# Toward a Framework for Evaluating Extreme Programming

Laurie Williams[1], William Krebs[2], Lucas Layman[1], Annie I. Antón[1], Pekka Abrahamsson[3]
[1]*North Carolina State University, Department of Computer Science*
*{lawilli3, lmlayma2, aianton}@ncsu.edu*
[2]*IBM Corporation, krebsw@us.ibm.com*
[3]VTT Technical Research Centre of Finland*, pekka.abrahamsson@vtt.fi*

## Abstract

*Software organizations are progressively adopting the development practices associated with the Extreme Programming (XP) methodology. Most reports on the efficacy of these practices are anecdotal. This paper provides a benchmark measurement framework for researchers and practitioners to express concretely the XP practices the organization has selected to adopt and/or modify, and the outcome thereof. The framework enables the necessary meta-analysis for combining families of case studies. The results of running framework-based case studies in various contexts will eventually constitute a body of knowledge of systematic, empirical evaluations of XP and its practices. Additionally,this benchmark provides a baseline framework that can be adapted for industrial case studies of other technologies and processes. To provide a foundation on the use of the framework, we present the initial validation of our XP evaluation framework based upon a year-long study of an IBM team that adopted a subset of XP practices.*

## 1. Introduction

Examining the efficacy of Extreme Programming (XP) [7] is a contemporary software engineering research challenge. Often, compelling empirical evidence is not available when a technology is introduced. Typically, such evidence evolves with the rate of adoption of the technology [13]. For example, strong empirical evidence of the value of the Software Capability Maturity Model (CMM) [31] came after wide initial adoption [13]. Similarly, XP is becoming increasingly popular.

Many articles have appeared in major software engineering publications and conference proceedings extolling the virtues of XP and questioning its weaknesses. Most often, these reports take the form of anecdotal success stories or lessons-learned from organizations that have adapted XP for a project [27, 28, 38]. While valuable, many of these articles do not fulfill the growing need for empirical case studies to provide scientifically-rigorous information about XP's virtues and shortcomings. Additionally, the metrics, if any, used in each of these studies are different. Thus, it is difficult to compare the study results or to establish a contextualized body of evidence to support or refute the anecdotal evidence of XP project results. Organizations and researchers need a framework to assess empirically XP's strengths and weaknesses in a variety of contexts. Additionally, researchers need ontological frameworks to guide data collection and analysis of industrial case studies.

Sim et al. challenged the software engineering community to create benchmarks, or a set of tests used to compare the performance of alternative techniques [37]. As such, benchmarks in the form of ontology-based frameworks[1] have been published in software engineering [16, 21, 24, 43]. The purpose of these frameworks is to reveal factors which might influence when and how a certain process/practice is beneficial compared with other processes/practices.

In this paper, we provide an ontology-based framework for expressing the XP practices an organization has selected to adopt and/or modify, and the outcome thereof. This framework, called the XP Evaluation Framework (XP-EF) is comprised of metrics that are focused, concise, and can be collected by a small team without a dedicated metrics specialist. The XP-EF is comprised of three parts: XP Context Factors (XP-cf), XP Adherence Metrics (XP-am) and XP Outcome Measures (XP-om). Use of this framework enables the necessary meta-analysis for combining families of case studies. The results of running XP-EF-based case studies by our and other research teams in various contexts will eventually constitute a body of knowledge of systematic,

[1] Such a framework consists of the enumeration of all the metrics and project characteristics that must be recorded in the case study and specific instructions about how the metrics are to be consistently collected (e.g. defect counting rules and classification schemes).

empirical evaluations of XP and its practices. This body of knowledge will be valuable to organizations awaiting stronger evidence of the efficacy (positive or negative) of the XP practices prior to adoption.

In the XP-EF, researchers and practitioners record essential context information about a project via the XP Context Factors (XP-cf). Factors such as team size, project size, criticality, and staff experience can help explain variations in the results of applying the practices.

The second part of the XP-EF is the XP Adherence Metrics (XP-am). Rarely, if ever, do software development teams fully exercise all XP practices in a "pure" form [13]; some employ only a few practices. The XP-am enables one to express concretely and comparatively the practices a team utilizes. 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); it provides researchers and practitioners a means to assess and report a team's project outcome from using a full or partial set of XP practices. The XP-om consists of traditional external software development measures, such as productivity and quality. Ideally, a team (comprised of researchers and/or practitioners) performing a case study will have a baseline product that can be used for comparison.

To guide the structure of empirical case studies of XP, we identified the following research goal via the GQM [2]:

G: *To build theories about whether the business-related results of a team change when XP practices are used.*

We refined these goals into questions which examined theories about five specific business-related results.

Q1: Does the pre-release quality improve when a team uses XP practices?
Q2: Does the post-release quality improve when a team uses XP practices?
Q3: Does programmer productivity improve when a team uses XP practices?
Q4: Does customer satisfaction improve when a team uses XP practices?
Q5: Does team morale improve when a team uses XP practices?

