# Re-engineering Issues and Opportunities in XP key adaptive practices

[1]K. Gowthaman, [2]K. Mustafa and [3]R. A. Khan
[1]Department of Computer Science,
[1,3]Jamia Millia Islamia (A Central University), New Delhi, India. [1]{gowthaman@computer.org},
[3]{rakhan_cs@jmi.ernet.in}
[2]Department of Information Technology,
[2]Al-Hussein Bin Talal University, Ma'an, Jordan {kmfarooki@ahu.edu.jo}

## Abstract

Legacy software must often be reengineered, a process that involves both reverse and forward engineering. No doubt, there is a lack of resource requirements, design, or design rationale documentation for legacy software. This lack of resource requirement means an unavailability of traces as well, making reengineering difficult and expensive. In this paper we arise the reengineering task and their issues at Design recovery side. We present the re-engineering project case study reports and share the experiences gained in a large-scale industrial reengineering project. We also refer the possible key practices from existing software process called Extreme Programming and their adoptive practices at other side. This is the extension of the work we published in Developer IQ [9].

**KeyWords:** Re-engineering, Reverse Engineering, Pair programming, Refactoring, Unit Testing.

## 1. Introduction

Reengineering is required when an organization faces the challenge of migrating from legacy systems to new target systems. Legacy system must often be reengineered through the process of reverse and forward engineering [3]. Figure 1 shows simple reengineering process flow diagram.

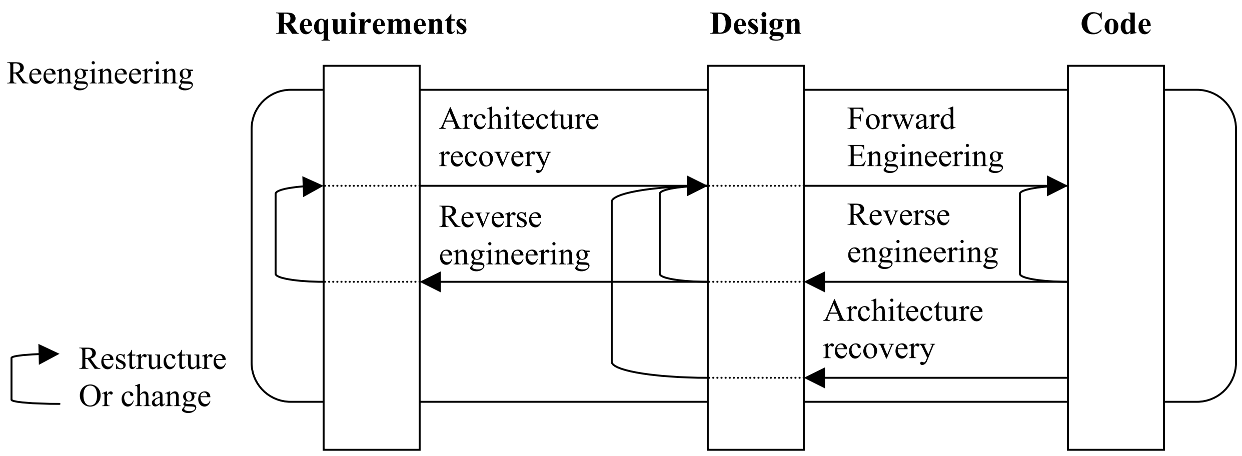

**Figure :1 Simple reengineering process flow diagram.**

The reengineering approaches include user interface compatibility, database compatibility, transition support, system interface compatibility, and training etc. The main limitations of current approaches include, may not be limited to, the following.

• Various third-party tools are available that translate source program from one programming language to another programming language. These Language translation tool does not provide design models of the application.

• Programs produced by language translator are poorly structured, contain cryptic variable names, use non-optimal data structures. Maintainability of such programs is difficult [7].

• When a change is made to a software program, most of the time the entire program is reengineered. This problem can be solved by identifying the 'core' of the system, or parts of the system that tend to remain stable [6].

• Execution effectiveness and artifacts consistency is not repeatable as it depends on development team's skill set and application knowledge the team possesses[13].

