Part IV

XR: Extreme Reality—Real-Life Experiences

Part IV reports on a series of experiences in applying XP. The goal is to guide future applications of XP by showing what went well and what did not work.

This is a first step toward establishing a body of knowledge about XP in which different experiences can be classified and compared. To ease the comparison, the authors have been required to follow a fixed set of headings as follows:

1. Research hypotheses, in which the goals of the trial of XP are described
2. Description of the context of the experience, detailing the environment where XP (or a portion of it) was tried
3. Results from the experience, with numeric quantification of the outcome, whenever possible
4. What actions were taken as a result of the experience

We have also encouraged the authors to follow a rigorous and factual style in reporting their XP experiences.

A comparison with what happens in medical studies may help in understanding the purpose and the scope of our work. Usually, a drug

becomes available for general use at the end of a four-phase process. We think that a similar approach would benefit software engineering in general and especially XP. These are the four phases.

- Phase 1—The drug is tested to determine whether it is harmful; in our case, we would like to know if a given methodology would be extremely harmful for an organization to try. We think that in XP we have already completed this phase.
- Phase 2—The drug is administered to volunteers to determine whether it has the beneficial effects it claims to have. We are at exactly this step in XP. We are "administering" the XP practices to the "volunteer" organizations that think that XP would benefit them.
- Phase 3—The drug is tested on a large scale, to perform a general, unbiased evaluation of its effects.
- Phase 4—The drug is released for general use but still under scrutiny for possible unexpected results and contraindications.

However, the first trials of novel techniques are usually performed by those adept at such techniques, those who have a direct interest in showing that the tried technique does indeed work. Remember, we are in phase 2 of our experimental study. Therefore, the language is often emphatic, and the reporters are clearly biased toward XP. This does not limit the validity of the study, because some of the results are definitely positive, while others require more careful investigation, and a few denote a negative impact, especially in the managerial aspects.

Table IV.1 summarizes the results.

Taking the same experimental approach as with medical studies, we think that now is the time to move in two directions.

- Extend these phase 2 experimentations, to broaden the scope in situations where people are not sure how to apply XP, such as with large or geographically dispersed teams.
- Initiate phase 3 experimentations—that is, systematically apply the XP approach to cases where it appears it would be useful.

TABLE IV.1 Results of Real-Life Experiences with XP

Chapter	Authors	Hypotheses	Context	Results
30	Hodgetts and Phillips	1. XP increases the rate of development. 2. XP reduces development costs. 3. XP increases the correlation of software to business needs. 4. XP reduces the release cycle. 5. XP increases the product quality.	Internet start-up; typical enterprise technology, including ASP, J2EE, relational db.	All hypotheses have been verified.
31	Kini and Collins	1. XP works well in general. 2. XP is easily accepted by the customer. 3. Some practices have worth when requirements or conditions change. 4. Most practices are not difficult to implement.	XP-based company, with specific hardware and infrastructures for XP. Four developers and one tester.	All hypotheses have been verified; areas of improvement include a formalization of the in-house XP process.
32	Schalliol	1. The story cards and the resulting code are a sufficient guide for the project. 2. Dividing application functionality into story cards is learned once and then reapplied for all subsequent iterations.	J2EE development project that switched to XP halfway through its three-year life.	Each hypothesis raises some concerns. 1. There is a lack of the "big picture," perhaps because of the size of the project. 2. It is not always true, because sometimes the division is not "obvious."

Table continued on next page.

TABLE IV.1 Results of Real-Life Experiences with XP (*cont'd.*)

Chapter	Authors	Hypotheses	Context	Results
32 (*cont'd.*)	Schalliol	3. It is easy to identify the single customer who drives the planning game for the XP process. 4. Such a customer can drive the planning game and is the primary author of all functional tests.	Fifty-person team: about 30 developers, eight quality assurance testers, and eight analysts. Overall, about 500,000 lines of executable code.	3. The large size of the application would require more than one customer in charge of leading the project. 4. The customer often was uneasy in defining the acceptance tests. 5. There were also aspects in which XP worked really well.
33	Elssamadisy	1. XP provides customers with positive feedback via quick releases. 2. XP "manages expectations" more easily because of constant customer involvement. 3. XP results in a better-quality product, delivered on time to customers. 4. XP eliminates integration problems caused by the presence of several subteams.	Large distributed leasing application; typically about 25 developers, ten business analysts, ten quality assurance people. About 600,000 lines of executable code in 6,500 classes.	All the hypotheses have been verified. In large teams, though, there are problems in meeting, applying pair programming, staying within the 40 hours a week, and defining suitable metaphors.

