Motivations and Measurements in an Agile Case Study

Lucas Layman
North Carolina State University
900 Main Campus Dr., Rm. 197
Raleigh, NC 27695
+1.919.513.5082
lmlayma2@ncsu.edu

Laurie Williams
North Carolina State University
900 Main Campus Dr., Rm. 198
Raleigh, NC 27695
+1.919.513.4151
williams@csc.ncsu.edu

Lynn Cunningham
Clarke College
60 South Algona Street
Dubuque, IA 52001
+1.563.556.4637
lynn.cunningham@clarke.edu

ABSTRACT
With the recent emergence of agile software development technologies, the software community is awaiting sound, empirical investigation of the impacts of agile practices in a live setting. One means of conducting such research is through industrial case studies. However, there are a number of influencing factors that contribute to the success of such a case study. In this paper, we describe a case study performed at Sabre Airline Solutions evaluating the effects of adopting Extreme Programming (XP) practices with a team that had characteristically plan-driven risk factors. We compare the team's business-related results (productivity and quality) to two published sources of industry averages. Our case study found that the Sabre team yielded above-average post-release quality and average to above-average productivity. We discuss our experience in conducting this case study, including specifics of how data was collected, the rationale behind our process of data collection, and what obstacles were encountered during the case study. We also identify four factors that potentially impact the outcome of industrial case studies: availability of data, tool support, co-operative personnel and project status. We believe that recognizing and planning for these factors is essential to conducting industrial case studies, and that this information will be helpful to researchers and practitioners alike.

Categories and Subject Descriptors
K.6.3 [**Management of Computing and Information Systems**]: Software Management — *software process*; and D.2.9 [**Software Engineering]**: Management — *life cycle, programming teams*

General Terms
Management, Measurement, Experimentation, Human Factors

Keywords
Agile software development, extreme programming, case studies

1. INTRODUCTION
The introduction of Extreme Programming (XP) [5] into mainstream software development has been met with both enthusiasm and skepticism. For decision-makers, an empirical, quantitative investigation is beneficial for investigating XP's efficacy. A survey of 90 software engineering researchers and practitioners [24] revealed that industry is influenced by compelling evidence on the effectiveness of a technique in live situations in an environment such as their own. One method for conducting research in a live industrial setting is through realistic, methodologically-defensible case studies. Case studies are valuable because they involve factors that staged experiments generally do not exhibit, such as scale, complexity, unpredictability, and dynamism [18]. However, Zelkowitz and Wallace [25] reported that less than 10% of papers in the respected journals they examined involved a case study.

In order for case study results to have meaning, it is necessary to record contextual information and to implement measures that satisfy the goals of the study. For instance, in our case studies of XP practices [16, 21], we record context information about the team and project under study, we measure the team's usage of XP practices, and we measure the business-related results (such as productivity and quality) of the project. However, data collection and software process measurement are not simple tasks. Furthermore, when studying agile processes in particular, it is desirable that any metrics collection program is lightweight and unobtrusive to the team's daily activities. In the course of conducting our case studies, we have observed several critical factors that impact data collection. We have found that the ability to collect even a lightweight set of metrics is heavily influenced by the presence of historical data, by the co-cooperativeness of personnel, by the availability of data, and by tool support.

This paper discusses the process of conducting an industrial case study with an agile team at Sabre Airline Solutions. In order to facilitate data collection and to guide our agile case studies, we have created the Extreme Programming Evaluation Framework (XP-EF) [21]. We discuss our rationale behind the processes for collecting certain metrics so that we can create a methodologically defensible case study. We analyze those factors which both enabled and prevented the collection of the full range of XP-EF metrics. This information will be useful to those practitioners who are considering the implementation of a software metrics program in conjunction with their agile process.

The remaining sections of this paper are organized as follows: Section 2 describes related work on case study research, the XP-EF, and XP studies in general; Section 3 describes the Sabre Airline Solutions case study and the team's adoption of XP practices; Section 4 provides a discussion of our case study findings and lessons learned.

2. RELATED WORK
The following sections present background information on case study research, the XP-EF, and existing empirically-based XP studies.

2.1 Case Study Research

Case studies can be viewed as "research in the typical" [8, 13]. As opposed to formal experiments, which often have a narrow focus and an emphasis on controlling context variables, case studies in software engineering test theories and collect data through observation of a project in an unmodified setting [25]. However, because the corporate, team, and project characteristics are unique to each case study, comparisons and generalizations of case study results are difficult and are subject to questions of external validity [14]. Nevertheless, case studies are particularly important for industrial evaluation of software engineering methods and tools [13]. Researchers become more confident in a theory when similar findings emerge in different contexts [13]. By performing multiple case studies and/or experiments and recording the context variables of each case study, researchers can build up evidence through a family of experiments.

When conducting a case study, or any form of research, it is important to consider the validity of that research. Yin [23] describes four components of experimental validity. *Construct validity* regards whether the measures and metrics in place are appropriate for capturing the desired results. *Internal validity* concerns changing or influencing factors in the object(s) under study (in our case, a software project and software team) that may impact the results. *External validity* regards whether the results of one study can be generalized outside the context of that study. Finally, *experimental validity* concerns whether the research was executed in a scientifically-defensible manner with suitable attention to detail. Replication of case studies addresses threats to experimental validity [3].

