Rational Unified Process Evaluation Framework Version 1.0

William Krebs[1], Chih-Wei Ho[2], Laurie Williams[2], Lucas Layman[2]
[1]*IBM Coporation, krebsw@us.ibm.com*
[2]*North Carolina State University, {cho, lawilli3, lmlayma2 }@unity.ncsu.edu*

1. Introduction

Software development is a complicated human endeavor. It is used in many domains and comes in all shapes and sizes. Development teams use an assortment of software development methodologies to develop software applications. To understand and evaluate the efficacy of these methodologies in a given context, we need to measure a variety of aspects of the development effort and the resulting product. Sound, empirical information on the efficacy of various software development methods can help software engineers make informed decisions on the best method to apply to their individual situation. To facilitate software process evaluation, we have created several process evaluation frameworks designed to capture a variety of information about the development team, the product, and the business-related outcome of different software methodologies. Our first work of software process evaluation framework was the Extreme Programming (XP) Evaluation Framework, or XP-EF. The XP-EF provides a benchmark measurement framework for researchers and practitioners to assess concretely the extent to which an organization has adopted XP practices and the result of this adoption. After several successful and encouraging results, we started to develop evaluation frameworks for other processes. Our goal is to set up a software process evaluation model and a comparison standard across different software development projects, even when different process models are used. In this report, the process evaluation framework for the Rational Unified Process® is introduced. Using the same three-component structure with modifications tailored to the practices of the Rational Unified Process (RUP®), this new addition to the process evaluation framework family, RP-EF, enables us to measure the development effort in a RUP project, and to investigate the effects of process adherence.

This report is organized as follows. Section 2 explains the importance of software development measurement. Section 3 to 5 give the detailed description of the three components in RP-EF, including how each metric is gathered and evaluated. Several examples are also included. Section 6 lists the limitations that a software development case study should report. Section 7 brings the conclusion for this report. Finally, Appendix A provides a subjective adherence survey, Appendix B shows the industry averages of the outcome measures, and Appendices C and D provide interview protocols for stakeholders and team members.

2. The Need for Software Development Measurement

We need measurements to make progress. Take railroad technology for example. The French once held the world record with their TGV lines, which run at an average speed of 157 miles per hour. The Japanese Shinkansen Bullet Trains top TGV lines with an amazing average speed of 162 miles per hour. The French are developing a new TGV line that will run over 200 miles per hour. If we did not know how to measure the speed, we would not know how to compare the speed of the trains running in different countries, and the concept of improvement would never exist.

The measurements for software and software development effort are more complicated. In physical sciences, we can find many rigorously defined rules for measurements. However, software measurements are less refined, although they exist and software engineers make important decisions based on them [FEN91].

In this paper, we propose an evaluation framework for the Rational Unified Process [KRU03] for the purpose of assessing whether transitions to RUP from another software development methodology is beneficial for the team, and to provide a benchmark for comparing case studies of teams using RUP. The evaluation framework for RUP (RP-EF) is based on Extreme Programming Evaluation Framework (XP-EF) [WKL04]. RP-EF contains metrics that we believe are essential to understand the development team, the process, and the software product. Furthermore, these metrics are designed to be:

- Parsimonious and simple enough for a small team to measure without a metrics specialist and with minimal burden
- Concrete and unambiguous
- Comprehensive and complete enough to cover vital factors

As in XP-EF, RP-EF has a three-part structure: RUP Context Factors (RP-cf), RUP Adherence Metrics (RP-am), and RUP Outcome Measures (RP-om). Context factors show the differences and similarities from a team to another; adherence metrics show how rigorously the development follows a particular software method; and outcome measures report how successful the project is. Using RP-EF, we can assess the extent to which an organization has adopted RUP and the result of this adoption. In the rest of this report, we will describe the metric suite in detail, using Goal-Question-Metrics technique [BAS94].

3. RUP Context Factors

Drawing general conclusions from empirical studies in software engineering is difficult because the results of any process largely depend upon the specifics of the study and relevant context factors. We cannot assume a priori that a study's results generalize beyond the specific environment in which it was conducted [BAS99]. Therefore, recording an experiment's context factors is essential for comparison purposes and for fully understanding the similarities and differences between the case study and one's own environment.

Under the guidelines of the GQM, we consider our goal for the defining the context factors:

GOALS:

- *To be able to understand in what ways a particular RUP case study is similar or different from another industrial case study using our process evaluation framework.*
- *To be able to understand in what ways a particular RUP project is similar or different from another industrial project.*
- *To be able to understand the differences between a baseline project /team and the RUP project/team under study.*
- *To build up a body of evidence about RUP and to enable meta-analysis of multiple RUP case studies*

QUESTION: *In what ways can software projects differ*?

Jones [JON00] 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 RP-EF templates are correspondingly organized into these six categories, though we modify the last factor (international) to geographical. We also include developmental factors that use a risk-driven approach for determining whether a project would be most successful using an agile or plan-driven approach [BT03a].

3.1. Software classification

Different types of software from the perspective of the types of customer are often associated with different risk factors and differing characteristics. We provide six classifications to choose from, as prescribed by Jones [JON00] and described in Table 1.

Table 1: Software classification

Systems software	Software that is used to control physical devices, such as a computer or telephone switch. Embedded systems fall into this category.
Commercial software	Software that is leased or marketed to external clients. These software applications are produced for large-scale marketing to hundreds or possibly millions of clients. This includes software such as word processors, spreadsheets, and project management systems.
Information systems	MIS systems are produced in-house to support a company's business and administrative functions, such as payroll and e-business.
Outsourced software	Outsourced software is software built under contract to a client organization. The customer may be external or internal to the company.
Military software	Software produced for a military organization, or that adheres to Department of Defense standards.
End user software	End user software refers to small applications that are written for personal use by people who are computer literate.

3.2. Sociological factors

The sociological factors of the RP-EF are designed to record the personnel characteristics of the project. Personnel and team makeup are documented as top risk factors in software development [BOE91]. Consequently, factors such as team size, experience, and turnover can substantially affect the outcome of a project.

An example of sociological factors is presented in Table 2.

Table 2: Example sociological factors

Context Factor	Metric
Development Team Size	11 developers + 1 tester
Specialist Available	GUI Designer
Team Education Level	Bachelors: 9 Masters: 2
Experience Level of Team	>20 years: 2 10-20 years: 3 5-10 years: 0 <5 years: 2 Interns: 4
Domain Expertise	High
Language Expertise	High
Experience of Project Manager	10 years, High
Software Methodology Experience	10-20 years: 1 5-10 years: 2 <5 years: 4
Personnel Turnover	22%
Morale Factors	Manager change

1) Team Size: Team size and make up can potentially affect project outcome. The complexity of team management grows as team size increases, and communication between team members and the integration of concurrently developed software becomes more difficult for large teams [BRO95].

Count	Do count full time developers and full time testers dedicated to the project under study.
Exclude	Do not count specialists who aid the team part time (such as performance, user interface design, management, project management, or localization specialists, technical writers). These will be recorded in another section.
Additional	Record the team's interaction with any full-time tester(s).

2) Specialist Available: Specialists can help speed up the development process by focusing on specific tasks that may be unfamiliar to other developers, such as GUI design. Other specialists, such as security specialists, can help catch flaws during development and improve deliverable quality.

Count	Include any person not on the core team that helps with the project. Examples would include performance, localization, install, build, team building, database, security, privacy, reliability, usability.
Exclude	Do not count anyone who has already been counted as a full time developer or tester on the team.

3) Team Education Level: Education level may be considered another measure of experience or domain expertise.

