Group Report on the

CS7CS3 Advanced Software Engineering Group Project

Group: *6*

Weiwei Wan

Madhura Vishal Nagle

Sharon Andrea Gomez

Mansi Paradkar

Junwei Yu

Prishita Singh

Zhiqiang Cheng

Long Pan

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?

To apply eXtreme Programming in our project, the following strategies were applied during our development processes.

Planning step:

- We clearly defined the project's scope, objectives, and requirements, created functional architecture and technical architecture to define the details of the project’ s components.
- We created a prioritized list of functions, features, and tasks that need to be completed based on their importance to the project.
- We defined the specification of coding standards.
- We planned a series of short iterations, each lasting one week, to develop and release new features and functions, and prioritized the features and tasks in the backlog for each iteration.

Implementation step:

- We emphasized communication between group members. Each week we worked together and we also had weekly meetings to talk about the process of our project and problems we met.
- We used pair programming to develop our project, two members worked together on the same code to share knowledge and improve code quality.
- We used continuous integration techniques to build and test the code frequently, to ensure that it is of high quality and free from errors.
- We used testing and test-driven development to ensure that the code is reliable, robust, and meets the requirements.
- We used regular retrospectives to reflect on our team's performance, identify areas for improvement, and adjust the development process as needed.
- All team members of our group take responsibility for the codebase to promote collaboration and ensure that the code is maintainable over time.

By following these steps, our software development team can effectively apply eXtreme Programming to our project, and deliver high-quality software quickly and efficiently.

Benefits

eXtreme Programming offers several benefits to our team, including:

- XP’ s main advantage is saving costs and time required for project realization. Time savings are available because of the fact that XP focuses on the timely delivery of final products. XP teams save lots of money because they don’ t use too much documentation but use group meetings instead.

- XP group could create extremely simple code that can be improved at any moment.
- XP is designed to handle changes in requirements and priorities throughout our project, making it ideal for situations where requirements may be uncertain or change frequently.
- XP emphasizes quick development cycles, with small, frequent releases. This approach allows our team to deliver software more quickly, which can help to keep stakeholders engaged and satisfied.
- XP's emphasis on testing and continuous integration helps to ensure that code is of high quality and free from errors.
- XP is particularly useful in situations where requirements are changing rapidly by focusing on collaboration and communication. XP can help ensure that everyone involved in the project has a shared understanding of the goals and progress of the project.
- XP encourages frequent communication between our team members, which can help to ensure that everyone is on the same page and working toward the same goals.
- XP's emphasis on testing and feedback can help to identify and address issues early in the development process, reducing the risk of costly errors and delays later on.
- XP encourages pair programming and collective code ownership, which can help to improve teamwork and collaboration among our team members.

Overall, eXtreme Programming can help our team to deliver high-quality software more quickly, with greater flexibility, improved communication, and reduced risk.

Drawbacks

While eXtreme Programming has many benefits, it also has some drawbacks, including:

- XP is focused on the code rather than on design. That may be a problem because good design is extremely important for software applications. It helps sell them in the software market. Additionally, in XP projects the defect documentation is not always good. Lack of defect documentation may lead to the occurrence of similar bugs in the future.
- XP does not measure code quality assurance. It may cause defects in the initial code.

- XP’s emphasis on collaboration and quick iterations requires team members who are experienced in their roles and able to work effectively together. However, in our group most members were inexperienced and unfamiliar with our project, they struggled to keep up with the pace and level of collaboration required by XP.
- XP places little emphasis on documentation, which can make it challenging to maintain or modify the code in the future. This can be particularly problematic for projects with long lifetimes or those that require extensive maintenance.
- XP is not the best option if programmers are separated geographically.

Overall, eXtreme Programming can be an effective methodology for software development in the right circumstances, but it may not be suitable for some special projects or teams. Its emphasis on collaboration, communication, and quick iterations requires experienced team members.

Lessons learned

After working on this XP project with our group, we have learnt the following lessons:

- XP encourages us to work in small increments, to get fast feedback on the work, and to continuously improve the code.
- Pair programming is very important for XP, where two developers work together on the same code. This can help us to share knowledge, improve code quality, and reduce the risk of errors.
- XP emphasizes the importance of collaboration and communication between our team members. This can help to ensure that everyone is on the same page and working toward the same goals.
- Testing and test-driven development is very important for XP, to ensure that the code is reliable, robust, and meets the requirements.
- XP emphasizes the importance of continuous feedback and improvement. By regularly reflecting on the team's progress and adjusting the development process as needed, our team can continuously improve the quality and efficiency of work.
- XP encourages rapid prototyping and iterative development, which can help us to quickly identify and address issues with the design and functionality of the project.

