Exploring Extreme Programming in Context: An Industrial Case Study

Lucas Layman[1], Laurie Williams[1], Lynn Cunningham[2]
[1]North Carolina State University, Department of Computer Science, {lmlayma2,lawilli3}@ncsu.edu
[2]Clarke College, lynn.cunningham@clarke.edu

Abstract

A longitudinal case study evaluating the effects of adopting the Extreme Programming (XP) methodology was performed at Sabre Airline Solutions™. The Sabre team was a characteristically agile team in that they had no need to scale or re-scope XP for their project parameters and organizational environment. The case study compares two releases of the same product. One release was completed just prior to the team's adoption of the XP methodology, and the other was completed after approximately two years of XP use. Comparisons of the new release project results to the old release project results show a 50% increase in productivity, a 65% improvement in pre-release quality, and a 35% improvement in post-release quality. These findings suggest that, over time, adopting the XP process can result in increased productivity and quality.

1. Introduction

The introduction of Extreme Programming (XP) [4] into mainstream software development has met with both enthusiasm and skepticism. Reports both extol the virtues and question the shortcomings of XP. Most often, these reports take the form of anecdotal success stories or lessons-learned from organizations that have adapted XP for a project [15, 16, 24]. However, many organizations remain skeptical regarding XP's value. For these decision-makers, an empirical, quantitative investigation is beneficial for demonstrating XP's efficacy. We increase the existing evidentiary base of empirical XP knowledge with a detailed study of an industrial team. We present this case study within the context of the XP Evaluation Framework [25, 26]. Our findings are useful for organizations seeking scientific investigation into the real-world impacts of utilizing XP practices.

In this single, longitudinal, holistic [28] case study, we examine a product created by an XP software development team at Sabre Airline Solutions™ in the United States. We evaluated and compared two releases of the Sabre team's product. One release was completed just prior to the team's initial adoption of XP; the other release was completed after two years of stabilized XP use. This ten-person team develops a scriptable GUI environment for external customers to develop customized end user and business software.

XP originators aimed at developing a methodology suitable for "object-oriented projects using teams of a dozen or fewer programmers in one location" [10]. Abrahamsson et al. [2] contend that the XP methodology is "situation appropriate" in that the methodology can be adjusted to different situations. The characteristics of the Sabre project placed it in the agile home ground [5] and allowed the use of XP in a nearly "pure" form.

As discussed in Section 3, several XP case studies were performed at Sabre. To differentiate this case study from other case studies performed at Sabre, we heretofore refer to the team in this study as Sabre-A (Agile). The Sabre-A team was among the first to use XP at Sabre Airline Solutions. The perceived success of this and the other early XP projects led to the use of XP with over 30 teams with more than 200 people throughout the organization.

In our case study, we examined five null hypotheses regarding XP's effectiveness. Because we are reporting a single case study, we cannot conclusively reject or accept these hypotheses. Our results add to the weight of evidence in support or in refutation of these propositions. We triangulate upon this support or refutation via objective and subjective quantitative methods and via qualitative data collection and analysis. The null hypotheses were as follows:

When used by teams operating within the specified context, the use of XP practices leads to no change in:

***$H1_0$**: pre-release quality (as measured by defects found before product release)*

***$H2_0$**: post-release quality (as measured by defects found by the customer after release)*

***$H3_0$**: programmer productivity (as measured by both user stories and lines of code per person-month)*

***$H4_0$**: customer satisfaction (measured via interview and customer feedback)*

***$H5_0$**: team morale (assessed via a survey)*

The remainder of this paper is organized as follows. Section 2 provides background information, and Section 3 describes the context of the case study. Section 4 presents the results of the case study. Section 5 discusses the case study limitations. Finally, Section 6 summarizes our findings and future work.

2. Background and related work

In this section, we discuss the advantages and limitations of case study and qualitative research in software engineering. We also discuss the Extreme Programming Evaluation Framework created by the authors and provide a brief survey of other XP research.

2.1. Case study research

Case studies can be viewed as "research in the typical" [7, 12]. As opposed to formal experiments, which often have a narrow focus and an emphasis on controlling context variables, case studies test theories and collect data through observation of a project in an unmodified setting [29]. However, because corporate, team, and project characteristics are unique to each case study, comparisons and generalizations of results are difficult and are subject to questions of internal validity [13]. Nonetheless, case studies are valuable because they involve factors that staged experiments generally do not exhibit, such as scale, complexity, unpredictability, and dynamism [20]. Case studies are particularly important for industrial evaluation of software engineering methods and tools [12]. Researchers become more confident in a theory when similar findings emerge in different contexts [12]. By recording the context variables of multiple case studies and/or experiments, researchers can build evidence through a family of experiments. Replication of case studies addresses threats to experimental validity [3].

2.2. Qualitative research

Qualitative methods can be used to enrich quantitative findings with explanatory information that helps to understand "why" and to handle the complexities of issues involving human behavior. Seaman [23] discusses methods for collecting qualitative data for software engineering studies. One such method is interviewing. Interviews are used to collect historical data from the memories of interviewees, to collect opinions or impressions, or to understand specific terminology. Interviews can be structured, unstructured, or semi-structured [23]. Semi-structured interviews, as were conducted in this case study, are a mixture of open-ended and specific questions designed to elicit unexpected types of information.

