Extreme Programming for Distributed Legacy System Reengineering

Bin XU
College of Computer Science & Information Engineering,
Zhejiang Gongshang University, 310035 Hangzhou, P. R. China
binxu@ieee.org

Abstract

Reverse engineering is an imperfect process when comprehending a legacy system with large volume of source code and complicated business rules. It is important for the adopted software process to shorten the time to market and minimize the risks especially in distributed environment. In this paper, Extreme Programming (XP) was evaluated in a distributed legacy system reengineering project to handle the imperfect system requirement and response to rapid business request combination while the customer was offshore. Some important adjustment was made to the XP process according to the project environment. The reengineering tasks of large scale were divided into several subtasks through evolving reengineering. XP made these tasks comparatively independent, reduced the workload of analysis in reverse engineering, and improved the performance of analysis. Localized analysis made testing and tracing easier, so the complexity of reengineering project was reduced. Evolving reengineering helped us to conduct and fulfill reverse engineering and forward engineering in parallel and shorten project lifecycle. XP enabled us to deliver better quality code in a shorter period of time with low cost.

Keywords: Reengineering, Extreme Programming, reverse engineering, forward engineering, evolving reengineering

1. Introduction

While legacy systems [1] still conduct the vital business in enterprises, the decades' degradation results in several issues when the enterprises want to expand the service scope and scalability of these system:

- The business rules and methods inside the legacy system are complex, and they are difficult to be comprehended and modified.
- The systems depend on some outdated technologies. As a result, it is difficult to comprehend the source code or to add new functionality into the system.
- The hardware platform and third-party libraries used in the system are out of date. It is difficult and very costly to get continual support.

The cost, schedule and risks are the main factors that determine whether to maintain or to redesign a legacy system or just to purchase a different one [3, 4]. Continuous maintenance induces high cost annually, and the rising cost is unacceptable. If a legacy system is replaced with another system, not only its customers would have difficulties to adapt it, but also high investment risk would be involved [2].

Reengineering may be a good choice to expand the service scope and scalability of a legacy system. However, there are some risks in matching the business value and scheduling the tasks in a reengineering project [8]. It is a complicated and long process to comprehend a legacy system with large volume of source code and complicated business rules. Reverse engineering is an imperfect process driven by imperfect knowledge in such projects [7]. Since the system requirements are generated from the reverse engineering of original legacy system and combined with current business request, new system requirements for the target system is imperfect before the entire reverse engineering is finished. In distributed environment, there is more obstacle in communication and coordination with the knowledge, requirement and asserts transfer [9].

Extreme Programming (XP) is famous for its short release, incremental planning, evolutionary design and its ability to response to changing business needs. XP values communication, simplicity, feedback, and courage [5]. Recently, XP has been widely used in software development, such as maintenance environment [6] to deal with the frequent business requirement changes. In this paper, XP is evaluated in a distributed legacy system reengineering project to handle the imperfect system requirement and response to rapid business request combination while the customer is offshore. Lattice®

Trading System reengineering project is identified for this analysis.

2. XP Organization for Lattice System Reengineering

Three parties in the world were involved in Lattice system reengineering project:

- State Street Boston (SSB, located in Boston, USA)
- State Street Hong Kong (SSHK, located in Hong Kong, China)
- State Street Zhejiang University Technology Centre (SSZUTC, located in Hangzhou, China)

SSZUTC was in charge of concrete reengineering implementation including reverse engineering and forward engineering. The securities trading business department and the securities trading IT department at SSB were responsible for Lattice system reengineering project. The business departments of SSB raised the business requirements and provided consultation to SSZUTC. The IT departments of SSB and SSHK validated the outcomes from SSZUTC.

2.1 Roles and responsibilities

Generally, in XP projects, there are roles of programmer, customer, tracker, consultant, tester, coach and big boss. In Lattice project, the developers in SSZUTC served as the role of programmer, the business departments of SSB was the customer, and as it was far away from the site of developing—SSZUTC, it was called remote customer. The IT department in SSHK was the tracker while IT department in SSB was the consultant. Both of these two departments also worked together to run the acceptance test as the tester. The project management in SSZUTC acted as coach and several senior managers of State Street Corporation acted as the role of big boss.

