Extensions to Extreme Programming and its Usage in Education

Ashwin Tumma¹, Nikhil Sharma² and Harshada Kshirsagar³

ABSTRACT

Delivering the most value to the business, efficient usage of resources, faster development, better quality solutions, more enduring systems and higher productivity are some of the needs of today's software development businesses. Agile Software Development best suffices all these purposes with showing the ability to respond to change. Extreme Programming is one such instantiation of agile process. Since its inception in 1999, it has evolved to a very large extent.

In this paper, we discuss the study of evolution of Extreme Programming and propose an update to Extreme Programming which leverages the benefits of cleanroom software engineering and component-based development. User stories modeled as box structures of cleanroom software process are explained. Reuse of existing software components in conjunction with Extreme Programming and cleanroom software process is proposed. The paper also discusses the implications of the proposed update of Extreme Programming methodology in research works and education.

INTRODUCTION

In his seminal book Software Engineering—A Practitioner's Approach¹, Roger Pressman defines agile methods as “the philosophy that encourages customer satisfaction, early incremental delivery of software, small but highly motivated project teams, informal methods, minimal software engineering work products and overall development simplicity”. In agile approach to software engineering, all the software engineers and the project stakeholders work hand-in-hand with each other. The agile team is the one that decides its own way of operation and project track. Communication amongst all the members involved in the project, is an important factor that holds the pinnacle in agile approaches.

¹ Department of Computer Engineering, College of Engineering, Pune, Shivajinagar, Pune.
² Department of Computer Engineering, College of Engineering, Pune, Shivajinagar, Pune.
³ Department of Computer Engineering, Bharati Vidyapeeth’s College of Engineering for Women, Pune, Katraj, Pune.
The remainder of this paper is organized as follows: Section on Agile Approach gives a brief description of agile approach towards development of software engineering projects. Section on Evolution of XP presents the evolution of XP and its values and principles, Section on Cleanroom Software Engineering provides a brief explanation of the cleanroom software engineering approach. Section on Component Based Software Engineering presents the intricacies of component based software development. Section on proposal for update describes our proposals of update in XP and how agile methodologies can be coupled with cleanroom software engineering and component based software engineering terminologies. Section on usage explains how our proposed method of XP can be applied in research and education. Section on conclusions summarises the conclusions of the paper.

**The Agile Approach**

Today, the customer of software industry is becoming more and more smarter. He now understands the potential that lies in software projects and their intricacies, and thus keeps on changing the requirements very often as the project proceeds. The change in these requirements are very difficult to predict beforehand, nonetheless, they must be accommodated in the project. Also, due to increase in outsourcing and offshore deployments the market conditions change very rapidly, new threats can pop up at any time without any prior knowledge. This severes the necessity of opting a project model that manages these issues very well. They key requirement is that the project team must be highly adaptable to the then project and technical circumstances. An agile process model best accomplishes these issues. It says that the process model should adapt incrementally and have support for a constant customer feedback. A high customer feedback assists in managing changes because the team can learn the most appropriate adaptions. It emphasizes that the software process model be a continuous and an iterative one. The components of the software need be built and delivered early to the customer and seek his feedback on the same. This approach has umpteen advantages in most of the business projects carried out in the software industry.

The implementation of the agile principles is done using certain process models. Some of the most popular process models are as follows: Extreme Programming, Scrum, Agile Modelling, Feature Driven Development, Adaptive Software Development, Dynamic Systems Development Method. These agile models conform to the Manifesto for Agile Software Development.

XP is one such popular agile model. XP is a methodology that allows the software teams to develop a software in very dynamic environments. As Kent Beck says in his book, *Extreme Programming Explained: Embracing Change*, XP is beneficial to both the software team as well as to the customers and managers. From the programmers perspective, XP guarantees them work on the most appropriate issue and assists in taking the most apt decisions. As far as the customers and managers are concerned, it assures that they get the most out of every programming week. It also allows them to see significant progress in lesser time and enables them to change the requirements of the project in the midst of its development without much costs.