To perform the initial validation of our framework, we utilized the "industry-as-laboratory" [32] approach in which software engineering researchers worked closely with industry to create and evaluate solutions [32]. In a year-long, "in-vivo" (in the field, under normal conditions) [4] case study, the XP-EF was applied within an IBM software development team. This seven-person IBM team develops Servlet/XML applications for a toolkit that other IBM teams utilize to create products for external customers. The team adopted and sustained the use of a subset of XP practices deemed "safe" and appropriate for their team culture and project characteristics. This case study is the first of a family of XP-EF case studies, providing initial validation of the XP-am metrics. Additionally, it demonstrates how to conduct an XP assessment utilizing the XP-EF framework.

The paper provides a foundation on the use of the XP-EF. The remainder of this paper is as organized as follows. Section 2 discusses measurement frameworks and surveys related XP research. Section 3 sets the context for our IBM case study. Section 4 presents the results of the case study. Section 5 presents the analysis of our results and our plans for future work.

## 2. Background and related work

In this section, we discuss existing software measurement and assessment strategies, frameworks and metrics. We then provide a survey of prior XP studies.

### 2.1. Strategies and frameworks

Software measurement is imperative for companies competing in a rapidly-changing environment. McGarry proposes several project management strategies to establish a customized measurement framework [30]. Grady offers similar strategies tailored to customer satisfaction, defect prevention, analysis, and removal [17]. The ISO/IEC 14598 standard [19] establishes guidelines for measuring and assessing software quality.

Kitchenham et al. [23] proposed a detailed set of guidelines for designing, collecting data, analyzing, and reporting the results of empirical studies. The use of their guidelines is purported to improve the quality of individual studies and to increase the likelihood that meta-analysis can be used to combine the results of multiple related studies. These guidelines have been incorporated in the composition of the XP-EF.

Williams et al. [41] introduced a metric suite for empirically assessing an agile methodology's effectiveness. Several hypotheses and metrics are proposed for productivity, cycle time, externally-visible pre- and post-release quality, responsiveness to customer change, internal code structure, and job satisfaction. This prior work serves as a foundation for the framework proposed herein.

An ontology-based framework for software maintenance was developed by Kitchenham et al [24]. These authors developed the ontology for four reasons, consistent with the purpose of benchmarks:

- to allow researchers to provide a context within which specific questions about maintenance can be investigated;
- to help understand and resolve contradictory results observed in empirical studies;
- to provide a standard framework to assist in reporting of empirical studies in such a manner that they can be classified, understood, and replicated and to help industrial adoption of research results; and
- to provide a framework for categorizing empirical studies and organizing them into a body of knowledge.

The intent of the XP-EF is consistent with these four purposes. Additionally, a common concern with case studies is that they can lack rigor and the use of systematic procedures [44]; the use of detailed instructions and metric suite of an ontology-based framework can dissuade this concern.

### 2.2. Metrics and comparisons

The XP-EF is a compilation of validated and proposed metrics. Metric validation requires convincing demonstration that (1) the metric measures what it purports to measure and (2) the metric is associated with an important external metric, such as field reliability, maintainability, or fault-proneness [14]. Schneiderwind proposes that six criteria be applied when validating software metrics: association, consistency, discriminative power, tracking, predictability, and repeatability [35]. In empirical studies, comparisons are informative. For example, a new project's measures can be compared against a prior project's measures within the same organization. Alternatively, comparisons can be made to industry standards and/or benchmarks. Jones has compiled data from many software organizations and provides benchmarks, best practices, and statistics for a range of software development topics [21].

### 2.3 Combining Methods for Triangulation

Through triangulation, two or more distinct methods are used to study the same phenomenon. Convergence and agreement between the result of these multiple methods enhances our belief that the results are valid and not a methodological artifact [20]. Triangulation also captures a more complete, holistic, and contextual portrayal of the factors under study [20].

Empirical research in software engineering often involves quantitative metrics. However, qualitative methods can be used to enrich quantitative findings with explanatory information, helping to explain "why" and to handle the complexities of issues involving human behavior. Seaman [36] 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 explain terminology that is used in a particular setting. Interviews can be structured, unstructured, or semi-structured [36]. Semi-structured interviews are a mixture of open-ended and specific questions designed to elicit unexpected types of information. The XP-EF provides a template of suggested interview questions.

Another form of qualitative research that can be used to study complex real-life problems and the immediate concerns of practitioners is action research [1, 6]. Action research is an iterative process in which researchers and practitioners collaborate on a cycle of activities, including problem diagnosis, action intervention, and reflective learning. Avison et al. [1] encourage the use of action research as a way to make academic theories relevant to practitioners in real situations. However, action research may inject bias to the study.

