is described.

5.2.2 Personnel

There is an ongoing debate about whether or not agile requires "good people" to be effective. This is an important argument to counter as "good people" can make just about anything happen and that specific practices are not important when you work with good people. This suggests that perhaps the success of Agile methods could be attributed to the teams of good folks, rather than the practices and principles. On the other hand, participants argued that Agile Methods are intrinsically valuable.

Participants agreed that a certain percentage of experienced people are needed for a successful Agile project. There was some consensus that 25%-33% of the project personnel must be "competent and experienced."

"Competent" in this context means:

- Possess real-world experience in the technology domain
- Have built similar systems in the past
- Possess good people & communication skills

It was noted that experience with actually building systems are much more important than experience with Agile development methods.

The level of experience might even be as low as 10% if the teams practice pair programming [22] and if the makeup of the specific programmers in each pair is fairly dynamic over the project cycle (termed "pair rotation"). Programmers on teams that practice pair rotation have an enhanced environment for mentoring and for learning from each other.

5.2.3 Criticality, reliability, safety issues

One of the most widespread criticisms of Agile methods is that they do not work for systems that have criticality, reliability and safety requirements. There was some disagreement about suitability for these types of projects. Some participants felt that Agile Methods work if performance requirements are made explicit early, and if proper levels of testing can be planned for. Others argue that Agile best fits applications that can be built "bare bones" very quickly, especially applications that spend most of their lifetime in maintenance.

There was also some disagreement about the best Agile Methods for critical projects. A consensus seemed to form that the Agile emphasis on testing, particularly the test-driven development practice of XP, is the key to working with these projects. Because all of the tests have to be passed before release, projects developed with XP can adhere to strict (or safety) requirements. Customers can write acceptance tests that measure nonfunctional requirements, but they are more difficult and may require more sophisticated environments than JUnit tests.

Many participants felt that it is easier to address critical issues since the customer gives requirements, makes important issues explicit early and provides continual input. The phrase "responsibly responding to change" implies that there is a need to investigate the source of the change and adjust the solution accordingly, not just respond and move on. When applied right, "test first" satisfies this requirement.

5.3 Introducing Agile Methods: Training requirements

An important issue is how to introduce Agile Methods in an organization and how much formal training is required before a team can start using it. A majority (though not all) of the participants felt that Agile Methods require less formal training than traditional methods. For example, pair programming helps minimize what is needed in terms of training, because people mentor each other [23]. This kind of mentoring (by some referred to as tacit knowledge transfer) is argued to be more important than explicit training. The emphasis is rather on skill development, not on learning Agile Methods. Training in how to apply Agile Methods can many times be done as self-training. Some participants have seen teams train themselves successfully. It was the conclusion that there should be enough training material available for XP, Crystal, Scrum, and FDD.

5.4 Project management

5.4.1 Success factors

One of the most effective ways to learn from previous experience is to analyze past projects from the perspective of success factors. The three most important success factors identified among the participants were culture, people, and communication.

To be Agile is a cultural thing. If the culture is not right, then the organization cannot be Agile. In addition, teams need some amount of local control; they must have

the ability to adapt working practices as they feel appropriate. The culture must also be supportive of negotiation as negotiation is a big part of the Agile culture.

As discussed above, it is important to have competent team members. Organizations using Agile use fewer, but more competent people. These people must be trusted, and the organization must be willing to live with the decisions developers make, not consistently second-guess their decisions.

Organizations that want to be Agile need to have an environment that facilitates rapid communication between team members. Examples are physically co-located teams and pair programming.

It was pointed out that success factors are not free and that organizations need to carefully implement these success factors in order for them to happen. The participants concluded that Agile Methods are more appropriate when requirements are emergent and rapidly changing (and there is always some technical uncertainty!). Another factor that is critical for success is fast feedback from the customer. In fact, Agile is based on close interaction with the customer and expects that the customer will be on site for the quickest possible feedback because customer feedback is viewed as such a critical success factor.

5.4.2 Warning signs

A critical part of project management is recognizing early warning signs that indicate that something has gone wrong. The question posed to participants was: How can management know when to take corrective action to minimize risks?

Participants concluded that the daily meetings provide a useful way of measuring problems. Because of the general openness of the project and because discussions of these issues is encouraged during the daily meeting, people will bring up problems. Low morale expressed by the people in the daily meeting will also reveal that something has gone wrong and that the project manager has to deal with it. Another indicator is when "useless documentation" is getting produced, even though it can be hard to determine what useless documentation is. Probably the most important warning sign is when the team is getting behind on planned iterations. As a result, having frequent iterations is very important for frequent monitoring of this warning sign.

5.4.3 Refactoring

A key tenet of agile methodologies (especially in XP) is refactoring. [24] Refactoring means improving the design of existing code without changing the functionality of the system. The different forms of refactoring involve: simplifying complex statements, abstracting common solutions into reusable code, and the removal of duplicate code.