**EXTREME PROGRAMMING AND ITS EVOLUTION**

The ideas and intuitions associated with extreme programming started coming into light in the late 1980s, but were formally documented, presented and published by Kent Beck in 1999. It houses is set of methodologies are fitted in the context of the four framework activities: planning, design, coding and testing. During the time of its inception, XP was based on four main values viz. Communication, Simplicity, Courage and Feedback. It also followed twelve practices which are: On-site Customer, Metaphor, Small Releases, Planning Game, Pair Programming, Collective Code Ownership, Testing, Refactoring, Simple
Design, Continuous Integration, Coding Standards, 40-Hour Week. To get best of it, the software teams that would choose XP as their development model were emphasized to employ all the twelve practices in their work. But in due course of time, it was noticed that it is practically difficult to use all the practices since some practices would not fit properly in different development environments. To have a solution to this difficulty, it was proposed that, the software teams could use only those practices that can be best adapted by their respective software development environments and needs.

Later, Kent Beck and Cynthia Andres, published the second edition of the book Extreme Programming Explained: Embracing Change. There were noteworthy updates to XP in this new edition. An additional value named, Respect, was proposed. The original twelve practices were dissolved to form thirteen primary practices which are: Sit Together, Whole Team, Informative Workspace, Energized Work, Pair Programming, Stories, Weekly Cycle, Quarterly Cycle, Slack, Ten-Minute Build, Continuous Integration, Test-First Programming, and Incremental Design. In addition to these, eleven corollary practices were also added which are: Real Customer Involvement, Incremental Deployment, Team Continuity, Shrinking Teams, Root-Cause Analysis, Shared Code, Code and Tests, Single Code Base, Daily Deployment, Negotiated Scope Contract, Pay-Per-Use. Beck lucidly outlined the principles of the methodology that should serve as guidelines when translating values into practices. Those fourteen principles are: Humanity, Economics, Mutual Benefit, Self Similarity, Improvement, Diversity, Reflection, Flow, Opportunity, Redundancy, Failure, Quality, Baby Steps, Accepted Responsibility.

These values and principles have been of great aid to large number of projects in the software industry. The primary reason of its wide acceptance can be summarized in the following words: XP approach best appeals to all the stakeholders and the software team members in business and enterprise projects.

Over the years, it is seen that XP is also successful in emphasizing the importance of people factor. The most appropriate assessment of the user’s requirement is done by user stories. These help in clearly identifying the customer’s requirements and their apt development. When the first release of the project is done, the team computes the project velocity, which is the number of customer stories implemented in that release. It is a provides an effective measure for estimation of delivery dates and further releases. XP focuses on keeping the design as simple as possible. It takes care of any difficult phases of the projects. If a difficult design is encountered then XP suggests the building of a spike solution which is an operational prototype of the design. This prototype is then implemented and tested in accordance with the user stories. Refactoring is a construction technique proposed by Martin Fowler that does not change the external behaviour of the code but still improves the internal structure.

In the coding framework activity, initially, a set of unit tests are created for the user stories and then coding begins on them. XP recommends the use of pair programming. In pair programming, two people work together on a single user story on a single computer. This approaches proves useful so that the programmers are confined to the user story, and new ideas pop up quickly which can be immediately implemented. Working on a single computer, one programmer could pay attention to the coding details while the other takes care of the coding standards that are necessary from the XP point of view. Unit tests form another distinguishing feature of XP which are specified by customer and focus on the features and functionalities of the increment.

Many books and journals are published that talk in depth about XP. Kent Beck has written two editions of Extreme Programming Explained: Embracing Change which give a thorough explanation of Extreme Programming. The contribution of Martin Fowler is remarkable. His books and articles have made a mammoth contribution to XP and have helped people understanding XP. Kent Beck and Martin
Fowler have explained the intricacies of extreme programming practices in their book, *Planning Extreme Programming*. Martin Fowler’s website is a good startup point for understanding agile concepts.