2.3. Extreme Programming Evaluation Framework

The Extreme Programming Evaluation Framework (XP-EF) is an ontology-based benchmark for expressing case study information [25]. The XP-EF records the context of the case study, the extent to which an organization has adopted and/or modified XP practices, and the result of this adoption. The necessity for common ontologies emerges from the need to exchange knowledge [19]. The XP-EF is composed of three parts: XP Context Factors (XP-cf); XP Adherence Metrics (XP-am); and XP Outcome Measures (XP-om).

In the XP-EF, researchers and practitioners record essential context information of their project via the XP Context Factors (XP-cf). Recording context factors such as team size, project size, criticality, and staff experience can help explain differences in the results of applying the methodology. The second part of the XP-EF is the XP Adherence Metrics (XP-am). The XP-am use objective and subjective metrics to express concretely and comparatively the extent to which a team utilizes the XP practices. When researchers examine multiple XP-EF case studies, the XP-am also allow researchers to investigate the interactions and dependencies between the XP practices and the extent to which the practices can be separated or eliminated. Part three of the XP-EF is the XP Outcome Measures (XP-om), which enable one to assess the business-related results (productivity, quality, etc.) of using a full or partial set of XP practices.

A more detailed discussion of the XP-EF, its creation, rationale, and shortcomings may be found in [26]. An initial validation of the XP-EF may be found in [26]; further work in this area is ongoing. Instructions and templates for measuring and reporting an XP case study data via XP-EF Version 1.3 have been documented by the authors of this paper [25, 26].

2.4. XP studies

Practitioners and researchers have reported numerous, predominantly anecdotal and favorable studies of XP. A number of these reports discuss the use of XP with small, co-located teams. Wood and Kleb [27] formed a two-person XP team and analyzed the productivity of their project as part of a pilot study at NASA to assess XP in a mission-critical environment. When the project results were normalized with past comparable projects, the XP approach was approximately twice as productive.

Abrahamsson [1] conducted a controlled case study of four software engineers using XP on a data management project at a Finnish research institute. The project lasted eight weeks with a fixed development schedule and fixed resources. Comparison between the first and second releases yielded the following results: planning estimation accuracy improved by 26%, productivity increased by 12 lines of code (LOC)/hour, and the defect rate remained constant at 2.1 defects/thousand lines of code. Similarly, Maurer and Martel [17] reported a case study of a nine-programmer web application project. The team showed strong productivity gains after switching from a document-centric development process to XP.

Reifer reported the results of a survey of 14 firms spanning 31 projects [21]. Most projects were character-

ized as small pilot studies, for internal use only, and of low risk. It was reported that these projects had average or better than average budget performance and schedule adherence. Projects in the software and telecommunications industry reported product quality on par with nominal quality ratings; e-business reported above par quality ratings; and the aerospace industry reported below par quality ratings for their agile/XP projects.

A year-long case study structured using the XP-EF was performed with a small team (7-11 team members) at IBM [26] to assess the effects of adopting XP practices. Through two sequential software releases, this team transitioned and stabilized its use of a subset of XP practices. The use of a "safe subset" of the XP practices was necessitated by corporate culture, project characteristics, and team makeup. The team improved productivity and improved their post-release defect density by almost 40% when compared to similar metrics from the previous release. These findings suggest that it is possible to adopt a partial implementation of XP practices and to yield a successful project.

El Emam [6] surveyed project managers, chief executive officers, developers, and vice-presidents of engineering for 21 software projects. El Emam found that none of the companies adopted agile practices in a "pure" form. Project teams chose which practices to adopt selectively and developed customized approaches to operate within their particular work contexts. The Sabre-A team showed evidence of an almost pure adoption of XP, with some customizations to fit their environment.

Boehm and Turner acknowledge that agile and plan-driven methodologies each have a role in software development and suggest a risk-based method for selecting an appropriate methodology [5]. Their five project risk factors (team size, criticality, personnel understanding, dynamism, and culture) aid in selecting an agile, plan-driven, or hybrid process (see Figure 1 for an example). The Sabre-A team in this case study is an example of a team that can be classified as characteristically agile.

Robinson and Sharp [22] performed a participant-observer study based on ethnography. The researchers participated with an XP team to examine the relationship between the 12 XP practices and the four XP values: communication, feedback, simplicity, and courage. Robinson and Sharp concluded that the XP practices can be used to create a community that supports and sustains a culture that includes the XP values. However, the specific 12 practices are not the only means for achieving the same underlying values; teams that adopt a subset of the practices can produce a similar culture.

3. Sabre Airline Solutions case study

We add to the body of knowledge about XP by reporting a case study with a Sabre Airline Solutions development team. This study was done as a part of a cooperative research effort between North Carolina State University and several development teams at Sabre Airline Solutions. The Sabre-A team was selected as an example of a team that was characteristically agile and did not need to scale or re-scope XP. Team selection was also influenced by data availability, team size, and the cooperativeness of the team with the researchers. The last factor proved important because the research team was working within a limited time frame.

