

How to Get the Most out of Extreme Programming/Agile Methods

Donald J. Reifer, President
Reifer Consultants, Inc.
Torrance, CA 90510

Abstract: This paper reports the results of an analysis of thirty-one extreme programming (XP)/agile methods early adopter projects completed by fourteen firms who have embraced the techniques in the form of lessons learned. The survey results show that early adopters have cut costs, improved productivity and reduced time to market through the use of these methods. To get the most from these methods, fifteen lessons learned have been developed that build on the experiences of others. Several of these lessons run counter to the teachings of the methodology developers. The paper next provides a scorecard that rates XP's performance in eight application domains. The paper concludes by summarizing four critical success factors for early adopters.

1. Introduction

The software industry seems to be embracing yet another change in the way it does business. Because of their emphasis on agility and time-to-market, many software shops have made the move to extreme programming/agile methods. Such methods focus on building working products instead of the documents and formal reviews that are frequently used to demonstrate progress in more classical developments. To implement these methods adherents embrace XP practices like pair programming, refactoring and collective code ownership to generate their products. These releases, which are working versions of the product, not prototypes, are used to demonstrate functions and features to stakeholders who help shape their form through refactoring [1] and continuous integration.

Initial reports from the field from early adopters about extreme programming/agile methods are encouraging. However, as is the case with anything new, some practices work better than others and some don't seem to work well at all. The purpose of this paper is to address what works in practice by summarizing the initial experiences of early adopters in the form of lessons learned. The paper's goal is to help those contemplating the move to XP/agile methods to take advantage of the experience of others as they try to use these practices productively.

2. The Survey

A survey was conducted to determine if extreme programming methods have merit and if they cut costs, reduced time-to-market and impacted product quality. The survey was conducted across eight segments of the industry [2]. The four goals that were established for the survey were:

- Determine what practices early adopters of agile methods were using.
- Assess the scope and conditions governing their use.
- Assess the cost/benefits associated with their use.
- Identify lessons learned relative to getting the most from their use.

The approach used in the survey was question and answer. The questionnaire was structured using project phases to glean what early adopters felt were the most important experiences. We also tried to gather hard data to determine cost/benefits. The questionnaire was sent to software managers in eight industry groupings who responded to the original call for information. Verification of findings was achieved via selective interviews and data analysis.

The demographics of the thirty-two organizations from twenty-nine firms (i.e., several large firms had more than one organization trying to use XP techniques) that responded to our call for information are summarized in Table 1. When queried, the fourteen firms that were using or had used XP/agile methods generally characterized the thirty-one projects that they were pursuing as follows:

- They were using XP/agile methods primarily to decrease the time needed to bring software products/application to markets. Cutting costs was a secondary concern.
- Over ninety percent of the projects involved were relatively small (typically less than ten participants) and being pursued as pilots or pathfinders. Only two of the projects involved more than twenty engineers.
- All thirty-one projects were in-house developments (as contrasted to contracted efforts), one-year or less in duration and low risk (risk was in the methods, not the software development).
- Stable requirements, established architectures and a high degree of development flexibility characterized almost all of the projects involved.
- Almost eighty-five percent of the software being developed were mostly quick-to-market applications (mostly web-based and client-server oriented).
- Teams were for the most part cohesive and staffed with motivated, experienced performers. The team members were relatively young and open to new ideas. Over ninety percent of the staff had little or no experience using XP/agile methods.
- While there was some skepticism about how well they would work, most of those involved with XP/agile methods were enthusiastic about the prospects.

Table 1 – Survey Responses

Industry	No. Firms Polled	No. Firms Responding	No. Firms Using XP/Agile Methods	No. Projects	Percentage (Responses/Polled)
Aerospace	30	5	1	1	26%
Computer	20	4	2	3	20%
Consultants	10	3	1	2	30%
E-business	20	6	5	15	30%
Researchers	10	2	1	1	20%
Scientific	10	2	0	0	20%
Software	20	3	2	4	15%
Telecom	20	4	2	5	20%
Totals	140	29	14	31	22.9%

As expected, different people from different firms had different views about what constituted best practices. When queried, the following practices were cited as being agile or extreme:

- Collective ownership
- Concurrent development
- Continuous integration
- Customer collaboration
- Daily standup meetings
- Product demos instead of documents
- Frequent product releases
- Full stakeholder participation
- Just in time requirements
- Metaphors instead of architectures
- Nightly product builds
- Pair programming
- Rapid Application Development
- Refactoring
- Retrospectives
- Stories for requirements
- Team programming
- Test-driven development

In other words, just about any new practice that literature says speeds products/applications to market were considered to be agile or extreme. Because we received such a wide range of definitions, we decided to limit further investigations to the twelve practices of extreme XP that are outlined in Kent Beck’s book [3]. Several recent good books have been written exploring use of these practices [4, 5] and the current literature is rich with many excellent articles.