**CLEANROOM SOFTWARE ENGINEERING**

Cleanroom software engineering is an approach to software development that emphasizes to build a correct software without any defects at the time when it is being developed itself. The traditional approach to software development follows analysis, design, coding, testing and debugging cycles. But an alternative to the same is available, named, Cleanroom software engineering. The key principle of this approach is the eliminate the cost-intensive defect removal process. This is done by writing the code in such a way that it is free from defects and verify their correctness before testing. These correct codes are then integrated in the main system. A strong emphasis is on verify the mathematical correctness of the code. It follows a cost-effective, time-effective and effort-effective approach that takes into account the defects that may arise in the software. Instead of building the software first and then taking more efforts to remove the defects, it is better to manufacture the software in a manner that is defect-free. Thus, reduction in the costs, time and efforts results in a more quality and productive work and also lessens the failure rates of the software. This significant improvement would be difficult or impractical to achieve if the traditional models are used.

Albeit, considering its mammoth benefits, cleanroom software engineering has not experienced widespread usage because, most people believe that it is too mathematical and theoretical to be used, Unit Testing is replaced by correctness verification and statistical quality control that are some what off the track from the traditional software that are developed today, and the software industry is has not been that mature to make flawless and efficient usage of the cleanroom software engineering process. But still, its proponents argue that its potential benefits far outweigh the issues that are caused if it were not used.

**The Strategic Details**

Cleanroom software engineering process follows the terminology of developing functional increments of the software system. The software team develops successive increments of the software and those are then delivered. In *Software Engineering—A Practitioner’s Approach*, Roger Pressman highlights the tasks that occur in the cleanroom software engineering process. They are explained further.

As mentioned above, an incremental approach to software development is followed, *Increment Planning* creates different schedules and other details required for that particular increment. Integration issues are also addressed in this task. *Requirements gathering* task gets elaborate descriptions of the customer’s requirements. *Box structure specification* is the core aspect of cleanroom software engineering that we will focus on. Box structures (will be explained in next section) facilitate the isolation of different behavioural, data and procedural aspects of the system. These box structures are representatives of the *Formal design* of the system. The design and code of the module are followed by its *Correctness verification* wherein, the team carries out a certain correctness verification activities on the design and the code. Following this the task of *Code generation, inspection and verification* are carried in which the increment is undergone a statistical quality control (explained in further sections).

The most important and noteworthy difference for cleanroom software engineering process and other traditional process models is that, in cleanroom software engineering, the unit testing and debugging tasks are absent. Instead they are replaced by, correctness verification and statistically based testing. These differences are the ones that make it more cost-effective, time-effective and effort-effective.
Box Structures

Cleanroom software engineering approach mainly focuses on the implementation details of the software and makes use of box structure specification. A box is a representative of the system or any increment at some level of detail. As and when we start elaborating the box, they are refined into a hierarchy, without depending on the implementation of any other box. Three types of boxes are used: black box, state box, clear box, their explanation follows.

The black box represents the behaviour of the system or part of it. It stands at the highest level of abstraction of the system. State Box holds the state data and services. It also contains the state transitions of the system. Clear box forms the finest level of refinement. The clear box consists the procedural implications for the state box. A rigorous correctness verification of each clear box is done before the software is released. Most of the proponents of cleanroom software engineering model say that it produces more better code than unit testing because, unit testing checks the consequences of executing only specific paths, on the other hand cleanroom approach verifies every possible effect on all data. Also, unit testing is a time consuming process since it takes time for preparation, execution and checking the system. Verification, on the contrary, can be completed in lesser time.

Correctness Verification and Statistical Quality Control

Cleanroom software engineering differs from traditional software engineering process models in testing approach. The cleanroom software testing teams calculate the usage probability distribution of the software. The black box for each increment is analysed for this propose. The team records the stimuli of the user to the software system, then determines and assigns a probability of use of each stimuli. The use of probability and statistics in this approach strengthens the success rate of the system, thereby improving the acceptance rate of the software system. Next, the certification teams certify the components of the system for reliability. A reliability measure of each component is specified in terms of mean-time-to-failure (MTTF).

