Extreme Programming Implementation in Academia for Software Engineering Sustainability

Lipsa Sadath
Faculty, Information Technology
Amity University
Dubai, UAE
lsadath@amityuniversity.ae

Kayvan Karim
Chief Technology Officer
WOXOMO
Dubai, UAE
kayvan@woxomo.io

Prof. Stephen Gill, PhD
Head, School of Mathematical and Computer Sciences
Heriot-Watt, Dubai, UAE
S.Gill@hw.ac.uk

Abstract—***The practice of making workable and maintainable software to meet the requirements of the use case is best defined as a software engineering practice. Fundamentally this is different from other engineering practices due to the abstraction level involved in the production. This fact produces different approaches for sustainability in the software industry. The importance of this practice towards a sustainable software industry in academia is very important. We propose a framework XPIA (Extreme Programming In Academia) that uses proven industry practices in software engineering with a focus on pair programming which is a practice in extreme programming to build a common knowledge and technical base in academics.***

Keywords— **Extreme Programming, Pair Programming, Software Engineering practices, XPIA, Academia**

I. INTRODUCTION

Software engineering is the art of making sure that your software has the expected behaviour in terms of usability. To ensure sustainability of engineering practices the produced software should include factors like functionality, consistency, simplicity, familiarity, safety and accessibility.

Extreme programming is a programming practice generally based on the values of its communication, simplicity, feedback and courage to execute the tasks. An educational experience moves someone along the path to expertise, preparing him or her to more effectively learn the knowledge that is required to perform appropriately in a new situation rather simply pouring expertise into his or her mind from which it can simply be recalled and applied [2].

The aim of our paper is finding the right gap between the software development practices learned in Universities and the level of knowledge and technical skills needed in the industry. Agile methods were argued to be born out of the frustration that software development teams were having with the waterfall methods followed in the software industry practices [1].

The proposed framework XPIA (Extreme Programming in Academia) uses proven industry practices in software engineering with a focus on pair programming which is a practice in extreme programming to build a common knowledge and technical base in academics. The main advantage of our framework is increased learning efficiency, knowledge and skill osmosis between pairs and getting students ready for the industry. We also discuss the concerns in academia during the phase.

As part of literature review and related works analysed, we have studied a few cases. They have proved to be vital for displaying information about real practices in organizations as they are a record of research performed. These studies performed on extreme programming at University levels and industries gives a thorough understanding and comparison of both aspects of education and industry including the distributed environment in the extreme programming field.

Section I is the introduction to software engineering, Section II is explanation of extreme programming practices, Section III is the proposed framework-XPIA and its proof working in pair programming in academia, Section IV is the implementation discussions, Section V covers related works reviewed from both academics and industry, Section VI are concerned touching academia practices, Section VII is a conclusion with suggestions for industry expertise as section VIII.

II. EXTREME PROGRAMMING IN PRACTICE

Extreme Programming is one of the agile techniques. It is a common practice among software developers to build a common understanding of a project and share the knowledge between the team members [3]. XP (extreme programming) as a framework like all the frameworks has some simple basic building blocks. Those blocks are following the main purpose of the agile methodology as well as providing a platform for the users of this framework to know what to do and how to do their job [4]. According to Newkirk et.al valued human communication over the paper, valued good code over good diagrams, valued quick iterations with customer feedback, and valued upfront software design to manage dependencies [5].

Authorized licensed use limited to: Northcentral University. Downloaded on February 28,2021 at 16:48:11 UTC from IEEE Xplore. Restrictions apply.

A. Whole Team

Everybody from managers, developers to the customer should be involved in the process of software production. Customer and developers should work as closely as possible. If possible, they should work in the same room with the developers. If that is not possible customer and developers should be as close as possible. They are aware of each other problems and they are working together to solve those problems.
Customer definition can change based on each project. Sometimes the customer can be a person or a team who can define and prioritize the product features. The customer can be some marketing analysts or business analysts or maybe even the user of the program. Whoever is the customer in a specific project, he should be available to the team and should consider as part of the team.