3. Project Startup

The hardest thing for most organizations to do seemed to be startup. Even though they had enthusiastic staff who wanted to try the techniques, they had difficulty in convincing management that XP/agile methods were more than a passing fad. Because most of the firms involved had embraced processes that were in tune with the Software Engineering Institute’s Software Capability Maturity Model (CMM) [6], there was hot debate over whether the twelve principle practices of XP were suitable for firms with established processes that were assessed at least a level 2. In addition, many of the line managers seemed content to stick their heads in the sand and argue “Why change? We’re doing all right.” Those who wanted to try XP responded uniformly: “Because it takes too long to get our products to market.”

To resolve the debate, most early adopters decided to use XP/agile methods on some form of pilot or pathfinder project. As previously summarized, these projects were relatively small, low risk and done with an in-house workforce. Based on preliminary positive results summarized in Table 3, the results so far relative to increasing productivity, cutting costs and reducing time-to-market are encouraging. When queried, only three industries had moved XP/agile methods into production (e.g., E-business, software and telecommunications). However, one can build a convincing business case for their use using the hard and soft data that we captured [7].

Table 3 – Summary of Results of Data Analysis+

Hard Data	Soft Data
<p><u>Productivity improvement:</u> 15 to 23 percent average gain</p>	<ul style="list-style-type: none"> ▪ Most used some form of survey to capture stakeholder opinions ▪ All used recruitment, morale and other intangibles to build a case for trying and retaining agile methods ▪ All argued passionately for continued use of agile methods based on qualitative factors ▪ All pressed for help in working the issues which revolved around technology transfer <p><u>Note:</u> Hawthorne effect may apply because the sample size was relatively small</p>
<p><u>Cost reduction:</u> 5 to 7 percent on average</p>	
<p><u>Time to market compression</u> 25 to 50% reduction in time</p>	
<p><u>Quality improvement:</u> 5 of the firms had data which showed that their defect rates were on par with their other projects when products/applications were released</p>	

+ These results were developed based upon an analysis of both hard and soft data supplied by the firms that participated in the survey. Soft data was collected via questionnaire. Hard data was collected via a form developed for that purpose. This data was then normalized, validated and statistically analyzed using standard regression techniques. To ensure the integrity of the database, standard statistical tests were run to test it for homogeneity and co-linearity.

After permission to start was given, the three biggest issues associated with startup were associated with release planning, requirements specification and architecture. Because these issues are interrelated, we will have to treat them together.

XP places less emphasis on requirements and architecture than classical methods. During the elaboration phase [8], XP seeks an effective metaphor for putting a skeleton in place that frames the development. Requirements are captured iteratively as releases are formulated and plans are finalized after functions and features for the new release are decided upon. Requirements are captured on 3X5 index cards based using stories that users/customers tell to communicate what they want the system to do and what their priorities are. Releases are scheduled frequently, normally in three month increments. By planning to develop a working version of the system as the first release, they assume the pieces will come together while the system is still simple.

According to early adopters, it takes some time to get used to the XP metaphor, i.e., slim requirements and skinny architectures [5]. It also takes getting used to the concept of developing systems from scratch using only a conceptual idea of what you think the customer/user wants the system to do. Indeed, some software engineers don't seem to know how to start this definition process. In response, a coach may be needed to kickoff the effort and get the team started off on the right track. The coach should start by conducting training sessions to bring the team up to speed on XP practices and to set realizable expectations.

The two most important lessons that were learned follow. These lessons are new and should be used by others to shape the metaphor adopted. The first addresses how to structure your stories to get the most out of them. It suggests putting a premium on capturing performance and quality expectations in your stories in addition to functions and features. The second addresses ways to reduce potential architecture and XP/agile method mismatches by focusing the effort on the application layer of the architecture. In all cases, the early adopters that were surveyed suggested that XP might not be appropriate for non-precedented systems. Precedentedness is a measure of similarity with previously developed applications [9]. This statement is controversial because advocates argue that this is where agile methods shine.