To encapsulate, the cleanroom approach to software projects results in an outstanding quality of software. Usage of box structures clearly presents the details of analysis and design modelling. Different levels of abstraction can be easily identified and worked upon. Verification is employed for software testing, which is far superior than the traditional unit testing approach. Statistical use testing assists in reduction of failure rate and increases the reliability and acceptance rate of the software.

COMPONENT-BASED SOFTWARE ENGINEERING

As explained in previous sections, today, the software projects are such that they need to be built in a short duration of time without compromise in their performance and quality. Inorder to meet the customer needs and demands the software team tries to complete the projects in a hurry which might cause to deliver a low quality software or a software with defects. Also, there are many instances wherein the genre of the project to be built is similar to that of the previously developed system. Certain modules and designs can be imported from the already-available components, in short they can be reused. Component based software Engineering (CBSE), is one such process model that emphasizes reuse of existing components. It guides the software team to design and construct the software from existing reusable software components and off-the-shelf components.

Large software can be built in shorter time and easily by making use of existing components. A component is an independent part of a system that can be replaced by any other similar object and that accomplishes the desired purpose in the context. There are many projects in the software industry that are
carried out. Many a times it happens that certain standardized components are available to fit a specific architectural design. These components can be reused in the current projects thereby achieving a significant reduction in development time and costs.

Albeit, the components are available, they should not be integrated in the new software in a haphazard manner or merely for the purpose of adding or as an escape from developing a new similar component. When a component shows a potential for its inclusion in the system’s architecture, it should be qualified to meet the needs in that particular context. Later, the component must be aptly adapted to its new environment and must undergo a thorough testing. In their article on Engineering of Component-Based Systems [7] Brown, A. W., and K. C. Wallnau, have highlighted some the factors that should be taken into account whilst component qualification: application programming interface (API), development and integration tools required by the component, run-time environments, resource usage, timing or speed, network protocol, service requirements, security features, embedded design assumptions, exception handling. After the component is qualified to fit in the software system, it needs to adapt to the current environment and then is to be integrated in the architecture. While integration it is necessary to make sure that the integrated component works in the desired manner and all other functions of the system are comfortable and consistent with its presence.

PROPOSAL FOR EXTENSIONS TO EXTREME PROGRAMMING

With reference to Section on evolution of XP, despite using pair programming and unit tests of XP, it might happen that, in the endeavour of delivering early incremental releases of the software, there might be serious bugs and defects in the software. These defects might go undetected by both the (pair) programmers. These defects need to be corrected in the debugging phase which incurs more cost, time and effort. So, this could result in slowing down the pace of developing the software increment because of the additional debugging task involved. Also, the software needs to be tested for integration issues. Thus, here we identify a need to fasten the development work of a software project. Cleanroom software engineering emphasizes the correct building of the software at the time of its development itself. So, while building the software, or in other words, whilst developing test cases and coding (pair programming) the software increment, if the terminologies of cleanroom software engineering are applied to the traditional XP approaches, then the software team and other stakeholders could reap maximum benefits of using the XP methodology for software project development.

In XP, the requirements of the customers are available in the form of user stories. The software teams works to complete the designing and implementation of those user stories. Applying Cleanroom approach, we propose that if each user story is modeled to a separate box structure of cleanroom software process, then each user story will enjoy the advantages of cleanroom approach. A user story can be represented as a black box which will represent the functional behaviour of the user story. Many a times, it might happen that the user stories are told vaguely, but black box structure would simplify its understanding and other intricacies. Further, the stories can be refined into respective state boxes that will represent the corresponding state transitions of that function. Clear boxes will then be brought into picture for the final implementation of the user story. Following this, the user story design and implement will undergo rigorous verification by the cleanroom software team. A statistical use testing is employed on the developed implementation of the user story. As mentioned in Section on Cleanroom Software Engineering, the advantages of verification outweigh those of unit testing. Once certified for reliability the increment can be delivered. Since, the increment has undergone a statistical quality control its defect rate is reduced tremendously and thus it causes a significant increase in the acceptance rate of the increment.
As explained in Section on Component Based Software Engineering, the most software projects can use reusable software components inorder to cut the expenses on time, costs and efforts. This terminology can be “reused” in XP. As explained in previous paragraph the user story implementation is accomplished with by the use of box structures. These can be accompanied by the reusable software components. If a user story demands implementation of component which is already existing, that component should be reused in the current project. This will enable the software team and other stake holders of the project to get the most of XP in a short duration of time. Reuse of off-the-shelf and reusable components reduces the time for development of the increment, and thus the software team can pay attention to other details of the increment and achieve an early delivery of the increment.