B. User Story

To plan a project there should be a list of requirements. These requirements also should be used to estimate the project cost and time. However, there should not be so much detail in these requirements. It is very likely that these details will change when the system is putting to gather, and the customer can see the result of requirements therefor capturing too much of details is considered as a waste of time and resources.
While talking to the customer the requirements of the system are getting defined, but the details related to those requirements are not getting captured. Rather, the customer writes a short sentence on an index-card as a reminder of that conversation and both the customer and the developer should agree that those words are enough to remember the requirement. The same time the customer is writing down the words for a specific requirement, the developers should estimate that requirement and put a number in front of the requirement. That number will be used later for prioritization and cost-based analysis. This index card is called a user story. A user story is a placeholder for an ongoing conversation related to a requirement and will be recalled many times during the life of a project.

C. Short Cycles

Every two weeks the XP team should provide a functional software with some of the requirements implemented in it. Stakeholders should review the system and their feedback should be received by the developers. These Short Cycles usually called iterations. Each small cycle produces minor changes to the system and after a certain number of changes, developers make a major release.

i. The Iteration Plans:

The duration of each iteration is usually two weeks. After each iteration, a minor deliverable software update should be available these minor updates might get into the production or can wait for the next batch release. Before each iteration, customer based on the defined budget by the developers, select the user stories for that iteration. Developers define the budget based on the previous delivery they had been achieved. Both sides should agree on the deliverable and customer should make the prioritization of the user stories. When the iteration started the requirements, user stories and the priorities should not be changed.

ii. THE RELEASE PLANS:

After some iterations, all the minor changes wrap into a major release and put into production. It can be the result of six iterations. The release plan budget is being set by the developers based on the previous release plans and customer priorities the release features and both sides agreed on them, but the release plan has the flexibility to get change based on the business requirements.

D. Acceptance Test

During the development of a user story or when a user story is getting implemented an acceptance test also is getting defined by the customer. Acceptance tests are the details of a user story and they should be written in a scripting language so that they can be run against the production code. When an acceptance test passed it will never be allowed to fail again and it stays as a proof of the implementation of the specific detail of a user story.

E. Pair programming

To spread the knowledge among all the developers and team members, two developer pair up and work at one workstation at the same time. This is like a driver and a navigator that one developer has the keyboard start writing code and the other developer watches closely and reviews and debugs at the same time. The positions can be changed several times for one hour and pairs get switched between each other during a day so that at least one developer should work in two pairs during a day. Pair programming increases the productivity of a team and increases the knowledge of everyone about the whole project.

F. Test-Driven Development

To write some production code developers should first write a simple test that fails and then they should write the minimum code required to pass that test. These processes produce lots of small unit tests that evolve during the implementation of the production code and any failure on those tests can be an indicator of broken features. Moreover, these unit tests help to refactor and improving the code quality and protecting the code base for future changes.
The core principles of Object-Oriented programming help developers to design and develop much better and more testable units and make each unit to be able to test independently. With patterns like dependency injection, developers can make each dependable module independent and test them against a controlled environment.

G. Collective Ownership

Every pair should check out any module, improve it and check it back again. Everybody should work from a Graphical user interface (GUI) to middle layers to services. Nobody holds an

Authorized licensed use limited to: Northcentral University. Downloaded on February 28,2021 at 16:48:11 UTC from IEEE Xplore. Restrictions apply.

ownership against a part of the software and everybody should work on all the code parts.

H. Continues Integration

The rule is simple, the first pair checks in first, wins and the others should merge into that code base. The code is get checked in several times during a day. XP teams use none blocking source control or distributed source control. These sources controls let developers check out any part of the source code regardless of that if anybody else had already checked that out. The first team checks in and the other teams merge to the same code base. Every team is responsible to run all the unit tests before checking in and no team should check in a broken code. The smaller the changes the easier to manage the changes so teams check in their code during a day several times.

I. Sustainable Pace

XP team work on a software project like running a marathon, not a sprint. Nobody should be burned out and no overtime is allowed. Overtime only allowed on some exceptional times like close to the release date. It is very important that the team should remember to work intentionally at a steady pace.