Record	Ask team members their level of post-secondary education (Associate, Bachelors, Masters, Ph.D.)
Exclude	Do not count degrees which have been partially completed.

4) **Experience Level of Team:** Developer experience is a critical factor that influences code development. The more experienced the programmer, the more likely he or she is to be able to identify recurring problems during development and draw upon personal knowledge to resolve the issue in an expedient manner. The presence of experienced programmers on a development team can also help less experienced developers gain practical knowledge about the problem domain or the software process in general.

Count	The number of years spent developing software professionally by each full-time team member. Also record the number of interns and/or co-ops.
Exclude	Do not count years spent developing software for personal use or in an academic setting.

5) **Domain Expertise:** A team familiar with the domain of their project may not be subject to the learning curve associated with development that must address an unfamiliar problem. For instance, a team developing web applications may have difficulty adjusting to developing embedded systems, but can produce software efficiently in their own domain.

Record	Estimate or ask about the team's experience with technical factors of the project other than the programming language and including the industry. Examples would be familiarity with database systems, communications, web programming, banking, retail system, agents. Use a scale of Low, Moderate, High to denote the average domain expertise of the team.
Exclude	Programming Language expertise (covered below)

6) **Language Expertise:** Similar to domain expertise, the team may incur significant learning costs when using an unfamiliar programming language. While it may be possible to switch between languages of a similar type (such as object-oriented languages) with a relatively small learning curve, adjusting from the object-oriented paradigm to functional languages may prove more difficult and thus slow development.

Record	Estimate or ask about the team's proficiency with the programming languages used on the project. The scale is Low, Moderate, High to denote the average language expertise of the team.

7) **Experience of the Project Manager:** The familiarity of the project manager with the managerial role can influence the productivity of a team by drawing upon scheduling, cost estimation, and conflict resolution experience. Also, a project manager familiar with the development team may be able to draw upon personal knowledge of the individuals on the team to optimize development by assigning tasks to team members appropriate to their strengths and weaknesses.

Record	Estimate the level of experience of the project manager in that role. Do not consider years of experience in general development unless he/she also served as project manager during that time. Record the number of years as a project manager, and the perceptive level of project management. The scale is Low, Moderate, and High.

8) **Software Methodology Experience:** If a team is more experienced with the software development methodology it is using, the developers can perform the activities and produce the artifacts specified in the methodology more efficiently. Software methodologies require some getting used to before the developers can be productive. When adopting a new software methodology, there will inevitably be a "productivity dip" [BR04]. The dip gets smaller when the team is more experienced with the methodology.

Count	The number of years of experience each full-time team member has for the software development methodology.

9) **Personnel Turnover:** The objective of this metric is to examine potential deficiencies or strengths of how TSP handles people joining or leaving the team. In RUP, artifacts are produced as results of activities. Artifacts are externalized knowledge. However, several studies suggest that externalized knowledge cannot work effectively without tacit, internalized knowledge [HAN99, ROD99]. Having knowledge dispersed across the team increases project safety since the system is not dependent upon the availability of a specific person or persons. Although RUP defines an array of artifacts to keep the software development knowledge, personnel turnover might still seriously hamper the team's performance.

Turnover for a given period of time is measured by counting how many people joined or left the team compared with the team size. This assumes an interval of time under study, such as one release. The counting rule of the number of personnel is the same as used in Team Size.

Count	When counting the number of people on the team, consider only those individuals considered full time members of the team. We use a development period of a release as an example. *Ending Total*: Count the number of people on the team at the end of the release. *Added*: Count the number of people added to the team just prior to or during the release. *Left*: Count the number of people who left the team just prior to or during the release. *Turnover = (Added + Left) / Ending Total*
Example	At the end of the release under study, the team has 11 people. At the beginning of the release, two people move to another project. In the middle of the release, one more person leaves and one other joins the team. At the end of the measurement interval, the team has nine people. Turnover = (1+(2+1)) / 11 = 36%

10) **Morale Factors:** The success or failure of a project is dependent largely on the performance of the development team. When a team suffers from low morale (a possible side effect of extremely long hours), code quality may slip. Similarly, a morale boost may result in a more productive team and/or higher quality code.

Record	Record any factors "outside" of software development that may have affected the team's performance during development, such as abnormal stress levels, cutbacks, holidays, manager change, or personnel transfers.

3.3. Project specific factors

The project-specific factors are designed to help quantify project measurements in terms of size, cost, and schedule. Gauging the size of the project can be done in many ways. An intuitive size measurement is lines of code. A development team may also use their own size measurement. In the rest of this section, we provide detailed description of the project-specific factors for RUP. Table 3 provides an example of the project-specific factors.

Table 3: Example of Project-Specific Factors

Metrics	**Values**
Use Case Change	Elaboration: added 15, dropped 1 Construction: added 2, dropped 1
Domain	Web aplication
Person Months	28.8
Elapsed Months	5
Nature of Project	Enhancement
Relative Feature Complexity	High
Product Age	1.5
Constraints	Date constraint: the delivery date is fixed.

Delivered Use Cases	76
New & Changed Classes Total Classes	139 431
New & Changed Methods Total Methods	486 3,715
New & Changed KLOEC Delta set	9.8 35.5
Component KLOEC	42.8
System KLOEC	240.1

1) **Use Case Change:** Changes are inevitable in software projects. Although changes allow a software team to provide the customers with better solutions, changes also mean more work, more cost, and probably project overdue. Because the cost of software change goes higher in the later stages of software development [BOE81], RUP attempts to accommodate changes as early as possible. Most of requirements efforts are made in the Inception and Elaboration phase. RUP requires a vision statement that all stakeholders agree on at the end of the Elaboration phase, and feature freeze at the end of the Construction phase. We use the change of use cases to show the requirements dynamism. The use case change metrics record the amount and time the use cases are changed.

Count	Count the number of added and dropped use cases after the initial plan, for each phases.

2) **Domain:** Domain is also an important consideration. Different risks are associated with different software domains. Web applications may be concerned with support of large number of concurrent transactions and of a variety of different languages. Yet, the primary concerns of a database project may be scalability and response time. The medical domain also has unique security and/or privacy concerns.

Record	Record the domain in which the application will be used. Potential choices include web application, intranet service, plug-in, developer toolkit, database development, and industry-specific technology.

3) **Person Months:** This metric provides basic documentation of the amount of effort spent on the project. Both Person Months and Elapsed Months are included as context factors because some people work part time on other projects.

Count	Do count the number of person months spent by each person on the project, including partial months and part-time work.

4) **Elapsed Months:** This metric provides basic documentation of the overall schedule of the project in terms of calendar days.

Count	Count the calendar days from the project kick-off to the code is final.
Exclude	Do not count time to manufacture media after code is final (time for pressing CDs, for example).

5) **Nature of the Project:** Effort will be focused in different areas depending upon the nature of a software project. We use the following list to describe the nature of the project.
   - An *enhancement project* may focus on a specific piece of the product, thus narrowing the project's scope and possibly simplifying the overall process.
   - A *new project* is a "Greenfield" start of a new project.
   - A *maintenance project* is primarily concerned with correcting bugs, but must also prevent further defect injection while fixing the system.
   - A *migration project* is a conversion to a new platform, which may involve understanding a significant amount of new technology.
   - A *reengineering effort* is a reimplementation of a legacy system or code translation, for instance, converting a 16-bit system to a 32-bit application.

Record	Whether the project is an enhancement of a previous release, a new product, a maintenance effort, a

	migration, or a reengineering effort.