One more inherent advantage of using cleanroom software process in conjunction to component based development are that the reusable components need to undergo a verification and statistical quality control review only once, later on, they can be used seamlessly in the software systems. Though, we say they can be used seamlessly, an overhead of component qualification, adaption and integration are always associated with component based developments. However, their advantages are such that the software team can always bear these tiny overheads inorder to maintain the achieve an early delivery of software, reduction in expenses and to maintain the quality of the software.

USAGE IN EDUCATION

This section explains how our proposed model can be employed in research and education. Here, we present three case studies that describe how our proposed model can be effectively used in the respective research and academic projects. First case study is about an undergraduate research work being carried in the field of data mining. The area of research is to find an algorithm that can correctly classify the data from the skewed classes or in order words finding algorithm for classification of imbalanced datasets. Second case study is about an undergraduate project work being carried out in the field of storage and virtualization. In this project work, we trying to extend support for Fiber Channel over Ethernet (FCoE) by providing it a multiple initiator support. These two cases are the projects that we are working on as our undergraduate projects. The third case study is an industrial project and research work for development of an application using Spring dynamic module Open Service Gateway initiative framework. In this section, we show how our above mentioned XP approach assists us in effective utilization of costs, time and efforts.

CASE STUDY I : ALGORITHM FOR CLASSIFICATION OF IMBALANCED DATASETS

In this research work, the aim is to develop an algorithm that will correctly classify imbalanced datasets. In many real life applications, the datasets are such that there are many negative instances and only a few positive instances. The few positive instances are of utmost importance to the user, but traditional classifiers in data mining are biased towards the majority class and tend to ignore the minority class, classifying them as noise. Here, arises a need of developing a classifier which will correctly classify the few positive instances in their respective classes.

This research work requires that we study the existing classifiers and their functionalities. Existing implementations of most of these classifiers are available on the world wide web. Our approach of XP is a good choice for the process model. The research guide specifies the requirements of the research work in terms of user stories. The participants of the research work form an agile team and work on the problem statement. The user stories are then modeled in terms of different box structures.
If any reusable component is available in hand, then that is employed for the same. For instance, if we consider that the requirement or user story talks about proposing the most appropriate sampling method for probability distribution of the given dataset, then this is considered as a user story. This is a plausible task whose completion is possible in two-three weeks. Cleanroom approach comes into picture. Before starting the implementation, the user story is broken down into box structures. At the highest level of abstraction is the black box structure that holds the behavioural aspects of the user story, which says that the sampling method must be such that it has an acceptable coefficient of skewness and is not biased towards any class. Further, refinement is done to form the state box that illustrates the state transitions that occur when the sampling is taking place. Clear box specification includes the implementation details of the probability distribution. Now, we search if any existing component is available to suffice our purpose. Most of the data mining tools are equipped with implementations of standard and benchmark algorithms in classification of datasets. The sampling method named Poisson sampling is chosen and its implementation is readily available which can be used in our algorithm implementation. This implementation now needs to be edited according to our requirements of further changes in the research work. Other subsequent tasks can also be carried out in this way and as the research work evolves the increments can be integrated with the software.