Overall, XP has provided several valuable lessons and insights into software development, and has helped teams to deliver high-quality software quickly and efficiently.

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?

We had devised a plan at the start of the project. During the initial times, we followed what we had scheduled. Later, this sheet was turned into a constantly updating document and according to the progress and the features we had already implemented, we recalculated what needed to be done and what was already on track. We changed the schedule every few weeks according to the state of the project and the priorities of the features and also keeping the errors and the difficulties we faced in mind.

Benefits

Having a project plan offers several benefits, including:

- A clear direction: The objectives, goals, schedules, and tasks necessary to execute a project are described in a project plan. Everyone involved in the project, including team members get understanding and direction from it.
- Resource allocation: Effective resource allocation is made possible by a project plan. It lets team members predict the resources—such as time, money, and people necessary for each activity and make appropriate plans.
- Communication: A project plan offers a framework and common language for team members to communicate. It guarantees that everybody is on the same page and pursuing the same objectives.
- Accountability: Each team member’s tasks and responsibilities are described in a project plan, encouraging responsibility and making sure that everyone is aware of what is expected of them.
- Time management: The project remains on track and deadlines could be reached according to the timelines and schedules provided in the project plan.

Drawbacks

While project plans offer many benefits, there are also some potential drawbacks, including:

- Rigidity: Sometimes a project plan is overly stiff, which limits its ability to be flexible or adjust to changing conditions. This could be a problem if unanticipated problems or opportunities show up, requiring a reevaluation and adjustment of the project.
- Overcomplication: Sometimes a project plan might become unnecessarily complicated, which makes it challenging to comprehend or implement. This may lead to uncertainty, delays, and errors.
- Unrealistic expectations: An inaccurate expectation of what can be accomplished within a specified schedule or budget may occasionally result from a project strategy. If the project does not live up to these expectations, this may cause disappointment and irritation.
- Limited creativity: A project plan can occasionally block innovation and originality as team members concentrate on finishing tasks inside a predetermined framework rather than considering fresh concepts or methods.

Lessons learned

We learned a lot of valuable lessons regarding the planning and scheduling of the project. Planning is an ongoing process, and it is crucial to regularly revisit and adjust the plan as necessary to ensure the project's success. As a team, we referred to the document every few weeks to check our progress and reassess the schedule depending on what we had already implemented. It was very helpful once we started readjusting the schedule and keeping it flexible without falling into the rigidity trap.

Planning the project in great detail works out well as long as we allow ourselves to make changes in the plan according to the current situations and use the plan only as a guideline and not stick to it very rigidly.

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?

Since our VR project is heavily dependent on the visual experience of the customer, with each integration we made, we made sure to test it with our classmates outside our group and strand and got valuable insights such as the VR headsets making them nauseous and the UI being not large enough. To counter such issues we modified our application accordingly.
Instead of using hand movements to do UI and asset interaction we made all interaction controller based to reduce movements. We also increased the size of the UI and added modifications to our kinematics code controlling movement specifications regarding position, speed and rotation of character to each reaction by a user using Quaternions. These changes were found to be acceptable by our peers in terms of countering the issue of feeling nauseated.

To automate the testing process we had tests written for the basic and integrated components of our application using jest and enzyme for ReactJS and using Nunit and Unity Test Runner for Unity the VR part and these tests were designed in such a way that they would break incase of any coupling between our different modules or incase of integration issues and point us to the source of the issue.

Benefits

Defining automated acceptance tests for each desired feature has several benefits:

- Ensures that the feature works as intended: By defining automated acceptance tests, customers can verify that the feature is working as intended and meets their requirements. This helps to catch any bugs or issues early on in the development process.
- Improves the quality of the product: Automated acceptance tests help to improve the overall quality of the product by ensuring that each feature works correctly and meets the customers' needs. This reduces the risk of defects and customer complaints.
- Saves time and resources: Automated acceptance tests can be run automatically and repeatedly, saving time and resources compared to manual testing. This helps to speed up the development process and reduces the cost of testing.
- Increases customer satisfaction: By involving customers in the testing process and ensuring that each feature meets their requirements, it increases their satisfaction with the product. This can lead to positive word-of-mouth and increased customer loyalty.
- Facilitates collaboration: By defining automated acceptance tests, customers and developers can work together to ensure that the product meets the cus

tomers' needs. This facilitates collaboration and communication between the different stakeholders involved in the development process.