J. Open Workspace

The workspace can be different for each XP team but in general, a war room model works best for this practice. Tables with two to three stations on each and two chairs in front of each station is the practice. The walls are covered with Unified Modelling Language (UML) diagrams and the sound in the room is the conversation between pairs regarding the feathers or the implementation problems.

K. The Planning Game

To divide the responsibilities between different parts of the team, business side decides which user stories have the priority to get implemented for the next iteration and developers provide with the cost of each user stories and the total budget that they can implement. The cycles are short, so these numbers get adjusted very fast during the time of a project.

L. Simple Design

One of the most fundamental parts of the XP is the simple design. XP teams keep their focus and design for the current iteration, they should not worry about the things that will come or plan. Design should be as simple as possible. The initial assumption for the design is that they are not going to need it. This will prevent implementing or maintaining complicated unnecessary features or design that might just increase the overhead and not add anything to the overall value of the project.

M. Refactoring

Refactoring keeps the behaviour of the code externally but improves the code quality internally. After lots of iteration and debugging to prevent codes to rust or to turn in to unmaintainable spaghetti, XP developers review and refactor the codes frequently. After each refactoring, all the unit tests should run and pass to prove the fact that refactoring was successful, and it didn't break anything from the software. Refactoring is one of the key features in making high quality, flexible and maintainable software [6].

III. THE PROPOSED FRAMEWORK- EXTREME PROGRAMMING IN ACADEMIA-XPIA

The framework XPIA (Figure:1) is explained with assumptions, arguments and proof of knowledge and technical skill transfer during the deep pair programming practice as part of XPIA. The participants are considered to be university students working on the same project. Their objective and time deadlines are considered to be the same, with any discrepancy in knowledge and technical skills.

PAIR PROGRAMMING IN ACADEMIA

The concept helps students switch between ideas and developments among themselves bringing out the best in them. Since the methodology helps one developer work on the codes and the other work on bugs the confidence levels are better than working all alone. This imparts a lot of productivity in students to think deeper and distribute ideas to others. In a way, the method makes students capable of disseminating their ideas to the peers which are most required as a software designer and developer in the industry to take the lead. Bringing the practices of extreme programming to the system should be compared to pairs getting ready for the submission deadlines as part of project release after tests and iterations.

a) Assumptions:

We have the following assumptions as:
Let A = {$\alpha1, \alpha2, \alpha3, \ldots.., \alpha n$} be the skills of participant 1 possess.
Let B = {$\beta1, \beta2, \beta3, \ldots.., \beta n$} be the skills of participant 2 possess.
From the above let $\alpha1$, $\alpha2$ to be the knowledge skills and technical skills of Participant 1 and $\beta1$, $\beta2$ be the knowledge skills and technical skills of Participant 2 respectively.

b) Arguments:

The framework argues that there are a transfer and sharing of knowledge that takes place between the participants. Therefore, we argue $\alpha1$ and $\beta1$ (Knowledge skills) are transferable after implementing the XPIA among students. But the skills of a person continue to remain with him/her. But we prove that there is a component that is commonly generated by the participants.

c) Proof:

1. Before the XPIA practice

$P1=\alpha1+ \alpha2$
$P2= \beta1+\beta2$

2. After the XPIA practice and knowledge sharing

$P1= (\alpha1+ \alpha2).\beta1$
$P2= (\beta1+\beta2).\alpha1$

Authorized licensed use limited to: Northcentral University. Downloaded on February 28,2021 at 16:48:11 UTC from IEEE Xplore. Restrictions apply.

3. Results Achieved on completion of the project

$P1 = (\alpha1 + \alpha2).\ \beta1 = \boldsymbol{\alpha1.\ \beta1} + \alpha2.\ \beta1$
$P2 = (\beta1 + \beta2).\alpha1 = \boldsymbol{\beta1.\alpha1} + \beta2.\alpha1$

We call this generated common skill factor **α1. β1** between the two participants as the achievement sharing partial technical skills along with knowledge skills.