Considering the different features between reverse engineering and forward engineering, the development team in Lattice system reengineering project was divided into two groups: Reverse engineering group and Forward engineering group. Reverse engineering group comprehended the source code and asked the remote customer for business requirement confirmation when necessary.

In order to shorten the time in software validation as well as to improve the software quality, a develop QA team which was composed of personnel of reverse engineering group was organized to perform integration test, functional test, and system test in SSZUTC.

2.2 On-site customer issue

On-site customer is the key of leading XP project to success [5]. However, it is too expensive to keep one group of specialists from SSB (U.S.) working together with the development team in SSZUTC (China) throughout the whole project duration; therefore the “On-site customer” practice was abandoned. But it was impossible to comprehend and master the complex business rules in Lattice system only from the source code. In fact, the analysis result of business rules worked out by reverse engineering group should get the confirmation from the remote customer. A hybrid role named “remote customer backed reverse engineering group” (CREG) was created to realize the onsite consultation [Figure 1].

In this project, reverse engineering group analyzed the legacy system and delivered the business logical analysis result to the remote customer in period. When all the parts of a component were analyzed and had gotten the confirmation, reverse engineering group would generate it into a story. When forward engineering group were available, reverse engineering group would allocate the stories to them. During the implementation of a story, forward engineering group might consult reverse engineering group. When the content of consultation exceeded its current knowledge, reverse engineering group would expand the scope of analysis and consulted the remote customer as needed.

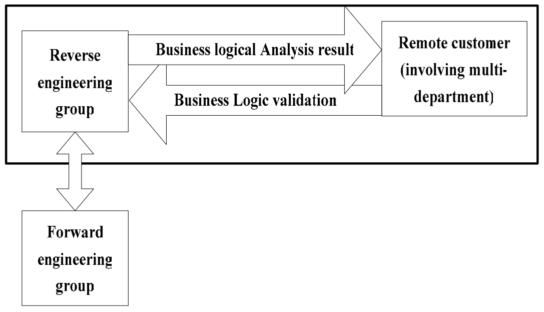
Reverse engineering group
Business logical Analysis result
Business Logic validation
Remote customer (involving multi-department)
Forward engineering group

Figure 1. Remote Customer Backed Reverse Engineering Group(CREG)

In this way, the remote customer and reverse engineering group served together as customer in XP parlance, and forward engineering group served as the original programmer in this project. The develop QA team not only fulfilled the original task as the tester, but also ran some functional tests and system tests.

3. XP Procedures for Lattice System Reengineering

There are 12 effectual rules in XP. Though most of the practices could be adopted in this project, some adjustments were needed in the light of the special features of this reengineering project [Table 1].

The refactoring was conducted when pair programming or peer review. The main goal of refactoring in Lattice project was to benefit the future improvement. For example, if develop groups felt a piece of code was not readable, they would try to clarify its logic or simplify its design as well. Sometimes, they also made several duplicated pieces of code into a relative component, which might reduce the cost in later modification.

Simple design is not only one of the important practices in XP, but also may help the new system to be maintained more easily in the future. Since one goal of Lattice project was to make the new system flexible and adaptable, every member that was involved in Lattice reengineering project focused on simple design all the time. In Lattice system reengineering, besides refactoring which is introduced above, the CREG model and pair programming also impelled simple design. When communicating the design with the remote customer, the develop groups must make sure the customer will understand all the design completely, so the final design will be surely communicative. When developing in pairs, the programmers did not write any code beyond the test scope, so that the implementation was consistent with the test case.

Because the customer of Lattice project was offshore, there was a certain interval when they were communicating with the offshore develop team. Assuming the functional tests were only conducted by the customer, the developers might face the trouble in apprehending the possible bugs found by QA. In order to obtain rapid feedback and reduce the cost in communication, some basic functional tests were also conducted by develop groups. Locally functional test also enabled the brainstorming which was important in bug fixing.