In this study, we compare the third and the ninth releases of the Sabre-A team's product. From this point forth, we refer to the third release as the "old release" and the ninth release as the "new release." The team used a traditional, waterfall-based software process in the old release. Development for the old release began in early 2001 and lasted 18 months. Work on the new release commenced in the third quarter of 2003. In the two and half years that passed from the beginning of the old release to the beginning of the new release, the team became veterans of XP and customized their XP process to be compatible with their environment.

Detailed data was collected for each release, and much of this data was gathered from historical resources. The old release was developed approximately two years prior to the beginning of this study. The researchers were not present for the old release, and the team was not aware that any research would be done on their product or on their documentation. Consequently, some in-process XP-EF metrics were not available. The research team was present only for a portion of the new release development. Many of the XP-EF metrics were readily available for the new release by examining source code, defect tracking systems, build results, and survey responses. Qualitative data was gathered from team members to aid in understanding quantitative findings. Six of the team's ten full-time members were interviewed during the new release. The interviews were semi-structured, and each interviewee was asked the same set of questions.

The Sabre-A case study will now be described in terms of the XP-EF and its sub-categories. Section 3.1 presents the context of the Sabre-A case study so that results can be interpreted accordingly. Section 3.2 outlines the Sabre-A team's XP use to help understand the extent to which the team actually employs XP.

3.1. Context factors (XP-cf)

The XP-cf utilize six categories of context factors outlined by Jones [11]: software classification, sociological, geographical, project-specific, technological, ergonomic, and an additional category, developmental factors, based upon work by Boehm and Turner [5].

Software classification. In the XP-EF, projects are classified as one of six software types: systems [used to control physical devices]; commercial [leased or marketed to external clients]; information systems [for business

information]; outsourced [developed under contract]; military; or end user [private, for personal use]. The Sabre-A team's product is funded both internally and by customer contribution. No single customer dictates requirements, though customer suggestions are integrated into the product. Since the product is built and marketed to appeal to many customers, we classify this project as *commercial software*.

Sociological factors. Team conditions for both releases are shown in Table 1. In the old release, the turnover consisted of two developers leaving the team and two joining the team. The team gained a new member each time an old member left. These personnel changes were distributed over an 18 month period, making the transitions easier for the team. Three of the team members in the new release worked on the old release. In the new release, developers were under pressure to incorporate more features into the product as the release deadline approached. This impaired the rigorous testing of the product and may have contributed to lower code quality.

Table 1: Sociological factors

Context factor	Old	New
Team Size	6	10
Highest Degree Obtained	None: 1 Bachelors: 3 Masters: 2	None: 0 Bachelors: 8 Masters: 2
Experience Level of Team	6-10 years: 4 <5 years: 2	6-10 years: 5 <5 years: 5
Domain Expertise	Medium	High
Lang. Expertise	Medium	High
Exp. of Proj. Mgr.	Low	Medium
Specialist Available	System architect Delivery manager	
Personnel Turnover	67%	10%
Morale Factors	Layoffs, manager change	High delivery pressure, expected layoffs and team reassignments

Geographical. Table 2 documents the geographical factors. The number of customers more than doubled between the time of the old release and the time of the new release. This led to increased usage of the product as well as the generation of a higher number of requirements.

Table 2: Geographical factors

Context factor	Old	New
Team location	Co-located	
No. customers	Approx. 5	Approx. 11
Customer location	Remote, multi-national, several time-zones.	
Supplier	None	

Project-specific factors. As shown in Table 3, though the number of actual new, changed, and deleted lines in the new release is relatively small, these changes affected numerous classes and functions. The size of the new and changed classes are similar for each release, though many more functions and classes were altered in the old release. Thousands of lines of executable code (KLOEC) are non-blank, non-comment lines.

Table 3: Project-specific factors

Context factor	Old	New
New & Changed User Stories	N/A	32
Domain	Scriptable GUI environment	
Person Months	108	14.7
Elapsed Months	18	3.5
Nature of Project	Enhancement/maintenance	
Constraints	Date constrained, scope constrained, semi-resource constrained	
New/Chged Classes Total Classes	276 903	180 1,337
New/Chged Method Total Methods	1,777 8,233	710 13,301
New/Chged KLOEC	32.4	6.5
KLOEC of New & Changed Classes	67.9	68.9
Component KLOEC	133.8	193.4
System KLOEC	133.8	221.6

Technological. The team's technology factors are summarized in Table 4. Some XP practices, such as continuous integration and collective code ownership, were already a part of the team's waterfall process. These practices were not part of planned XP development but were elements of the existing process. The team used Microsoft® Project to organize and schedule project tasks during the old release. In the new release, the planning game was used to establish release and iteration plans. User stories and task estimates were recorded in a Microsoft Excel spreadsheet that was also used to forecast release points and iterations based on the team's project velocity.