## 3. The XP-EF via an IBM case study

Experimentation in software engineering is challenging. Formal, controlled experiments, such as those conducted with students or professionals, over relatively short time periods are often viewed as "research in the small" [15]. These experiments offer the ability to produce statistically significant results yet may suffer from external validity limitations. Alternatively, case studies can be viewed as "research in the typical" [15]. Concerns with case studies involve the internal validity of the research [11] because the baseline and new treatments generally are not identical projects and/or teams, and case studies are difficult to replicate [45]. Finally, case studies seldom yield statistically significant results due to a small sample size. Nevertheless, case studies are valuable because they involve factors that staged experiments generally do not exhibit, such as scale, complexity, unpredictability, and dynamism [32]. Researchers' confidence in a theory increases when similar findings emerge in different contexts. By performing multiple case studies and/or experiments and recording the context variables of each case study, researchers can build knowledge through a family of empirical assessments. Replication addresses threats to experimental validity [5].

In this paper, we provide initial validation of the XP-EF and add to the knowledge of XP via a case study with an IBM development team in the United States. In our research, we compare the second and third releases of a product, heretofore referred to as the "old release" and the "new release" respectively. In the old release, the team began their initial adoption of XP practices. The team then increased and stabilized their XP adoption in the new release. This case study will be described in

terms of the XP-EF. Detailed instructions and templates for measuring and reporting the XP case study data via XP-EF Version 1.3 have been documented by the authors of this paper [42] to aid other researchers in replicating our case studies.

### 3.1. XP-cf: Context factors

Drawing conclusions from empirical studies in software engineering is difficult because the results of any process largely depend upon the relevant context variables. One cannot assume *a priori* that a study's results generalize beyond the specific environment in which it was conducted [5]. Therefore, recording an experiment's context factors is essential for fully understanding the generality and utility of the conclusions as well as the similarities and differences between the case study and one's own environment.

Software engineering has no well-defined standards for determining what contextual information should be recorded [23]. Jones [21] states that software projects can be influenced by as many as 250 different factors, but that most projects are affected by 10-20 major issues. He organizes key factors to be accounted for in every assessment into six categories: **software classification, sociological, project-specific, ergonomic, technological,** and **international**. The XP-EF framework templates are correspondingly organized into these six categories, though we modify the last factor (international) to **geographical** to capture the locale of the team, its suppliers, and customers. We also include **developmental** factors that use a risk-driven approach [8, 10] to determine whether a project would be most successful using an agile or plan-driven approach. In this subsection, we complete the XP-cf templates with data from the IBM case study.

**Software classification.** According to Jones [21], projects can be classified as one of six software types: *systems* (used to control physical devices); *commercial* (leased or marketed to external client); *information systems* (for business information); *outsourced* (developed under contract); *military*; or *end user* (private, for personal use). The IBM team developed software under contract for another IBM organization that ultimately marketed the product to external customers. We thus classify this project as *outsourced software*.

**Sociological.** Team conditions for both releases are shown in Table 1. Personnel is often considered one of the most prominent risk factors in software development [9], therefore, is it important to capture relevant information about team makeup. Sociological factors capture the development experience of the personnel, as well as their knowledge of the problem domain.

As shown in Table 1, the new release had a smaller team. XP and all agile methodologies rely upon tacit knowledge transfer to alleviate the challenges of personnel turnover. The turnover rate was calculated by adding the number of people who joined or left the team and dividing by the team size at the end of the release. Table 1 also classifies their years of experience. The team members are comparable; four left, including two agile veterans with high domain knowledge, but the remaining team members were the same.

**Table 1. Sociological factors**

| Context Factor | Old Release | New Release |
|---|---|---|
| Team Size (Develop) | 11 | 7 |
| Team Education Level | All: Bachelors<br>Two: Masters | All: Bachelors<br>Two: Masters |
| Experience Level of Team | 20 years: 2<br>10 years: 3<br><5 years: 2<br>Interns: 4 | 20 years: 1<br>10 years: 3<br><5 years: 1<br>Interns: 1 |
| Domain Expertise | High | |
| Language Expertise | High | |
| Experience Proj Mgr | High | |
| Specialist Available | GUI Designer | |
| Personnel Turnover | 22% | 36% |
| Morale Factors | | Manager change |

**Project-specific.** Projects of varying size and scope are subject to differing risk factors that may substantially affect development quality and schedule, making it necessary to record this context information. Table 2 compares the project-specific factors for the two releases. Based upon the number of new classes, methods, and lines of code (LOC), the new release is approximately half the size of the old release. In the Table 2, KLOEC are thousands of lines of executable (non-blank, non-commented) code. The team under study was responsible for the Component KLOEC, which this shipped as part of a larger product, the System KLOEC.

**Table 2: Project-specific factors**

| Context Factor | Old Release | New Release |
|---|---|---|
| New & Chng User Stries | 125 | 60 |
| Domain | Web | Web |
| Staff Months | 95.5 | 28.8 |
| Elapsed Months | 10 | 5 |
| Nature of Project | Enhancement | |
| Constraints | Partially date constrained | |
| New & Changed Classes<br>Total Classes | 203<br>395 | 139<br>431 |
| New & Changed Mthods<br>Total Methods | 1,110<br>3,229 | 486<br>3,715 |
| New or Chnged KLOEC | 19.2 | 9.8 |
| Component KLOEC | 38.8 | 42.8 |
| System KLOEC | 231.2 | 240.1 |

**Ergonomic.** The physical working environment can have direct impact on communication flow and overhead. This is particularly important to XP's core values of communication and feedback. Table 3 documents the projects' ergonomic factors. Because both the old and new releases had the same conditions, no comparison is made. Ideally, an XP team has an open space office environment. The IBM team sat in one aisle of cubicles with room for two people to pair program. A white noise generator protected other development groups from the distractions of talking pairs. However, these white noise generators might also impede peripheral information between sets of pairs. Cockburn [12] and others emphasize the importance of this peripheral information flow. The IBM team, however, was unable to modify their facility.