Drawbacks

Defining automated acceptance tests for each desired feature can have some drawbacks:

- Requires additional time and resources: Creating automated acceptance tests can be time-consuming and require additional resources such as specialized software, hardware, or personnel. This can increase the cost and time required for development.
- May lead to narrow testing: Customers may only test for the features they are interested in or that are most important to them, leading to a narrow focus on testing. This may result in other features being overlooked, and bugs or issues may go unnoticed.
- Can create additional complexity: Defining and maintaining automated acceptance tests can add complexity to the development process, which can lead to more errors and issues. It can also be challenging to keep the tests up-to-date with changes to the codebase.
- May not cover all edge cases: Automated acceptance tests are designed to test for specific scenarios, but they may not cover all edge cases or unusual scenarios that can occur in real-world usage. This can result in issues going undetected until they are discovered by users.
- May not align with developer priorities: Customers may prioritize different features and aspects of the product than developers. This can lead to a mismatch between the tests defined by the customers and the testing priorities of the development team.

Lessons learned

Defining automated acceptance tests for each desired feature has provided several important lessons:

- Customer involvement is crucial: Involving customers in the testing process and allowing them to define acceptance tests for each desired feature helps ensure that the product meets their needs and expectations. It also increases customer satisfaction and can lead to increased loyalty.
- Testing is essential for quality: Automated acceptance tests are critical for ensuring that each feature works correctly and meets the customers' needs. They help to catch bugs and issues early on in the development process and reduce the risk of defects and customer complaints.

- Collaboration is key: Defining automated acceptance tests requires collaboration and communication between different stakeholders, including customers and developers. This facilitates a better understanding of customer requirements and priorities and can help to align the testing priorities of the development team with the needs of the customers.
- Automation can save time and resources: Automated acceptance tests can be run automatically and repeatedly, saving time and resources compared to manual testing. This helps to speed up the development process and reduces the cost of testing.
- Test coverage is important: Automated acceptance tests are designed to test specific scenarios, but it is essential to ensure that they cover all critical scenarios and edge cases. This helps to ensure that issues are caught early on in the development process and reduces the risk of defects in the final product.

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?

To achieve the functionalities requirement of the system, we conducted a careful analysis of the system's requirements, use cases and user feedback. We made UML diagrams to get a better understanding of our project’s requirements. Once the design was in place, we tested it to see if it was meeting the ends and doing the modifications as required. We checked if our code is free of duplication by modularization, encapsulation and abstraction. We broke the system into more manageable modules, each with a well-defined function and interface. To save time, improve code quality and reduce maintenance costs, we used existing design patterns, frameworks and libraries to build the system.

Benefits

The benefits of keeping a design exactly suited for the current functionality of the system, passing all tests, containing no duplication, expressing everything the authors want expressed, and containing as little code as possible include:

- Improved maintainability: By minimizing the amount of code and removing duplication, it becomes easier to maintain the system over time. Fewer lines of code means fewer potential bugs and issues to deal with, and easier navigation for future developers.

- Improved efficiency: When the design is suited exactly for the current functionality of the system, there is no unnecessary code, making the system more efficient and faster to execute.
- Reduced development time: By focusing on only what is needed for the current functionality, the team can avoid wasting time on features that may not be needed or used.
- Easier to understand: When code expresses everything the authors want expressed, it becomes easier for other developers to understand the code and make changes if needed.
- Increased testability: A system that passes all tests is more reliable and easier to test for future changes and updates.

Overall, keeping a design suited for the current functionality of the system and containing as little code as possible can lead to a more maintainable, efficient, and reliable system that is easier to understand and update in the future.

Drawbacks

While designing a system that is tailored to its current functionality and passing all tests may seem like a good approach, it can have some potential drawbacks:

- Limited scalability: By focusing solely on the current functionality, the design may not be flexible enough to accommodate future changes or additions to the system. This may result in the need for significant rework or a complete overhaul of the system in the future.
- Lack of adaptability: The design may not be able to handle unforeseen scenarios or edge cases that were not accounted for during the initial design phase.
- Reduced readability: By striving to minimize the amount of code, the design may become more complex and difficult to read, making it harder for other team members to understand and maintain.
- Increased risk of errors: While a design that passes all tests may seem like it is error-free, there may be cases where the tests do not cover all possible scenarios, leaving room for potential errors or bugs in the system.
- Difficulty in collaboration: By having a design that is tailored to the current functionality, it may be more difficult for other team members to contribute to the design or make changes, leading to a lack of collaboration and potentially suboptimal outcomes.