Defect prevention and removal practices also changed between releases. In both releases, the team had dedicated testers to perform system- and integration-level testing. However, ad hoc testing by developers and testers served as the primary means to identify potential problems in the code during production of both releases. A code review policy was in place during the old release, but it was not strictly enforced. Pair programming took the place of code reviews in the new release. If a programmer did not pair when developing source, it was

common to ask a colleague for a peer review. Customer tests were also integrated into the new release.

Table 4: Technological factors

Context factor	Old	New
Software Development Methodology	Waterfall with some XP practices	XP
Project Management	Microsoft Project	Planning game
Defect Prevention & Removal Practices	Code reviews, QA team, ad hoc testing	Pair programming, unit testing, QA team, ad hoc testing by developers and customer
Language	Java, C++	Java
Reusable Materials	Third party libraries	Third party libraries, unit test suites, automated build machine

Ergonomic. As shown in Table 5, in the old release the team sat in semi-private cubicles, most of which were along the same hallway. In the new release, the team sat in an open office environment with furniture and pair programming stations arranged by the developers. A large whiteboard was available, and printouts of iteration plans and process metrics were attached to the walls. Test equipment and an automated build machine were also present in the room.

Customer interaction changed considerably between the two releases. In the old release, the product delivery team and the lead developers were responsible for customer communication. Requirements were delivered to developers in a variety of ad hoc ways, and the developers frequently visited customer sites around the world. In the new release, a representative from Sabre's product marketing department served as the XP customer. This representative made final decisions on product functionality, and was on-site approximately half of the time and was available by phone or e-mail at other times.

Table 5: Ergonomic factors

Context factor	Old	New
Physical Layout	Semi-private cubicles	Open office XP lab
Distraction of Office Space	Low	Medium
Customer Communication	Visit customer sites worldwide, e-mail, other ad hoc methods	Customer representative on-site, available through phone and e-mail

Developmental. The Sabre-A development team's Boehm-Turner risk factors for the old release are graphed on a polar chart's five axes, shown in Figure 1. These factors were evaluated both through empirical information and through consultation with the team's development lead. The team's shape indicates that a hybrid "partially agile, partially plan-driven method" is appropriate. The developmental factor that appears to necessitate plan-driven practices is culture.

The team's risk factors for the new release are shown in Figure 2. The team's size increased, but it still remains on the agile portion of the axis. Also, culture and dynamism moved toward the agile ends of their respective axes. The change in culture can be attributed to the addition of personnel with more chaos-tolerant personality types. Also, as Sabre Airline Solutions shifted toward XP as a whole, all developers had to adapt to an environment that embraced agility and open-space programming labs instead of waterfall and private cubicles.

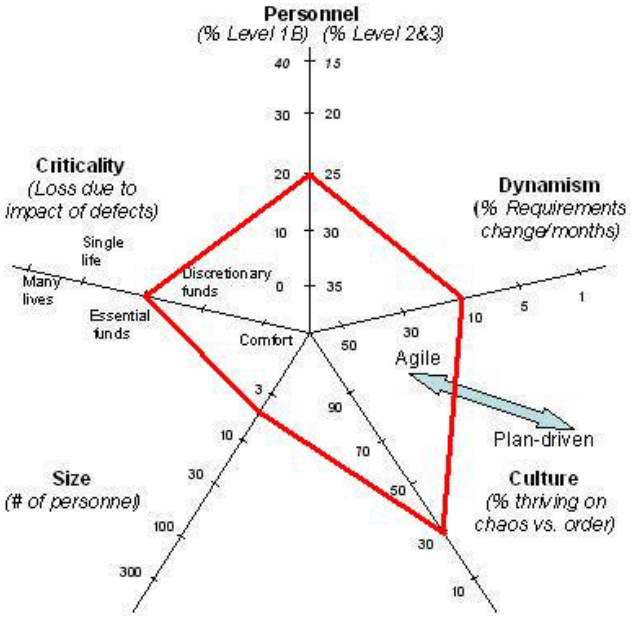

Figure 1: Old release developmental factors (adapted from [5])

The increase in dynamism may be attributed to several factors. The number of customer-requested changes and enhancements grew as the system evolved to include more features. Also, the number of external customers increased in the new release, escalating the number of customer-driven enhancement requests. The personnel factor, which pertains to skill level, remained the same. Though some personnel turnover occurred between the releases, the skill and experience of new team members were comparable to that of the team members that left.