- **Lesson 1** – When developing stories, focus on capturing performance and quality expectations in addition to the features and functions the user/customer wants. These can be defined in terms of acceptable end-to-end processing times and user tolerance for errors or desired levels of quality of service which include most non-functional requirements (maintainability, security, etc.). Recognize that poor quality will not be tolerated in most applications by users who are accustomed to higher standards of excellence [New].
- **Lesson 2** – Localize software to be built to the architecture's applications layer. That's where applications can be built quickly using XP techniques. Revert to more classical development methods if additional services are needed at the infrastructure layer of the architecture to mechanize the application (e.g., XP seems to work best when services available at the application layer of the architecture are adequate) [New].

After the metaphor has been developed, you can start developing a high-level plan for the first release. When thinking of such a plan, think of spending a week or two having the engineers estimate what it will take to deliver the capabilities outlined in the stories. Unlike document-driven methods, emphasis in planning is placed on having the engineers figure out the time and effort associated with story implementation. In the XP sense, such stories are testable because tests are developed typically in cooperation with customers/users prior to the start of coding to drive acceptance criteria.

As part of planning, the customer/user writes the stories, scopes what functions/features are needed, sets priorities and determines relative business value. The customers/users and software engineers collaborate to develop a way of realizing the high priority functions/features in a reasonable timeframe [10]. If the customer doesn't like the schedule, the team changes the content of the release with the user's blessing to reflect what the engineers believe can be done in the time allowable.

The following two lessons learned are aimed at getting the most out of these planning activities. Both lessons confirm experience others have had when trying to put these methods into practice. The first recommends that you assign a team leader to coordinate the development of release plans. The second focuses on using performance expectations to drive release content.

- **Lesson 3** – When planning releases, break the work down using stories to tasks that can be built by teams of two to ten people collaborating together in pairs to get the job done. Recognize that you must assign a team leader to coordinate assignments and to be ultimately held accountable for results. Also be extremely sensitive to matching personalities when staffing pairs. Else, there can be conflicts. [**Confirming**].
- **Lesson 4** – Consider performance expectations as you begin work on your release. Haste makes waste especially when performance considerations have to be factored into your design after the fact. Pin performance expectations down as early in the process as you can (see Lesson 1) and continuously exercise your working version to demonstrate their achievement via your test program [**Confirming**].

As these two lessons highlight, questions of scalability dominate the issues that larger teams have relative to harnessing the power of XP/agile methods. The data that we collected shows that XP/agile methods have merit for small applications. The jury is still out for larger projects because only two of the projects that we polled were staffed at greater than twenty people.

4. Project Implementation

You are ready to start coding once you have your stories and your release plan together. While the books say you can typically start coding within two to three weeks of starting the project, the survey indicates that startup takes four to six weeks. That's because you have to staff the team and prepare them to start using the XP/agile practices. Pair programming was singled out by many as the most controversial practice during this project phase. When asked why the controversy, most early adopters replied that assigning two people to work a job normally staffed by one was counter-cultural. In addition, several firms suggested that matching pairs was necessary to reduce potential personality conflicts. Two ways around these conflicts appear in our next two lessons learned which confirm other's experiences with XP/agile methods. Both address the need to be sensitive to how you establish pairs and who you assign to staff them.

- **Lesson 5** – Early experience with pairs indicates that personnel should be periodically rotated at least as often as work on new releases commences. This permits tasks to be staffed with pairs that have the prerequisite skills, knowledge and abilities to work on the problem at hand. It also facilitates mentoring to increase skill levels [**Confirming**].

- **Lesson 6** – Other early adopters have recommended that a short stand-up meeting be held daily in order for the team to review its plans, progress and problems. As a means to reduce conflict, pairs would be selected at this meeting based upon who could best contribute to getting the job done in the most effective manner [11] [**Confirming**].

The practice of having the customer/user on-site as a full-time project participant was identified as ideal. While great in theory, the survey suggests that this practice just does not seem to work in practice. The best that could be achieved in most situations by the firms surveyed was having the customer/user on-site for extended periods of time. This gives rise to the question: “Who were the users/customers for the typical applications being developed in the sites surveyed?” Unexpectedly, users/customers for applications being developed using by those in our survey ranged from the executives to focus groups representing users for applications like an enterprise-wide travel system on the web. Members of these groups included secretaries, engineers, managers, sales representatives and a variety of people from other specialties. Assigning a person from such a diverse group to work full-time as part of the development team was considered unacceptable because this person couldn’t make decisions for the group at large. Even a person from the travel department could not represent the user at large [12]. As another example, the customer for a web-based supplier management system being developed at one of the software firms was the Chief Technical Officer of the firm. Because he was focused on defining the firm’s next generation products, getting him to work full-time as part of the team for any prolonged period of time was simply out of the question. He participated in the development, but his average attendance was one day a week. This inability to staff the project full-time with a user/customer representative gives rise to our next lesson learned which differs from what others have reported in the literature.