34	Griffin	1. XP makes the development team more responsive to business needs. 2. XP increases the speed of releases and reduces the error rate. 3. XP improves the development cycle flow.	Provision of secure transaction settlement services for business and consumer e-commerce sites. Multitier architecture with ASP, Java/EJB, Oracle db, and XML.	All the hypotheses have been verified. However, the company was not happy with the "performance" of the XP team—no overtime, no stress, and so on. Therefore, the team was dismissed at the end of the project.
35	Johansen, Stauffer, and Turner	1. Adhering to the principles of XP would help overcome the major deficiencies usually encountered on "regular" projects. 2. XP creates a stronger team and an experience that is remembered with pleasure.	Small software development company. Eight-month, six-developer project, delivering a major upgrade to an enterprise software system.	All the hypotheses have been verified in general. However, the management and one of the developers felt that working more would have helped the project to deliver more.
36	Gittins, Hope, and Williams	1. The incremental addition of XP practices provides significant benefits to the overall development process. 2. There are difficulties of various kinds, which can be discovered using qualitative statistical methods.	Medium-sized software company committed to implementing XP.	Both hypotheses have been verified, with a careful analysis of the outcome of each XP practice.

Part IV

Chapter 30

Extreme Adoption Experiences of a B2B Start-up

—Paul Hodgetts and Denise Phillips

This chapter presents the results of adopting XP at an Internet business-to-business (B2B) start-up. We discuss our motivations and goals for adopting XP and the context under which the adoption efforts were evaluated. Then we present the results in terms of objective metrics and subjective evaluations. The results indicate that the project conducted under XP demonstrated significant improvements.

Escrow.com faced extreme challenges as a new Internet start-up. The business-to-business e-commerce market was, and remains, immature and rapidly changing. Time-to-market was critical to establish the business and gain market share. Quality was essential for the company as a provider of regulated online transaction settlement services.

This chapter discusses our experiences in adopting Extreme Programming (XP). Escrow.com was in a unique position to compare nearly identical projects conducted under both XP and non-XP processes. By collecting a small set of simple metrics, we were able to measure the relative success of the adoption effort. As we'll see, even under difficult circumstances, this adoption effort produced tangible improvements over the prior project.

Extreme Programming has gained considerable fame and acclaim over the past several years. Although the published body of theoretical and how-to knowledge has grown quickly, more experience reports are needed to provide convincing evidence of XP's effectiveness in real-world project contexts. This report strives to contribute to this body of evidence.

Research Hypotheses

By October 2000, Escrow.com was experiencing severe and growing problems in its development efforts, including a dramatically slowed pace of delivery, increasing development costs, poor product quality, and a deteriorating state of the code base. Early process improvement efforts were largely unsuccessful at solving these core problems.

Senior management recognized that a fundamental change to the development process was needed. Spearheaded by a team of senior developers, research was conducted using XP. Afterward, we hypothesized that adopting XP would deliver the following benefits:

1. *Increased rate of development*, enabling the development efforts to keep pace with the increasing demand for new features
2. *Reduced development costs*, enabling a reduction in the size of the development team or deploying existing resources to new projects
3. *Increased correlation of software to business needs*, bringing the delivered releases in line with the requirements of the end users and eliminating unnecessary or low return-on-investment features
4. *Reduced release cycle*, enabling frequent releases of high-priority features
5. *Increased product quality*, reducing the quantity and severity of defects that delayed production releases
6. *Increased quality of implementation*, reducing the code entropy that hindered the addition of new features and increasing the maintainability of the code base

Existing project management and change control practices produced metrics that measured developer effort and defect discovery rates. Combined with code analysis, this data enabled objective measurements of the improvement from adopting XP.

Description of the Context of the Experience

Escrow.com is a provider of online settlement services. Escrow.com's services are delivered via Internet-based enterprise systems, using typical enterprise technologies including Active Server Pages (ASP), the Java 2 Platform, Enterprise Edition (J2EE), and relational databases.

To attack the emerging business-to-business e-commerce market, Escrow.com began development of a flexible and full-featured transaction-processing engine. This project, creatively dubbed "Version 2," or "V2," was conducted using many elements of "traditional" defined processes, and required significant expenditures for personnel, tools, and technologies. The V2 project is used as the baseline for our process improvement comparisons.