6) Relative Feature Complexity: Assess the relative complexity of the features (user requirements) of this project compared to past similar team projects. More complex features may have schedule and quality implications.

Record	Estimate the overall complexity of the features in this project compared to past similar team projects. The scale is Low, Moderate, High to denote the overall feature complexity.

7) Product Age: Product age relates to both the availability of product knowledge as well as product refinement. An older product might be considered more stable with fewer defects, but there may be a lack of personnel or technology to support the system. Furthermore, making significant changes to a legacy system may be an extremely complex and laborious task. Working with a newer product may involve instituting complex elements of architectural design that may influence subsequent development, and may be prone to more defects since the product has not received extensive field use.

Record	Record the age of the product in years.

8) Constraints: The presence of constraints significantly increases the amount of risk associated with a project. This factor describes if the project is time boxed, or if the list of requirements is fixed and the date moves to accommodate the content. A fixed-delivery date may force a product out the door before it has been thoroughly tested. Conversely, severe reliability constraints may influence the amount of new functionality that can be introduced into the system.

Count	Any constraints by which the project is bound, e.g. fixed delivery dates, fixed-price contracts, team size limitations, reliability constraints, etc. Examples would be Date Constrained, Scope Constrained, Resource Constrained and any combinations and degrees of these.

9) Project Size: These next three factors document the relative size of the project in terms of number of use cases, lines of code, number of methods, and number of classes. Differencing tools such as Beyond Compare[1] can help in counting these measurements.

9.1) Delivered Use Cases: Because RUP is a use case-centric process, we use the number of use cases for the size measurement. By themselves, neither lines of code nor counts of use cases can adequately represent the whole picture. The number of new (not previously-completed) use cases and the amount of work captured by each use case may vary greatly between organizations and projects. By combining lines of code and new and changed use cases, we hope to offset the disadvantages of each.

Count	Count each use case or requirement completed during this release. Any use cases from a previous release that are re-opened to modification in the current release should be considered new use cases.
Exclude	Defects and deleted use cases.

9.2) Number of Classes: A class is a construction unit in object-oriented programming languages. This measure shows the number of logical units in the system. For non-object-oriented development projects, this measure can be ignored or replaced with another measure for the logical unit of the programming language.

Count	Number of new and changed classes: Count the number of new, changed, and deleted classes. Count the total number of classes. If possible, include details of the number of inner classes as well as the number of regular classes. We consider a changed class to be any class whose functionality is changed by the additional, removal, or modification of code. This does not include formatting or the addition or removal of comments. Also, we count deleted classes as representative of refactored

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

	work. Do not count empty classes – classes that contain no functionality. Changes to abstract, inherited, or interface classes should be counted as one changed class. Also count the number of total classes in the delivered system.

9.3) Number of Methods: A method is a construction unit in object-oriented programming languages. This measure shows the number of logical units in a finer granularity than the number of classes. For non-object-oriented development projects, this measure can be ignored or replaced with another measure for the logical unit of the programming language.

Count	New and changed methods: Count new, changed, and deleted number of methods. Count the total number of methods. Also count the number of total methods in the delivered system.

9.4) Lines of Code: The Delta set is defined as the total lines of code for all classes in which any lines of code were added, changed, or deleted. The Delta set serves as a basis for performing calculations related to quality and productivity in the RP-om section.

Count	Count new, changed, and deleted lines of code. Count total lines of code for all classes in which any lines of code were added, changed, or deleted; this is called the Delta set.

10) Component KLOEC[2]: If the project/release is a component or set of components of a larger project, document the total size of the component. This metric provides information on how much other code the developers need to intimately understand. It also relates to the complexity of the project.

Count	Count total number of non-blank, non-comment lines of the component or components.
Include	All the lines of code in the component or set of components, not just the new and changed lines of code.

11) System KLOEC: Record the size of the system the component or set of components is a part of. This metric provides information on how much other code the developers may need to understand. It also relates to the complexity of the project.

Count	Count total number of non-blank, non-comments lines of the system.
Include	All the lines of code in the system, not just the new and changed lines of code.

3.4. Ergonomic factors

Office environment has profound effects on how people work [DL99]. If a team currently operates in a cubicle or private office environment, the developers may have difficulties with direct communication. Furthermore, physical layouts, such as whether a team works in private offices or in cubicles, or whether there are white-noise generators present, may influence the team's ability to adopt collaborative techniques. Conversely, too much verbal interaction between the developers may become a noisy distraction.

Customers are important to software projects. A project is not successful if it cannot satisfy the customers' need. Therefore, good communication between the development team and the customer can be a critical factor for a successful project. Documenting how communication occurs with the customer can help understand the varying levels of feedback the customer provides during the project. Boehm and Turner point out that, in their analysis of 100 e-services projects, collaborative, representative, authorized, committed, and knowledgeable (CRACK)

[2] Thousands of Lines of Executable (non-blank, non-comment) lines of code

customer representatives are a success factor [BT03a]. We adopt this idea, and use five subjective measures for customer participation. Table 4 is an example of ergonomic factors.

Table 4: Example ergonomic factors

Physical Layout	Cubicles large enough to allow pair programming
Distraction level of office space	Distraction: The location is close to the vending room. Some times a few people gather around the vending machines talking. Facilities to lessen distraction: White noise generators, semi-private cubicles. Overall distraction level: Low.
Customer Communication	E-mail, chat programs, phone, and databases
Customer Participation	Collaborative: 4 Representative: 5 Authorized: 5 Committed: 4 Knowledgeable: 4

1) Physical Layout: This factor records the setting in which the development team works. An open space encourages casual communication among the team members, while cubicles respect personal privacy. The presence of distributed teams is recorded later.

Record	Describe if the team members are located in private offices with doors, in cubicles, in open space, or in some other arrangement. Describe if they are in the same aisle, building, or site. Note if the facilities are conducive to easy communication with team members. It may also be useful to list if a team meeting room or a team recreation room is available.

2) Distraction Level of Office Space: Note distractions the team must deal with during development. An abnormal level of distraction in the working environment can have negative effects on productivity. Distraction reduces concentration, which may result in design or implementation flaws going unnoticed. However, extreme isolation can have negative morale implications.

Record	1. Describe any distractions the team might suffer due to a crowded workspace, proximity to a busy conference room, construction, etc. 2. Note any factors that lessen the amount of distraction, such as white noise generators, working in a sound-proof lab, or private offices. 3. Rate the overall level of distraction on a simple scale such as “Low, Moderate, or High.”

3) Customer Communication: This item describes how closely the team works and communicates with its customers. It may be an on-site customer, or a proxy solution, such as frequent communication with the customer or customer representative.

Record	Describe how often and what means are used to communicate with customers. Note if customer dialog/problem databases, e-mail, phone, chat programs, video conferencing, etc, are used. Also, document if the customer is on-site, or if the primary interaction with the customer is through a representative in the development company.

4) Customer Participation: Software development teams rely on customer representatives to provide them valid requirements. Boehm and Turner point out that CRACK customer representatives are an important factor to a successful project. Without good customer participation, the project might miss the dead line, the development team might deliver a wrong product, and the team morale might be affected. [BT03a] We use subjective measures for the five properties of good customer participation.

Record	Describe the CRACK properties of the customer representatives, on a scale from 1 (very bad) to 5 (very satisfying). This measure should be evaluated by the persons who have direct interaction with the customer representatives. If more than one person have direct interaction with the customer representatives, the average of the evaluations should be recorded.