Not all participants were comfortable with refactoring the architecture of a system because refactoring would affect all internal and external stakeholders. Instead, the approach should be frequent refactoring of reasonably sized code, keeping the scope down so that changes would more local. Most participants felt that large-scale refactoring is not a problem, because they are frequently necessary anyway and as a matter of fact, are more feasible using Agile Methods. There was a strong feeling among participants that traditional "BDUF"[5] is rarely on target, but lack of applicability is not

[5]Big Design Up Front

fed back to the team that created the BDUF so they do not learn from experience. It was again emphasized that testing is the major issue in Agile. Big architectural changes do not need to be risky, for example, if a set of automated tests is provided as a "safety net."

5.4.4 Documentation

Product and project documentation is a topic that has drawn much attention in discussions about Agile. Is any documentation necessary at all? If so, how do you know how much? Scott Ambler commented that documentation becomes out of date and should be updated only "when it hurts." Documentation is a poor form of communication, but sometimes it is necessary in order to retain critical information over time. Many organizations demand more documentation than is needed. Organizations' goal should be to communicate effectively and documentation should be one of the last options to fulfill that goal. Barry Boehm mentioned that a documented project makes it easier for an outside expert to diagnose problems. Kent Beck disagreed, saying that, as an outside expert who spends a large percentage of his time diagnosing projects, he is looking for people "stuff" (like quiet asides) and not technical details. Bil Kleb said that with Agile Methods, documentation is assigned a cost and its extent is determined by the customer (excepting internal documentation). Scott Ambler suggested his Agile Documentation essay[6] as good reference for this topic.

6. Conclusions

Whether or not to use a certain software development methodology is not trivial and depends on many factors. Our approach to support selection of methodologies is based on collecting and analyzing experience from the application of methodologies as well as the context under which the experience was gained. This experience forms an experience base and as new experience is gained, the previous experience is refined and the experience base grows. The experience base evolves over time into an asset that can support and guide future projects in selecting the most appropriate methodology for the task at hand.

This expert discussion attempted to collect experience from applying Agile Methods. It was conducted by identifying and analyzing some of the most important factors related to Agile Methods and their characteristics. A post analysis of the discussion refined and structured the results. Several lessons can be learned from this discussion; lessons that seed the experience base and that can be useful to those considering Agile Methods in their organization. These lessons should be carefully examined and challenged by future projects and the circumstances for when they hold and when they do not should be captured.

[6] http://www.agilemodeling.com/essays/agileArchitecture.htm

The lessons gained were discussed in the paper. A summary is provided below:

- Any team could be agile, regardless of the team size, but size is an issue because more people make communication harder. There is much experience from small teams. There is less for larger teams, for which scale-up strategies need to be applied.

- Experience is important for an Agile project to succeed, but experience with actually building systems is much more important than experience with Agile methods. It was estimated that 25%-33% of the project personnel must be "competent and experienced", but the necessary percentage might even be as low as 10% if the teams practice pair programming due to the fact that they mentor each other.

- Reliable and safety-critical projects can be conducted using Agile Methods. The key is that performance requirements are made explicit early, and proper levels of testing are planned. It is easier to address critical issues using Agile Methods since the customer gives requirements, makes important things explicit early and provides continual input.

- Agile Methods require less formal training than traditional methods. Pair programming helps minimize what is needed in terms of training, because people mentor each other. This is more important than regular training that can many times be completed as self-training. Training material is available in particular for XP, Crystal, Scrum, and FDD.

- The three most important success factors are culture, people, and communication. Agile Methods need cultural support otherwise they will not succeed. Competent team members are crucial. Agile Methods use fewer, but more competent people. Physically co-located teams and pair programming support rapid communication. Close interaction with the customer and frequent customer feedback are critical success factors.

- Early warning signs can be spotted in Agile projects, e.g. low morale expressed during the daily meeting. Other signs are production of "useless documentation" and delays of planned iterations.

- Refactoring should be done frequently and of reasonably sized code, keeping the scope down and local. Large-scale refactoring is not a problem, and is more feasible using Agile Methods. Traditional "BDUF" is a waste of time and doesn't lead to a learning experience. Big architectural changes do not need to be risky if a set of automated tests is maintained.

- Documentation should be assigned a cost and its extent be determined by the customer. Many organizations demand more than is needed. The goal should be to communicate effectively and documentation should be the last option.

We have an ambitious goal of collecting relevant empirically based software engineering knowledge. Based on our experiences on the topic of Agile Methods, the eWorkshop has been shown to be a mechanism for inexpensively and efficiently capturing this information. It has been useful for discussing important Agile topics, and we have obtained critical information regarding experience from real world projects using Agile Methods. To continue this activity, we will run a second eWorkshop on Agile Methods in 2002. It will be a more detailed discussion focusing on a more specific set of topics in order to collect even more detailed information about Agile Methods and their characteristics. We believe this is an important activity as practitioners need to understand when and under what circumstances a certain method or process is optimal and how it should be tailored to fit the local context.