**Table 3: Ergonomic factors**

| | |
|---|---|
| Physical Layout | Cubicles large enough to allow pair programming |
| Distraction level of office space | Low. White noise generators, semi-private cubicles |
| Customer Communication | E-mail, chat programs, phone, and databases |

**Technological.** General software development tools and practices, such as code inspections, project management, and 4th generation languages, can have a dramatic effect on project productivity and quality. While the XP-am captures the use of XP practices, it is important to document other technological influences on a project's outcome as well. During the three years prior to the old release, the IBM team had used successfully a blend of waterfall phases and informal small team practices that resembled those of XP. The team culture was small, informal, skilled, and adverse to heavy process. Due to their past success and their aversion to heavy process, the team often omitted heavyweight waterfall-development practices, including formal UML design documents and formal code inspections. In the new release, the team adopted more of the XP practices. The project environment was marked by constraints that limited the team's ability to adopt all 12 XP practices to their full extent, as discussed in Section 4. The team's technology factors are summarized in Table 4.

**Table 4: Technology factors**

| Context Factor | Old Release | New Release |
|---|---|---|
| Software Develop-ment Methodology | Waterfall, with XP practices | Primarily XP |
| Project Management | Planning Game Gantt charts | Planning Game |
| Defect Prevention & Removal Practices | Design Reviews | Pair Program, Customer Test, Unit Test |
| Language | Java | Java |
| Reusable Materials | XML test data | XML test data, IDE techniques |

**Geographical.** Team location and customer location may greatly impact the feedback cycle length during software development. Table 5 documents the geographical factors. Because both the old and new releases had the same conditions, no comparison is made.

**Table 5: Geographic factors**

| Team Location | Collocated |
|---|---|
| Customer cardinality and location | Multiple; remote; multi-national, several time zones, some very far away |
| Supplier cardinality and location | Multiple; both remote and local; two time zones |

**Developmental.** 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 [8, 10]. Their five project factors (team size, criticality, personnel understanding, dynamism, and culture) aid in selecting an agile, plan-driven, or hybrid process. Criticality indicates the magnitude of loss due to a defect, ranging from loss of many lives to loss of comfort. Personnel indicates the team's ability, ranging from ability to perform procedural methods to ability to revise a method in an unprecedented situation. Dynamism is a measure of requirements volatility, and culture indicates the attitude of the team toward change.

These factors are graphed on a polar chart's five axes, as shown in Figure 2. When a project's data points for each factor are joined, shapes distinctly toward the graph's center suggest using an agile method. Shapes distinctly toward the periphery suggest using a plan-driven methodology. More varied shapes suggest a hybrid method of both agile and plan-driven practices. The IBM development team's factors are shown in Figure 1. The shape indicates that a hybrid "mostly agile, somewhat plan-driven method" is appropriate, which is what the team followed. The developmental factor that appears to necessitate plan-driven practices is criticality.

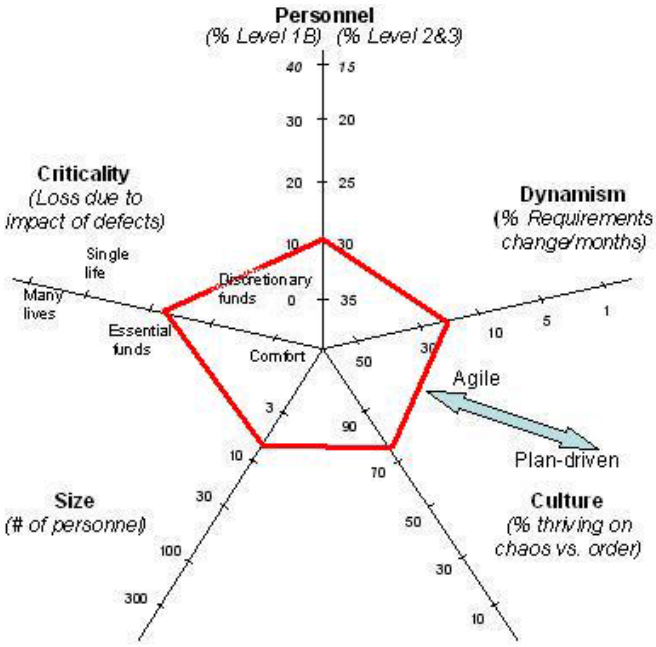


**Figure 1: Developmental factors [adapted from [8, 10]]**

### 3.2. Case study limitations

We examine the design of our case study based upon the four criteria for judging the quality of research design set forth by Yin [44].