• Mapping platform services from source to target environment has always been a manual task as this is difficult to automate[7].

The requirement engineer ( or re-engineer) should have at least a partial understanding of exiting software application, before starting the reengineering process. During the process, the requirement engineer used to acquire the details from available sources to understand the legacy system. Table-1 depicts the tasks and corresponding issues.

The common sources and understanding issues are:
- Studying legacy system documentation
- Viewing existing source code
- Discussing with Legacy system developers

**Table-1**

| Re-engineering task | Re-engineering issues |
|---|---|
| 1. Studying legacy system documentation | Documentation of all types is frequently out of date [14]<br><br>System often have too much of documentation. [14]<br><br>Documentation is poorly written<br><br>Finding useful content in documentation can be so challenge that people might not try to do so.<br><br>Much mandated documentation is so time consuming to create that its cost can outweigh its benefits [14] |
| 2. Viewing existing source code | Difficult to understand the source code [13]<br>Code is more complexity [6]<br>Lot of unused codes, it difficult to estimate which is used and which is not used. |
| 3. Discussing with Legacy system developers | Legacy system developer were no longer available ( resigned or left the organization) [14] |
| 4. Re Design | Legacy system developer is not full knowledge of entire modules. [8] |
| 5. Development Skills | Team member does not having the latest skill set of experiences. |

A major cost factor in the life cycle of software system is program understandings[13] i.e. trying to understand an existing application for the purpose of planning, designing, implementing, and testing changes [1]. This suggests that paying attention to program understanding issues in software process could well pay off in terms of higher quality, longer life time, fewer defects, lower costs and higher job satisfactions[2][4].

In order to contribute these re-engineering issues, we will carefully analyze the role of program understanding in XP with reengineering process. We have referred the six key practices of XP[10]: Pair Programming, Refactoring, Coding standard, testing, and simple design as team activity.

## 2. XP: Core Practices

Extreme Programming (XP) is a set of values, principles and practices for rapidly developing high-quality software that provides the highest value for the customer in the fastest way possible. XP is extreme in the sense that it takes 12 well-known software development "best practices" to their logical extremes - turning them all up to "10" (or "11" for Spinal Tap fans). Kent Beck's has proposed 12 core practices of XP in his book titled Introduction to Extreme Programming Explained [Kent Beck 2002]. They are listed in the following section.

1. **The Planning Game**: Business and development cooperate to produce the maximum business value as rapidly as possible. The planning game happens at various scales, but the basic rules are always the same. Business comes up with a list of desired features for the system. Each feature is written out as a **User Story**, which gives the feature a name, and describes in broad strokes what is required. User stories are typically written on 4x6 cards. Development estimates how much effort each story will take, and how much effort the team can produce in a given time interval (the iteration). Business then decides which stories to implement in what order, as well as when and how often to produce a production releases of the system.
2. **Small Releases**: Start with the smallest useful feature set. Release early and often, adding a few features each time.
3. **System Metaphor**: Each project has an organizing metaphor, which provides an easy to remember naming convention.
4. **Simple Design**: Always use the simplest possible design that gets the job done. The requirements will change tomorrow, so only do what's needed to meet today's requirements.
5. **Continuous Testing**: Before programmers add a feature, they write a test for it. When the suite runs, the job is done. Tests in XP come in two basic flavours.

**a. Unit Tests** are automated tests written by the developers to test functionality as they write it. Each unit test typically tests only a single class, or a small cluster of classes. Unit tests are typically written using a unit testing framework, such as JUnit.

**b. Acceptance Tests** (also known as Functional Tests) are specified by the customer to test that the overall system is functioning as specified. Acceptance tests typically test the entire system, or some large chunk of it. When all the acceptance tests pass for a given user story, that story is considered complete.