Lessons learned

- Design for current functionality: When designing a system, it’s important to focus on meeting the current requirements and functionality, rather than

trying to anticipate future needs. This approach can help to ensure that the design is not overly complex or bloated, and that it's optimized for the current use case.
- Testing is crucial: It's essential to thoroughly test the system to ensure that it meets all requirements and functions as intended. This can help to identify and address any issues or bugs before they become bigger problems.
- Avoid duplication: Duplicating code can lead to unnecessary complexity, increased maintenance costs, and potential errors. By minimizing duplication, the design can be made more efficient and easier to maintain.
- Express ideas clearly: It's important to express everything that the authors want to communicate clearly and concisely. This can help to ensure that the design is easily understood and implemented by other team members.
- Strive for minimalism: By aiming to use as little code as possible, the design can be made more efficient and easier to maintain. This approach can also help to reduce the risk of errors and bugs in the code.

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?

Initially, in the plan we assigned the pairs of programmers for each week along with the task they will be working on. This document was later updated as the weeks went by. Each week, we swapped the pairs. One person working on the component stayed, while another programmer from another component joined them so that they understood the current progress and could join in on an ongoing task with their own inputs and the time was used productively, making progress on that component.

Benefits

Pair programming is a practice in software development where two programmers work together on the same code, often sitting at the same computer. The benefits of pair programming include:

- Improved code quality: Due to the fact that two people are looking at and discussing the code as it is being written, pair programming can result in higher quality code. Less bugs and better-designed solutions may result from this.

- Increased knowledge sharing: Pair programming encourages team members to share expertise. This can aid the team as a whole by dispersing knowledge and experience and enhancing team capabilities.

- Improved team communication: Due to the continual communication that pair programming requires, team members are better able to work together and comprehend one another's perspectives.

- Increased productivity: Even though it can seem ironic, pair programming may increase productivity. When two people collaborate, projects can typically be completed more quickly than when one person works alone, and the resulting code is frequently of greater quality.

- Improved morale: By encouraging companionship and teamwork, pair programming can raise team morale. Having a close working relationship with another team member can also help to lessen burden and stress.

Drawbacks

While pair programming has many benefits, it also has some potential drawbacks, including:

- Personality conflicts: Due to the tight cooperation and communication required for pair programming, personality conflicts between team members may arise. It can be challenging to operate effectively as a team if team members have diverse work styles, communication preferences, or personalities.

- Unequal contribution: When working as a pair of developers, one developer could contribute more than the other. If one team member believes that they are doing more work or bearing more of the weight, this can result in irritation and resentment.

- Reduced autonomy: Because pair programming necessitates close collaboration between developers, it can limit their freedom and ability to make independent decisions. Some programmers might feel they have less control over their work or less freedom to try out new ideas.

- Fatigue: For prolonged lengths of time, working in pairs can be mentally and physically draining. Developers may become fatigued, lose focus, or work less efficiently.

Lessons learned

Though a little tough to get used to in the start, pair programming can be an effective way to improve code quality, knowledge sharing, team communication, learning, productivity, and morale.

It helped us get to know all the components of the projects. It allowed us to work on parts of the project that were not under our skill set and in our comfort zone and learn new skills, since we were working with a team member who already knew about it.

It helped us understand all the code and held everyone accountable for every piece of code. The collective ownership of code was beneficial for all of us to work as a team and solve any issues that we come across without blaming anyone else in the team.

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?

In our team, Test-Driven Development (TDD) is a core practice that we apply to our software development process, including our React web project and Unity project. We work in short development cycles where we first write a failing test before writing any production code. This helps us to ensure that our code is thoroughly tested and meets the required specifications.

We use a testing framework such as Jest and enzyme to write our tests and ensure that our React components and other parts of the application function as expected. Jest is a popular JavaScript testing framework that we use in our React web project. It provides a simple and intuitive API for writing tests, including support for snapshot testing, mocking, and coverage reports. Enzyme is a JavaScript testing utility for React that provides a set of helper functions for testing React components. It allows us to simulate user interactions, such as clicks and input events, and inspect the output of a component. Enzyme works well with Jest, and we often use them together to write comprehensive and reliable test suites for our React components.