The collective ownership practice was adopted partly. There were two kinds of ownership in this project: direct ownership and collective ownership. The initial authors were the direct owners and all code was collectively owned. To reduce the integration time, the direct owners had the responsibility for refactoring and bug fixing. The collective ownership worked only if the direct owners were unavailable.

3.1 The main process

The whole project was carried out in a defined process [Figure 2]. The process comprised of 5 steps: architecting, planning game, iterations, release delivery, and acceptance test.

At first, reverse engineering group conducted the spikes on architecture (architectural spike) and built the System Metaphor with its result.

Then a planning game was conducted to define the release plans of the whole project (planning game ran later only when necessary).

After that, the tasks were generated from System Metaphor story by story according to the present release plan and the interdependent relations of the tasks through evolving iterations. Not only unit tests but also functional tests were run on the new system to find out the bugs as early as possible. The performance test, system test and stress test of the whole system and the key parts were also conducted to ensure the recovery and reliability of software delivery.

Finally, the acceptance test was conducted by customer QA team on the delivered releases.

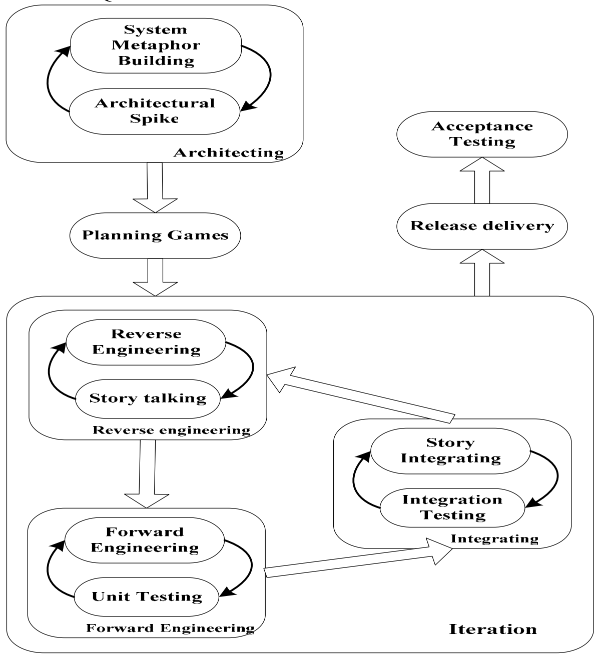

Figure 2. Main Process of Lattice Project

3.2 Architectural Spike and System Metaphor

The system architecture is important for the code comprehension. Before conducting the reengineering iteratively, it was essential to clarify the top level comprehension of the legacy system including the functionalities, interfaces of the sub-modules and the dependencies between them.

Some reverse engineering tools are useful when collecting such information and demonstrating it into several views in system metaphor. These views in system metaphor not only benefit the architecture compliant but also facilitate the communicating.

Table 1. XP Practices for Lattice System Reengineering

XP practices	Adoption status	Comments
System Metaphor	Fully	The System Metaphor was built on the architectural graph, and might be updated after story talking or when a story was finished. System Metaphor had a very important position as reverse engineering group picked up the tasks from it and it would provide a quick overview of the whole project.
Planning game	Fully	Planning game ran when RC changed the plan or if the manager thought it was needed. It was accompanied by weekly project meetings which summarized a week's work and planed for the next week. Develop groups, QA teams, the project manager, and several delegates of the customer participated in the project meeting.
Small release	Fully	The first release was delivered in one month. The subsequent releases were delivered every two weeks.
Simple design	Fully	Develop groups conducted the readable and communicative design and didn't design anything outside of the test case scope. Pair programming, CREG model and refactoring brought the chance to do simple design.
Testing	Enhanced	Basically unit tests were conducted in XP project while the functional tests were run by the customer. In order to achieve high quality products, some basic functional tests were also conducted by the developers to remove the defects early.
Refactoring	Fully	Refactoring was done during development and bug fixing. Develop groups refactored not only the production code but also unit test and functional test code.
Pair programming	Enhanced	Brainstorming was needed in planning, designing and debugging. Expanding pair-programming to team work was necessary when doing these tasks.
Collective ownership	Partly	There were two kinds of ownership in this project: direct ownership and collective ownership. The initial authors were the direct owners and all code was collectively owned. To reduce the integration time, the direct owners had the responsibility for refactoring and bug fixing. The collective ownership worked only if the direct owners were unavailable.
Continuous integration	Fully	The integration was conducted story by story.
40-hours week	Mainly	Recorded by the time sheet, the average work hours of the involved members was about 42 hours a week.
On-site customer	Not adopted	This was not adopted because of the high costs. The CREG model was conducted instead.
Coding standards	Fully	It was adopted throughout the project, and benefited communication very much. Develop groups and QA teams enjoyed the effective code/design sharing among the geographically distributed sites: Boston, Hong Kong, and HangZhou.

