Group Report on the

CS7CS3 Advanced Software Engineering Group Project

Group: *16*
SHAUNAK PEDGAONKAR
TANAY DARDA
ASHIQUR RAHMAN HABEEB RAHUMAN

This report is intended for you to provide a reflection on the use of eXtreme Programming in your group project.

Instructions:

1. 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].
2. Change the <group number> in the heading above to your group number
3. Change the <list of group members> to be your group members
4. Change the name of this file to replace the 'N' in the current 'GroupN' with your group number
5. Submit on Blackboard – note ONE SUBMISSION is sufficient for your whole group.

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?

Our group worked as a solid unit, and every member was present at meetings on a regular basis. These regular meetings provided a platform for sharing ideas, project planning, and challenges. Every member in this setting contributes actively to every conversation, which promotes a cooperative atmosphere. We all participated in the entire development process as an integrated group rather than working independently.

This thorough engagement began with the planning and designing stages and continued through the coding and testing phases of execution. Every team member offered their perspectives, knowledge, and work to all areas of the project, tried to build an accurate plan and increased the group's overall capabilities.

[1] It is highly unlikely (I would suggest not possible) that any Generative AI tool will be able to tell you what the experiences were in your group!

Benefits as experienced in your team

Working in groups allowed us to better understand each other's strengths and improve our communication skills when it came to planning and execution. This helped us in making effective decisions. Our productivity increased when we worked in groups and pairs. We promptly corrected our errors and devised a more effective strategy.

Drawbacks as experienced in your team

We occasionally had disagreements on project ideas and methodologies, but those were only ever settled then and then.

Lessons learned

We observed that working together is the most effective approach to regularly record team member feedback, iterate existing solutions, and generate ideas in an organised setting.

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?

Planning was an ongoing process, we planned our tasks for each week and assigned the work to each member within our group. At the end of every week we checked our progress, checked if our task were completed and if not what challenges are we facing at that moment, based on this we would plan upcoming tasks for next week

Benefits as experienced in your team

The key benefit of the above approach was its ability to swiftly change our plans in response to any unexpected alterations or difficulties that emerged. By making estimations and choosing characteristics within a brief time structure, we had the ability to focus on speedily delivering specific results, which preserved the group's productivity.

Drawbacks as experienced in your team

However, this approach occasionally leads to a tendency to prioritise work that might have been finished quickly, rather than focusing on those which were more crucial in the long run. The absence of an actual external customer sometimes posed difficulties for effectively evaluating the significance of tasks.

Lessons learned

We gained an understanding of maintaining a balance between the critical tasks that need to be completed and the overall objectives of the project. In the future, it would be beneficial for us to often pause and evaluate our priorities within the framework of the full project timeline.

3. Customer Tests

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

NOTE – This principle cannot be applied in your team, as you do not have a customer on your team. If you have an opinion on whether this would have been beneficial, please add it here.

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?

We exhibited our stand on the principle of simple architecture by carefully applying pruning approaches, ensuring that our codebase had only the essential elements required for its current functionality. We intentionally minimised unnecessary complexity, prioritising present requirements instead of potential future needs.

Benefits as experienced in your team

This design choice proved beneficial for quick adaptability and maintaining clear focus within each subgroup, ensuring that integration of AWS for data management and Google Maps for routing remained robust and efficient.

Drawbacks as experienced in your team

The initial simplicity of the system required modification as it progressed to incorporate more sophisticated features and integrations. This required careful consideration to enhance the system's abilities without the need for major reworking.

Lessons learned

We noticed that although simple architecture improves the development process and facilitates team migrations, it is crucial to plan for future expansion. Attempting to improve our ability to effectively manage the complex equilibrium between planning for development while preserving the fundamental layout is a goal we aspire to achieve in future projects.

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?

As a team of three, we completed most of the tasks as a team; but, for various assignments, we worked in pairs. Prior to agreeing on a strategy, the pair would work on implementing the feature. First, we would read and research the functionality that needed to be built. Periodically, the pairings were switched around to increase cooperation and disperse expertise among various areas of the codebase.

Benefits as experienced in your team

Ensuring adherence to coding standards and early error detection were made possible by reviewing the code continuously.
Pair programming promoted helping and sharing ideas among ourselves, bringing all members together in pairs to speed up and disseminate system knowledge throughout the team.
Strong working relationships and increased communication skills were developed among team members as a result of close cooperation.