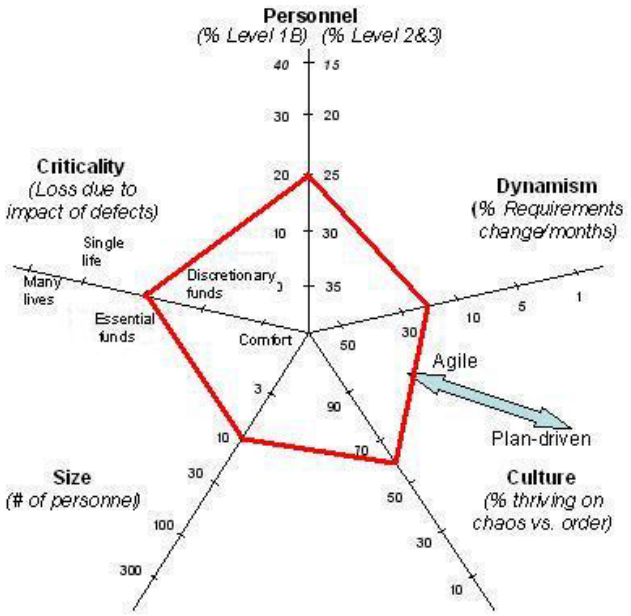

Figure 2: New release developmental factors

3.2. Adherence metrics (XP-am)

Most companies that use XP adopt the practices selectively and develop customized approaches to operate within their particular contexts [6]. The XP adherence metrics enable case study comparison, the study of XP practice interaction, and the determination of contextually-based, "safe" XP practice subsets. These metrics also provide insight into whether a team has adopted XP's core values. Unfortunately, many of the objective metrics in the XP-am could not be gathered for this case study since most of the metric information in this study is collected from historical data. For those metrics that could not be calculated, anecdotal evidence was solicited from the team leader, who was present in both releases. Interviews were also conducted with team members to aid in understanding and in substantiating this evidence.

The Shodan Adherence Survey (described fully in [25] and adapted from [14]) is an in-process, subjective means of gathering XP adherence information from team members. Survey respondents report the extent to which they use each practice on a scale from 0% (never) to 100% (always). The survey was taken during new release development; the survey was not administered during the old release. Eleven team members took the survey, including one team member who left during the new release.

We present the combined results of these adherence metrics based upon three categories: planning (Table 6), coding (Table 7), and testing (Table 8). For each section, we present the results of the XP-am metrics followed by the Shodan survey results. We also provide substantiation based on interviews with team members.

3.2.1. Planning adherence metrics. The release length in the old release was appropriate for the waterfall process and a less volatile requirement set. During the time of the new release, the team operated almost exclusively on iteration plans, and their product was continuously available via an automated build machine. Product versions were primarily dictated by the marketing department. Versioning served a means for the business unit to track production rather than as a development milestone. For the new release, stand-up meetings were conducted every morning in the lab. The marketing representative serving as the XP customer was on-site approximately half the time and was available through phone and e-mail the rest of the time.

According to team members, the daily stand-up meetings were a valuable asset for problem resolution and team communication. Team members also stated that the short iterations were helpful and allowed them to create more accurate estimates. Also, by estimating planning only a few weeks at a time, there was less anxiety about completing future tasks. When using the waterfall method, the developers were often under pressure to meet inaccurate task schedules that were created several months earlier in the development cycle.

Table 6: XP adherence metrics – Planning

Planning metric	**Old**	**New**
Release Length	18 months	3.5 months
Iteration Length	None	10 days
Requirements added or removed to Total Shipped Ratio	N/A	N/A
Subjective (Shodan)	**Mean (σ^2)**	**Mean (σ^2)**
Stand up meetings	N/A	92.7% (10.1)
Short Releases	N/A	91.8% (11.7)
Customer Access / On-site Customer	N/A	89.1% (12.2)
Planning Game	N/A	84.5% (10.4)

3.2.2. Coding adherence metrics. In the old release, pairing did not occur, but a code review policy was in place. According to the team leader, in the new release, the team paired approximately half the time. In interviews, some developers noted their dislikes for pairing because of differences in age and in expertise. Almost all interviewees disliked mandated pairing and saw no value in pairing on trivial tasks. However, everyone agreed that it was a valuable process for solving problems and overcoming technical difficulties. One developer noted that the time required to switch pairs is often underestimated when this occurs several times per day. Developers agreed that collective ownership is valuable and stated that the group shares code often. Yet, some people still regard their code as proprietary, particularly highly-specialized or fragile pieces of code. The metaphor score

was low since no analogy could be created for the system. However, the team used a system of names in the product.

Table 7: XP adherence metrics – Coding

Coding metric	Old	New
Pairing Frequency (anecdotal)	0%	50%
Inspection Frequency (anecdotal)	60%	20%
Solo Frequency (anecdotal)	100%	50%
Subjective (Shodan)	**Mean (σ^2)**	**Mean (σ^2)**
Pair Programming	N/A	67.3% (16.2)
Refactoring	N/A	66.4% (18.6)
Simple Design	N/A	78.2% (9.8)
Collective Ownership	N/A	70.0% (18.4)
Continuous Int.	N/A	81.8% (6.0)
Coding Standards	N/A	90.0% (6.3)
Sustainable Pace	N/A	80.0% (11.8)
Metaphor	N/A	56.4% (27.3)

3.2.2. Testing adherence metrics. The team's new release test coverage reflects a concerted effort to adopt automated unit testing. Test coverage is averaged over the entire component, not just the new and changed portions. The team wrote automated unit tests before adding or changing functionality and before refactoring code. The ratio of Test Classes to New and Changed classes indicates that more than half of the new and changed classes have a corresponding test class. Nearly all of the new classes written have a corresponding test class. Static legacy code was not always unit tested.