3.3 Evolving Iterations

Traditional reengineering usually views reverse-engineering and forward-engineering as two indivisible processes, and this brings difficulty to cost evaluation, schedule control and quality improvement in the reengineering project. In Lattice project, reverse engineering and forward engineering were conducted iteratively, therefore the whole project evolved in the progress of story.

According to the release plan and the interdependency of the tasks, reverse engineering group picked up the components from the System Metaphor and conducted the reverse engineering with clear aim. When the requirement of a component was clarified and confirmed by the remote customer, the requirement was provided to forward engineering group as a story through story talking which helped forward engineering group understand the requirement completely. Forward engineering group evaluated the specific subtasks and started to implement the story meanwhile reverse engineering group began to design and implement the functional test of the task. In Lattice project, the functional test was assigned as a special task of a story to enhance the testing. When the story passed the automated unit test, forward engineering group cooperated with the develop QA team to run the corresponding functional test. After a story passed the functional test, it would then be integrated to the target system by the develop QA team. This procedure was iterated as a cycle.

4. RESULTS

In the first phase of Lattice project, a representative and critical module named HostTicker was chosen which had 77.5KLOC source code and depended on a third party library named objectspace®. Six analysts, who formed the reverse engineering group, were assigned to implement architectural spike and acquired the System Metaphor. Then, four more developers, who formed the forward engineering group, joined the project. Reverse engineering group generated 22 stories with the basis of the System Metaphor and allocated these stories to forward engineering group. The reverse engineering and forward engineering were conducted in all these stories iteratively. Develop groups worked out 50.2KLOC code in 7 weeks. Finally, they successfully got the executable application with the performance twice as that of the original one. The related project management data was collected and compared with other reengineering projects which were conducted in SSZUTC [Table 2].

Table 2. Comparison of this project with other reengineering projects in SSZUTC

Process feature	This project (XP concept was used)	Average of the other projects
Productivity	143 LOC/per day	100 LOC/per day
Bug rate after delivery	0.04 / KLOC	0.3 / KLOC
Whole Staff-weeks	70 weeks	100 weeks
Project Duration	7 weeks	More than 11 weeks

With the adoption of XP, there were significant improvements in terms of productivity, quality, and time to market.

Traditionally this project could take around 8+3 weeks but the authours were able to complete in 7 weeks

4.1 Shorter duration of the project

According to the plan, the project should be finished in 8 weeks. As there was some dependency among the tasks, it will be postponed at least 3 weeks if the reverse engineering and forward engineering were run sequentially through the traditional mode. Multi reverse engineering tasks could be finished at the same time, and the forward engineering tasks were generated accordingly. In such way, the stories could be performed in parallel, which enabled us to employ more programmers into the project than traditional methodology. Four more programmers were added for the forward engineering and develop groups were able to finish the project in only 7 weeks, 1 week ahead of the deadline.

4.2 High quality of the source code

With the successful refactoring and simple design, the size of the source code dropped from 77.5KLOC to 50.2KLOC for approximately 1/3 and the throughput turned out to be doubled. In the whole process of the pilot project, the develop QA team found 7 bugs in total and the customer QA team found 2 bugs after the delivery. All these bugs were located very quickly and 7 of them were fixed in 30 minutes for each while a memory leakage bug took 2 hours to fix. The one left was a crash in the third party library, which was fixed in the patch given by the provider 4 days later. Develop groups removed almost 77% of the defects with the enhancement of testing and enjoyed the quick bug fixing with the simple design and team work. This project has notable low bug rate which is only 13% of that of the other project.

