Group Report on the

CS7CS3 Advanced Software Engineering Group Project

Group: *9*
Akshay Mathur, Nikhil Girraj Khandelwal, Sarvani Chakrabarty, Sujit Maruti Jadhav, Taranvir Singh, Utkarsh Bhardwaj

This report is intended for you to provide a reflection on the use of eXtreme Programming in your group project. Please give a "real" assessment of the actual benefits and drawbacks of using XP for your project under each of the 12 core practices of XP. Indicate when each category was not applicable, and why.

1. Whole Team

"All the contributors to an XP project – developers, business analysts, testers, etc. – work together in an open space, members of one team…."

How was this applied in your team?

- Took small steps and mitigated failures as they happened
- Worked together on all parts, from requirement analysis to code and maintained communication on a daily basis
- Made frequent small releases to enhance and adapt to any changes necessary
- Divided project into iterations and moved the teammates around every part of coding
- Unit tests were attempted to be developed first
- Carried out pair programming extensively

Benefits

- Pair programming helped the team members to work with and understand all segments of the project code
- Continuous integration and deployment made processes move quicker
- Since the teammates are moving across the different segments of the process, there is complete transparency and helps in clear communication

Drawbacks

- Measuring code quality assurance was difficult
- Due to COVID-19, pair programming suffered temporarily due to different geographical locations
- Since the process changes quite often, documenting each and every change becomes challenging

Lessons learned

The golden rule of XP is test first, then develop. Of course, there are risks when you go to "extreme" testing and then coding. Nevertheless, XP was the natural idea while working with an unfamiliar technology and this approach proved to be efficient and gave the coders the opportunity to work with all segments of the code.

2. Planning Game

"Planning is continuous and progressive. Every two weeks, for the next two weeks, developers estimate the cost of candidate features, and customers select those features to be implemented based upon cost and business value"

How was this applied in your team?

- Had meetings at a well defined interval, dedicating 10 hours a week to the project
- Requirements and commitments from the teammates were defined to meet the needs of the different parts of the project
- The team of 6 was divided into pairs, and the pairs were changed on a regular basis
- Previously decided plans were often changed and adjusted to meet the evolving needs of the project
- Necessary tasks yet to be completed to meet the upcoming iteration release were assigned to the developers and scheduled suitably (mostly in two week iterations)

Benefits

- Any bugs were addressed to and resolved quickly
- Since the process was flexible, the code could be moulded in different ways as more ideas streamed

Drawbacks

- XP requires a detailed planning from the start due to changing scope
- Project management related with the practice that changes during the life cycle is not always a smooth ride

Lessons learned

Release planning helped the team to estimate the difficulty level of the problem in hand. Our initial plan to switch pairs every week seemed to be too frequent and hence we switched to changing pairs every two weeks. Iteration planning kept the team on track and provided the clarity required to build different aspects of the application.

3. Customer Tests

"As part of selecting each desired feature, the customers define automated acceptance tests to show that this feature is working"

How was this applied in your team?

Customer tests were not quite possible. However, it was attempted to have a pair "act" as customers to see the overall flow of the application to check if it was fulfilling the requirements of the project from the customers' perspectives.

Benefits

Helped in noticing discrepancies in the UI

Drawbacks

Since there were no actual acceptance tests carried out, the true scenario is lacking

Lessons learned

Acceptance tests would have shed more light to the working of the application.

4. Simple Design

"The team keeps the design exactly suited for the current functionality of the system. It passes all the tests, contains no duplication, expresses everything the authors want expressed, and contains as little code as possible".

How was this applied in your team?

- Verified code by automated tests
- Tried to keep the code to minimum number of components
- Attempted to keep code duplication to the least
- Included refactoring
- Maintained a list of design flaws which required an explicit remediation through refactoring

Benefits

- Since the code was kept to the simplest form, it was possible to work on it and improve at any moment
- Mitigated the risk of overdesign

Drawbacks

Since all team members worked on every aspect of the codes, there were instances of duplication.

Lessons learned

Simple design helped the software to be easy to change, since most design decisions were agnostic to which particular changes were expected

5. Pair Programming

"All production software is built by two programmers, sitting side by side, at the same machine".

How was this applied in your team?

- The team of 6 was split into 3 pairs of team members
- Sprints of 1 week were made for first semester and 2 weeks for second semester
- The partners in the pair were rotated in every sprint throughout the team

Benefits

- Promoted better communication and team building
- Programmers learned from each other
- Provided equal opportunity to all the developers to work on all sections of the code

Drawbacks

- Geographical locations have a huge impact on pair programming
- A "watch the master" phenomenon arise sometimes when an experienced and a novice programmer pair up, with the novice becoming the observer
- Pair programming didn't seem to be resourceful in the initial requirements deciding phase as discussing the research done every time the pair changes was time consuming and there was ought to be some misinformation handling as well as missing out on valuable information in the process.

Lessons learned

Pair programming is beneficial in terms of fewer bugs and helps in the emergence of new ideas. It gives the programmers to discuss and exchange ideas. Our initial plan to switch pairs every week seemed to be too frequent and hence we switched to changing pairs every two weeks. Programming aloud instead of silently programming was a challenge. Our initial plan to switch pairs every week seemed to be too frequent and hence we switched to changing pairs every two weeks. Current COVID scenario temporarily affected the pair programming schedule, however, that was solved using Microsoft Teams and Hangouts.

6. Test-Driven Development

"The programmers work in very short cycles, adding a failing test, then making it work"

How was this applied in your team?