- **Lesson 7** – Getting a user/customer to be resident full-time is almost impossible in most organizations. The best that can be done is getting the user/customer resident full-time for weekly periods. The challenge is to decide when to schedule such participation. Most early adopters agree that user/customer interactions are most valuable when the functionality of new releases is being planned (e.g., as working products are released for review) [**New**].

Some critics believe that XP involves nothing more than hacking out code [13]. While this may be true in some cases, this was not observed in the firms involved in the survey. Many had adapted their existing software processes to encompass XP/agile methods. Most liked the emphasis placed by XP/agile methods on demonstrating working product releases and its disdain for excessive documentation. Many argued convincingly that XP/agile methods were compatible with the CMM process infrastructure endorsed by the Software Engineering Institute [14]. These experiences lead to the next lesson whose aim is getting the most out of these methods. Again, this lesson tends to be controversial especially when firms don’t have established processes.

- **Lesson 8** – When incorporating XP/agile methods into an existing process infrastructure, put a premium on reviewing actual working product at demos instead of paper reviews. Replace out-of-date practices whenever they conflict. However, keep the infrastructure because it makes sure that you address all of the right things, not a part of them. [**New**]

Everyone surveyed agreed that the concept of frequent small working releases of the product and continuous integration made a lot of sense. The debates that occurred among early adopters revolved around how often these releases were needed and whether or not nightly builds generating daily working versions of the system were appropriate [15]. Independent of the frequency of the builds, all agreed that a working version of the code that was under some form of version control should be available for testing the next day. In addition, all argued that flexibility to reprioritize release contents should be preserved by the team (assuming the user/customer is a member). Such priorities were best dictated when set by the user/customer a priori based upon some notion of business value of the functions and/or features involved. Based on these observations, the following additional four lessons learned were developed to get the most out of XP/agile methods.

- **Lesson 9** – Code must be put under some form of release control to manage changes being made to working versions. The goal is to preserve the integrity of what is released for review and testing. Early adopters tend to have their teams update code releases at least nightly after the initial working versions are delivered for open review by any team member. Changes to releases should be incorporated nightly after pairs run clean tests. A baselined working version should be released for use during the next day, if possible [**Confirming**].
- **Lesson 10** - A version of the system should be built nightly composed of those code units that have cleanly completed testing. This version is needed because it contains those units/objects that others must interface with when they test their units the next day. To expedite testing, this version should be made public each morning and placed under version control (see Lesson 10 for more on configuration management experience) [**Confirming**].
- **Lesson 11** – Releases should be planned every few months based upon a list of prioritized stories (situation-dependent; three months the average). When push comes to shove, schedules should be preserved by pushing low priority functions and features to future releases. Performance considerations should dominate the demonstration independent of function and feature content. Acceptance tests used for the release should be devised by customers/users whenever possible because they best know what is desired [**Confirming**].

One area that most early adopters felt needed more attention was standards. Many felt that focusing on only coding standards was suboptimal. Because many of the applications cited in the survey were web-based, additional attention was needed on hyper-media design and the use xml and html. In addition, many argued that more attention needed to be placed on reuse considerations especially when products employing multi-media were being developed iteratively with high degrees of refactoring. Finally, many interviewed felt that it was the software engineer's responsibility to design quality into the code. As a consequence, having quality assurance personnel check to ensure standards were followed was thought to be counter-productive. As a matter of fact, many pairs felt that XP's elimination of the many watchers and checkers (external QA personnel, the process police, etc.) typically assigned to a project was a good thing. The value these people added to the project was questioned repeatedly. These experiences gave rise to the next two lessons whose aim is to enable those using XP to exploit its many virtues.

- **Lesson 12** – Standards should embrace design and reuse considerations in addition to what is necessary for pair programming, refactoring, testing and continuous integration. The best way to communicate best standards is via some form of context-sensitive help provided on-line with examples of what is and isn't good [**New**].
- **Lesson 13** – Make sure that those assigned to the team add value, e.g., watchers and checkers should be put to work developing product instead of offering critical remarks [**Confirming**].