4.3 High productivity

The productivity of this project is 143LOC/per day, 1.5 times as other projects we had conducted in SSZUTC before. Although there was 100 weeks as planned, the project was successfully finished with only 70 weeks. In the project, we did not find the pair could code in double speed as the individual. However, XP practices ensured the quality of the code, and saved much time in testing and bug fixing in this project.

4.4 Effective collaboration

We didn't record the communication between forward engineering group and reverse engineering group due to its large volume, but we found that the communication between reverse engineering group and the remote customer was efficient and regular. Normally reverse engineering group got responses the next morning; occasionally they got the responses in 3 days or longer when the questions involved many departments of the customer. CREG model reduced the communication delay between SSB and SSZUTC to a tolerable level. It is proved that the CREG model can be run efficiently in the whole pilot project.

SSB informed us to add a new function into the system after the first release had been delivered. With the cooperation of the two develop groups, a new story was created and implemented, finally the new function was added into the system successfully. The new system and the XP practice proved to be flexible and adaptable.

5. Conclusions

Reverse engineering is an imperfect process when comprehending a legacy system with large volume of source code and complicated business rules. It is important for the adopted software process to shorten the time to market and minimize the risks especially in distributed environment. In this paper, Extreme Programming (XP) was adjusted and evaluated in Lattice Trading system reengineering project to handle the imperfect system requirement and response to rapid business request combination while the customer was offshore. The large scale reengineering tasks were divided into small scale subtasks which are comparatively independent through evolving reengineering. In such way, the reengineering goals were made clear, the analysis workload in reverse engineering was reduced, and the performance of analysis was improved. Localized analysis results made testing and tracing easier so as to simplify the reengineering project. Evolving reengineering enabled reverse engineering, forward engineering and functional test to be conducted and fulfilled in parallel. As a result, project lifecycle was shortened. The suggested CREG model can avoid the high cost that the on-site customer may bring, and achieve the effect of customer in XP practice as well.

Compared with traditional reengineering, evolving reengineering made the process of the entire project more visible, helped control the progress and cost better. More importantly, it enabled us to deliver better quality code in shorter time-to-market.

This research demonstrated that partial adoption of XP can also achieve marked effect. The key to successful application of XP is the coordination of practices and flexibility.

Acknowledgement

This research was sponsored by State Street Corporation. Lattice is a registered trademark of State Street Corporation. All other company and product names are trademarks or registered trademarks of their respective owners.

The contents of this paper are the opinions and conclusions of the author only and do not necessarily represent the position of State Street Corporation or its subsidiaries, affiliates, officers, directors or employees.

References

[1] M.M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", *Proc. IEEE*, Vol. 68, No. 9, 1980, pp.1060-1076.
[2] K. Bennett, "Legacy Systems: Coping with Success", *IEEE Software*, Vol. 12, No. 1, 1995, pp. 19-23.
[3] N. F. Schneidewind and C. Ebert, "Preserve or Redesign Legacy Systems?", *IEEE Software*, Vol. 15, No. 4, pp. 14-17.
[4] H.M. Sneed, "Planning the Reengineering of Legacy Systems", *IEEE Software*, Vol. 12, No. 1, 1995, pp. 24-34.
[5] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, Mass, 1999.
[6] C. Poole and J. W. Huisman, "Using Extreme Programming in a Maintenance Environment", *IEEE Software*, Vol. 18, No. 6, 2001, pp. 42-50
[7] J.H. Jahnke and A. Walenstein, "Reverse Engineering Tools as Media for Imperfect Knowledge", *Proceedings of the Seventh Working Conference on Reverse Engineering*, *IEEE*, 2000, pp.22-31
[8] H.M. Sneed, "Risks involved in Reengineering Projects", *Working Conference on Reverse Engineering*, 1999, p.204-211.
[9] J.D. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally Distributed Software Development", *IEEE Transactions on Software Engineering*, Vol. 29, No. 6, 2003, pp. 481-492