Pair programming brought out the strengths of all members and improved collective decision-making.

Drawbacks as experienced in your team

There were no drawbacks to pair programming because we made an effort to support one another when needed. If we were to list the drawbacks, they would be that there were instances of miscommunication and that scheduling might have been difficult because everyone had additional tasks they needed to focus on.

Lessons learned

Pair Compatibility: In order to create a peaceful and effective work environment, it was essential to take personality and working styles into account while forming pairs.
Task Appropriateness: In difficult, high-risk jobs where the benefits of teamwork exceed the possibility of a reduction in individual productivity, pair programming may be more advantageous.
Flexible Implementation: To optimise pair programming's advantages and minimise its disadvantages, it can be applied with flexibility. For example, it can be allowed for tasks where it may not be beneficial.

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?

We made test cases for each aspect of the project. We ensured that all of our project's features worked properly. We used tools and frameworks that were compatible with our development environment to automate our testing. We were able to rapidly run tests and get feedback on the state of our code thanks to this automation, which sped up the debugging and iteration process. Example: For our web pages, we always pre - decided our test cases: functionality testing, usability testing, browser compatibility testing, performance testing, or for our database, we build test cases like data integrity testing, data accuracy testing, validation testing, performance testing, and concurrency testing.

Additionally, we used a variety of frameworks and technologies that were appropriate with our development environment to automate our testing. We were able to rapidly run tests and get feedback on the state of our code thanks to this automation, which sped up the debugging and iteration process.

Benefits as experienced in your team

Test driven development made us consider our code's architecture up front, which led to the creation of cleaner, more modular implementations. We were able to find possible problems and edge cases early in the development process by creating tests before developing code, which resulted in more reliable solutions.

Debugging got easier since we created tests to ensure that our code was correct. The amount of time spent looking for hidden problems was decreased when a test failed and gave explicit feedback on what needed to be corrected.

As we made changes or refactored existing code, we felt confident knowing that our code was supported by extensive test coverage. We could run the test suite to immediately confirm that our changes had not resulted in regressions.

Drawbacks as experienced in your team

Making test cases for every functionality was challenging for us at first since sometimes our functionality failed to pass all of the tests for different frameworks and tools, forcing us to dedicate more time to finding a solution. However, as time went on, we became accustomed to and comfortable with creating test cases.

Sometimes it felt like more work to write tests before developing code, especially for things that are basic or easy to use.

Lessons learned

Mitigating the learning curve and promoting acceptance can be achieved by providing team members with sufficient training and resources on test driven development methodologies and testing frameworks.

Test-Driven Despite certain difficulties, our team project's development was mostly favourable. The first learning curve and perceived overhead were surpassed by the advantages of better code quality, quicker debugging, and more trust in our programming. We were ultimately able to produce a more dependable and maintainable project.

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?

We made sure that whenever we made modifications or added new features, we also enhanced the current codebase by incorporating refactoring into our everyday operations. This made it easier for us to keep the project's code quality at a high level.
Pair programming - by working in pairs, we were able to make sure that the code was well-structured and comprehensible and to identify possible problems early on. This cooperative method decreased the possibility of introducing inadequate code while encouraging information exchange.

Benefits as experienced in your team

Enhanced Readability: By emphasising clear and expressive code, team members were better able to understand one another's contributions, which cut down on the amount of time needed for debugging and code review.
Faster Development: Although it may sound strange, our development process was ultimately sped up by putting a focus on code quality. Over time, productivity rose because cleaner code was simpler to grow and maintain.

Drawbacks as experienced in your team

Time-intensive: It took a lot of time and effort to maintain a high level of code quality. We occasionally discovered that we were spending more time writing tests and reworking than we were really implementing new things, which may be viewed as a short-term productivity setback.
Opposition to Change: Initially, a few team members found it difficult to accept the idea of constantly reworking code or to invest the effort in creating tests beforehand. It took persuasive language and skillful communication to overcome this opposition.

Lessons learned

Prioritising code quality is crucial, but it's equally critical to balance long-term maintainability with short-term deadlines. Sometimes, in order to satisfy urgent project objectives, rapid fixes may be required. However, in order to prevent racking up technical debt, refactoring efforts should come after quick fixes.

8. Continuous Integration

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

How was this applied in your team?