6. **Refactoring**: Refactor out any duplicate code generated in a coding session. You can do this with confidence that you didn't break anything because you have the tests.
7. **Pair Programming**: All production code is written by two programmers sitting at one machine. Essentially, all code is reviewed as it is written.
8. **Collective Code Ownership**: No single person "owns" a module. Any developer is expect to be able to work on any part of the codebase at any time.
9. **Continuous Integration**: All changes are integrated into the codebase at least daily. The tests have to run 100% both before and after integration.

10. **40-Hour Work Week**: Programmers go home on time. In crunch mode, up to one week of overtime is allowed. But multiple consecutive weeks of overtime are treated as a sign that something is very wrong with the process.
11. **On-site Customer**: Development team has continuous access to a real live customer, that is, someone who will actually be using the system. For commercial software with lots of customers, a customer proxy (usually the product manager) is used instead.
12. **Coding Standards**: Everyone codes to the same standards. Ideally, you shouldn't be able to tell by looking at it who on the team has touched a specific piece of code.

## 2.1 XP Key: Pair Programming

Pair programming means that *all production code is written with two people looking at one machine, with one key board and one mouse*[10]. It is the pair's job is to write the tests and production code for a given user story. To that end, they discuss design issues, as well as the particular piece of code they are looking at. While developer holding the keyboard is entering code, his partner is conducting an immediate code review, thinking about the overall design, additional test cases, and potential simplifications.

**How the Pair Programming key helps for Re-Engineering project?**

One way in which pair programming affects comprehension is that pair programmers explore a large number of alternatives than a single programmer alone might to. The partners may have different backgrounds and experiences, and different strategies for solving particular sets of problems [2]. Their combined strengths help them when understanding existing code. Moreover, when designing for change they may come up with ideas that neither of them would have found when working individually.

A consequence of this is that pair programming should result in better code. Indeed several studies into effects of pair programming have been conducted. which reports that pair programming results in better code, fewer code defects, better design, better team building. Moreover code written by two programmers together is more readable than code written by one programmer only.

Concerning team building, XP requires that programmers frequently change partner. As a result, system knowledge is shard between the team rather than isolated with some key programmers.

## 2.2 XP key: Refactoring:

The term refactoring specifically refers to a common activity in *programming and software maintenance: changing the structure of a program without changing its semantics* [11]. Often, refactoring precedes a program modification or extension, bringing the program into a form better suited for the modification step[5].

Refactoring [12] are applied when bad smells are deducted in the code, such as duplicate code (which violated one and only rule), feature envy (when a method seems more interested in the features of class other than the method is actually in) or switch statements.

**How Refactoring key helps for Re-Engineering project?**

The impact refactoring plans can have on the development team dealing with large refactorings, such as:
• Developers can recognize changes and by-passes inside the code base that are introduced as part of a large refactoring. Therefore they use the refactoring plan and connections between the plan and the code.
• The risk of getting lost within a large refactoring is reduced by the refactoring plan. Developers can watch the plan while diving down into the refactoring. They can check whether the current activity really provides a benefit for the overall refactoring or not.
• The team can track the progress of the refactoring. This can help to plan the refactoring effort to spend within current and future iterations.

**2.3 XP Key: Evolving a Simple Design**
XP's design philosophy is minimalist and pragmatic. It does not start with a full up-front analysis and design: instead, it only requires a quick analysis of the entire system and then begins working on the first iteration, building the smallest useful system in approximately three weeks. Moreover, XP does not distinguish between analysts, architects and programmers: every developer is responsible for both design and programming. Furthermore, developers evolve a design by continually simplifying the existing code base through refactoring.

**How Evolving a Simple Design helps for Re-Engineering project?**

XP aims at arriving at the simplest design that runs all the acceptance tests. Simplest is defined as follows [10]:

- The code and tests together communicate everything the developer wanted to communicate;
- The system contains no duplicate code;
- It has the fewest possible classes;
- Each class has the fewest possible methods.

Because of this requirement for simplicity, generalizations not needed to get the current acceptance tests running are not implemented. Thus, class structures promising simplifications in forthcoming iterations or features potentially needed at a later stage are not taken into account in the design. XP takes the position that such generalizations are costly to build and to maintain. They will be implemented only when they are needed to eliminate code duplication in the current system. In XP, the design is generally not explicitly documented. Software diagrams tend to get associated with heavyweight processes, in which a lot of time is spent on drawing and maintaining diagrams that are not actually used.