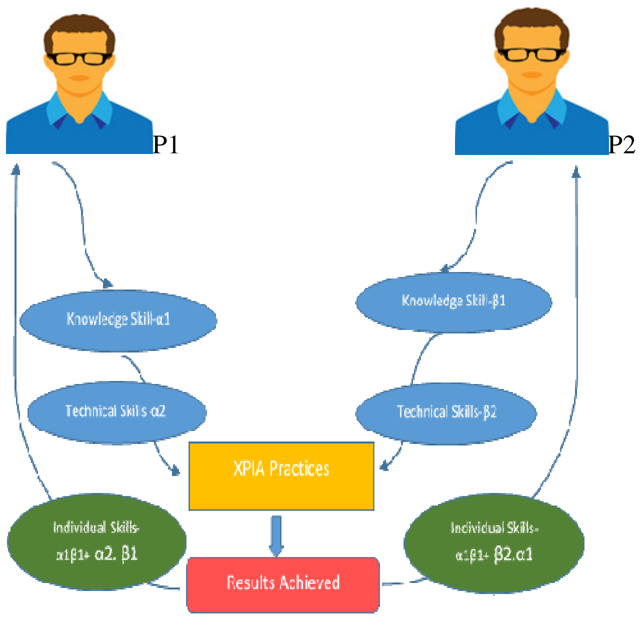

Figure: 1
The XPIA Practice

IV. IMPLEMENTATION

The practice is suggested to be implemented in institutions in the coming academic year as part of their curriculum so that students get through with the software engineering practices the first year of study onwards. A thorough knowledge of extreme programming practices is to be developed theoretically before they are given the practical disciplines of pair programming. XPIA can be practised over a year time in the final year study practices to ensure maturity of student thinking and technical skills.

V. RELATED WORK

Case Study I
This was a study performed at the University of Karlsruhe [7]. Their aim was to train students with the best extreme programming practices that could be incorporated into an evolving system quickly. All subjects considered were graduating students. The study focused on pair programming, iteration planning, testing, refactoring and scalability. The prime problems faced by the system was designed in small increments and writing test cases before coding. While advantages and disadvantages were quoted in the study about other aspects of extreme programming the main challenge was seen in testing. As there are two methods in testing in extreme programming, writing test cases before coding and making them automatically during regression, students were not convinced at the method in the beginning but later at the end, they were.
As extreme programming always required information exchange, it was observed that the team size was a crucial factor for the project. Small groups always did better and kept networking better than large groups. It was studied that extreme programming was meant for small teams due to communication overheads. Though small increments were found to be difficult, test cases were written before coding.

Case Study II
Another university type of study done in the Department of Computer Science at the College of College of Computer and Information Sciences, King Saud University [8] was about a project that went on for eleven weeks and had three releases. The study also showed that the best value in extreme programming was communication. The study followed extreme practices like planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective code, continuous integration and coding standards checked. Other extreme programming strategies were an incremental change, small initial project investment, stand-up meetings, tracking progress, minimal documentation, teaching strategies and experiments. The task given to the students was the development of ATM system. The aim was to give a simple but real problem to the students. It was suggested that id the problem is not really to be simulated students may lose confidence in extreme programming.
The results showed that there was the partial adoption of extreme programming sub-practices like planning game, pair programming, collecting code, unit testing, simple design and use of coding standards. The results as part of a survey through students' responses were welcoming. Two surveys were conducted to know the acceptability of the methodology. Both were great. On survey showed 60% acceptability while the other one showed a 33% of students commenting as the system a good idea but it will not work. Another 66% commented extreme programming as a system they have tried and have loved it.
Generally, the study showed that students were comfortable with extreme programming as it is code-oriented and did not follow the upfront detailed analysis of the traditional waterfall methodology. The system was also accepted for the small size of documentation. However, the challenge was more on on-site customer acceptability especially if the customer was an inexperienced person. It was observed that the team developed the full functionality of the required products with less work. Response to changes in extreme programming was faster compared to waterfall methodology.

Authorized licensed use limited to: Northcentral University. Downloaded on February 28,2021 at 16:48:11 UTC from IEEE Xplore. Restrictions apply.

Recommendations were given to incorporate extreme programming into the early stages of the curriculum at universities. Cost evaluation of the practices of extreme programming was also recommended.