We used Continuous Integration techniques in our team project to maintain system integration at all times. Our version control system was GitHub, and each time we made a change, team members pushed code to a repository.
In order to make sure that changes were recorded and tracked gradually, team members committed frequently to their individual branches. To help with collaboration and to isolate changes, we used branching strategies like feature branches and bug fix branches.
Using CI/CD systems that were integrated with GitHub, we set up automated build and test pipelines. These pipelines made sure that updates were automatically created, tested, and checked for accuracy with each push to the repository.

Benefits as experienced in your team

We were able to find conflicts, errors, and integration problems early in the development process thanks to continuous integration. We kept a high standard of

code quality and reduced the possibility of introducing regressions by identifying problems as soon as they might be discovered.

Teamwork was aided by the application of branching techniques and version control. Changes could be easily merged and integrated when they were ready, allowing each member to work freely on their allocated tasks without interfering with the work of others.

Drawbacks as experienced in your team

There was additional complexity added by setting up CI/CD pipelines and maintaining several branches. To guarantee that the modifications were properly combined and integrated, close collaboration and communication were needed.

Lessons learned

Processes for development and integration were streamlined by using a consistent branching strategy. To prevent confusion, we discovered how crucial it is to have explicit naming rules and documentation for branches.

The successful adoption of continuous integration required excellent team communication. To maintain alignment and prevent disputes, we have learnt to discuss modifications, updates, and integration plans on a frequent basis.

9. Collective Code Ownership

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

How was this applied in your team?

We were working in pairs, and as a whole, every task and code were discussed with each member before finalising. Individual tasks were assigned, but everyone shared their opinions about each aspect of the project.

All sections of the codebase were accessible to team members in our collaborative, open atmosphere. This made it possible for everyone to examine, comprehend, and add to any part of the code as needed.
Our team often engaged in pair programming sessions where members collaborated on a range of projects. By encouraging knowledge exchange and teamwork, this approach allowed people to contribute to parts of the code that they would not have been familiar with at first.
As a part of our development process, we held frequent code reviews. Regardless of who authored the initial code, any team member could review and comment on changes made by others to the code.

Benefits as experienced in your team

Enhanced Code Quality: As a result of shared code ownership, overall code quality increased. Multiple sets of eyes allowed for the more efficient identification and correction of problems including bugs, design flaws, and code smells.
Faster Problem Solving: Since any team member may work on any section of the software, issues could be settled swiftly. When one or more team members need to debug or improve a specific area of code, they may work together to develop solutions.
Knowledge Sharing and Skill Development: Team members' ability to share and improve knowledge was encouraged by collective code ownership. Individuals were exposed to new technologies, coding styles, and best practices while working on various sections of the codebase.

Drawbacks as experienced in your team

Although shared code ownership promoted cooperation, there was a chance that it might also give rise to disputes or arguments about code modifications. Disputes about implementation details, design patterns, or coding conventions can occasionally impede development.
In certain situations, a lack of responsibility may result from unclear ownership of particular codebase sections. Important parts of the code might not get the attention they need or might be missed during maintenance if there isn't a designated owner.

Lessons learned

Effective communal code ownership requires clear communication and teamwork. We discovered that in order to reduce conflict, it is crucial to communicate and work together on coding standards, design choices, and implementation strategies.

Establishing policies and procedures for code review, merge requests, and conflict resolution is essential while keeping an open codebase. Well defined a process to promote uniformity and preserve the calibre of the code.

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?

We established a coding standard to ensure consistency across our project's codebase. This became particularly significant after the team's division into two smaller units, focusing on simplifying our committed use cases due to reduced manpower.

Benefits as experienced in your team

Implementing a uniform coding standard for both subgroups ensured that even while duties were made simpler, the quality and readability of our code remained at a high level. This enhanced the efficiency of the workflow and simplified the process of maintenance.

Drawbacks as experienced in your team

The main challenge was adapting the coding standard to a reduced feature set while preventing over-simplification that could undermine the system’s functionality.

Lessons learned

11. Metaphor

“The team develops a common vision of how the program works”

How was this applied in your team?

Over the time of our project, we had entitled a common metaphor for work - “City as a living source”. This mainly helped to conceptualise the dynamic and interconnected nature of the response to disaster response systems.This metaphor was particularly effective in aiding both understanding the system’s functionalities and the flow of the data from the database to aid the emergency coordination of the incident response.