Table 8: XP adherence metrics – Testing

Testing metric	Old	New
Test Coverage (stmnt)	N/A	32.9%
Test Run Frequency (anecdotal)	None	1.0
Test Class to Story Ratio	N/A	3.22
Test Classes to New/Changed classes Ratio (JUnit only)	0.036	0.572
New Classes with corresponding Test Classes (JUnit only)	4.8%	80.0%
Test LOC / Source LOC	0.054	0.296
Subjective (Shodan)	**Mean (σ^2)**	**Mean (σ^2)**
Test First Design	N/A	67.3% (14.2)
Automated Unit Tests	N/A	78.2% (23.2)
Cust. Acceptance Tests	N/A	56.4% (20.2)

In interviews, team members stated that automated tests were difficult to write for some GUI components because of limitations in available unit testing technology. Developers noted that they enjoy the rapid feedback afforded by unit testing. The developers' view corresponds with that of Mugridge [18], who likens test-driven development to the repeatable experiments of the scientific method. Some developers stated that there was an adequate amount of customer acceptance tests, however these tests were not automated. The XP customer sometimes had difficulty creating acceptance tests due to time constraints and system scope.

4. Results (XP-om)

Of utmost importance to decision makers is whether or not adopting XP aids in productively creating a higher quality project. We provide quantitative output measurements of productivity and quality, as well as qualitative information gathered from team member interviews. We performed member checking with team members to review our findings and receive any final feedback on their XP experience.

Defect information was collected from the team's defect tracking system. We now explain the method that was used to identify and to classify defects for both the old and the new releases. Internal (pre-release) defects were identified by examining the "originating customer" field of each defect header. According to the team's quality tracking lead, a defect was found internally if the originating customer field did not refer to an external customer. If the originating customer field indicated an external customer, the defect was considered to be a post-release defect. Furthermore, any Severity 3 defects were classified as internal defects since the team's severity schema states that all Severity 3 defects were found during development regardless of their impact on the system. Also, all defects are reviewed by a quality assurance (QA) panel to determine if they are, indeed, software defects. Any defect entry that had not been classified by the QA panel as a defect was not counted. All entries that addressed the same problem (duplicate defects) were counted as one defect.

In interviews conducted during the new release, team members were asked to discuss their likes and dislikes of the XP process. All team members individually and independently responded that increased communication amongst the team was the most beneficial aspect of XP. Stand-up meetings allowed the developers to understand better the work being done on all aspects of the project. A general increase in team communication also allowed for problems to be resolved more quickly. It was stated that often one developer would overhear a problem someone else was having and was able to suggest a solution or the two were able to collaborate on the problem. Developers also enjoyed the rapid feedback afforded by unit

testing and pair programming during the code and design phases.

When citing their dislikes about their XP process, most developers cited mandatory pair programming as one drawback to their process. While they considered pair programming to be valuable for problem solving and implementing difficult functionality, it was often considered an inefficient use of time to pair on menial tasks. The interviewees also noted that, while the open lab increased communication, the noise level could sometimes become a distraction.

The Sabre business-related results, structured via the XP-Outcome Measures (XP-om), are shown in Table 9. A relative scale is used to protect proprietary information. Results are measured with respect to the "component under study," which is the body of code listed as the KLOEC of New & Changed Classes in Table 4. The entire Component KLOEC is not factored into the result because it does not accurately reflect the amount of work that took place during new release development. The rest of the System KLOEC is not counted because it is designed and built as a separate component from the component under study.

Table 9: XP outcome measures (relative scale with the old release at 1.0)

XP Outcome Measures	Old	New
Internally-Visible Quality (test defects/KLOEC of code)	1.0	0.35
Externally-Visible Quality (released defects/KLOEC of code four months after release)	1.0	0.64
Productivity (stories/person-month) Relative KLOEC person-month	N/A 1.0	- 1.46
Customer Satisfaction (approx)	N/A	High
Morale (via survey)	N/A	68.1%

Internally-Visible Quality. Internal (pre-release) defect density, which concerns defects identified by Sabre testers, improved by 65%. These findings support the alternative hypothesis ***H1$_1$***: *when used by teams operating within this specified context, the use of a specified subset of XP practices leads to an improvement in pre-release quality*. Testing was done by the dedicated testers associated with the Sabre-A team and the developers performing ad-hoc functional testing and unit testing throughout development. We temper these results by noting that these measurements may be skewed because the old release was subject to 18 months of continuous internal testing, while the new release was internally tested for only 3.5 months.

Externally-Visible Quality. We observed that the number of defects found in the customer's production system has improved by approximately 35%. These findings support the alternative hypothesis ***H2$_1$***: *when used by teams operating within this specified context, the use of a specified subset of XP practices leads to an improvement in post-release quality.* The defect numbers presented reflect a collection period of four months after each release.

Post-release defect counts were impacted by several important factors. One major influencing factor was the doubling of the number of external customers between the old and the new releases. The old release was not used extensively since most customers were awaiting the completion of a new, concurrently-developed version of the product in progress at that time. However, the new release was used significantly by more customers, some of which had a more complex problem domain than customers of the old release. Evidence of similar customer use of the product in the old and new releases and an assessment of feature complexity would aid in determining the accuracy of the post-release defect comparison.