2.2 Extreme Programming Evaluation Framework

The Extreme Programming Evaluation Framework (XP-EF) is a benchmark for expressing XP case study information [21]. The XP-EF is a compilation of validated and proposed metrics designed for expressing the XP practices an organization has selected to adopt and/or modify, and the outcome thereof. We desired for all metrics to be parsimonious and lightweight so that they could be collected by a small team without a dedicated metrics specialist. 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 about their project via the XP Context Factors (XP-cf). Recording 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 enables one to express concretely and comparatively via objective and subjective metrics the extent to which a team utilizes the XP practices. By examining multiple XP-EF case studies, the XP-am also allows 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 enables one to assess and to report how successful or unsuccessful a team is when 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 [21]. Instructions and templates for measuring and reporting XP case study data via XP-EF Version 1.4 have been documented in [20].

2.3 XP Studies

Practitioners and researchers have reported numerous, predominantly anecdotal and favorable, studies of the XP methodology. However, some empirically-based XP case studies do exist. Abrahamsson [1] conducted a controlled case study of four software engineers using XP to implement data management software. Comparison between the first and second releases of the project yielded increases in planning estimation accuracy and productivity while the defect rate remained constant. 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 an industrial survey conducted to determine if agile methods/XP reduce costs and improve development time [19]. Results from 14 firms spanning 31 projects were collected. Most projects were characterized as small pilot studies, for internal use only, and of generally low risk. Most 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 a below par quality rating for their agile/XP projects.

A year-long case study was performed with a small team (7-11 team members) at IBM to assess the effects of adopting XP [21]. The case study was structured using the XP-EF. Through two software releases, this team transitioned to 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, reduced pre-release defect density by 50%, and achieved a 40% reduction in the post-release defect density when compared to the same metrics from an earlier release. A similar case study was performed with another team at Sabre Airline Solutions [16]. This team (denoted as the Sabre-A team) was selected as an example of a "characteristically agile" team, whereas the team in this case study is "characteristically plan-driven," as discussed in section 3.2.1. The study compared the business-related results of a product developed by the Sabre-A team using traditional methods and the same product further developed using XP. The study showed a 65% reduction in the product's pre-release defect rates, a 35% reduction in the post-release defect rates, and a 50% improvement in productivity (as measured by code output) in the XP release.

3. SABRE-P CASE STUDY

In this section, we describe the details of applying the XP-EF framework to our case study at Sabre Airline Solutions. We provide discussion of the important components and influencing factors of our study, and discuss the method behind recording various measures.

In this study, we examine the 13th release of the Sabre team's product. To differentiate between the Sabre team in this study, and those in other case studies, we refer to this team as Sabre-P (plan-driven). At the time of this release, the team had been using XP for approximately 20 months. The release began in the second quarter of 2003 and lasted 5 months. Data collection for this case study was largely dependent on historical data. Integration and build tools allowed the researchers to gather code samples, and a web-based defect-tracking system provided a means for gathering quality information. Similarly, the team's project planning and tracking tool (an augmented Microsoft® Excel spreadsheet) was used for productivity and schedule analysis. The Shodan Adherence Survey (described in Section 3.3) was used to gather subjective information of XP practice usage from team members during the new release. An e-mail questionnaire was sent to developers approximately eight months after the completion of the release under study in order to solicit qualitative information about their use of XP practices and any obstacles in adopting these practices.

We now describe this case study in terms of the XP-EF and its sub-categories. In the case of all data and metrics collected, we discuss our methods of collection and the rationale behind those methods, and we highlight difficulties thus encountered. In each of the data tables below, we include a column to indicate how the data was collected based on the following key:

Table 1: Data Source Key

Source	Key
Defect tracking	DT
Development leader	DL
Developer questionnaire	DQ
Observation	OB
Project tracking	PT
Source code	SC
Survey	SU
Test suite / test tools	TS

3.1 Team and Project Selection

Our case study was performed with a Sabre Airline Solutions development team (the Sabre-P team) in the United States. This study was done as a part of a cooperative effort between researchers at North Carolina State University and several development teams at Sabre Airline Solutions. Before conducting any case study, it is important to select a sample that is representative of the company or organization as a whole in order to reduce threats to external validity. Kitchenham notes that project selection is not always a decision that can be made by the researcher, since a participating company may only commit one or two teams/projects to the research [13]. One can select based on project age, programming language, development methodology, risk factors, etc.

The team reported in this paper was selected to participate in the study because they were classified as "characteristically plan-driven" based upon the five developmental risk factors suggested by Boehm and Turner [6] for evaluating a team's agile or plan-driven characteristics. These risk factors include requirements dynamism, team size, personnel skill, criticality, and team culture. We were also able to select other teams for separate case studies based on varying risk factors. By selecting multiple teams with differing risk factors, we hoped to acquire samples that were representative of teams within the development organization. A more thorough discussion of the Sabre-P team's risk factors may be found in Section 3.2.1. Team selection for our research project was also influenced by data availability, team size, and the cooperativeness of the team with the researchers.

3.2 Context Factors

Drawing conclusions from empirical studies in software engineering is difficult because the results of any process largely depend upon the project setting. One cannot assume a priori that a study's results generalize beyond the specific environment in which it was conducted [3]. Therefore, recording an experiment's context factors is essential for fully understanding the generality and the utility of the findings. Context is also beneficial for understanding the similarities and the differences between the case study and one's own environment. The XP-Context Factors utilize developmental factors, based upon work by Boehm and Turner [6], and the seven categories of context factors outlined by Jones [9]: software classification, ergonomic, sociological, project-specific, technological, geographical.