In Unity we used the Unity Test Runner and Nunit package to write tests for the application.
The basis of Unity Test Runner is assemblies, they help organize code, reduce compilation time and create dependencies. The assemblies in unity help attain cohesion of code. These are executable dlls, and we have created one assembly for each group of code that should work independent from other pieces of code, they will form compilation errors if unnecessary dependency is formed avoiding coupling.

To test a React component using Jest and Enzyme, we typically follow a three-step process:

1. Set up the test environment: We first configure Jest and Enzyme to work together in our test files. We import Enzyme and set up a test renderer to render our React component. We also set up any necessary mock data or external dependencies that the component may need.

2. Write the test case: We then write a test case that describes the behavior of the component. We use Enzyme’s helper functions to simulate user interactions and verify that the component renders the correct output. For example, we might test that a button component displays the correct label when clicked, or that an input field correctly updates its value when the user types into it.

3. Run the test: Finally, we run the test using Jest’s test runner. Jest will execute the test case and report any failures or errors. We can also generate coverage reports to see how much of our code is covered by our tests.

By following this process, we can ensure that our React components are thoroughly tested and function as expected. We can catch bugs and issues early on in the development process, which saves time and effort in the long run. Overall, TDD has been a valuable practice in our team, and it has helped us to develop high-quality software efficiently.

Benefits

- Better code quality: TDD encourages developers to write modular and testable code, which can lead to fewer bugs and more maintainable code.

- Faster feedback: TDD provides developers with immediate feedback on their code changes, allowing them to catch issues early on in the development process.

- Increased confidence: TDD helps developers to feel more confident in their code changes, as they have a comprehensive test suite to ensure that their code is working as expected.

- Collaboration: TDD can promote collaboration among team members, as they can share their tests and ensure that their code is consistent across the project.

Drawbacks

- Time-consuming: TDD can be time-consuming, as developers need to write tests in addition to their production code. This can slow down the development process, particularly for large or complex projects.
- Learning curve: TDD requires a shift in mindset and can take some time for developers to learn and apply effectively.
- Over-reliance on tests: TDD can lead to over-reliance on tests, with developers focusing too much on passing tests rather than ensuring that their code is meeting the business requirements.

Lessons learned

Our team was working on a new feature of renting a book and a seat for a web application that required some complex business logic. We initially wrote a set of tests that covered all the different scenarios we could think of. However, we quickly realized that our tests were too complex and difficult to maintain.

We then decided to take a step back and revisit our approach to writing tests. Through some research and discussions, we learned about the "Three A's" of good tests: Accurate, Automated, and Actionable. We also learned about the importance of writing tests that are focused on one thing at a time, and that use clear and descriptive names.

We applied these lessons to our test suite by refactoring our existing tests and writing new ones. We made sure that each test was focused on a single scenario, and that the test name clearly described what was being tested. We also made sure that our tests were accurate, by using realistic data and covering edge cases.

As a result of these changes, our test suite became more maintainable and easier to understand. We were able to catch bugs more quickly, and our tests gave us more confidence in our code changes. Overall, we learned that writing good tests is a key part of the TDD process, and that taking the time to write focused and accurate tests can save time and effort in the long run.

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?

To implement design improvement during eXtreme Programming, the following strategies were applied during our development.

- We defined coding standards for the team to follow before we start the project to ensure consistency and readability of the code, including standards for naming conventions, code formatting, and code organization.
- We used refactoring techniques to improve the design of the code while maintaining the same functionality, including simplifying code, removing duplicated code, improving code organization, and reducing complexity.
- We conducted code reviews with our team members to ensure that the code is well-designed, maintainable, and meets coding standards. This can also help to identify areas for further improvement.
- We used pair programming during our development. It's particularly effective for improving code design, as it allows our team members to discuss design decisions and identify areas for improvement.
- We used continuous integration techniques to build and test the code frequently to ensure that it is of high quality and free from errors. This can help to identify design issues early in the development process.
- We identified areas in the codebase that could benefit from improved design, including code that is difficult to maintain or modify, inefficient or slow, or overly complex or difficult to understand.

By following these steps, our team can effectively implement code design improvement during XP, and continuously improve the quality and maintainability of the codebase.

Benefits

There are several benefits of design improvement during XP in our project including the following features.