**CASE STUDY II: MULTIPLE INITIATOR SUPPORT FOR FCOE**

Today, the enterprise data centers are expanding at a very fast rate. Everyday there are new proposals of update in the existing technologies in the field of storage and virtualization. Each application class has its own interface. For instance for ethernet have Network Interface Cards (NIC) and for storage protocols we have separate interfaces which work on fiber channel. These result in different networks and sets of hardware and cables. Fiber channel over Ethernet is one such standard that simplifies customer environments by transferring fiber channel frames encapsulated in the ethernet frames and allowing the industry to avoid creating another, separate protocol for I/O consolidation. But, currently FCoE architecture supports only point-to-point protocol and does not support multiple initiators for one target. The purpose of this project work is to provide multiple initiator support which will help in the initial testing of switches leading to verification of the implementation.

Applying our model in the development of the project, we find that we can fit our model as the process model very well. As in the previous case study the user gives his/her requirements in the form of a user story which is, for instance, development of a single initiator on a Linux machine. We model this user story in our box structure and identify that the module that we develop should possess the characteristics of an initiator. The state box further refines the internal details of various transitions that are possible in the different kernel modules and its compilation process. The clear box then holds the implementation of the initiator module. Being an open source project in OpenFCOE standards, there are umpteen modules available off-the-shelf that can function as an initiator. These modules can be chosen judiciously and brought into our project work. Then, the kernel needs to be recompiled with this module added or can be added as a pluggable kernel module. This module now has to undergo a series of statistical quality control tests to judge its reliability. The team members perform the verification of the module and then certify the reliability of the module. Further, the same approach will be applied to other user stories that the customer specifies.
CASE STUDY III : APPLICATION USING SPRING DYNAMIC MODULE OPEN SERVICE GATEWAY INITIATIVE FRAMEWORK

The objective of this project work is to take the combined advantages of both, the Spring framework in J2EE and Open Service Gateway initiative (OSGi), the application development framework. Spring offers advantages like inversion of control, dependency injection, aspect oriented programming and JDBC templates. While OSGi supports dynamic updation of modules wherein every module is segregated. There is no dependency between the modules. These modules can be installed, uninstalled, started, stopped without affecting any other module. OSGi also supports versionity that allows co-existence of multiple versions of the same software to be functional on the same system. Project work mainly focuses on having Spring module within the OSGi framework and act as an OSGi service, thereby taking the advantages of both the technologies.

Applying our model, if the requirement is such that a banking application is to be developed that will support the bank employees in maintaining the customer records. Thus, the Spring and OSGi approach segregate the features of adding an account, updating the account, etc. into different modules. These modules are treated as user stories. The customer specifies them and the software team breaks it into the box structure with black box highlighting the behaviour of the module. For example, if we consider, adding an account as one user story, then the behaviour of the module should be to ask for the customer details and addition of records to the bank database. Spring and OSGi offer versionity, so an additional option of version selection also has to be supported. Further, the state box contains the state transitions of the application calls and the backend database. The clear box contains the implementation details of the module. To make the application a loosely coupled one, we follow a new procedure for creation of objects in a Java class. If class A calls a method of class B, then instead of creating an object of class B in class A, we pass the parameters to a XML file and it takes care of the parameter passing and object reference intricacies. If this were not done, then the traditional unit testing approach would have to be employed which would test, class A first, then class B, since it is instantiated in class A. Now, the components of class A or class B are already available and can be reused in our current code. This fastens the development time and the testing efforts. Since, Spring and OSGi already offer the independence of modules, the verification and statistical quality control can also be easily employed in it. Thus, the software is rigorously tested for reliability, and there is an increase in the acceptance rate of the software application.

CONCLUSION

In this paper we have proposed an update to the traditional extreme programming agile model. Cleanroom software engineering along with component based development techniques are coupled with extreme programming. Terminologies of cleanroom approach like box structures can be representatives of the traditional XP user stories. The implementation of user stories can be done in conjunction with reusable software components. Verification and statistical quality control techniques of cleanroom approach can aid in reduction of costs, time and efforts. If this model is employed in research works and academic projects we believe that there will be a significant reduction in defects and a substantial increase in the acceptance rate of the software.

REFERENCES