3.5. Technological factors

Different teams use different methodologies and tools when working on their software projects. Factors such as rigorous project management and a large repository of reusable materials may significantly influence planning accuracy and coding efficiency. Recording this factor may provide insight as to whether hybrid methodologies (using both plan-driven and agile practices) are viable options in software development. Furthermore, different methodologies are associated with different means of project management. For example, users of the waterfall process may use use-cases for feature specification and Gantt charts for scheduling. RUP can be configured to tailor the development process for a given team using plug-ins or the Development Case artifact. Recording the techniques used may provide insight into their effectiveness at estimating schedule and cost. An example of technological factors is displayed in Table 5.

Table 5: Example technological factors

Context Factor	Values
Software Development Methodology	RUP with business modeling support plug-in
Project Management	Gantt charts
Defect Prevention & Removal Practices	Design Reviews
External/System Test	On-site, after code completed, little interaction other than to discuss problems
Language	Java
Reusable Materials	XML test data

1) **Software Development Methodology:** While this is an RUP framework, case studies can compare with pre-RUP project and/or pre-RUP release. As a result, it is important to record the development methodology of the baseline project. Also, recording this factor may provide insight as to whether hybrid methodologies (using both plan-driven and agile practices) are viable options in software development.

Record	Record the development paradigm employed during the project, e.g. waterfall, spiral, RUP, RAD, etc. If you are using a hybrid or customized process, outline what aspects are used from the different paradigms. For documenting RUP customization, Rational provides several RUP tools (for example, RUP Builder and RUP Modeler) that can serve this purpose. Bergström and Båberg also suggest several formal and informal documents to describe the RUP adoption [BR04]. While we do not use a specific technique to record RUP adoption in this framework, we do need to record what process components are used and how they are use.

2) **Project Management:** A variety of project management (PM) tools and techniques may be used create a PM suite tailored to individual organizations. Recording the techniques used may provide insight into their effectiveness at estimating schedule and cost.

Record	Record any project management tools used during development. This may include a project management application, Gantt charts, release planning sessions, planning game, etc.

3) **Defect Prevention and Removal Practices:** Some traditional defect-prevention practices, such as design reviews and code inspections, have been shown to improve the quality of a software product [FAG76, MEY78,

TTN95]. RUP does not specify any form of defect prevention and removal techniques. To find out the effectiveness of different defect prevention techniques, it is important to document which practices are used in order to frame the quality results in the proper context.

Record	Note practices such as design inspections, code reviews, unit testing, pair programming, and employing external testing groups that are meant to reduce the amount of delivered defects.

4) **External/System Test:** Often organizations perform software testing with a group other than the original developers before the code is shipped to the field. The location and the amount of interaction between the developers and this testing group can vary.

Record	The location of the testing group relative to the developers. The timing of when system test activities started. The degree of interaction between the testers and developers.

5) **Language:** Different programming languages share different characteristics. A strongly-typed language may eliminate certain types of bugs while an object-oriented language emphasizes reusability. Also, the particular programming language can drastically affect code volume. Certain metrics may not be available depending on the type of language used in the project as well.

Record	Record the programming languages used by the team during the release.

6) **Reusable Materials:** Reusable materials can greatly speed up the development process by eliminating the need to reproduce the same artifacts over and over. Reusable test data (such as regression tests) can also be used by the team to provide rapid feedback on the impact of a change in code. However, artifacts may become obsolete over time and require effort to remain up-to-date.

Record	Record any items that your development team repeatedly uses during development. These items may include test suites, documentation templates, code templates, code libraries, third-party libraries, etc.

3.6. Geographic factors

Distributed development has become more commonplace in industry, and brings new challenges to the development teams. An example of geographical factors is shown in Table 6.

Table 6: Example 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

1) **Team Location:** Distributed teams that communicate via the Internet are becoming more commonplace, and it is possible that team location and accessibility may influence a software project. Even with groupware and other communication technologies, distributed collaboration is still difficult [OO00]. A distributed team faces more challenges than a co-located team during development. Communication and feedback times are typically increased when the team is distributed over many sites.

Record	Record whether the team is collocated or distributed. A collocated team is found in the same building and area, such that personal interaction is easily facilitated. If the team is distributed, record whether the distribution is across several buildings, cities, countries, or time zones.

2) **Customer Cardinality and Location:** Dealing with more than one customer for a given project may result in a conflict of interest. Customers are the source of software requirements. The development team usually depends on the customer to clarify the requirement statements. The presence of a responsive customer can drastically reduce feedback time, and may improve schedule adherence, performance, and customer satisfaction in the final product.

Record	Record the number of customers (or customer representatives) with which the team interacts. Document if the customer is located on-site, in another city, country, or time zone. Also document if the customer is from a different culture or uses a different language.

3) **Supplier Cardinality and Location:** Some teams are "supplied" code by other development teams which must be integrated into the component(s) or product under development. The presence of a supplier adds to the complexity of the project because the team must understand at least the interfaces of the supplied code and is dependant upon the work of the supplier team(s).

Record	Record the number of suppliers with which the team interacts. Document if the customer is located on-site, in another city, country, or time zone. Also document if the customer is from a different culture or uses a different language.

3.7. Developmental factors

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 [BT03a, BT03b]. Their five project factors (team size, criticality, personnel understanding, dynamism, and culture) are used to select an agile, plan-driven, or hybrid process. These factors are graphed on a polar chart's five axes as shown in Figure 1. The agile risk factors appear toward the graph's center and the plan-driven risk factors appear toward the periphery. 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. As an industry, it would be beneficial to run as many case studies with varying polar chart shapes to validate the risk-based approach of Boehm and Turner.

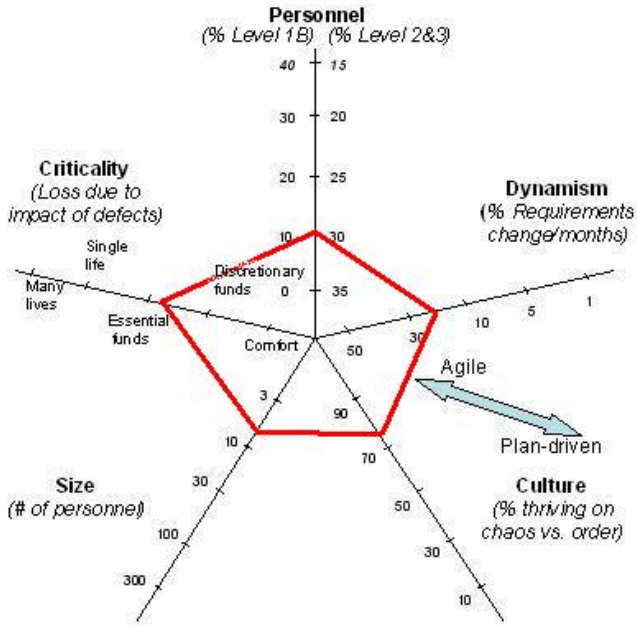

Figure 1: Example developmental factors polar chart

1) **Dynamism:** Dynamism is the percentage of requirements change per month. Assuming there is some sort of plan (a release/iteration plan), how much does actual development deviate from that plan at the end of the iteration/release? Are user requirements volatile due to changing customer requests? It is best to be as quantitative and accurate as possible. We use the following formula to assess the dynamism of a RUP project:

$$\frac{\#\text{ of changed use cases (including deleted and deferred use cases)} + \#\text{ of added use cases}}{\text{total number of use cases delivered}}$$

However, formally tracking requirements changes can be difficult and, if so, a subjective measure can be used. Although accurate requirements change rate can only be measured after the development started, estimating expected requirements dynamism can aid in the choice of an appropriate methodology.