To address problems with the V2 project, Escrow.com radically retooled its development process and culture. The resulting project, called "Version 3," or "V3," used XP as its development process and is the focus of this experience report.

The XP adoption efforts at Escrow.com were conducted under actual production circumstances. Because of the immediate need to address problems and the company's small size, we had neither the time nor the resources to conduct a pilot XP project. The switch to XP was immediate and complete. Following a brief, two-week preparation period, all development was conducted using the full set of XP practices.

Based on our observations, the context under which XP was adopted at Escrow.com is similar to the development environments at many small to medium-sized companies. We believe our results can be generalized to similar development environments and projects.

Results from the Experience

In this section, we evaluate the results of the XP adoption efforts by using metrics gathered from both the V2 and V3 projects. The results are summarized in Figure 30.1.

We must point out that these comparisons were not made in the context of a controlled, scientific study and are therefore anecdotal. Some comparisons are based on subjective criteria. We nevertheless believe even anecdotal evidence in this context to be relevant and valuable, particularly because of the rarity of scientific studies conducted in actual production environments.

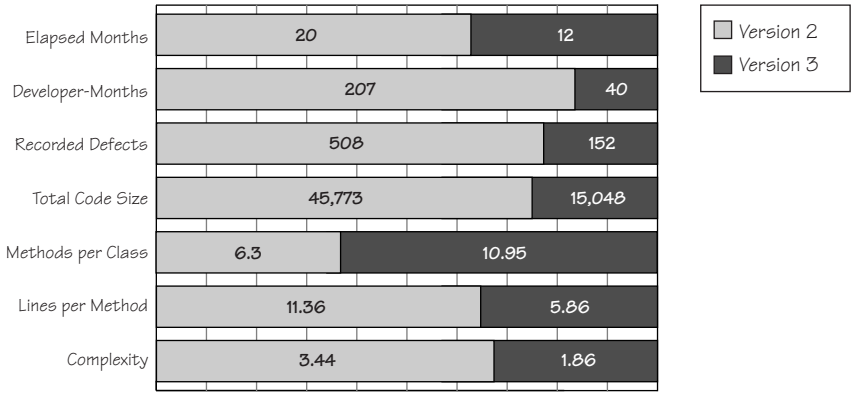

FIGURE 30.1 Version 2 versus Version 3 comparison of results

Comparing V2 and V3 Functionality

For the purposes of this evaluation, we must establish a comparison of the overall functionality of the V2 and V3 products. In the absence of objective function-point measurements, we must subjectively compare the two products.

In our subjective judgment, the V2 and V3 products were virtually equivalent in the total amount of functionality. Although the V2 product incorporated more complexity in its features, the V3 product targeted a wider set of simpler features.

Increased Rate of Development

The V2 project delivered its business value over a period of 20 months before the project was stopped because of the excessive costs of ownership.

The V3 project was suspended after nine months of development. At the established velocity, V3 would have delivered its total business value over a period of 12 months.

This result represents a 67% increase in the overall development velocity, as measured in terms of the rate of business value delivered over time.

Reduced Development Costs

The V2 project employed a total of 21 developers during its existence. The team size ranged from three developers to a maximum of 18 developers at its peak. The V2 project cost a total of 207 developer-months of effort.

The V3 project began with two developers. After four months, the team size was increased to four developers. Overall, using the estimated schedule, the V3 project would have cost a total of 40 developer-months.

These results represent an 80% reduction in developer-month effort and its corresponding personnel and overhead costs. We note that the V3 team was staffed by senior developers, and their expertise probably contributed to the productivity gains.

Increased Correlation of Software to Business Needs

The V2 project delivered features and technological capabilities beyond the requirements of the customer, at the expense of delivering revenue-generating features.

The V3 project's use of the planning game focused delivery only on clearly identified business requirements. As the expertise of the customer team increased, the development effort increasingly correlated directly to specific revenue-generating opportunities.

Reduced Release Cycle

The V2 project was unable to produce meaningful production releases in cycles of less than two to three months. The quality assurance cycle alone normally lasted two or more weeks.

The V3 project delivered production-ready releases in iteration cycles of two weeks. Because of the increased clarity and prioritization of the planning game, meaningful feature releases were produced in cycles of one to three iterations, representing a substantial improvement in the time between production releases.