Benefits as experienced in your team

The metaphor provided a significant role in promoting a shared development mindset among every team member, which became especially significant following the division of our team into two separate groups. It assisted in maintaining a uniform development strategy, guaranteeing that despite working on distinct components, the combined system operated smoothly and effectively.

Drawbacks as experienced in your team

The idea of "City as a Living Source" helped understand and design the system, but it often made it difficult to address the technological issues, especially in handling data in real-time flows and system integration.

Lessons learned

Though the metaphor effectively streamline the groups vision to enhance the understanding though the group were divided, it helped to avoid technical complexities.We learned that, enhancing the flow of the project through any technical

roadmaps and clear strategies to enhance the communication between members.This will ensure that while our metaphors provide a cohesive and inspiring vision, they are also grounded in practical, executable plans that anticipate and address the specific technical challenges of our projects.

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?

We implemented a continuous integration of the code base using GitHub as the platform to manage our own repository for each.We setted up to automate the builds and tests to the code with every time each member committed a change. This had ensured that our codebase remained in a deployable state,which was crucial.After an clear assessment of each code working,it would be pushed to the main branch.

Benefits as experienced in your team

It minimised integrating issues, allowing us to easily recognize and resolve conflicts and faults. This method was very successful in sustaining a superior level of code excellence and guaranteeing consistent alignment of outputs from both subgroups.

Drawbacks as experienced in your team

A significant challenge held here was maintaining it especially under tight deadlines.Occasionally, when a new push of code was done, it led to premature commits that broke the build, causing delays and requiring debugging of the initial phase of code.

Lessons learned

The experience highlighted the need of complying strictly to Continuous Integration methods.This approach facilitated enhanced productivity and encouraged seamless collaboration among our team members across various project components.

13. Overall Project

Overall we learned so many valuable lessons from this projects as we worked through each part of the project .Everyone felt encouraged to participate and exchange ideas because of the collaborative atmosphere our team created. This culture of cooperation encouraged creativity and invention, which produced stronger ideas and better results.

Benefits as experienced in your team
Everyone felt emboldened to participate and exchange ideas because of the collaborative atmosphere our team created. This culture of cooperation encouraged creativity and invention, which produced stronger ideas and better results.

We maintained our flexibility and adaptability throughout the project in response to shifting priorities and requirements. Our ability to adapt allowed us to change course as needed and modify our strategy to suit the project's changing requirements.

Continuous improvement was given top priority during the course of the project. We improved our procedures and produced better results by adopting techniques like agile approaches, continuous integration, and collective code ownership.

Drawbacks as experienced in your team

There were difficulties in integrating data from many sources, such as discrepancies in data quality, inconsistent data formats, and compatibility problems. Overcoming these obstacles took a lot of planning and work.

There have occasionally been misunderstandings or breakdowns in communication. These difficulties occasionally led to misplaced expectations or difficulties in the completion of thesome of the functionalities of the project.

Decision-makers faced difficulties in figuring out the proper response plans and in analysing different data streams due to their complexity. Determining how to balance variables like traffic conditions, resource availability, and event severity took significant thought and occasionally caused delays in decision-making.

Lessons learned

The course of our Work taught us a lot. The functioning of the disaster recovery management system was discussed to us. We gained knowledge on how to construct complex algorithms into applications and how to synchronise every feature necessary for an application to function. We had the opportunity to study new technology. We gained knowledge about how to work in pairs, hold regular group meetings, and use brainstorming sessions results better project outcome

For a project to be successful, scope and expectations must be actively managed. We gained knowledge on how to set precise project goals, rank requirements, and periodically assess and share any modifications.

14. Analysis of the use of Generative AI tools in your project, as experienced in your team

Strengths of Gen AI tools as experienced in your team across the whole development life cycle (where relevant). Please give explicit examples.

1. Debugging error : Django server error code can be provided to gpt and it gives right areas to check and fix the bug.
2. Syntax correction: Ask for syntax and it provides good quality code.

Limitations of Gen AI tools as experienced in your team across the whole development life cycle (where relevant). Please give explicit examples.

1. Inefficient in complex logic or architecture design.
2. Provides biassed results for less popular programming languages and services like cpython server configuration or wsgi setup as per the project architecture.

Lessons learned

1. Do research by browsing around reading code repositories, trying logics and fixing the things as per requirement.
2. Clear understanding of requirements with good details can provide expected results in generative AI.