- Coded unit tests first
- Created automated tests to fail, then developed the code to make them pass

- Frequently provided feedback through unit testing and acceptance testing

Benefits

- Regular testing ensured early detection of bugs
- Helped programmers to have a clearer idea of the invariants and constraints that the implementation will have to satisfy
- Refactoring was easier to be done as needs and designs changed

Drawbacks

- Took longer to get to writing the final code
- Spent time in writing test cases which were ended up not being used
- TDD is difficult to achieve along with works which tend to be Proof-of-Concept in nature.

Lessons learned

The TDD process seemed slower at the beginning as we had to keep all scenarios in mind while writing them, however, things moved much faster in the development phase and codes were easier to understand.

7. Design Improvement

"Don't let the sun set on bad code. Keep the code as clean and expressive as possible"

How was this applied in your team?

- Coding design was kept as simple as possible
- Carried out refactoring whenever and wherever possible which was followed by testing to make sure nothing is broken
- Functionalities were not added in the early stages
- Maintained naming conventions for easier communication

Benefits

- Refactoring helped in removing redundancy and increased the code cohesion
- Naming conventions helped in quick navigation and search for specific functionalities

Drawbacks

- Since XP focuses more on coding than design, the prospect of a good design is lacking

Lessons learned

Simple and clean code helped in better understanding and refactoring helped reduce clutter in the code.

8. Continuous Integration

"The team keeps the system fully integrated at all times"

How was this applied in your team?

- Each pair wrote their code and integrated it together
- Testing followed refactoring to make sure nothing is broken
- Integrated testing and changes in the development phase continuously

Benefits

- Continuously integrating ensured that everyone has the latest version of the project
- Helped in detecting compatibility problem at an early stage

Drawbacks

- There would be instances when multiple developers would want to integrate their codes around the same time which led to slowing down the process.

Lessons learned

Continuous Integration was extremely beneficial in maintaining the latest version and promoting working at a granular level.

9. Collective Code Ownership

"Any pair of programmers can improve any code at any time"

How was this applied in your team?

- Adhering to a consistent coding standard, each member contributed in a way which can be deciphered by all team members
- If any part of the code is incomprehensible, they were replaced by simpler codes to make it understandable to everyone.
- Unit tests made it possible for team members to apply changes to source code without affecting the overall functionality

Benefits

- Collective ownership of codes, through unit tests, ensured that versions which are incompatible with the team's codes are fixed as soon as they develop.
- Increased accountability for all team members

Drawbacks

There were no such drawbacks since everyone had their guard against the chaos of "no ownership" and took equal ownership of all sections of the code.

Lessons learned

This was a huge advantage in the project since the collective code ownership meant equal responsibility and the team was at par with the latest versions. This proved to be

more efficient as the deadline loomed closer. It also helped when pair programming had to be temporarily suspended during the initial stages of the lockdown.

10. Coding Standard

"All the code in the systems looks as if it was written by a single – very competent – individual"

How was this applied in your team?

- Mobile Application:
  The link for the coding standard document which will be followed by the team for development of the Mobile Application is given below. The document serves as the complete definition of Google's coding standards for source code in the Java™ Programming Language. Like other programming style guides, the issues covered span not only aesthetic issues of formatting but other types of conventions or coding standards as well. However, this document focuses primarily on the hard-and-fast rules that we follow universally and avoids giving advice that isn't clearly enforceable (whether by human or tool).
  Link: https://source.android.com/setup/contribute/code-style
- Server Code:
  The link for the coding standard document which will be followed by the team for the server is given below. This document gives coding conventions for the Python code comprising the standard library in the main Python distribution.
  Link: https://www.python.org/dev/peps/pep-0008/
- Variable naming conventions were consistent throughout the code. They described the data they contained.

Benefits

- Helped in better understanding and readability of the code
- Since pairs rotated on a regular basis, adhering to a consistent standard of code was useful
- It was possible to understand the code where we had left it even after a time gap

Drawbacks

- Imposes some restriction on exploring the creativity element of programming

Lessons learned

Coding standards provide clarity to the purpose of the code. A neat well laid out code with optimum commenting is easy to comprehend and enhance.

11. Metaphor

"The team develops a common vision of how the program works"

How was this applied in your team?

- Ensured that team members would work for the end goal and not get swayed by things which aren't necessary to the project.

Benefits

- Everyone had a similar understanding of the bigger picture
- Makes communication among members easier

Drawbacks

None

Lessons learned

Following metaphor has only helped in enhancing the overall structure of the project.

12. Sustainable Pace

"The team is in it for the long term. They work hard, at a pace that can be sustained indefinitely. They conserve their energy, treating the project as a marathon rather than a sprint"

How was this applied in your team?

- 10 hours a week was set for the project
- Meetings were fixed for 2-3 days a week.
- After the lockdown, the number of days were increased with the number of hours per call reduced due to the different time zones. However, the total number of hours per week was maintained.

Benefits

- Gave an idea of exactly how much time we have to invest to get the work done
- Was easier to plan other assignment meetings as we had fixed timings for ASE

Drawbacks

- It was initially difficult to define timetable due to members having different electives

Lessons learned

Having a sustainable pace helped us to give equal attention to other projects and assignments.

13. Overall Project

Benefits

- Helped in learning Android which none of us had experience with
- Stronger sense of responsibility was built throughout the team
- Harmony was maintained throughout the project, even with the pandemic fiasco

Drawbacks

- Quite some time was spent in designing tests
- Initially, found it difficult to work in pairs

Lessons learned

It was a different experience for all the team members. Extreme Programming was truly “extreme” at times, as it requires a lot of effort. However, this approach proved to be efficient and the learning curve was huge, not only from the technical aspect, but also from the people involved.