2 **Criticality:** Criticality is the impact on the customer of a software failure. What is the worst thing that can happen if the software fails? Is it possible that many people will lose their lives, that the company will lose critical funds, or that it will simply be a loss of comfort? A failure in an air traffic control system could be catastrophic, whereas a failure occurring in an information kiosk may simply be an inconvenience. The company may also lose very important monies, or a small amount.

3) **Personnel:** Record the percentage of personnel on the development team at the various Cockburn personnel levels [BT03a, modified from COC01] described in Table 7.

Table 7: Cockburn personnel levels [BT03a, modified from COC01]

Level	Team member characteristics
3	Able to revise a method, breaking its rules to fit an unprecedented new situation.
2	Able to tailor a method to fit a new situation.
1A	With training, able to perform discretionary method steps such as sizing stories to fit increments, composing patterns, compound refactoring, or complex COTS integration. With experience, can become Level 2.
1B	With training, able to perform procedural method steps such as coding a simple method, simple refactoring, following coding standards and configuration management procedures, or running tests. With experience, can master some Level 1A skills.
-1	May have technical skills but unable or unwilling to collaborate or follow shared methods.

4) **Size:** The number of full-time people on the team, as recorded in the sociological factors (Table 2).

5) **Culture:** Culture measures the percentage of the team that prefers chaos versus the percentage that prefers order. Does most of the team thrive in a culture where people feel comfortable and empowered by having their roles defined by clear policies and procedures? If so, they tend toward order. If your team is more comfortable by having many degrees of freedom, then they tend toward chaos.

4. RUP Adherence Metrics

The second part of the RP-EF is the RUP Adherence Metrics (RP-am). RUP is a process framework that defines a full set of roles, activities, artifacts, workflows, and disciplines [KRU03]. RUP provides tools for all aspects of software development, but it is impractical, or even impossible, to use all the techniques specified in RUP. A software team needs to customize the process based on their need. The adherence metrics measure how closely the team follows the principles of RUP [KRO01] and RUP best practices [KRU03].

Under the guidelines of the GQM, we consider our goal for the defining the adherence metrics:

GOAL: To be *able to understand how closely a team actually follows the RUP principles and best practices.*

QUESTION: *How much do the team members follow each of the RUP principles and best practices*?

The RP-am enables one to express concretely and comparatively the extent to which a team follows RUP. The RP-am also allows researchers to investigate and compare the outcome of different RUP adoption cases. The RP-am contains subjective and objective measures (described below) as well as qualitative analysis about the team's use of RUP to triangulate the extent to which a team follows.

4.1. Objective Measures

The principles of RUP [KRO01] and RUP best practices [KRU03] are more guidelines than concrete software development methodology. Consequently, most of them cannot be measured with objective metrics. The only two RUP practices and principles that can be measured objectively are iterative development and continuous quality verification. Therefore, the objective measures listed in this section are all related to those two practices. We rely on subjective measures (Section 4.3) to measure the rest of RUP adherence. Table 8 is an example of RP-EF objective adherence metrics.

Table 8: RP-EF Objective Adherence Metrics Example

Metrics	Values
Iteration Length	Inception: 10 days Elaboration: 10 days Construction: 15 days Transition: 10 days
Test Coverage	78.2%
Test Run Frequency	23%
Test Class to Use Case Ratio	23.6
Test Code to Source Code Ratio	2.7
New and Changed Classes with a Corresponding Test Class	85.2%
Percentage of New Class with a Corresponding Test Class	94.3%
Levels of Testing	Unit testing and system testing
Testing Methods and Tools	Unit testing: automated JUnit testing System testing: manual
Unit Test Testware Volume	87.2 KLOC
Review Styles	Unit test code: none Source code: peer review Other documentations: peer review and group review

1) Iteration Length: RUP is an iterative process. In RUP, the software development lifecycle is divided into four phases: Inception, Elaboration, Construction, and Transition. Each phase contains one or more iterations. Iterative development helps to surface risk factors earlier. This metric describes the length of iterations.

Count	Count the average length of the iterations, in the number of days, for different phases.

2) Test Coverage: The objective for this metric is to measure what proportion of the lines of code is executed via automated unit tests.

Count	Count the lines of the *delta set* (new and changed code) executed during execution of the total test suite (RUN). Count the total lines of code of the delta set (TOTAL). Divide RUN by TOTAL.

3) Test Run Frequency: While a set of test cases may be available, the QA team or the developers will run them. This metric determines how often test suites are run.

Count	• Count the number of person days per week. • Count the number of times each person runs the team's automated unit test suite (not only their own tests). This number can be collected via automated or manual means.

	• Divide number of runs by the number of person days.
Example	Example: on a team of 10, one person runs the test suite 3 times per day, 8 people run it once per day, and one person never runs them. Average these numbers. The average for the week is 1 person @ (3 runs / day * 5 days / week) + 8 people @ (1 run / day * 5 days / week) + 1 person @ (0 runs / day * 5 days / week) = (15 + 40) / (10 people * 5 days / week) = 55 / 50 = 1.1

4) **Test Class to Use Case Ratio:** This metric describes the amount of automated testing effort for each use case.

Count	Count the number of automated test classes (CLASS) and the number of use cases (UC). Divide CLASS by UC.

5) **Test Code to Source Code Ratio:** Examine the ratio of test lines of code to source lines of code as a relative measure of how much test code is written by the team.

Count	• Count the KLOEC of automated test code run frequently written to test the delta set (TEST LOC). • Count the KLOEC of new and changed production source code (SOURCE LOC). • Divide the TEST LOC by SOURCE LOC.

6) **New and Changed Classes with a Corresponding Test Class:** Assessing the amount of test code written via source code ratios or statement coverage can be misleading. When dealing with a legacy system, there may be a significant amount of source code that has not been unit tested. To assess the testing effort on the current project, we measure the number of New and Changed source classes that have a corresponding test class. In accordance with the principles of refactoring, every new and changed piece of code should have a corresponding unit test.

Count	Identify the source classes with new, changed, or deleted code. Count these classes (CHURNED CLASS). Identify the unit test classes that test the new and changed classes in the delta set. Count these unit test classes (TEST). Divide TEST by CHURNED CLASS.

7) **Percentage of New Classes with a Corresponding Test Class:** This metric supplements the Test Classes/New & Changed Classes metric by identifying what percentage of new source classes have a corresponding unit test class.

Count	• Count the number of new source classes (NEW CLASS). • Identify the unit test classes that test these new source classes. Count these unit test classes (TEST CLASS). • Divide NEW CLASS by TEST CLASS.

8) **Levels of Testing:** This metric describes the levels of testing the development team develops and runs throughout the development lifecycle.

Record	Record the levels of testing (unit testing, integration testing, system testing, and acceptance testing) facilitated by the team.

9) **Testing Methods and Tools:** This metric describes the methods and/or tools that are used for testing.

Record	Record the methods and/or tools that are used for different levels of testing. For manual testing, write "Manual." For automated testing, record the tools and languages that are used.

10) **Unit Test Testware Volume (KLOC):** This metric describes that amount of automated unit testing code.

Count	Count the lines of automated test code, in KLOC.

11) Review Styles: The metric describes the review styles that are used by a team. There are different types of artifacts in RUP, and different review styles might be used for different artifacts. Describe the review styles with as much detail as possible.

Record	The review styles (for example, peer buddy review or formal group review) that are used for different artifacts.