Case Study III

Another case study considered was performed at Sabre Airline Solutions [9]. This was an industry-based case study. The team already had the same product released three years before experimenting extreme programming. The try was again done releasing the same product after the team became experts in extreme programming. This helped the case a better study comparing the old and the new systems. The new release was amazing. It showed a 55% increase in productivity of programmers, 65% increase in pre-release quality and 35% on post-release quality. The findings of the case study were used for scientific investigations into the real world as well. The team had a good stabilized extreme programming use.

The study considered five hypotheses to measure the pre-release quality, post-release quality, programmer productivity, customer satisfaction and team morale. The team used an ontology-based benchmark [9]. It recorded the context of the case study and the organization's adoption to modify extreme programming practices.

First, the team selection of the team was done and then the introduction of the system was done. The framework of XP-EF consists of three different kinds of matrices for the three sub-categories XP-cf, XP-am and XP-om. XP-cf is context factors that check and compares factors like software classifications, sociological, geographical, project-specific, technological, agronomical and developmental factors between old system & new system. XP-am which is the adherence matrix compared the planning adherence, coding adherence and testing adherence between the two systems. The third subcategory XB-om which stands for outcome measures compares the outcome measures between both the systems. The measures of the outcome signified that the hypotheses were all proven to be true while operated by teams within a specified context. While proving the hypothesis for customer satisfaction and morale one customer expressed satisfaction that the product released was the most professionally developed one.

However, the case study did not provide insight into applying extreme programming to larger groups. Even though the productivity and quality were higher the study revealed that 20% complexity in the system. Team members came up with feature complexities of extreme programming compared to waterfall models. But an overall post-released quality of 35% and commendable 50% programmer productivity was seen as an achievement.

Case study IV

The case study was found to be successful to share the experiences of building an extreme programming system in a distributed environment [11]. The scenario shares the case study in WDS Global were three independent development regions – Singapore, Pole & Seattle practised extreme programming in a distributed environment. The system required a lot of motivation to work as a global team. It was observed that the three regions had some common and differing needs. The company was in a practice of developing web-based tools in a known extreme programming environment. This caused the teams developing duplication of application programming interface. The system was getting expensive as well. There were a number of obstacles when a distributed extreme programming was thought about, especially when the teams did not have frequent communication with each other earlier. Again, cultural difference and time zone differences also mattered. Technical backgrounds of the extreme programming teams were another concern. Some had knowledge of object-oriented programming, some accepted and learned extreme programming as and when for meeting requirements, others who couldn't cope up with the new set up gradually left.

During the transition phase in the US & Singapore regions there were no pairing machines available, but eventually, they were taken care of. The system suffered an ownership problem when the UK team who developed the original core back-end was not ready to share the ownership with the other regions. Teams had to be made to understand the requirements of sharing in the system. Regional coaches were introduced to get regions equipped with Agile and extreme programming. To overcome the challenges in the new system boot camps in extreme programming were given in the UK office which had already moved to extreme programming. By the time non-UK developers were ready for extreme programming training and boot camps, the teams came together to form a virtual team. It was found that the network bandwidth was little too low to transfer and handle large traffic over the network.

VI. ACADEMIA CONCERNS

Basic concerns in academia are always the level of knowledge the student has benefited from the work he did when there is a partner to share the load. From the XPIA methods, we proved that knowledge should no more be a concern in academia after a pair programming practice.

Another concern is the level of plagiarism which academia suspects as to what is actually the amount of technical input by a particular student, this can be eliminated by the already existing method of viva voce that is practised in universities to know the student contributions in the project. However, our pair programming focuses on students doing an equal amount of work both intellectually and technically.

VII. CONCLUSION

It was observed that when a study was performed on the learning and implementation of extreme programming that was imposed on students, always a correction for the situation was suggested. The main suggestions included improvement in the method delivered and peer cooperation as students. Still, researchers have found always benefits of extreme programming in the education domain. Studies performed on engineers with good software engineering backgrounds didn't

Authorized licensed use limited to: Northcentral University. Downloaded on February 28,2021 at 16:48:11 UTC from IEEE Xplore. Restrictions apply.

bother much about future corrections and suggestions on the development of extreme programming but were concerned about economic constraints and consequences of the projects.