## 2.4 XP key: Unit Testing

Unit testing is at the heart of XP. Unit tests are written by the developers, using the same programming language as used to build the system itself. Tests are small, take a white box view on the code, and include a check on the correctness of the results obtained, comparing actual results with expected ones. Tests are an explicit part of the code, they are put under revision control, and all tests are shared by the development team (any one can invoke any test). A unit test is required to run in almost zero time. This makes it possible (and recommended) to run all tests before and after any change, however minor the change may be.

**How Unit testing key help for Re-Engineering project?**
First, XP's testing policy encourages programmers to explain their code using test cases. Rather than explaining the behaviour of a function using prose in comments or documentation, the extreme programmer adds a test describing that behaviour.

Second, the requirement that all tests must run 100% at all times, ensures that the documentation via unit tests is kept up-to-date. With regular technical documentation and comments, nothing is more difficult than keeping them consistent with the source code. In XP, all tests must pass before and after every change, ensuring that what the developer writing the tests intended to communicate remains valid.

## 3. Re-engineering projects: A Case Study Report:

We got chance to do the re-engineering case study at three different public sector organization. The table 2.0 provides the case study report about their practices as related to XP keys. The organization 'A' is one of leading power sector organization has successfully executed the re-engineering project called as Inspection Call Management System (ICMS). The purpose of this re-engineering is to facilitate primarily the logistics for monitoring inspection activity and to have an online information system regarding status of inspection activity against various contract for different items/ equipment, the web based ICMS, developed and implemented in-house, can be considered as a milestone as well as a exemplary direction for the organization.

The drawback of previous legacy system was not fully automated as per latest technologies. In the past, inspection calls were raised by filling up a prescribed form with some manual operations. This application was developed by VB 5.0 with MS-Access 97 environment. The new work flow process has been implemented using ASP 3.0 Access 2000 database. After implementing this ICMS project, The on-line data helps helps to identifying the requirement of Inspection Engineers for a particular period in particular inspection office, optimizing time and resources by clubbing outstation inspection. Further, it has also helped discipline the logistics of inspection activities. Contractors are now bound to submit correct and all relevant information required for carrying out inspection. The performance of the suppliers during inspection is also analyzed. Consistent good quality supplier are identified from the lot and trend of rejections, item wise or supplier wise, helps in arriving at a judicial decision regarding waiver of inspections. The development team fully aware of their nature of job and system workflow details. All of module source has written with their own coding standard guidelines. Their key success almost match with XP key practices. The organization have a training cell, which provides the training for all team member to gain the latest technologies.

The organization 'B' is one of leading software development company , recently re-engineered their in-house project in to .NET framework. The project is an application namely Resource Request Form (RRF) which is basically developed for associates who are working for the organisation. The project has following features.

- Each request form will have a unique ID, which will be used for tracking and status viewing.
- Automates all functions of the workflow pertaining to hardware or software installation processing.
- Automatic mail generation between users to draw their attention.
- Approving Authorities name, time and date will be endorsed from the terminal.
- When the form is accepted / submitted by higher authorities, the associate will receive reply e-mail with date and time of the approval of the request.
- E-mail notification will be sent at every transition of the flow of the process.

This application is maintained by the team with concept of owner and co-owner. These owners are having full work flow knowledge of current application and they are able to do any change request action at near feature. This organization is have Confident Development Centre, which conducts the training program about software process and their guideline activities.

The organization 'C' had done a re-engineering project with a bit of delay and excess cost. The reasons are

1. The legacy application was built third-party vendor, due to licence agreement they have not supplied the original source code. The actual team members are left the job from this organization.