4.2. Qualitative Inquiries

Qualitative methods can be used to enrich quantitative findings with explanatory information, helping to understand "why" and to handle the complexities of issues involving human behavior. Seaman [SEA99] 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 [SEA99]. Semi-structured interviews are a mixture of open-ended and specific questions designed to elicit unexpected types of information. The RP-EF uses semi-structured interviews to gather qualitative data to aid in understanding the RP-am's quantitative metrics. Interviews are preferably conducted in person [KAN03], and interview data should be anonymous. No identifying information should be collected from the interviewee, though developer experience may be solicited to help understand the background of the interviewees. The information obtained from the interviews is used to explain the phenomenon displayed in the RP-cf and RP-am metrics and the overall project results (RP-om). An example team member interview is found in Appendix D.

4.3. Subjective Measures

The RUP version of an in process survey, which is shown in Appendix A, is adapted from the XP Shodan Adherence Survey [KRE02]. The Shodan Survey is an in-process, subjective means of gathering RUP adherence information from team members. The survey, answered anonymously via a web-based application, contains 18 questions gauging the extent to which each individual follows the RUP principles and best practices. The survey is taken by the developers only, not by any of the support personnel (such as tester, security specialists, etc.). A survey respondent reports the extent to which he/she uses each practice on a scale from 0 (never) to 100% (always). The results of the survey can be used by team leaders to see how much a practice is used, the variation of use among team members and, trends of use over time. Tracking The Shodan Survey results for a team is useful. However, since the Shodan Survey is subjective, it is not advisable to compare survey results across teams.

In addition to the benefit of serving as a cross check to other adherence metrics, the survey has other benefits. If the survey is administered throughout development, early indication of failing to follow a critical RUP principle or practice allows the course adjustment before the release is over. The survey can also help encourage the use of development activity by serving as a reminder of what the activities are. Also, these subjective measurements are oftentimes the only ones available since it is sometimes extremely difficult to determine an objective metric for evaluating a practice. Table 9 shows an example of the Shodan Survey result.

Table 9: Example subjective metrics

Subjective metric (Shodan)	Mean (std dev)
Vision Statement	55% (22.2)
Iterative Development	67% (22.1)
Iteration Feedback	78% (6.9)
RUP Milestones	90% (14.1)
Manage Risk	77% (9.4)
Manage Requirements	87% (4.7)
Manage Change	85% (10.0)
Architecture-Central Development	68% (14.6)
Component Based Architecture	57% (14.9)
Visual Modeling	78% (6.9)
Continuously Verify Quality	83% (7.5)

Automated Unit Tests	78% (13.4)
Test Run Frequency	82% (3.7)
Use Case Based Requirements	77% (9.4)
Use Case Driven Design	43% (18.9)
Use Case Driven Tests	47% (10.1)
Feedback and Retrospective	87% (15.3)
Morale	86% (7.4)

5. RUP Outcome Measures

Part three of the RP-EF is the RUP Outcome Measures (RP-om), which are business-oriented metrics that enable one to assess and report how successful or unsuccessful a team is when using RUP to develop software. Software quality, development productivity, and customer satisfaction are often considered successful factors, and are thus included in the outcome measures. Another measure of interest is team morale, which shows how the team accepts the software process.

In our experience, confidentiality is usually the concern when reporting outcome measures. One possible way to deal with this issue is to report the measures on relative scales to industrial averages. We provide the industrial averages from Jones' work [JON00] in Appendix B.

Under the guidelines of the GQM, we consider our goal for the defining the results metrics:

GOAL: *To build theories about whether the business-related results of a team change when RUP is adopted.*

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

QUESTION 1: Does the pre-release quality improve when a team uses RUP?
QUESTION 2: Does the post-release quality improve when a team uses RUP?
QUESTION 3: Does programmer productivity improve when a team uses RUP?
QUESTION 4: Does customer satisfaction improve when a team uses RUP?

The RP-om is comprised of quantitative output measurements of productivity and quality, as well as qualitative information gathered from team member interviews. Interviews are also used to obtain customer feedback on the process. The RP-om metrics can be reported on a relative scale to protect proprietary information as needed. Table 10 shows an example of outcome measures and the industrial averages.

Table 10: Example and averages of outcome measures

XP Outcome Measure	Measurements
Pre-release Quality	Test defect/KLOEC of code: 2.1 Severe defect: 2.3% Moderate defect: 23.5% Minor defect: 74.2%
Post-release Quality	6 months after release Defect/KLOEC of code: 1.3 Severe defect: 6.3% Moderate defect: 15.7% Minor defect: 78.0%
Productivity KLOEC / PM Putnam Productivity Parameter	 2.3 1.92
Customer Satisfaction	Very satisfied

5.1. Quality Measures

1) **Pre-release Quality (test defects/KLOEC):** This metric reflects quality exposed during test before it is release to a customer such as is done by an external testing group within an organization. The metric is a *surrogate* measure of quality [KIT96]; it is also a measure of testing efficiency.

Count	Count the number of defects found in the new and changed code. These defects are found during final system-level testing before the product is released to a customer. This testing can be done by the development team or by an external testing team in the development organization. Count the lines of code of the Delta set.
Exclude	Do not count defects discovered or reported during the release but not in the new or changed lines of code. Do not count those defects found by the developers during unit testing. Do not count bugs in reused binary code libraries from other teams not a part of the study. Do not count bugs that were reported but were duplicates or irreproducible. Do not count bugs in test or sample code not shipped to customers. Do not count bugs found by the compiler or during pairing or inspections. Do not count bugs injected after the release (those belong in Post-release Quality).
Additional	Include information on the severity of the defects found as an indication of whether defects were critical or minor. Also, it is advisable to include information about the testing effort exhibited by the testing team. Testing effort can be measured in terms of person-months, number of scenarios tested, etc. Also document when during development external testing takes place, e.g. throughout development, just prior to release, etc. This information is very important for comparing pre-release defect density. A lower pre-release defect density could be due to improved product quality or a non-thorough testing effort.

2) **Post-release Quality (released defects/KLOEC):** Perhaps of greater business importance is the post-release quality metric, which documents those defects found by the customer after release.

Count	Count the number of defects found in the new and changed code by a customer after release. Count the lines of code of the Delta set.
Exclude	Do not count defects discovered or reported but not in the new or changed lines of code. Do not count bugs that were reported but were designated as something like duplicates, works as designed, or irreproducible.
Additional	Include information on the severity of the defects found as an indication of whether defects were critical or minor. Additionally, provide the amount of time since the product shipped.

5.2. Productivity Measures

1) **KLOEC/PM:** LOC is a precise and easy-to-calculate metric, but does not capture the number of system requirements met of features delivered to the customer. The lines of code per person-month (PM) metric is gathered for comparison to traditional data and to serve as a cross-check to the user stories per PM metric.

Count	Count the lines of code of the Delta (see New and Changed Lines of Code in Section 3.3). Divide both these measures by total person-months on the project

2) **Putnam Productivity Parameter:** To adjust for differences in the size and the duration between projects, the Putnam productivity parameter (PPP) [PUT78, PM92] is 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 [PM92]. The PPP is calculated via the following equation:

$$PPP = \frac{\text{SLOC}}{(\text{Effort}/B)^{1/3} \times \text{Time}^{4/3}}$$

Putnam based this equation on production data from a dozen large software projects [PM92]. 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 (Table 11). SLOC is source lines of code, or the Delta set, and Time is number elapsed years of the project.

Table 11: Putnam system size factor *B* [PM92]

Size (SLOC)	***B***
5-15K	0.16
20K	0.18
30K	0.28
40K	0.34
50K	0.37
> 70K	0.39

5.3. Customer Satisfaction

Currently in the RP-EF, interviews are used to qualitatively assess the level of customer satisfaction and an interview instrument is provided in Appendix C. The outcome measure that is reported is the customer's answer to the question "What is your overall satisfaction with the project?" on a scale from 1-5 (very dissatisfied, dissatisfied, neutral, satisfied, very satisfied). In the event of multiple customers, averaging the answer may be a plausible alternative, or documenting the individual answers. The other questions in the template survey may serve as helpful explanatory information and to help guide process improvement.