7. Acknowledgements

We would like to recognize our expert contributors: *Scott Ambler* (Ronin International, Inc.), *Ken Auer* (RoleModel Software, Inc), *Kent Beck* (founder and director of the Three Rivers Institute), *Winsor Brown* (University of Southern California), *Alistair Cockburn* (Humans and Technology), *Hakan Erdogmus* (National Research Council of Canada), *Peter Hantos* (Xerox), *Philip Johnson* (University of Hawaii), *Bil Kleb* (NASA Langley Research Center), *Tim Mackinnon* (Connextra Ltd.), *Joel Martin* (National Research Council of Canada), *Frank Maurer* (University of Calgary), *Atif Memon* (University of Maryland and Fraunhofer Center for Experimental Software Engineering), *Granville (Randy) Miller*, (TogetherSoft), *Gary Pollice* (Rational Software), *Ken Schwaber* (Advanced Development Methods, Inc. and one of the developers of Scrum), *Don Wells* (ExtremeProgramming.org), *William Wood* (NASA Langley Research Center). This work is partially sponsored by NSF grant CCR0086078, establishing the Center for Empirically Based Software Engineering (CeBASE).

References

1. Humphrey, W.S., *A Discipline for Software Engineering*. SEI Series in Software Engineering, ed. P. Freeman, Musa, John. 1995, Reading, Massachusetts: Addison Wesley Longman, Inc.
2. Highsmith, J., *Agile Software Development Ecosystems*. The Agile Software Development Series, ed. A. Cockburn and J. Highsmith. 2002, Boston, MA: Addison-Wesley.
3. Boehm, B., *Get Ready for Agile Methods, with Care.* IEEE Computer, 2002. **35**(1): p. 64-69.
4. Basili, V.R. and A.J. Turner, *Iterative Enhancement: A Practical Technique for Software Development.* IEEE Transactions on Software Engineering, 1975. **1**(4).
5. Cockburn, A., *Agile Software Development*. The Agile Software Development Series, ed. A. Cockburn and J. Highsmith. 2001, Reading, Massachusetts: Addison Wesley Longman.

6. Marchesi, M., et al., eds. *Extreme Programming Perspectives*. XP Series, ed. K. Beck. 2002, Addison Wesley: Boston.
7. Marchesi, M. and G. Succi, eds. *Extreme Programming Examined*. XP Series, ed. K. Beck. 2001, Addison Wesley: Boston.
8. Highsmith, J., *Does Agility Work?* Software Development, 2002. **10**(6): p. 30-37.
9. Shull, F., et al. *What We Have Learned about Fighting Defects*. in *International Software Metrics Symposium*. 2002. Ottawa, Canada.
10. Basili, V.R., et al. *Building an Experience Base for Software Engineering: A Report on the first CeBASE eWorkshop*. in *Profes (Product Focused Software Process Improvement)*. 2001.
11. Highsmith, J. and A. Cockburn, *Gile Software Development: The Business of Innovation.* IEEE Computer, 2001. **34**(12).
12. Cockburn, A. and J. Highsmith, *Agile Software Development: The People Factor.* IEEE Computer, 2001. **34**(11).
13. Beck, K., et al., *The Agile Manifesto.* 2001: p. http://www.agileAlliance.org.
14. Rakitin, S., *Manifesto Elicits Cynicism.* IEEE Computer, 2001. **34**(12).
15. Beck, K., *Extreme Programming Explained: Embrace Change*. 2000, Reading, Massachusetts: Addison-Wesley.
16. Auer, K. and R. Miller, *XP Applied*. 2001, Reading, Massachusetts: Addison Wesley.
17. Jeffries, R., A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. The XP Series, ed. K. Beck. 2001, Upper Saddle River, NJ: Addison Wesley.
18. Schwaber, K. and M. Beedle, *Agile Software Development with SCRUM*. 2002: Prentice-Hall.
19. Coad, P., J. deLuca, and E. Lefebvre, *Java Modeling in Color with UML*. 1999: Prentice Hall.
20. Stapleton, J., *DSDM: The Method in Practice*. 1997: Addison Wesley Longman.
21. Ambler, S.W., *Agile Modeling*. 2002: John Wiley and Sons.
22. Williams, L., et al., *Strengthening the Case for Pair-Programming*, in *IEEE Software*. 2000. p. 19-25.
23. Palmieri, D., *Knowledge Management through Pair Programming*, in *Computer Science*. 2002, North Carolina State University: Raleigh, NC.
24. Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*. 1999, Reading, Massachusetts: Addison Wesley.