5. Project Completion

The most important factor is that the project successfully delivers a high quality product on schedule and within budget constraints. In addition, the knowledge gained by the project should be captured so that others can capitalize on it. This means that firms need to put in place a process to take advantage of the lessons learned and any hard data that resulted.

One of the practices most firms added that wasn't in the recommended set was retrospectives [16]. Such reviews allow us to consider past experience, capture it and use it on our next project. Because most of these projects were pilots, it was natural to assess them and their outcomes. However, we strongly suggest adding this practice to continue to assess experience to exploit the knowledge gained. This recommendation gives rise to our next lesson learned which follows.

- **Lesson 14** – Plan to conduct a retrospective when you complete your projects to pinpoint your lessons learned and capture hard data. Then, plan to use the results to help shape how you implement XP/agile practices and the coding standards [**New**].

Before I close, I must comment on the forty hour week practice heavily endorsed by the XP/agile community. Results from early adopters indicate that forty hour weeks are still unachievable. The rationale behind this conclusion which is summarized as follows:

- Many software engineers live their work and enjoy learning on the job. Even when they don't have to, they stay in the office and work on their project or professional capabilities. Taking advantage of spare time is something new and novel for them.
- Many other software engineers are perfectionists. They need to learn when "good enough" is acceptable.
- Finally, experience shows that XP projects start slowly and end with a bang. The reason for this is that more and more hours are needed as the code requires more and more refactoring later in the project. Contrary to others, our experience shows effort levels are not flat.

These points lead to our fifteenth and final lesson learned which is as follows:

- **Lesson 15** – Plan for more than forty hour weeks towards the end of the project. Effort tends to increase proportionately with the amount of refactoring done. That is because performance problems that occur as the product is built up tend to become harder and harder to resolve (i.e., the cost to fix curve is not flat) [**New**].

This lesson runs counter to the teachings of the methodology developers. Yet, the hard data from our survey and that of others [17] tends to show that the effort associated with refactoring is not flat. Instead, the staff needed goes up as the project progresses. Such an increase in effort late in the project forces staff to expend more than forty hours per week to deliver what's promised.

6. A Scorecard for XP/Agile Methods

The scorecard for XP/agile methods on these thirty-one projects is summarized in Table 4. While seemingly really positive, one must remember that the majority of these projects were XP/agile method pilots and pathfinders. The projects are small, short in duration and low risk. The question on the minds of most early adopters we surveyed was “Will XP/agile practices scale appropriately when used on larger, more risky developments?” While initial experiences were promising, the jury is still out based on the hard data we analyzed. Therefore, we need to continue to monitor results closely to determine how to get the most out of XP/agile methods

Table 4 – XP/Agile Method Scorecard

Industry	Number of Projects	Budget Performance*	Schedule Achievement*	Product Quality+
Aerospace	1	Better than Average	Better than Average	Below Par
Computer	3	Average	Better than Average	No data
Consultants	2	Average	Average	No data
E-business	15	Better than Average	Better than Average	Above Par
Researchers	1	Average	Average	No data
Scientific	0	N/A	N/A	N/A
Software	4	Average	Average	Par
Telecom	5	Better than Average	Better than Average	Par
Totals	31			

* Average represents what is normal performance for similar projects (using existing metrics)

+ Par represents nominal quality rating for their projects (using metrics already in place)

We tried to compare the results/experiences of the larger teams to the general population dominated by small projects to answer these scalability concerns. Our efforts were not fruitful statistically because we had just two projects employing teams in excess of twenty people in our XP database. Scalability therefore remains an issue that we will have to track because we don't have enough data to develop meaningful conclusions.

7. Summary and Conclusions

We have summarized the fifteen lessons that fourteen early adopters have learned on thirty-one projects. These lessons are aimed at helping those contemplating the move to XP/agile methods to get the most out of these new and promising techniques. The lessons are based on the insights gained by early adopters who all were high on the techniques and willing to use them further. As indicated, some lessons are conformational, while others are new. To take full advantage of these lessons, we suggest that you keep the following four critical success factors in mind:

- **Proper Domain of Fit** – Recognize that XP/agile methods currently have been shown to work best on relatively small projects (less than ten people) where the systems being developed are precedented, requirements are stable and the architecture's is well established. This conclusion runs counter to rapid change which is one of the big selling points for XP/agile methods. But, scalability and application of the methods to high risk situations tend to be the two issues that continue to concern those considering embracing the methods.