3.2.1 Developmental factors

Some proponents of XP claim that it can work in almost any setting, while others warn that XP may not be suitable for large teams or in a safety-critical environment [6]. Recording a team's developmental factors can aid in investigating these claims. The Sabre-P development team's Boehm-Turner [6] risk factors for the release under study are graphed on a polar chart's five axes, shown in Figure 1. Because most of the data points are toward the periphery of the graph, the Sabre-P team's shape indicates that a hybrid "partially plan-driven, partially agile method" is appropriate. The developmental factors that appear to necessitate plan-driven practices are personnel, dynamism and culture.

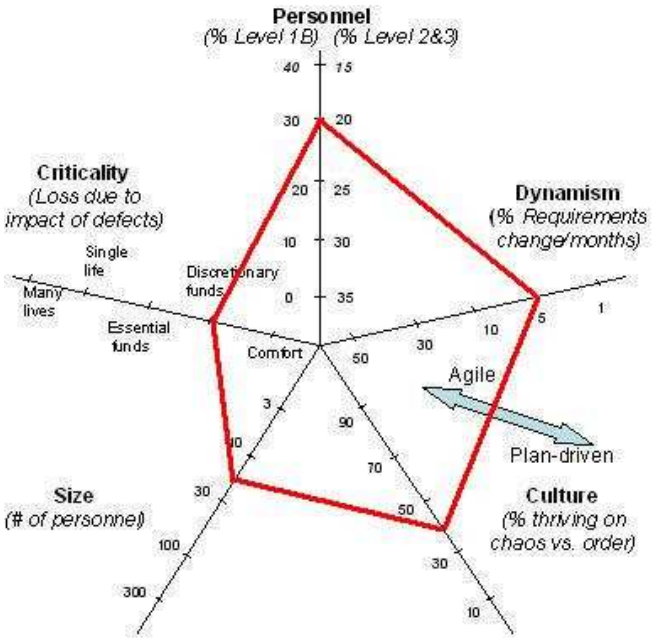

Figure 1: Sabre-P Developmental Factors

3.2.2 Sociological

The team's sociological factors are summarized in Table 2. The Sabre-P team was comprised of 15 developers, one dedicated tester, and several specialist personnel (such as DBAs, UI layout designer, etc.). The team members had varied amounts of experience and education. Personnel turnover during the release under study was low (5%) and consisted of one person leaving the team prior to development began on this release.

When documenting personnel characteristics, it is important to distinguish whether one is counting only software developers, or whether one is including project managers or testers in the measurement. In this study, we count only software developers, but include a dedicated tester in the counts for team size. This tester was present in the software lab and contributed to development on a regular basis. Collecting the personnel turnover rate involved a consultation with the team leader, who identified those team members who left prior to and during each release. The accuracy of such information may be in question when dealing with a non-recent release. However, calculating turnover would be trivial if an XP tracker or someone in a similar role recorded the number of developers present during each iteration/release. Assessing factors such as domain expertise and language expertise also involved inquiry of the development lead. It may be possible to evaluate these factors in a more objective manner through a standardized examination, but this was beyond the scope of our study.

Table 2: Sociological Factors

Context Factor	Value	Source
Team Size (developers)	15+1 tester	PT
Team Education Level	Bachelors: 8 + 1 tester Masters: 6 PhD: 1	DL
Experience Level of Team	1-5 yrs: 6 + 1 tester 6-10 yrs: 5 11-15 yrs: 3 16+ years: 1	DL
Domain Expertise	Medium	DL
Language Expertise	High	DL
Experience of Proj. Mgr.	High	DL
Specialist(s) Available	Dedicated tester, dedicated DBA, configuration manager, web-services specialist	DL
Personnel Turnover	5%	PT
Morale Factors	None	DL

3.2.3 Project-specific factors

As shown in Table 3, the Sabre-P team's product is a large web application combined with a back-end batch component that together total over 1 million lines of executable code (>1,000 KLOEC). Development during the release under study lasted approximately five months and involved enhancements to the web application component of the system (487 KLOEC), with the total number of new and changed classes amounting to 202 KLOEC and new and changed lines of code totaling 26.4 KLOEC. The new and changed lines, methods and classes are with respect to the release developed immediately prior (12th version) to the release under study (13th version). Also, there was a two-month feature freeze before the release point during which the entire team engaged in end-to-end testing of the system.

The person-month metric is a traditional measure of effort that can be calculated by knowing number of personnel present during the course of a project as well as the elapsed time of development. Basic XP project tracking advocated by Beck [4] provides sufficient information to perform this calculation. Other effort metrics that we gathered involve the amount of change to the code base that occurs during release development. Tools exist that can compare two file systems and determine those files with new, changed, and deleted lines. One can thereby isolate those files with changes in them; in the case of object-oriented languages, this technique could possibly be used to identify new and changed classes. However, many tools are not context-aware and cannot separate significant changes (changes in executable lines) with insignificant changes (changes that do not impact functionality). Also, to our knowledge, no tool exists to count the number of new and changed methods in a project. In this case study, we used the Beyond Compare[1] tool to identity new and changed classes and to count the new, changed, and deleted lines of code. From these classes, new and changed methods were manually counted; a time-consuming task that may be prone to human error.