The reduction in the release cycle enabled product managers to flexibly and quickly respond to changing business conditions. This was dramatically demonstrated by the rapid succession of changes to the product priorities following the introduction of XP. In January 2001, at the start of the XP adoption, two product lines were under

development. In February 2001, two additional product lines were initiated, including the V3 product. In June 2001, development was stopped on V2, and another product line was suspended. Finally, in October 2001, V3 development was suspended, leaving one remaining active product line.

Increased Product Quality

For both projects, defects found in acceptance testing were tracked using a defect-tracking database. V2's policy was to verbally report minor defects without tracking, while V3 mandated that all defects be formally logged. Acceptance tests on both projects were manually executed, but the V3 project also used a suite of 1,230 automated unit tests. Acceptance testing on the V2 project was performed sporadically, while on the V3 project acceptance testing was performed repeatedly on all iterations.

The V2 project logged a total of 508 defects over its 20-month life cycle. Of these defects, 182 were logged during a difficult two-month period from September through November 2000. Another 123 defects were logged during the final one-month testing cycle before V2's last production release in June 2001.

The V3 project logged a total of 114 defects over its entire nine-month duration. All these defects were minor. Severe defects were discovered and fixed by the developers before concluding iterations. Assuming a linear increase in the number of defects had the project run to completion, V3 would have produced 152 total defects.

These results represent a 70% reduction in the number of defects discovered in acceptance testing. This reduction is even more significant when we take into account the lower defect severity levels.

Increased Quality of Implementation

Quality of design and implementation is difficult to measure objectively, because even the types of measurements are subject to debate in the software community. For these measurements, we chose a few relatively simple indicators of possible quality: total code size, the average size of classes, the average size of methods, and the average cyclometric complexity of methods. The total-code-size metric includes all types of sources (Java, JSP, ASP, and HTML), while the remaining metrics focus only on the Java sources.

Because the V3 project was suspended before completion, it was necessary to estimate the final total-code-size measurement, assuming a continuation of the observed linear growth.

Table 30.1 summarizes the comparison of these metrics.

Although these metrics are subject to differing analyses, when we combine them with our subjective reviews of the code base, we feel they represent an improvement in the quality of the implementation. The reduction in code size is indicative of a simpler implementation, assuming delivery of comparable functionality. The presence of a larger number of smaller methods per class, combined with the reduced complexity of methods, suggests an improved division of responsibility and behavior across methods.

What to Do Next

This chapter presents evidence that the Version 3 project produced significant measurable improvements over the prior Version 2 project. Many aspects of the two projects remained relatively consistent—the domain and feature sets, the tools and technologies, and the team. The primary difference between the two projects was the use of XP on Version 3. We must therefore conclude that XP contributed substantially to the improvements, a conclusion reinforced by our subjective day-to-day observations.

The XP adoption experiences at Escrow.com have proved to us that XP is particularly effective in today's fast-paced e-commerce environment, and we now make XP our process of choice. We plan to continue to measure and quantify the benefits gained as we adopt XP on future projects.

TABLE 30.1 Measurements of Implementation Quality

	Version 2	Version 3	% Change
Total code size	45,773	15,048	– **67%**
Average methods per class	6.30	10.95	**+ 73%**
Average lines per method	11.36	5.86	– **48%**
Average cyclometric complexity	3.44	1.56	– **54%**

Acknowledgments

We acknowledge and sincerely thank all our colleagues at Escrow.com whose efforts enabled these successes. In particular, we thank our former CEO, Russell Stern, for trusting our vision and enabling us to transform the business. We also must thank Robert Martin, Lowell Lindstrom, Brian Button, and Jeff Langr from Object Mentor, whose expert help proved invaluable. A special thanks to Beth Hechanova, Tamlyn Jones, and Paul Moore for their help as reviewers. Last, but certainly not least, we thank our families and friends for their limitless patience and support for our "extreme" endeavors.

About the Authors

Paul Hodgetts is the founder and principal consultant at Agile Logic, a provider of training, mentoring, and development services focused on agile development and enterprise technologies. Paul served as the director of product development and chief architect at Escrow.com during the V2 and V3 projects. Paul can be reached at http://www.AgileLogic.com.

Denise Phillips is an independent consultant working with companies to adopt agile development techniques and processes. She is the creator and instructor of the Extreme Programming certificate program at California State University, Fullerton. Denise served as the senior enterprise architect at Escrow.com during the V2 and V3 projects. Denise can be reached by e-mail at dyp@pacbell.net.