**Construct validity**. Construct validity involves establishing measures for the concepts being measured. Our case study had quantitative measures predefined; these were backed up by qualitative means. As a result, construct validity was not a limitation of our study.

**Internal validity**. *Experimenter expectation bias* could have occurred because the second author of this paper tested, coded, and led the IBM team while participating as an action researcher in this study. This intimate knowledge potentially introduces bias into the study. His enthusiasm for XP may have influenced the team's successful adoption of XP. However, his direct involvement significantly aided the research because his detailed project knowledge provided qualitative details. The team knew the study was occurring so a Hawthorne [29] effect is a concern. However, the team was wholly concerned with completing the project and was unconcerned with the fact that a case study was occurring. Additionally, an external peer team participated in part of the product test, which should help remove some bias.

The difference in size of the two releases is a *confounding factor*. The new release was approximately half the size of the old release and had a smaller development team. Smaller projects with smaller teams are often considered to be less complex. However, the new release involved understanding and updating the larger code base.

The *maturation effect* is a concern because the comparison is made between two consecutive releases, however, we aimed to reduce internal validity concerns by studying the same software project with a team comprised largely of the same personnel. There is also a learning curve to be considered. The team members were learning to use some new XP practices, such as test-driven development, and they became more comfortable with these practices in the new release.

**External validity**. The *representativeness of the project* is a point of concern. The IBM team in this case study was selected in large part due to the third author, who expressed eagerness to participate in an XP project. Also, as documented in the Developmental Factors in Section 3.1, the team's risk factors indicated that they were fairly well suited for using an agile development methodology. Since no formal selection of the case study team took place, it cannot be stated whether this team is representative of other teams in the organization.

The case study provides a *complex context* which impedes generalization even to all small, co-located teams. Most importantly, the IBM team did not have 100% process adherence to the XP practices, as will be discussed in Section 4.1.

**Experimental reliability**. Attention to detail contributes to experimental reliability [22]. The use of the ontology-based framework has a desirable effect on experimental reliability.

## 4. Framework/Results

This section explains and provides an example of use of the adherence and results metric suites.

### 4.1. XP-am: Adherence metrics

Determining and recording the subset of practices employed by a team is essential for comparison purposes. Additionally, organizations may be interested in the adherence to certain practices. For example, pair programming and test-driven development have been shown to improve quality [39, 40] and may be deemed high-priority practices. Adherence metrics also 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. The XP-am does not advocate high adherence as a universal benefit for all projects.

This case study provides initial validation of the XP-am metrics. The XP-am is comprised of both subjective and objective measures as well as qualitative analysis about the team's use of XP practices. The Shodan

Adherence Survey (described fully in [26] and adapted from [25]) is an in-process, subjective means of gathering XP adherence information from team members. The survey, answered anonymously via a web-based application, contains 15 questions gauging the extent to which each individual uses XP practices. Survey respondents report the extent to which he/she uses each practice on a scale from 0% (never) to 100% (always). Periodic survey data can be used by teams for in-process corrections based on degree of use, trends, and variation between individuals. However, since the Shodan survey is subjective, it is not advisable to compare survey results across teams. Seven team members took the survey for the old release, and six took it for the new release (all full time team members). The objective measures portray the quantifiable adherence to XP practices for the old and new releases.

We present the combined results of these adherence metrics based upon three categories: planning (Table 6), testing (Table 7), and coding (Table 8).

As shown in Table 6, though the customer was remote, the team was comfortable with their remote communication, feedback, and responsiveness via e-mail, chat programs, phone, and databases.

**Table 6: Planning adherence metrics**

| Planning Metric | Old Release | New Release |
|---|---|---|
| **Objective metrics** | | |
| Release Length | 10 months | 5 months |
| Iteration Length | Weekly | Weekly |
| Requirements added or removed to Total Shipped Ratio | N/A | 0.23<br>13 added,<br>1 removed,<br>60 delivered |
| **Subjective (Shodan)** | **Mean (std dev)** | **Mean (std dev)** |
| Stand up meetings | 72% (16.4) | 90% (14.1) |
| Short Releases | 78% (27.3) | 77% (9.4) |
| Onsite Customer | 60% (28.1) | 87% (4.7) |
| Planning Game | 75% (21.2) | 85% (10.0) |

The measurement shown in Table 7 is the average for the entire component, including code that was not modified or added. As such, this number underestimates the testing effort in the new release. Test-run frequency measures how often the automated tests are run. The data shown was manually calculated and partially estimated. Ideally, the measure should be automated, and the value should be at least 1.0, indicating that each team member runs the test suite at least once per day. The automated test classes per user story ratio allows the team to examine adherence to their goal of a test class for every user story. The team's goals for coverage and run frequency were 60% and 90%, respectively. "Quickset" is the set of automated unit tests each developer runs several times a day before checking in code. Customer acceptance tests were run manually.