Table 3: Project-specific Factors

Context Factor	Value	Source
New & Changed User Stories	46	PT
Domain	Web application	DL
Person Months	23.1	PT
Elapsed Months	5	PT
Nature of Project	Enhancement	DL
Relative Complexity	Moderate	DL
Project Age	10 years	DL
Constraints	Fixed-delivery date, utilize CM and quality-management processes	DL
New/Changed Classes Total Component Classes	1,200 2,721	SC
New/Changed Methods Total Component Methods	2,471 30,088	SC
New/Changed KLOEC	26.4	SC
KLOEC of New & Changed Classes	234.5	SC
Component KLOEC	487.4	SC
System KLOEC	1,014.8	SC

3.2.4 Technological factors

The Sabre-P team's development methodology throughout the release was XP. Release versions were dictated by the marketing department and served as development milestones. The team almost exclusively did their planning activities on the iteration level, and the product was continuously available to customers via an automated build machine. User stories and task estimates were recorded in a Microsoft® Excel spreadsheet that was also used to

[1] http://www.scootersoftware.com/

forecast release points and iterations based on the team's project velocity. The web-application component was developed using Java, and the team employed the JUnit[2] automated unit testing framework. The team also began preliminary evaluation of the FIT[3] acceptance testing framework during the release, though it was not used extensively. The team's technological factors are shown in Table 4.

Table 4: Technological Factors

Context Factor	Value	Source
Soft. Dev. Meth.	XP	DL/OB
Project Mgmt.	XP Planning game	DL/OB
Defect Prevention & Removal	Unit testing, dedicated tester, customer acceptance tests	DL/OB
External/System Test	Involvement throughout code development, daily interaction	DL/OB
Language	Java, C++	DL
Reusable Materials	Third party libraries, JUnit test suites, FIT tests, code template skeletons	DL

3.2.5 Ergonomic factors

The Sabre-P team's ergonomic factors are documented in Table 5. A representative from Sabre's product marketing department served as the XP customer on this project, was on-site 25%-50% of the time, and was available through e-mail at other times. When the marketing representative was not available, the team's XP tracker served as the proxy customer and was trusted by the marketing representative and product management to serve as an appropriate replacement. The team worked in two adjacent, open-space XP labs. Team members stated that, due to the number of people in open space, the work area could sometimes become a distraction. Again, information for these factors was solicited from the project lead and from developer questionnaire responses.

Table 5: Ergonomic Factors

Context Factor	Value	Source
Physical Layout	Two adjacent, open labs	DL/OB
Distraction of Office Space	Medium	DL/DQ
Customer Communication	Pseudo-customer. Primarily face-to-face and e-mail communication.	DL/DQ

3.2.6 Geographical factors

The entire development team was co-located, as indicated in Table 6. The team's product was used by three external customers. These customers were all remote: one domestic, two international. One of the international customers is based overseas.

[2] http://junit.org

[3] http://fit.c2.com/

Table 6: Geographical Factors

Context Factor	Value	Source
Team location	Co-located	DL/OB
Customer Cardinality and Location	3 (remote; multi-national; several time zones away)	DL
Supplier Cardinality	None	DL

3.3 Adherence Metrics

Most companies that use XP adopt the practices selectively and develop customized approaches that are appropriate within their particular organizational setting [7]. The XP adherence metrics enable case study comparison, the study of XP practice interaction, and the determination of contextually-based, safe XP practice subsets. Most of the adherence metrics are in-process metrics that must be planned for and documented during development. These metrics also introduce potential overhead in the measurement process. For example, there is currently no fully automated means of tracking the frequency of pair programming or how often unit test suites are run by individual members. Therefore, some of this information requires either a subjective estimate or manual data tracking by some or all of the team members. Where feasible, the XP-am contain objective, and often automated, measures that capture adherence information with minimal development overhead. Unfortunately, many of the objective metrics in the XP-am could not be gathered for this case study since most of the information in this study is collected from historical data.

The XP-am also contain subjective information in the form of the Shodan Adherence Survey (described fully in [20] and adapted from [15]). The survey is an in-process means of gathering XP adherence information from team members, and asks the question "How often do you perform each XP practice?" The survey is web-based and was taken by each developer during the time of the release under study. Also, a questionnaire was administered to the Sabre-P team approximately eight months after the completion of the release under study. The team was asked to discuss the various XP practices and why they felt that the team had difficulty adopting certain practices (based on survey responses). Team members were also asked which practices they felt were essential and which practices they felt were unnecessary. An additional question also asked if the team members believed that XP worked for a team of their size and to justify their answers. A summary of the Sabre-P team's survey responses are provided for context. We discuss the Sabre-P team's XP-am results under three categories: planning, testing, and coding.

3.3.1 Planning practice adherence

The entire team participated in daily stand-up meetings. Developers noted that this exercise could become tedious with their team size, since some points brought up by developers seemed irrelevant to others on the team who had little to do with other people's components. However, team leads, project managers, trackers, and some developers found the meeting extremely beneficial for evaluating the status of the project. Developers also believed that customer access was a problem, and that their representative was not available as often as needed to

make decisions. As such, team members felt that development would sometimes become hurried or ill-planned because important features needed to be incorporated or re-worked late in the release cycle at times when the customer was able to give input. Furthermore, developers noted that the planning game was often influenced by factors outside the team's control, such as deadlines set by upper management and feature creep. Business demands sometimes required the team to incorporate more features than were planned into an iteration or a release but without the feature tradeoff that XP mandates. Thus, planning became difficult and sometimes frustrating for the team. Releases ranged from three to five months in length, and iteration lengths were fixed at ten days. Many developers noted the value of small iteration plans because they provided a concise, focused set of tasks that must be completed in the coming weeks.