The team's defect rates were well-below industry averages [9, 11] in both the old and the new releases. Furthermore, no Severity 1 defects were reported for the new release. A Severity 1 defect is classified as a defect that causes the customer's system to be unusable, whereas a Severity 2 defect is a defect where the customer's system is working badly and their operations and/or revenue is negatively impacted, but a work-around exists for the defect. The post-release defect severity distribution for both releases is shown in Table 10.

Table 10: Post-release defect severity distribution

Severity	Old	New
1	12%	0%
2	88%	100%

Productivity. The results suggest that team had increased their productivity (in terms of KLOEC/PM) by approximately 50% between the old release and the new release. These findings support the alternative hypothesis ***H3$_1$***: *when used by teams operating within this specified context, the use of a specified subset of XP practices leads to an improvement in developer productivity.* A decrease in the relative complexity of implemented features can potentially affect this metric. Recording the amount of developer effort spent on non-production activities, such as installation, training, and customer support activities, would also help account for variations in these results. Furthermore, the team's increased familiarity with the application domain in the new release may have affected the results. Unfortunately, the user story per person month metric was unavailable since the team did not utilize user stories during the old release. This metric would help in gauging the amount of actual functionality produced by the developers, rather than the amount of code they produce.

Customer satisfaction. Proponents of XP profess that customers are more satisfied with the resulting project because the team produced what the customer *actually* wanted, rather than what they had originally expressed they wanted. In the future, we plan to author and validate a customer satisfaction survey instrument. Unfortunately, we could not contact the external customers for this project. Therefore, we can draw no inferences regarding the null hypothesis $H4_0$ regarding customer satisfaction. Anecdotally, however, the customers were very satisfied with the new release product. One customer expressed to the team that it was one of the most professionally-developed products he had ever used. Feedback from other customers was unavailable.

Morale. Morale was assessed via an additional question on the Shodan Adherence Survey. The question read, "How often can you say you are enjoying your work?" In interviews, several team members stated that they enjoyed their jobs and enjoyed the XP methodology more than the waterfall method. However, we cannot verify that the team members who made these comments worked on the old release. Survey results indicated that team members felt they were working at a sustainable pace. However, since there is no assured basis for comparison in the old release, we can draw no inferences about the null hypothesis $H5_0$ regarding morale.

5. Case study limitations

The Sabre-A team in this case study is characteristically agile and has organizational and managerial support to use the XP methodology. Therefore, their successful implementation may not be representative of teams that are not characteristically agile (e.g. large, distributed teams) and/or do not have management and organizational support. This case study does not provide any insight into extending the range of applicability of XP beyond small, co-located teams.

The comparison is made between two releases of the same product. We sought to reduce internal validity concerns by studying the same software project with a team comprised largely of the same personnel. However, there are many differing characteristics between the two projects that must be kept in mind when examining the results. The amount of new and changed LOEC in the new release was approximately one-fifth the size of the old release, and smaller projects are often considered to be less complex. However, the system size increased 65% between the old and new releases, and overall system cyclomatic complexity increased by 20%. Also, team members expressed that feature complexity was much higher in the new release when compared to the old release. Furthermore, the amount of test code written in the new release was not included in effort calculations or total system size. Therefore, productivity and effort results in the new release may be underestimated.

6. Summary and future work

As is often the case with software engineering innovation, empirical research on the efficacy of XP is lagging behind adoption of the methodology and its practices. Both commercial decision-makers and researchers are awaiting empirical evidence of the efficacy of XP. We contribute such evidence by performing an industrial case study with a Sabre Airline Solutions development team. This case study compares two releases of the Sabre team's product: one completed prior to adoption of XP and the other completed after two years of XP experience. We examined five hypotheses related to the results of the team's adoption of XP practices in their context. We summarize our case study findings in the format suggested by Fenton [8] in Table 11.

We remind the reader that these results are based on one case study in one particular context. Our results can be used to build up the weight of evidence about XP, but we cannot conclusively accept or reject the hypotheses. Our findings suggest that the adoption of XP practices can improve programmer productivity and can improve product quality. All findings should be taken in the context of a product that grew and evolved significantly in the time period between the two releases under study.

Table 11: Case study summary

When used by teams operating within the specified context, the use of a specified subset of XP practices leads to an improvement in . . .		
Number	**Alternative Hypothesis**	**Case study evidence?**
1	*. . . pre-release quality*	Yes – 65% reduction in defects found in test
2	*. . . post-release quality*	Yes – improved by a factor of 35% in defects found in customer systems
3	*. . programmer productivity*	Yes – 50% increase in code output
4	*. . . customer satisfaction*	N/A
5	*. . . team morale*	N/A

We are currently analyzing two other case studies conducted at Sabre Airline Solutions. Three additional case studies structured by the XP-EF are about to commence. The results of this family of case studies and that of other researchers will build an empirical body of results concerning XP in various contexts in various organizations.

Specifically, we intend to examine trends in the variety of subsets of XP practices that emerge.