**Table 7: Testing adherence metrics**

| Testing Metric | Old Release | New Release |
|---|---|---|
| **Objective metrics** | | |
| Test Coverage (quickset) | 30% of lines | 46% of lines |
| Test Run Frequency | < 10% | 11% |
| Test Class to Story Ratio | N/A | 0.45 |
| Test LOC / Source LOC | < 0.30 | 0.42 |
| **Subjective (Shodan)** | **Mean (std dev)** | **Mean (std dev)** |
| Test First Design | 17% (11.2) | 55% (22.2) |
| Automated Unit Tests | 43% (16.4) | 67% (22.1) |
| Custom Acc Tests | 63% (25.6) | 78% (6.9) |

The pairing frequency shown in Table 8 was calculated by examining file headers. In the program comment banner, the developers indicated who worked on any file creation or modification. Pairing frequency was calculated by searching for these comments; ideally a more automated, objective means of assessing pairing should be utilized. For the new release, people were given a choice of pairing, inspecting, or justifying why code was written alone. To satisfy remote stakeholders accustomed to traditional design artifacts, a Slim Design Up Front (SDUF) template was used that included the user story, test case, and design checklist.

**Table 8: Coding adherence metrics**

| Coding Metric | Old Release | New Release |
|---|---|---|
| **Objective metrics** | | |
| Pairing Frequency | 11% | 48% |
| Inspection Frequency | 2% | 3% |
| Solo Frequency | 87% | 49% |
| **Subjective (Shodan)** | **Mean (std dev)** | **Mean (std dev)** |
| Pair Programming | 32% (15.0) | 68% (14.6) |
| Refactoring | 38% (11.6) | 57% (14.9) |
| Simple Design | 75% (10.5) | 78% (6.9) |
| Collective Ownership | 58% (14.0) | 83% (7.5) |
| Continuous Int. | 58% (18.8) | 78% (13.4) |
| Coding Standards | 87% (7.0) | 82% (3.7) |
| Sustainable Pace | 57% (12.5) | 77% (9.4) |
| Metaphor | 32% (30.7) | 43% (18.9) |

### 4.2. XP-om: Outcome Measures

Of utmost importance to decision makers is whether or not adopting XP practices aids in productively creating a higher quality project. The IBM business-related results are shown in Table 9, using a relative scale to protect proprietary information.

**Table 9: XP Outcome Measures (relative scale with the old release at 1.0)**

| XP Outcome Measure | Old Release | New Release |
|---|---|---|
| Response to Customer Change (Ratio (user stories in + out) /total) | N/A | 0.23 |
| Internally-Visible Quality (test defects/KLOEC of code) | 1.0 | 0.50 |
| Externally-Visible Quality* (defects/KLOEC of code 6 months after release ) | 1.0 | 0.61 |
| Productivity | | |
| User stories / Staff-Month | 1.0 | 1.34 |
| KLOEC / Staff-Month | 1.0 | 1.7 |
| Putnam Product. Parameter | 1.0 | 1.92 |
| Customer Satisfaction | N/A | High (qual) |
| Morale (via survey) | 1.0 | 1.11 |

*Response to Customer Change*. The number of user stories added and removed based on customer priority/preference change is important because it relates to an XP team's degree of flexibility or agility. Response to Customer Change was not computed for the old release due to lack of user story availability. However, anecdotally, fewer requirements were added during the old release than the new release and the team responded to customer change during the new release.

*Internally-visible Quality*. Internal (pre-release) quality improved by a factor of two. This metric was based upon the defects identified by an external IBM testing organization prior to release to the customer. For the old release, 65 scenarios were tested; for the new release, 96 were tested. Therefore, we assess that the testing effort for the new release was at least as thorough as that of the old release.

*Externally-visible Quality*. The new release's post-release defect density of new code has improved by almost 40%. The defect numbers presented reflect a collection period of nine months after each release. The severity distribution of the pre- and post-release defects was similar between releases. A direct comparison cannot be made because the defect classification scheme changed for the new release. Evidence of similar use of the product by the customer in the old and new releases would aid in determining the accuracy of the post-release defect comparison for this project. According to the customer, the new release was utilized extensively during development (in the form of pre-release drivers), and immediately after official release. The customer states that the amount of use of the new release is comparable to that of the old release. These results support our theory that post-release quality is improved when a small, co-located team uses XP practices.

*Productivity*. The productivity calculation used both user stories and lines of code (LOC) because neither metric is perfect. LOC is a precise metric, but customers pay for features, not LOC; the IBM team tries to reduce LOC via code reuse and refactoring. A benefit of the user stories/PM metric is it creates no extra work for the team, but this metric has not been calibrated. Function points were not used in this metric because they require use of a trained specialist and one was not available. Function points can be estimated from LOC, but the result can be inaccurate [21].