Gathering the release and iteration length metrics was a matter of examining the team's project tracking tool. Requirements dynamism serves as our measure of requirements volatility. If the team is agile, then it should be able to withstand a high amount of requirements change. However, collecting this information potentially introduces overhead into the XP process. In the Sabre team's project tracking tool, no mechanism existed for recording which user stories were injected, removed, or changed. Deleted stories disappeared from iteration plans, but the reason for their removal (be it by customer request or because of feature completion) was not recorded. Similarly, it was unclear whether stories that were added to iteration plans were a part of the larger release plan, or whether an impromptu customer request was made to insert the story directly into the iteration. Collecting meaningful information for this metric was also difficult given that the team operated almost exclusively on iteration plans. Since the user story changes were not recorded at the release plan level, it is difficult to quantify the actual amount of requirement change that took place over the release period.

The team's planning adherence is summarized in Table 7. The objective metrics appear on the top and the Shodan; subjective metrics appear on the bottom. This format will be used for Tables 7, 8, and 9.

Table 7: Planning adherence metrics

XP-am Planning Metric	**Value**	**Source**
Release Length	3 months	PT
Iteration Length	10 work days	PT
Requirements dynamism	N/A	N/A
Subjective Metrics (Shodan)	**Mean (σ^2)**	
Stand-up meetings	98.0% (4.1)	SU/DQ
Short Releases	75.5% (22.6)	SU/DQ
Customer Access / On-site Cust.	70.0% (23.6)	SU/DQ
Planning Game	66.5% (17.3)	SU/DQ

3.3.2 Testing practice adherence

Table 8 summarizes the Sabre-P team's testing adherence metrics. The team's test coverage information was gathered using the Clover[4] tool that ran as a part of the automated build process. The large amount of legacy code was the most influential inhibitor to the team adopting the XP testing practices. The actual coverage provided by the unit tests is small (7.7% statement coverage). Test coverage is averaged over the entire component, not just the new and changed portions. Therefore, the coverage percentage is relatively low due to the large amount of legacy code in place before unit testing was instated as a development practice within the team. A considerable amount of effort would be required to unit test all of this code.

More revealing information regarding the team's testing effort can be inferred by examining the number of test classes that correspond to new and changed classes, as well as the ratio of test KLOEC to source KLOEC. This information was gathered using the Beyond Compare tool to identify the new and changed classes in the system, and then searching for a corresponding test class in the source tree. Since identifying new and changes classes could not be counted by the Clover tool, identifying corresponding test classes that corresponded with new and changed classes was a manual process. This counting could potentially involve significant overhead in a large system with a high amount of code churn. These remaining testing adherence measurements also turned out to be low, indicating that the team does not write unit tests often. Further consultation with the team lead revealed that the team's unit testing strategy centered on writing tests for the classes containing the business logic of the component, which in turn exercised many supporting classes.

Further insights about the team's difficulties in adopting unit testing were offered by the developer questionnaire responses. Writing unit tests for some of the legacy code was not perceived to be cost-effective by some individual developers. Furthermore, developers noted that, because there are many complex elements to their system, it is often difficult to write unit tests for all pieces of code. Some developers also cited limitations in existing unit testing frameworks (such as limited capabilities to test GUI applications) as a major hurdle in adopting unit testing and test-driven development. Also, developers often abandoned writing tests when under deadline pressure, striving to implement all the features promised in the release. This was done in spite of encouragement from the team leads and the XP coaches, who urged the team members to write unit tests for all new or changed pieces of code.

The team has just begun learning to automate their acceptance testing via the FIT framework. However, most of the acceptance tests were not automated. Developers stated that they frequently use acceptance tests, but that these tests are often not very detailed or extensive before work began on an associated user story or feature. The practice of writing an acceptance test to guide development was not strictly enforced at the time. Many developers found value in the tests as a mark of completion for a user story, but that acceptance tests that were better defined early in the process could help drive development more effectively.

[4] http://www.cenqua.com/clover/

Table 8: Testing adherence metrics

XP-am Testing Metric	Value	Source
Test Coverage (statement)	7.7%	TS
Test Run Frequency (quickset[5] runs / person-day) [anecdotal]	0.4	DL
Test Classes to New/Changed classes (JUnit only)	2.25%	TS
New Classes with corresponding Test Classes (JUnit only)	5.66%	TS
Test LOC / Source LOC (including test code embedded in the system)	0.061	TS
Subjective Metrics (Shodan)	**Mean (σ^2)**	
Test First Design	60.0% (21.0)	SU/DQ
Automated Unit Tests	74.0% (23.0)	SU/DQ
Customer Acceptance Tests	64.0% (26.6)	SU/DQ

3.3.3 Coding practice adherence

The Sabre-P's coding adherence metrics can be found in Table 9. The coding adherence metrics have been the most difficult to automate. An automated means of collecting these metrics is currently unavailable, though one suggested method involves examining comment banners to identify changes done by pair programmers rather than by solo programmers [22]. Manually tracking the amount of time spent pairing when working on a user story would be elementary, but would be tedious in that developers would have to record information every time that they worked on a specific item. Perhaps sampling the developers' time spent pairing and doing inspections at various phases of development is a plausible alternative.

Beck hypothesizes that continuous integration may be problematic on a large team since the integration software will have to handle multiple code streams simultaneously [5]. However, the Shodan survey response for the question on continuous integration averaged 89.5% (std. dev. of 7.6), indicating that, on average, the team members checked their code into the integration machine more than once per day. The practice was considered essential by some developers as it forced them to design simply and to code in smaller increments. Many developers also noted that the constant integration provided feedback (in the form of automatically-run unit test suites) helped identify errors quickly.