It was studied that the success of extreme programming relied on a shared set of values like standard coding, refactoring and test-driven environments developed. Whenever a crisis came team members of the systems were seen to blame the remote groups. The global team was thus struggling with vision and values.

In a distributed system, communication was found to be the most important in extreme programming. The system used video conferencing, time zone adjustments and pairing machines to get connected. To understand the market situation and cultural values of each region team members were made to travel to other regions for weeks. That was a very effective move. Coaches also communicated with each other continuously.

Teams who introduced high programming standards without informing others were not encouraged. The architecture of the system was found to be another problem in the distributed environment. The feedbacks taken from each region was time-consuming. A testing framework was adopted by the team for quality assurance. The quality was to be assured form the customer end.

The methods were seen to be helpful as the values of the extreme programming were maintained even in the distributed environment. The teams were to be balanced and same skill levels to be assured for the system.

This way, the organizations and institutions found the whole transition to extreme programming a success. The studies proved that extreme programming was a successful methodology implemented in any system tried. Education or industry the demand to know and learn more experimenting and experiencing extreme programming system was amazing according to our study. The method makes students to practice knowledge and skill sharing along with getting ready for industry expertise.

VIII. SUGGESTIONS AND FUTURE WORK

The concept of XPIA is unarguably a practice that can be taken up in universities without doubt as this enhances the student ability to be smarter for a better industry experience to get established.

In future, we plan to implement the practice in the first year of the study itself so that students come up with better quality projects and industry efficient products by the end of final year completion. This will enable the students absolutely capable to face the whiteboard testing practices as well in the industry.

IX. REFERENCES

[1] Pearman, Greg., James. Goodwill, SpringerLink, and LINK. Pro .NET 2.0 Extreme Programming Greg Pearman and James Goodwill. Berkeley, CA: Apress, 2006. Expert's Voice. Web.

[2] M. Muller, E. Tichy (2001), “Case Study: Extreme programming in a University Environment”. [accessed 05/11/16]. http://faculty.salisbury.edu/~xswang/research/papers/serelated/xp/xpinuniversityenvironment.pdf

[3] Bahli, B.; Zeid, E.S.A., "The role of knowledge creation in adopting extreme programming model: an empirical study," International Conference on Information and Communication Technology, pp. 75-87, 2005.

[4] R. C. Martin and M. Martin, Agile principles patterns and practices in c#, Boston: Pearson Education, Inc., 2007.

[5] Newkirk. J, Martin. C. R, Extreme Programming in Practice, OOPSLA 2000 Companion Minneapolis, Minnesota © Copyright ACM 2000 1-58113-307-3/00/10

[6] M. Fowler, Refactoring: Improving the Design of Existing Code, 1999: Addison-Wesley.

[7] L. Layman, L. Williams, L. Cunningham,(2004) “Exploring Extreme Programming in Context: An Industrial Case Study” [accessed on 05/11/16] http://collaboration.csc.ncsu.edu/laurie/Papers/ADC.pdf

[8] G. Assassa, H. Mathkour, H. Dossari, “Extreme Programming: Paper: “Extreme programming: A case study in software engineering courses”, 233-240.

[9] L. Williams, W. Krebs, and L. Layman, "Extreme Programming Evaluation Framework for Object-Oriented Languages Version 1.3," North Carolina State University Department of Computer Science, Raleigh, NC, TR-2004-11, April 6, 2004.

[10] "Case Study: Extreme programming in a University Environment," [Online]. Available: http://faculty.salisbury.edu/~xswang/research/papers/serelated/xp/xpinuniversityenvironment.pdf. [Accessed 05 11 2016].

[11] M. Yap(2010), “Implementing Distributed Extreme Programming: A Case study” [accessed on 05/11/16]. https://www.solutionsiq.com/docs/implementing-distributed-xp-case-study.pdf

Authorized licensed use limited to: Northcentral University. Downloaded on February 28,2021 at 16:48:11 UTC from IEEE Xplore. Restrictions apply.