- **Suitable State of Organizational Readiness** – Realize that the move to XP/agile methods represents a culture change in most firms. In order use these methods successfully, you must prepare for and foster change to this new way of doing business. Preparation is best achieved by educating and training the workforce so that they become equipped with the skills, knowledge and abilities to get the job done on schedule and within budget constraints. In addition, you must also convince your customers/users to adopt new business practices. Then, you must work with your customers/users to set realizable expectations.
- **Process Focus** – Adapt and refine rather than throw away your existing processes when you adopt XP/agile methods. Experience shows that XP/agile methods work best when they are integrated into and become part of an existing process framework that establishes the way your firm develops software. Recognize that by using this approach you can adapt and exploit supportive processes (version control, design practices, etc.) whenever they make sense and are appropriate to the task at hand.
- **Appropriate Practice Set** – Don't be afraid to put additional practices into place when they are needed to get the job done (daily standup meeting, project retrospective, etc.). Being overly zealous implementing new concepts often interferes with your organization's ability to perform. Make sure that what results as you extend the practice set stays aligned with the guiding principles of the agile manifesto [18]. This manifesto should provide the overarching concepts to frame your implementation.

Acknowledgments

I would like to acknowledge and thank those individuals from the fourteen firms surveyed who supplied me the information upon which this paper is based. I appreciate their time and insights. I would also like to thank Dr. Barry Boehm, Dr. Sunita Chulani, Dr. Steven Fraser and my other reviewers for making some very positive suggestions for improving this paper.

References

- [1] Fowler, Martin, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
- [2] Reifer, Donald J., "How Good Are Agile Methods?" *IEEE Software*, July/August 2002.
- [3] Beck, Kent, *Extreme Programming Explained*, Addison-Wesley, 2000.
- [4] Auer, Ken and Miller, Roy, *Extreme Programming Applied*, Addison-Wesley, 2002.
- [5] Wake, William C., *Extreme Programming Explored*, Addison-Wesley, 2002.
- [6] Paulk, Mark C., Weber, Charles V., Curtis, Bill and Chrissis, Mary Beth, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995.
- [7] Reifer, Donald J., *Making the Software Business Case: Improvement by the Numbers*, Addison-Wesley, 2001.
- [8] Kruchten, Philippe, *The Rational Unified Process: An Introduction*, Addison-Wesley, 1998.
- [9] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, Bert Steece, *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000, pp. 31-33.
- [10] Highsmith, James A. III, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House Publishing, 2000.

- [11]Williams, Laurie, “Extreme Programming (and Pair Programming),” *Proceedings of the University of Southern California-Center for Software Engineering Annual Research Review*, Los Angeles, CA, March 2002.
- [12]Constantine, Larry L., *The Peopleware Papers*, Yourdon Press, 2001 (see thoughts on consensus building on pp. 13-16).
- [13]Rakitin, Steven R., “Manifesto Elicits Cynicism,” Letter to the Editor, *IEEE Computer*, December 2001, p. 4.
- [14]Paulk, Mark C., “Extreme Programming from a CMM Perspective,” *IEEE Software*, November/December 2001, pp. 19-26.
- [15]Cusumano, Michael A. and Selby, Richard W., *Microsoft Secrets*, Simon & Schuster: A Touchstone Book, 1998. (see notes on builds on pp. 263-271).
- [16]Kerth, Norman L., *Project Retrospectives: A Handbook for Team Reviews*, Dorset House Publishing, 2001.
- [17]Boehm, Barry, *Private Communications*, 5/03/02.
- [18]Cockburn, Alistair, *Agile Software Development*, Addison-Wesley, 2002.

About the Author



Donald J. Reifer is one of the leading figures in the fields of software engineering and management with over thirty-five years of progressive experience in government, industry and academia. During that time, he has managed software organizations, major programs and government initiatives. He has grown a successful business and served as a trusted advisor to many Fortune 500 firms. Currently, he serves as President of Reifer Consultants, a software consulting firm, and as a visiting associate at the University of Southern California's Center for Software Engineering. Some of Reifer's many honors include the AIAA Software Engineering Award (2002), the Secretary of Defense's Medal for Outstanding Public Service (1996), the ISPA Frieman Award (1991), the NASA Distinguished Service Medal (1989), membership in Who's Who in the West and the Hughes Aircraft Masters Fellowship. Reifer has more than 100 technical publications including his two newest books: the *IEEE Software Management Tutorial (6th Edition)* and *Making the Business Case: Improvement by the Numbers*