To adjust for differences in the size and duration of the old release versus the new release, the Putnam productivity parameter (PPP) [33, 34] was calculated. This parameter is a macro measure of the total development environment such that lower parameter values are associated with a lesser degree of tools, skills, method and higher degrees of product complexity. The opposite holds true for higher parameter values [33]. The PPP is calculated via the following equation:

$$PPP = (SLOC)/[(Effort/B)^{1/3} * (Time)^{4/3}]$$

Putnam based this equation on production data from a dozen large software projects [34]. Effort is the staff years of work done on the project. B is a factor that is a function of system size, chosen from a table constructed by Putnam based on the industrial data. SLOC is source lines of code, and Time is number elapsed years of the project. The data for SLOC, effort, and time can be found in Table 2. The increase in PPP for the new product supports the increased productivity demonstrated by the other measures, fortifying the results by a normalization of project size and duration. These results support the theory that productivity is improved when a small, co-located team uses XP practices.

*Customer satisfaction*. XP proponents 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. The XP-EF Version 1.3 contains a systematic customer satisfaction survey. This customer satisfaction survey was not available during the IBM case study. Therefore, the customer feedback was obtained qualitatively via a phone interview. For the IBM project, qualitatively the customer was very satisfied with the team's work.

*Morale*. Team morale was assessed via an additional question placed on the Shodan Adherence Survey. The question read, "How often can you say you are enjoying your work?" The survey results indicated

an overall increase in morale as the team utilized more XP practices.

## 5. Discussion and Future Work

The XP-EF framework provides informative feedback utilizing streamlined process and project metrics appropriate for a lightweight software process. Software measurement is a challenging and time-consuming task. Small software development teams require a smaller, more manageable metrics set that provides constructive feedback about their development process. Our proposed metrics were comprehensive enough for this software development team to evaluate the efficacy of their XP practices, while not imposing excessive burden. The framework provided constructive feedback throughout development and allowed the team to improve their adherence to XP practices. However, we acknowledge much work remains to further validate and extend this framework, particularly with regard to the XP adherence metrics.

An active continuation of our research is refining and validating our suite of objective metrics, focusing on those metrics that can be automated. The composition of the XP-EF will be evolved and validated via additional case studies. We consider the following three validation criteria for the XP-EF metric suite:

- The XP-EF will consist of the minimal set of factors necessary to explain the variance in outcome measures observed by various teams.
  - This criteria ensures that the metrics necessary to explain variance between teams are included.
  - This criteria prevents overfit (the collection of metrics that do not provide additional information about variance between teams) and reduces the overhead of metrics collection.
- All internal metrics, particularly those of the XP-am, are validated via the criteria of Schneiderwind [14] in Section 2.2.

Through out international collaboration, we are currently replicating this study with multiple industrial projects[2] to compare the results. We welcome interested researchers to do the same. This family of case studies can be used to create an Experience Factory [3] of XP efficacy knowledge. Once a sufficient number of case studies are available, we will synthesize our results via statistical meta-analysis techniques such as Forrest plots and Rosenthal's file-drawer statistics [18]. We particularly seek unsuccessful XP case studies (that are often tucked away in "file-drawers") which will aid in determining critical success factors for XP. Finally, the structure of the XP-EF provides an adaptable structure for use with case studies with other process methodologies and technologies.

[2] For details of the European case studies, see http://www.agile-itea.org/public/info.php


### Acknowledgements

We wish to thank the IBM team for participating in this case study. Additionally, the participants of the Data Workshop at the XP/Agile Universe conference, Philip Johnson, and the NCSU Software Engineering Reading Group provided helpful feedback. We also wish to thank the anonymous reviewers for their many helpful comments. This research was supported by NCSU CACC Core Grant 02-02.


## References


[1] D. Avison, F. Lau, M. Myers, and P. A. Nielsen, "Action Research," *Communications of the ACM*, vol. 42, no. 1, pp. 94-97, January 1999.

[2] V. Basili, G. Caldiera, and D. H. Rombach, "The Goal Question Metric Paradigm," in *Encyclopedia of Software Engineering*, vol. 2: John Wiley and Sons, Inc., 1994, pp. 528-532.

[3] V. Basili, G. Caldiera, and H. D. Rombach, "The Experience Factory," in *Encycopedia of Software Engineering*, J. C. Marciniak, Ed.: John Wiley, 1994.

[4] V. Basili, "The Role of Experimentation: Past, Present, Future (keynote presentation)," International Conference on Software Engineering, 1996.

[5] V. Basili, Shull, F.,Lanubile, F., "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. Vol. 25, No.4, no., 1999.

[6] R. L. Baskerville, "Investigating Information Systems with Action Research," *Communications of the Associate of Information Systems*, vol. Volume 2, no. 19, October 1999.

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

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

[9] B. Boehm, "Software Risk Management: Principles and Practices," *IEEE Software*, no. pp. 32-41, January 1991.

[10] B. Boehm and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *IEEE Computer*, vol. 36, no. 6, pp. 57-66, June 2003.

[11] D. T. Campbell and J. C. Stanley, *Experimental and Quasi-Experimental Design for Research*. Boston: Houghton Mifflin Co., 1963.

[12] A. Cockburn, *Agile Software Development*. Reading, Massachusetts: Addison Wesley Longman, 2001.

[13] K. El Emam, "Finding Success in Small Software Projects," *Agile Project Management*, vol. 4, no. 11.