Record	Document the customer's satisfaction with the given release of the product, either through interviews or via a customer satisfaction survey. A template customer satisfaction survey has been attached in Appendix C that can be used as a guide for customer interviews.
Exclude	Do not record customer satisfaction information unrelated to the product or the development team, e.g. customer service interaction, installation services, documentation, etc.

6. Limitations

The limitations of the case study should be enumerated. We suggest the use of the four criteria for judging the quality of research design set forth by Yin [YIN03]. The four criteria are listed in this section.

1) **Construct validity:** Construct validity involves establishing measures for the concepts being measured. We need to make sure that the metrics we use really measure the concepts in a case study.
2) **Internal validity:** Internal validity shows whether a study actually addresses the researcher's intension. There are several factors that might threaten internal validity. When working with only one case, it is difficult to determine that whether some factors outside of the research affect the result. This is called *single group threats*. When working with multiple groups that are not strictly comparable, there might be some other causes outside of the study that affect the results. This is called *multiple group threats*. Another kind of threat, *social interaction threats*, occurs when different subjects in a study have some interaction that can affect the result. For example, when doing a study about how a software development practice affects the outcome, we might apply the practice to some teams, and that might make other teams feel treated unfair. The threats are usually referred to as *confounding variables*.
3) **External validity:** Sometimes called *generalizability*, is the degree to which a study can be generalized. When conducting a case study, we need to make sure that the representativeness of the subject is relevant.
4) **Experimental reliability:** Experimental reliability shows whether the effect of a case study can be reproduced in another. This implies that if two teams have similar context and adherence factors, they should also have similar outcome. If this is not the case, there are some missing factors.

7. Conclusion

The RP-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. There proposed metrics are comprehensive enough for RUP development teams to evaluate the efficacy of their process adoption, while not imposing excessive burden. The framework provides constructive feedback throughout development and allows the team to improve their adherence to essential RUP process components. However, much work remains to be done to validate and extend this framework, particularly with regard to the adherence metrics for RUP. We hope we can apply this framework to RUP projects of different scales to show the effectiveness of the framework and refine the metrics at the same time. An active continuation of this research is refining and validating the suite of objective metrics, focusing on those metrics that can be automated. We are interested in working with researchers around to world to establish a family of RUP case study research and to use meta-analysis to combine the results.

Trademarks

References

[BAS94] Basili, V., G. Caldiera, and D. H. Rombach, "The Goal Question Metrics Paradigm," in *Encyclopedia of Software Engineering*, vol. 12, pp. 528-532, John Wiley and Sons, Inc., 1994.

[BAS99] Basili, V., F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, 1999.

[BOE81] Boehm, B., *Software Engineering Economics*, Prentice Hall, 1981.

[BOE91] Boehm, B., "Software Risk Management: Principles and Practices," *IEEE Software*, vol. 8, no. 1, pp. 32-41, 1991.

[BRO95] Brooks, F. P., *The Mythical Man-Month: Essays on Software Engineering Anniversary Edition, Addison-Wesley*, 1995.

[BR04] Bergström, S. and L. Råberg, *Adopting the Rational Unified Process: Success with the RUP*, Addison-Wesley, 2004.

[BT03a] Boehm, B. and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison Wesley, 2003.

[BT03b] Boehm, B. and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *IEEE Computer*, vol. 36, no. 6, pp. 57-66, 2003.

[CH05] Coplien, J. O. and N. B. Harrison, *Organizational Patterns of Agile Software Development*, Pearson Prentice Hall, 2005

[COC01] Cockburn, A., *Agile Software Development*, Addison-Wesley, 2001.

[DL99] Demarco, T. and T. Lister, *Peopleware: Productive Projects and Teams, 2nd Edition*, Dorset House Publishing Company, 1999.

[FAG76] Fagan, M. E., "Advances in Software Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, vol. 15, pp. 182-211, 1976.

[FEN91] Fenton, N., *Software Engineering with Formal Metrics*, QED Publishing, 1991.

[HAN99] Hansen, M.T., N. Nohria, and T. Tierney, "What's Your Strategy for Managing Knowledge?" *Harvard Business Review*, March-April 1999, pp. 106-16.

[JON00] Jones, C., *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, 2000.

[KAN03] Kan, S. H., *Metrics and Models in Software Quality Engineering, Second edition*, Addison-Wesley, 2003.

[KIT96] Kitchenham, B., *Software Metrics: Measurement for Software Process Improvement*, Blackwell, 1996.

[KRE02] Krebs, W., "Turning the Knobs: A Coaching Pattern for XP through Agile Metrics," *Proceedings of Extreme Programming/Agile Universe*, 2002, 60-69.

[KRO01] Kroll, P., "The Spirit of the RUP," *The Rational Edge*, December, 2001. Also available online at http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/dec01/TheSpiritoftheRUPDec01.pdf

[KRU03] Kruchten, P., *The Rational Unified Process: An Introduction, 3rd Edition*, Addison-Wesley, 2000.

[MEY78] Meyers, G., "A Controlled Experiment in Program Testing and Code Walkthrough/Inspection," *Communications of the ACM*, vol. 21, pp. 760-768, 1978.

[NEI05] Neiderman, F., "Staffing and Management of E-Commerce Programs and Projects," Proceedings of the 2005 ACM SIGMIS Conference on Computer Personnel Research, pp. 128-138, Atlanta, Georgia, 2005.

[PM92] Putnam, L. H. and W. Myers, Measures for Excellence: *Reliable Software on Time, Within Budget*, Yourdon Press, 1992.

[PUT78] Putnam, L. H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering*, vol. SE-4, pp. 345-361, 1978.

[TTN95] Takagi, Y., T. Tanaka, N. Niihara, K. Sakamoto, S. Kusumoto, and T. Kikuno, "Analysis of Review's Effectiveness Based on Software Metrics," *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pp. 34-39, Toulouse, France, 1995.

[OO00] Olson, G. M. and J. S. Olson, "Distance Matters," *Human-Computer Interaction*, vol. 15, pp. 139 – 178, 2000.

[ROD99] Rodhain, F., Tacit to Explicit: "Transforming Knowledge through Cognitive Mapping – An Experiment", *Proceedings of the 1999 ACM SIGCPR Conference on Computer Personnel Research*, pp. 51-56, New Orleans, Louisiana, 1999.

[SEA99] Seaman, C. B., "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, 1999, pp. 557-572

[WKL04] Williams, L., W. Krebs, and L. Layman, "Extreme Programming Evaluation Framework for Object-Oriented Languages Version 1.4," North Carolina State University Department of Computer Science, TR-2004-18, Raleigh, NC, June 2004.

[YIN03] R. K. Yin, *Case Study Research: Design and Method, 3rd Edition*, Sage Publications, 2003.

Appendix A: Shodan In-Process Survey for RUP v1.2

This survey determines which software development 'best practices' you tend to use. While taking the survey you may learn more about some new practices. The survey should take about 15 minutes to complete. You may want to take this after every iteration as part of your iteration assessment.

The 'How often' questions use a scale from 0 to 10:

10: Fanatic (100%)	9: Always(90%)	8: Regular(80%)
7: Often (70%)	6: Usually (60%)	5: Half n Half(50%)
4: Common(40%)	3: Sometimes(30%)	2: Rarely(20%)
1: Hardly ever(10%)	0: Disagree with this practice	

The date, team, and user handle are collected in addition to the survey questions. In addition to these questions, you may ask others at the same time of special interest to the team such as code review style and frequency.