**Table 2.0 : Re-engineering projects: A Case Study Report**

| Organization | Pair Programming | Refactoring | Evolving Simple Design | Unit Testing |
|---|---|---|---|---|
| A | Partially adopted | Fully adopted | Fully adopted | Fully adopted |
| B | Partially adopted | Fully adopted | Fully adopted | Fully adopted |
| C | Not adopted | Not adopted | Partially adopted | Fully adopted |

2. The reverse engineering process are performed with domain expertise with available executable application.

3. The entire GUI has been re-designed and re-developed.

While we learned many things during this project so far, there are some issues which seem outstanding concerning their relevance to its outcome:

- Selection of an appropriate methodology
- Acknowledgment of the importance of communication
- Ensuring the propagation of the chosen methodology in the team.
- Knowledge sharing among the team.

Especially concerning release deadlines and release management, it became very obvious to us how developing in a large team requires a solid amount of discipline from everyone. Not to the least, this can (and had to) be enforced by putting up conventions that have to be abided by. Thus, one of the premier lessons was to accept the demand for certain degree of formalism when working in the large team towards a release date. This goes to a good amount into the direction of the quality management paradigm of being able to accomplish the same task repeatedly with the same quality. Within such this environment, key practices like pair programming, Refactoring could be applied well and efficiently [5][10][12].

## 4. Conclusion:

In this paper, we have seen that XP practices of Pair programming, Refactoring, Unit testing and Simple Design. Testing can help to get code that is easier to understand, testing can be used to document program understanding. Refactoring can act as a restructuring of existing code, removal code clone, etc. Pair programming helps the team can track the progress of the refactoring. This can help to plan the refactoring effort to spend within current and future iterations. Thus we believe implementing the XP key practices into existing application gives the solution for re-engineering application issues.

## References

[1] Alistar Cockburn, "Agile Software Development", Published by Pearson Education 2002, ISBN 81-7808-768-5.

[2] Arie Van Dersen, "Program comprehension Risks and Opportunities in Extreme Programming", Proceedings of the 8th Working Conference on Reverse Engineering 2002, pp 176-182.

[3] Chikofsky, Elliot J. & James H. Cross II, "Reverse Engineering & Design Recovery: a Taxonomy", *IEEE Software*, Jan1990, pp. 13-17,

[4] Gerald Ebner,Hermann Kaindl , "Tracing All Around in Reengineering ",IEEE Software, May/June 2002, pp 70-77.

[5] Jim Shore, "Continuous Design", IEEE Software Jan 2004, pp 20-22.

[6] John Bergey, Northrop Linda, Dennis Smith, "Enterprise Framework for the Disciplined Evolution of Legacy Systems" Software Engineering Institute, CarnegieMellon University 1997. Technical Report: CMU/SEI-97-TR-007

[7] John Bergey, Dennis Smith, Nelson Weiderman, "DoD Legacy System Migration Guidelines", (CMU/SEI-99-TN-013, ADA370621). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 1999.

[8] John Bergey; Dennis Smith, Scott Tilley, Weiderman, Nelson; Woods, Steven. "Why Reengineering Projects Fail", Software Engineering Institute, CarnegieMellon University, 1999. Technical Report (CMU/SEI-99-TR-010).

[9] K.Gowthaman, K.Mustafa, Raees A Khan , "Design recovery issues and opportunities in Extreme Programming", Developer IQ, Techmedia, Feb 2005, Vol. 5, No. 2, pp. 72-76.

[10] Kent Beck, "extreme Progamming explained, Embrace Change", Published by Pearson Education 2002, ISBN 81-7808-667-0.

[11] Martin Fowler, “Refactoring: Improving the Design of Existing Code”, Addision Wesley Longman, 1999, ISBN 0201485672

[12] Martin Fowler, “Separating User Interface Code”, IEEE Software Apr 2001, pp 96-97.

[13] Tilley, Scott R.; Smith, Dennis B “Towards a Framework for Program Understanding” Proceedings of the 4th Workshop on Program Comprehension (WPC), March 29-31, 1996, Berlin, Germany, pp. 19-28.

[14] Timothy C. Lethbridge, Janice Singer, Andrew Forward, “How Software Engineers Use Documentation: The State of the Practice”, IEEE Software, Dec 2003, pp 35-39.