[14] K. El Emam, "A Methodology for Validating Software Product Metrics," National Research Council of Canada, Ottawa, Ontario, Canada NCR/ERC-1076, June 2000.

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

[16] H. Gallis, E. Arisholm, and T. Dybå, "An Initial Framework for Research on Pair Programming," International Symposium on Empirical Software Engineering, Roman Castles (Rome), Italy, pp.132-142, 2003.
[17] R. Grady, *Practical Software Metrics for Project Management and Process Improvement*. Englewood Cliffs, NJ: Prentice Hall, 1992.
[18] J. E. Hunter and F. L. Schmidt, *Methods of Meta-Analysis*. Newbury Park: Sage Publications, 1990.
[19] ISO/IEC, "DIS 14598-1 Information Technology - Software Product Evaluation," no., 1996.
[20] T. D. Jick, "Mixing Qualitative and Quantitative Methods: Triangulation in Action," *Administrative Science Quarterly*, vol. 24, no. 4, pp. 602-611, December 1979.
[21] C. Jones, *Software Assessments, Benchmarks, and Best Practices*. Boston, MA: Addison Wesley, 2000.
[22] B. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case Studies for Method and Tool Evaluation," *IEEE Software*, vol. 12, no. 4, pp. 52-62, July 1995.
[23] 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," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721-733, 2002.
[24] B. A. Kitchenham, G. H. Travassos, A. v. Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang, "Towards an ontology of software maintenance," *Journal of Software Maintenance: Research and Practice*, vol. Volume 11, no. 6, November 1999.
[25] W. Krebs, "Turning the Knobs: A Coaching Pattern for XP Through Agile Metrics," Extreme Programming/Agile Universe, Chicago, IL, 2002.
[26] W. Krebs, L. Layman, and L. Williams, "The Extreme Programming Evaluation Framework Version 1.1," North Carolina State University Department of Computer Science TR-2003-17.
[27] M. Marchesi and G. Succi, "Extreme Programming Examined," in *XP Series*, K. Beck, Ed. Boston: Addison Wesley, 2001.
[28] M. Marchesi, G. Succi, D. Wells, and L. Williams, "Extreme Programming Perspectives," in *XP Series*, K. Beck, Ed. Boston: Addison Wesley, 2002.
[29] E. Mayo, *The Human Problems of an Industrial Civilization*. New York: Macmillan, 1933.
[30] J. McGarry, D. Card, C. Jones, B. Layman, E. Clark, J. Dean, and F. Hall, *Practical Software Measurement: Objective Information for Decision Makers*. Boston, MA: Addison Wesley, 2002.
[31] M. C. Paulk, B. Curtis, and M. B. Chrisis, "Capability Maturity Model for Software Version 1.1," Software Engineering Institute CMU/SEI-93-TR, February 24, 1993.
[32] C. Potts, "Software Engineering Research Revisited," *IEEE Software*, no. pp. 19-28, September 1993.
[33] L. H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, Within Budget*. Englewood Cliffs, NJ: Yourdon Press, 1992.
[34] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering*, vol. SE-4, no. 4, pp. 345-361, June 1978.
[35] N. Schneidewind, "Methodology for Validating Software Metrics," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, pp. 410-422, May 1992.
[36] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557-572, 1999.
[37] S. E. Sim, S. Easterbrook, and R. C. Holt, "Using Benchmarking to Advance Research: A Challenge to Software Engineering," International Conference on Software Engineering, Portland, pp.74-83, 2003.
[38] D. Wells and L. Williams, "Extreme Programming and Agile Methods -- XP/Agile Universe 2002," in *Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 2002.
[39] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair-Programming," in *IEEE Software*, vol. 17, 2000, pp. 19-25.
[40] L. Williams, E. M. Maximilien, and M. Vouk, "Test-Driven Development as a Defect-Reduction Practice," IEEE International Symposium on Software Reliability Engineering, Denver, CO, 2003.
[41] L. Williams, G. Succi, M. Stefanovic, and M. Marchesi, "A Metric Suite for Evaluating the Effectiveness of an Agile Methodology," in *Extreme Programming Perspectives*, M. Marchesi, G. Succi, D. Wells, and L. Williams, Eds. Boston, MA: Addison Wesley, 2003.
[42] L. Williams, W. Krebs, and L. Layman, "Extreme Programming Evaluation Framework for Object-Oriented Languages -- Version 1.3," North Carolina State University, Raleigh, NC Computer Science TR-2004-11 available at: http://www.csc.ncsu.edu/research/tech/reports.php, April 7. 2004.
[43] L. Williams, W. Krebs, L. Layman, and A. Antón, "Toward a Framework for Evaluating Extreme Programming," Empirical Assessment in Software Engineering (EASE) 2004, Edinburgh, Scotland, to appear 2004, available at: http://www.csc.ncsu.edu/research/tech/reports.php.
[44] R. K. Yin, *Case Study Research: Design and Methods*, vol. 5, Third ed. Thousand Oaks, CA: Sage Publications, 2003.
[45] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," *IEEE Computer*, vol. 31, no. 5, pp. 23-31, May 1998.