Vision Statement
A vision statement helps align the team on a common goal. *To what extent does your team have an explicit vision for the product?*

Iterative Development
Iterative development allows code to be broken down into manageable units. This reduces risk, and allows early test and user feedback. Iterations of 1, 2, or 4 weeks would be typical. *Do you break down development deliveries into bite size chunks?*

Iteration Feedback
It's important to not just iterate the drivers, but also to gather stakeholder feedback. *How often do you gather feedback on each iteration?*

RUP Milestones
Do you identify the Inception, Elaboration, Construction, and Transition milestones in your product life cycle? *To what degree to you use the four RUP lifecycle milestones?*

Manage Risk
Managing risk up front will save potential cost later in the cycle. *Do you have a Risk List? Do you identify risks and plan to avoid them up front?*

Manage Requirements
Careful focus on storing requirements allows the list to be managed under changing pressures. *What proportion of your requirements are tracked and managed in a database?*

Manage Change
Change management is important not just because it uses a library or repository to check in and check out code, but because it can notify other team members when new information is available. *How often are people notified of changes to designs, defects, or processes?*

Architecture
The architecture of your product serves as a foundation for future change. Its documentation can remove errors and can provide a tool to teach the product to new team members. An executable architecture gives you an early picture of the system. *How often do you use your design model?*

Component Based Architecture
Your designs break the system down into reusable components. Reuse is one way to scale productivity up to needed levels. *To what extent is your design based on a Component Architecture?*

Visual Modeling

Graphing your architecture in a model in UML notation allows developers to better understand the design, and if it's linked with the code, will help developers keep the code in sync with the model. *How often do you use UML Modeling to describe your designs?*

Continuously Verify Quality
Artifacts and Use-Cases should be tested near the end of each iteration in which they are produced. This gives early feedback and warning of design problems. In addition, the testing should be repeated for key iterations. This differs from the traditional model where most testing occurs later in the cycle. The tests should be derived from Use-Cases. *Do you emphasize early and continuous test efforts?*

Automated Unit Tests
You have automated unit tests for your code (such as JUnit or Rational Functional Tester). *What % of your changes are tested with automated unit tests?* They may be run after the code is checked in and built.

Test Run Frequency
You run an quick set of automated regression tests before you check in your code. *What % of your changes are checked in **After** you run a quick set of automated regression tests?*

Use-Case based Requirements
How often do you describe requirements from use cases or a scenario perspective

Use-Case based Design
How often do you derive designs from use cases or from a scenario perspective

Use-Case based Tests
How often do you derive tests from use cases or a from a scenario perspective?

Lessons Learned
The team reviews how to get better after every release.

Morale
How often can you say you're enjoying your work?

Appendix B: Industrial Averages of the Outcome Measures

The following information is adopted from [JON00].

Category	MIS			Outsourced			Systems		
Size (FP)	100	1,000	10,000	100	1,000	10,000	100	1,000	10,000
Pre-Release Quality									
Test defect/FP	3.00	4.90	5.90	3.00	4.80	5.60	5.00	5.50	6.60
Post-Release Quality									
Test defect/FP	0.150	0.588	1.062	0.090	0.240	0.392	0.250	0.440	0.726
High-Severity Defects (%)	13.33	14.97	15.00	11.11	15.00	15.00	15.20	15.00	15.00
Productivity									
FP /Staff month	27.80	9.46	3.44	30.49	12.68	4.82	10.10	4.13	2.05

Category	Commercial			Military			End User
Size (FP)	100	1,000	10,000	100	1,000	10,000	
Pre-Release Quality							
Test defect/FP	5.00	5.50	6.60	5.25	5.75	6.80	1.8
Post-Release Quality							
Test defect/FP	0.250	0.495	0.792	0.263	0.518	0.816	
High-Severity Defects (%)	15.20	15.01	15.00	14.83	15.00	15.00	
Productivity							
FP /Staff month	12.68	5.13	2.07	7.69	2.48	0.88	26.00

Appendix C: Customer Satisfaction Survey

Project-specific questions

When prompted, the scale for the questions below is:
1. Very Dissatisfied 2. Dissatisfied 3. Neutral 4. Satisfied 5. Very Satisfied

1) What product do you use?

2) What version of this product do you use?

3) On a scale from 1-5, how satisfied are you with the *reliability* of this product?

4) What are the effects of any *reliability* problems in this product? Please comment.

5) On a scale from 1-5, how satisfied are you with the product's *capabilities?* That is, does it meet your needs in terms of features and functionality?

6) In what ways do the product's *capabilities* fail to meet your expectations? Please comment.

7) On a scale from 1-5, how satisfied are you with *communication* with the development organization? Rate this on the basis of communicating with and receiving feedback from the development team or marketing representative. Do not base this comparison on communications with customer support or other avenues.

8) Please describe *who* you interact with at the development organizations, e.g. a marketing representative, project manager, developers themselves. If you interact with more than one contact, please indicate which is the primary contact.

9) On a scale from 1-5, what is your overall satisfaction with the product?

Longitudinal Product Comparison

When prompted, the scale for the questions below is:
1. Much worse 2. Worse 3. About the same 4. Better 5. Much better

10) On a scale from 1-5, how would you rate this product's *reliability* in comparison to past versions of the *same* product? If there is no basis for comparison, write N/A.

11) What factors do you believe contributed to this change, if any?

12) On a scale from 1-5, how would you rate this product's *capabilities* in comparison to past versions of the *same* product, i.e. are your expectations comparably met to those of past versions of the product? If there is no basis for comparison, write N/A.

13) What factors do you believe contributed to this change, if any?

14) On a scale from 1-5, how would you rate the level of *communication* with the development organization in comparison to past versions of the *same* product? If there is no basis for comparison, write N/A. Rate this on the basis of communicating with and receiving feedback from the development team or marketing representative. Do not base this comparison on communications with customer support or other avenues.

15) What factors do you believe contributed to this change, if any?

Latitudinal Product Comparison

16) On a scale from 1-5, how would you rate this product's *reliability* in comparison to other products you have purchased from software development organizations? If there is no basis for comparison, write N/A.

17) Please comment on your answer to 15.

18) On a scale from 1-5, how would you rate this product's *capabilities* in comparison to other products you have purchased from software development organizations, i.e. are your expectations comparably met to those of past versions of the product? If there is no basis for comparison, write N/A.

19) Please comment on your answer to 17.

20) On a scale from 1-5, how would you rate the level of *communication* with the development organization in comparison to other software development organizations? If there is no basis for comparison, write N/A. Rate this on the basis of communicating with and receiving feedback from the development team or marketing representative. Do not base this comparison on communications with customer support or other avenues.

21) Please comment on your answer to 19.

Appendix D

Example developer interview template

Hypothetical, based upon survey response for chosen practices
Refactoring: 57%
Pair programming: 68%
Test-first design: 55%

Developer experience:
Years developing software:
Years with company:
Years with this team:

Project experiences
What do you like and dislike about the XP practices?

Do you think your project is better using agile methods as opposed to a plan-driven traditional method?

Practice-specific questions
Refactoring was a low survey score. Why do you think so?

Is there resistance to programming in pairs?

What keeps people from pairing?

When does pairing take place?

Is your coach supportive of pair programming?

What was the software testing process adopted in the project?

Were customer acceptance tests available? Do you feel they are important?

What were the major hurdles in adopting automated tests?