The survey responses also indicated that the team paired approximately 60% of the time. Questionnaire respondents noted that the team used "intelligent pairing," wherein they only paired on those problems perceived to be suitably complex. However, many questionnaire respondents stated that pairing was often discarded due to impending deadlines. Developers felt that they must work solo in order to meet these deadlines, despite strong encouragement from the coach and the team leads to continue to pair program. Developers noted that refactoring is sometimes a neglected practice due a fear of injecting defects into existing production code. They state that a more robust suite of unit tests would help alleviate this fear, but, because of the large amount of legacy code, refactoring this code to be enable unit testing is not a viable option. When asked if they followed the rules of simple design, most developers stated that the practice is often followed. However, when faced with a complex problem, a simple design approach is not sufficient and more detailed planning and design is required.

When asked to comment on collective code ownership within the team, developers stated that it provided some benefit in distributing knowledge of the system. However, because the system is large and complex with different modules, there are still some team members who retain specialized knowledge and are the only ones qualified to work on certain tasks. One drawback several developers mentioned about collective ownership is that it has lead to a decreased amount of responsibility for poorly-written code. For instance, a developer might choose to implement the quickest solution, but not one with a sound design or an optimal structure since he/she is not the only person responsible for the performance and the effects of that code. Without this responsibility, there is less motivation to write code that conforms to standards and/or is well-designed. When asked if they felt that they were working at a sustainable pace, the developers all agreed that they were not. Since the team operated with fixed and aggressive deadlines without the ability to reduce scope, the team worked consistent overtime to meet promised features and delivery dates. Underestimation of the time it took to complete user stories also contributed to the problem.

Table 9: Coding adherence metrics

Coding Metric	Value	Source
Pairing Frequency (anecdotal)	70%	DL
Inspection Frequency (anecdotal)	0%	DL
Subjective Metrics (Shodan)	**Mean (σ^2)**	
Pair Programming	61.5% (22.3)	SU/DQ
Refactoring	59.5% (20.4)	SU/DQ
Simple Design	69.0% (21.0)	SU/DQ
Collective Ownership	70.0% (21.0)	SU/DQ
Continuous Integration	89.5% (7.6)	SU/DQ
Coding Standards	80.5% (14.7)	SU/DQ
Sustainable Pace	61.0% (25.9)	SU/DQ
Metaphor	55.0% (25.7)	SU/DQ

3.4 Outcome Measures

Of utmost importance to decision makers is whether or not adopting XP aids in productively creating a higher quality product. Because adequate baseline data was not available for the Sabre-P team, their business-related results, structured via the XP-Outcome Measures (XP-om), are compared to industry averages documented by Capers Jones in [9] and the Bangalore SPIN group [2]. We selected these two sources because of their accessibility (the Bangalore SPIN report is available online and the Jones reference is available in many bookstores). Furthermore, these

[5] Quickset refers to a subset of the entire unit test suite that is run to excerise a particular module of the system

sources contained similar software process measures as those in the XP-om. The measures in these two sources were also documented clearly enough that data collection could be conducted in similar manner to that of the industry averages.

In our other case studies, we compared these results to results from a previous release of the same product. One can reduce internal validity concerns by studying the same software project with a team comprised largely of the same personnel. However, for this case study, no such comparison point could be established since the necessary artifacts for earlier releases (project tracking, defect information, etc.) could not be obtained.

In order to provide an informative comparison, we used published industry averages from two sources [2, 9]. When comparing with industry survey data, it is critical to ensure that the metrics are identical and that they were collected using the same methods. For example, when gathering defects, the same collection period must be established. This period can be either a set time period (e.g. six months after release) or a set of specific phases of development (post-release, system test, integration test). When interpreting the results, it is important to remember that the published data covers a broad range of projects and organizations. Therefore, it can be unclear how your own case study relates to the sample population from the industry averages. If the published information is organized into specific categories (e.g. team size, project duration, criticality, domain, etc.), then one may be able to draw more meaningful conclusions. When such data is unavailable, one must be cautious when interpreting the results since the specific context of one's study may be vastly different from those projects in the industry surveys.

3.4.1 Limitations

The case study results are presented relative to industry averages of the Bangalore SPIN Benchmarking group [2] (Table 10) and of Jones [9] (Table 11). The results in Table 10 are presented with regard to the 95% confidence bounds published in the report: Higher (greater than upper limit), Lower (less than the lower limit), or Similar (within the confidence interval). Results in Table 11 are presented in the same fashion, except that no confidence bound was given and only a point estimate is available. Interpreting the Jones surveys required converting the system size from KLOEC to function points. Function points were estimated from lines of code using the 1996 version of the Programming Languages Table[6]. This estimation technique, known as "backfiring," has been shown to have an accuracy of ± 20% [11]. The lines of New and Changed Code served as the basis for all computations in the table below. In our calculations for Table 11, we tested both the upper bound and lower bound of the function point estimate. Items with a results of "Higher" were higher than the upper bound and items marked "Lower" were lower than the lower bound of the given the ± 20% accuracy of function point estimation. Due to the inaccuracies in our function point estimation and the use of point estimates for comparison, we acknowledge that there are experimental validity concerns.

Another experimental validity concern involves the accuracy of our defect counts. It has been our experience that most teams have their own methods for recording defect information, even in a standardized or automated setting. Interpreting and codifying this information correctly is important to ensuring the validity of the data. Counting the defects in the Sabre-P project was a non-trivial process and required extensive input from both the team's tester and the project lead. When counting defects, there were several entries that were not classified as defects, but instead as enhancements, customizations, etc. We counted only those entries where the resolution type was a "Defect Correction." Furthermore, (as is consistent with both survey comparison points), we did not count defects uncovered during development or unit test. We also only counted defects which could be positively identified as attributable to the release under study (all entries in the Sabre-P team's defect tracking system had a "release" category). Some defects that were entered into the system during the defect collection timeframe (from the beginning of development through six months after release point) had an entry of ALL in the "release" category of the entries. Those entries that were attributable to the release under study were added to the defect counts and were identified by the development lead after examining the body of the defect information.