Acknowledgements

The authors wish to thank the individuals on the Sabre development team for participating in this study, and Kerry Rodgers and Scott Frederick for their invaluable assistance. We also wish to thank the anonymous reviewers for their valuable comments and the students in the North Carolina State University Software Engineering Reading Group for their helpful suggestions on this paper. This research was supported by Sabre Airline Solutions. Lynn Cunningham participated in this research through support from the National Science Foundation Distributed Mentor Project.

References

[1] P. Abrahamsson, "Extreme Programming: First Results from a Controlled Case Study," in *29th EUROMICRO Conference*. Belek, Turkey: IEEE, 2003.

[2] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New Directions in Agile Methods: A Comparative Analysis," in *International Conference on Software Engineering (ICSE 203)*. Portland, OR: IEEE Computer Society, 2003, pp. 244-254.

[3] V. Basili, F. Shull, and F. Lanubile, "Building Knowledge Through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25, pp. 456 - 473, 1999.

[4] K. Beck, "Extreme Programming Explained: Embrace Change." Reading, Massachusetts: Addison-Wesley, 2000.

[5] B. Boehm and R. Turner, "Balancing Agility and Discipline: A Guide for the Perplexed." Boston, MA: Addison Wesley, 2003.

[6] K. El-Emam, "Finding Success in Small Software Projects," *Agile Project Management*, vol. 4, 2003.

[7] N. E. Fenton and S. L. Pfleeger, "Software Metrics: A Rigorous and Practical Approach," Brooks/Cole Pub Co., 1998.

[8] N. E. Fenton, "Conducting and Presenting Empirical Software Engineering," *Journal of Empirical Software Engineering*, vol. 6, pp. 195-200, 2001.

[9] B. B. S. I. Group, "Benchmarking of Software Engineering Practices at High Maturity Organizations," Bangalore Software Process Improvement Network, 2001.

[10] R. Jeffries, A. Anderson, and C. Hendrickson, "Extreme Programming Installed," in *The XP Series*, K. Beck, Ed. Upper Saddle River, NJ: Addison Wesley, 2001.

[11] C. Jones, *Software Assessments, Benchmarks, and Best Practices*. Boston, MA: Addison Wesley, 2000.

[12] B. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case Studies for Method and Tool Evaluation," in *IEEE Software*, vol. 12, July 1995, pp. 52-62.

[13] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," in *IEEE Transactions on Software Engineering*, vol. 28, 2002, pp. 721-733.

[14] W. Krebs, "Turning the Knobs: A Coaching Pattern for XP Through Agile Metrics," in *Extreme Programming/Agile Universe*, L. Williams, Ed. Chicago, IL: Springer, 2002.

[15] M. Marchesi and G. Succi, "Extreme Programming Examined," in *XP Series*, K. Beck, Ed. Boston: Addison Wesley, 2001.

[16] M. Marchesi, G. Succi, D. Wells, and L. Williams, "Extreme Programming Perspectives," in *XP Series*, K. Beck, Ed. Boston: Addison Wesley, 2002.

[17] F. Maurer and S. Martel, "Extreme Programming: Rapid Development for Web-Based Applications," in *IEEE Internet Computing*, vol. 6, Jan/Feb 2002, pp. 86-91.

[18] R. Mugridge, "Test Driven Development and the Scientific Method," presented at Agile Development Conference, Salt Lake City, UT, 2003, 47-52.

[19] P. Nour, "Ontology-based Retrieval of Software Engineering Experiences," in *Computer Science*. Calgary, Alberta, Canada: University of Calgary, August 2003.

[20] C. Potts, "Software Engineering Research Revisited," in *IEEE Software*, September 1993, pp. 19-28.

[21] D. J. Reifer, "How to Get the Most out of Extreme Programming/Agile Methods," in *2nd XP and 1st Agile Universe Conference*. Chicago, IL: Springer LNCS 2418, August 2002, pp. 185-196.

[22] H. Robinson and H. Sharp, "XP Culture: Why the twelve practices both are and are not the most significant thing," presented at 1st International Agile Development Conference (ADC '03), Salt Lake City, UT, 2003, 12-21.

[23] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," in *IEEE Transactions on Software Engineering*, vol. 25, 1999, pp. 557-572.

[24] D. Wells and L. Williams, "Extreme Programming and Agile Methods -- XP/Agile Universe 2002," in *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2002.

[25] L. Williams, W. Krebs, and L. Layman, "Extreme Programming Evaluation Framework for Object-Oriented Languages -- Version 1.3," North Carolina State University Department of Computer Science, Raleigh, NC, TR-2004-11, April 6, 2004.

[26] L. Williams, W. Krebs, L. Layman, and A. Antón, "Toward a Framework for Evaluating Extreme Programming," presented at Proceedings of the Eighth International Conference on Empirical Assessment in Software Engineering (EASE 04), 2004, in press.

[27] W. Wood and W. Kleb, "Exploring XP for Scientific Research," *IEEE Software*, vol. 20, pp. 30-36, 2003.

[28] R. K. Yin, "Case Study Research: Design and Methods," in *Applied Social Research*, vol. 5, D. J. Rog, Ed., Third ed. Thousand Oaks, CA: Sage Publications, 2003.

[29] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," in *IEEE Computer*, vol. 31, May 1998, pp. 23-31.