Our final limitation involves external validity. The intention of this study was to compare the outcome measures of a team using XP with industry averages of software teams using many different development methodologies. Since we do not know the specific contexts of the industrial teams our industry average resources, it is unclear whether the Sabre-P team is comparable to those teams in terms of their contexts. Therefore, we cannot concretely identify whether the relationship between Sabre-P team's project outcome and the industry averages is due to their adoption of XP, or if there are other elements of Sabre-P team's context factors that may have influenced the results.

Table 10: XP outcome measures (compared to [2])

XP Outcome Measures	Result	Source
Pre-release defect density (test defect/KLOEC)	Similar	DT/SC
Total defect density (pre-release + post-release defects / KLOEC)	Lower	DT/SC
Productivity (LOEC/Staff Day)	Similar	SC/PT

Table 11: XP outcome measures (compared to [9])

XP Outcome Measures	Result	Source
Post-release defect density (released defects / KLOEC)	Lower	DT/SC
Total defect density (pre-release + post-release defects / Function Points)	Lower	DT/SC
Defect removal efficiency (test defects / total defects)	Lower	DT
Productivity (FP/Staff Month)	Higher	SC/PT

3.4.2 Pre-release Quality

Both pre-release defect density (test defects/KLOEC) and defect removal efficiency are indicators of pre-release quality. Kitchenham notes that pre-release quality is a surrogate measure of quality [12], and that it is in truth a measure of the testing

[6] http://www.spr.com/products/programming.htm

process. The Sabre-P team showed similar pre-release defect density and a lower defect removal efficiency than published averages. These results are subject to interpretation because the release underwent a two month feature freeze in which end-to-end testing of the product was performed. This concentrated testing effort may have led to more defects being uncovered than in a release where no extended testing period took place. The entire team participated in the testing process. Conversations with the team's tester revealed that, in many cases, defects found during this test phase were not always recorded in the defect tracking system: the defects were uncovered and dealt with immediately. As such, the count of pre-release defects may be an undercount of the actual number.

3.4.3 Post-release Quality

The team's post-release quality includes the number of defects delivered to and reported by the customer. We use a defect collection period of six months after the release point, a time box suggested by Kitchenham [12] and others [10], to allow the customer time to exercise and test the product. The Sabre-P team achieved lower than the average delivered defect density. Furthermore, the team's total defect density (including both pre-release and post-release defects) was lower than both sets of survey results. Again, these numbers may be influenced by the team's extended testing effort prior to the release of the product.

These results would be better understood if we had evidence regarding the customer's use of the system after the release. According to the development lead, the release under study was not received by all of the team's existing customers, and that new customers for the product activated only a subset of the features in this release. Therefore, since the system was not fully exercised by all available customers, the count of post-release defects may be lower than the team's usual numbers.

3.4.4 Productivity

Effort was calculated in both LOEC/Staff Day and Function Points/Staff Month. As previously mentioned, the New and Changed LOEC (Table 3) provided the basis for the sizing estimates; this is consistent with the method used in both surveys. The Sabre-P team's productivity was similar to the averages published by the SPIN group and better than average when compared to projects of similar size in the Jones publication. Again the team did not spend the entirety of the release updating and/or creating new features because of the two month feature freeze. This was taken into account both productivity measurements, where only those staff days the team spent working on development code (and not performing the end-to-end testing) were counted. Furthermore, the productivity counts do not account for the unit test code written during development.

4. DISCUSSION

This paper has provided one example of an agile case study. We discussed the difficulties involved in collecting information, provided some guidance for collecting various metrics, and showed an example of how comparisons can be made with published industry average data. The results of our case study may not be conclusive, but this paper illustrates several important considerations for performing a case study.

4.1.1 Availability of Data

The measurements we gathered were largely dependent on the availability of data. Our original intent for this study was to compare this release of the product to a release completed using a plan-driven methodology. However, because of difficulties in obtaining source code and because of changing defect repositories, none of our outcome measures were available for the desired plan-driven baseline. When establishing the goals of a study, it is important to identify what measurements will be necessary to enable those goals and if the required data is available.

4.1.2 Tool Support

Strong tool support allowed us to *collect* our data for analysis quickly and easily. In the case of the defect tracking system, the tools also allowed us to perform our analysis more quickly by providing simple sorting features, hyperlinks to extended descriptions, and the like. Manually collecting and sorting through this information would have required extended involvement of the development team. The last point is particularly important since we did not wish to pose excessive burden on the developers that might impact their agile process. However, even with tool support, the *analysis* portion of the study still required considerable effort on the part of the researchers.

4.1.3 Co-operative Personnel

Since many agile teams do not have a dedicated metrics specialist, having a team with personnel who are willing to cooperate in data collection and interpretation is essential. In our study, the team leader was the source for most of the context information, and both the team lead and the tester were essential to interpreting the defect information. Furthermore, the cooperation of the developers was necessary in taking the survey and in responding to questionnaires provided important qualitative information needed to help understand and interpret the quantitative findings of the case study. The qualitative information from the developers is also valuable for other practitioners reading this case study, as it allows them to see some of the obstacles and benefits of adopting XP practices that may not be explicitly captured by the measurements of the XP-EF framework.

4.1.4 Project Status

Project status is related to data availability. In the XP-EF, we employ several in-process metrics to determine XP practice adherence. Since the project had already begun at the time of the study, we could not gather all of these metrics since the data would have been incomplete. Project status is a factor in the case study that is dependent upon the nature of the data being collected. Furthermore, if the case study project has occurred several years or even several months ago, data artifacts and personnel associated with project may not be available to contribute valuable quantitative and qualitative information.

4.1.5 Conclusion and Future Work

Throughout this paper, we have discussed our experience of performing a case study to evaluate the effectiveness of XP practices with an industrial team at Sabre Airline Solutions. We employed the XP-EF in our study as a means for structuring our metrics collection, and describe various dependencies and

difficulties in collecting the data throughout. We present the Sabre-P team's business related results as related to two published industry averages. These results indicated that the team had better post-release quality than average and similar or better productivity than average. These results are discussed within the specific context of the case study.

We have also identified four important factors that impacted the progress of the study: availability of data, tool support, co-operative personnel and project status. The results collected from this case study were derived largely from historical data, such as archived code and documented defect information. This presented a number of unique challenges in collecting meaningful, informative data post hoc. There is a reliance on software process artifacts, such as archived code, user story information, and defect reports, to enable collection of historical data. The production of these artifacts may run contrary to the agile paradigm. However, the presence of CASE tools, including automated build tools, integration environments, and defect tracking systems, may alleviate much of the overhead associated with collecting these metrics. This information will be useful to those practitioners who are considering the implementation of a software metrics program in conjunction with their agile process.

We are currently analyzing one other case study conducted at Sabre Airline Solutions. Three additional case studies structured by the XP-EF are about to commence with a telecommunications firm in the United States. 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.

5. ACKNOWLEDGEMENTS

The authors wish to thank the individuals on the Sabre Airline Solutions development team for participating in this study, and Chris Shepperd, Brian Sullivan, Mahvash Hatamieh and Scott Frederick for their invaluable assistance. This research was supported by Sabre Airline Solutions. Lynn Cunningham participated in this research through support from the National Science Foundation Distributed Mentor Project.

6. REFERENCES

[1] Abrahamsson, P., "Extreme Programming: First Results from a Controlled Case Study," *29th IEEE EUROMICRO Conference*, Belek, Turkey, September 1-6, 2003, pp. 259-266.

[2] Bangalore Benchmarking Special Interest Group, *Benchmarking of Software Engineering Practices at High Maturity Organizations*. 2001, Bangalore Software Process Improvement Network.

[3] Basili, V., F. Shull, and F. Lanubile, "Building Knowledge Through Families of Experiments," IEEE Transactions on Software Engineering, vol. 25, No. 4, pp. 456 - 473.

[4] Beck, K. and M. Fowler, *Planning Extreme Programming*, Boston, MA: Addison-Wesley, 2001.

[5] Beck, Kent, *Extreme Programming Explained: Embrace Change*, New York: Addison-Wesley, 2000.

[6] Boehm, B. and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley, 2003.

[7] El Emam, Khaled, *Finding Success in Small Software Projects*, in *Agile Project Management*.

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

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

[10] Kan, S., *Metrics and Models in Software Quality Engineering*, Second ed, Boston, MA: Addison Wesley, 2003.

[11] Kemerer, C.F., "Reliability of Function Point Measurement: A Field Experiment," Communications of the ACM, vol. 36, No. 2, pp. 85-97.

[12] Kitchenham, B., *Software Metrics: Measurement for Software Process Improvement*, Cambridge, MA: Blackwell, 1996.

[13] Kitchenham, B., L. Pickard, and S. L. Pfleeger, "Case Studies for Method and Tool Evaluation," IEEE Software, vol. 12, No. 4, pp. 52-62, July.

[14] Kitchenham, B.A., 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," IEEE Transactions on Software Engineering, vol. 28, No. 8, pp. 721-733, 2002.

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

[16] Layman, L., L. Williams, and L. Cunningham, "Exploring Extreme Programming in Context: An Industrial Case Study," *2nd IEEE Agile Development Conference*, Salt Lake City, UT, June 22-26, 2004, pp. 32-41.

[17] Maurer, F. and S. Martel, "Extreme Programming: Rapid Development for Web-Based Applications," IEEE Internet Computing, vol. 6, No. 1, pp. 86-90, January-February 2002.

[18] Potts, C., "Software Engineering Research Revisited," IEEE Software, vol. No. pp. 19-28.

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

[20] Williams, L., L. Layman, and W. Krebs, "Extreme Programming Evaluation Framework for Object-Oriented Languages -- Version 1.4," North Carolina State University Department of Computer Science TR-2004-18, 2004.

[21] Williams, L., W. Krebs, L. Layman, and A. Antón, "Toward a Framework for Evaluating Extreme Programming," *8th International Conference on Empirical Assessment in Software Engineering (EASE 04)*, May 2004, pp. 11-20.

[22] Williams, Laurie, William Krebs, Lucas Layman, and Annie Antón, *Toward a Framework for Evaluating Extreme Programming*. 2004, North Carolina State University: Raleigh, NC.

[23] Yin, R.K., *Case Study Research: Design and Method*, Third ed, Vol. 5, Thousand Oaks, CA: Sage Publications, 2003.

[24] Zelkowitz, M.V. and D.R. Wallace, "Culture Conflicts in Software Engineering Technology Transfer," *NASA Goddard Software Engineering Workshop*, 1998.

[25] Zelkowitz, M.V. and D.R. Wallace, "Experimental Models for Validating Technology," IEEE Computer, vol. 31, No. 5, pp. 23-31, May 1998.