- Design improvement during XP could lead to a higher quality codebase that is easier to maintain, modify, and extend. Also reduce the likelihood of bugs and errors and lead to a more stable and reliable software system.
- Design improvement during XP could help to reduce complexity and make the codebase easier to understand and work with and improve developer productivity and reduce the time required to implement new features or fix bugs.

- Design improvement during XP could make our code easier to maintain the codebase over time, reducing the risk of technical debt and making it easier to make changes to the codebase as needed.
- Design improvement during XP could improve collaboration among team members by making the codebase more understandable and easier to work with, and improve communication and teamwork among team members.
- Design improvement during XP could make our project easier to implement new features and fix bugs, reducing the time and effort required to complete development tasks, and improve the efficiency of the development process and allow our team to deliver high-quality software more quickly.

Overall, code design improvement during XP can lead to a higher quality, more maintainable, and more efficient codebase, with improved collaboration among our team members.

Drawbacks

While there are many benefits of design improvement during XP, there are also some potential drawbacks, including:

- Improving the design of the code can be a time-consuming process, especially for large codebases, might lead to delays in project timelines, and may require additional resources to complete.
- Code design improvement may require significant changes to the codebase, which can be disruptive to the development process, might require additional testing and debugging, and can slow down the development process.
- Improving the design of the code can also increase the complexity of the codebase, especially when our team members are not experienced in design patterns and principles, could make the code harder to understand and work with the codebase, and might lead to additional technical debt.
- Improving the design of the code can introduce new bugs and issues, especially if the changes are not thoroughly tested, and could result in regressions and require additional time and effort to fix.

Overall, while design improvement can lead to many benefits, it is important to carefully consider the potential drawbacks and plan for them accordingly, involving balancing the benefits of design improvement against the costs and risks of disrupting the development process or introducing new issues into the codebase.

Lessons learned

After implement design improvement during XP in our project, we have learnt the following lessons:

- Improving the design of a large codebase can be overwhelming, so it's important to start small and focus on one area at a time, which can help us to build momentum and make it easier to see progress.
- Code design improvement should involve the whole team, not just one or two developers to ensure that everyone understands the changes being made and can provide input and feedback.
- When improving the design of the code, it's important to focus on maintainability, rather than just adding new features, which can help us to reduce technical debt and make it easier to add new features in the future.
- Automated tools can help to identify areas of the code that need improvement, such as duplicated code or code that is difficult to understand, which can save time and ensure that improvements are made consistently across the codebase.
- Design improvement should be an ongoing process, rather than a one-time event. Continuously review and iterate on the codebase, and involve the whole team in the process, to ensure that the code remains maintainable and efficient over time.
- It's important to keep the final goal in mind when making code design improvements. This may involve balancing the benefits of design improvements against the costs and risks of disrupting the development process or introducing new issues into the codebase.

Overall, design improvement is an important aspect of software development, and can lead to many benefits. By starting small, involving the whole team, focusing on maintainability, using automated tools, continuously reviewing and iterating, and keeping the goal in mind, can make significant improvements to our codebase over time.

8. Continuous Integration

“The team keeps the system fully integrated at all times”

How was this applied in your team?

In our team, we completed continuous integration by uploading code updates for front-end, back-end, and Unity components to our GitHub repository on a weekly basis. During our weekly meetings, we would discuss new features introduced in the updated code, and collaboratively resolve any conflicts or issues that arose.

Benefits

Implementing continuous integration in our team has provided numerous advantages. It has allowed us to identify and resolve issues early, reducing the likelihood of accumulating significant technical debt. The regular updates also promote better collaboration and communication among team members, as everyone stays informed about the latest changes. Additionally, this practice encourages the development of modular, adaptable code, which can improve the overall quality of our software.

Drawbacks

While continuous integration has been beneficial, there have been some challenges as well. The weekly update schedule can create pressure on team members to complete tasks in a limited time frame, potentially affecting work quality. Additionally, frequent code merges can lead to an increased risk of conflicts or integration issues, requiring extra effort from the team to resolve. In some cases, team members may have different levels of familiarity with certain aspects of the codebase, which can lead to misunderstandings or difficulties during the integration process.

Lessons learned

From our experience with continuous integration, we have learned several valuable lessons. First, it is essential to maintain clear communication channels within the team to avoid misunderstandings and ensure everyone stays informed of code changes. Second, we discovered the importance of implementing automated testing to detect and resolve issues quickly. Third, allocating time for regular code reviews can help improve code quality and foster a shared understanding of the codebase. Finally, we learned that providing support and training for team members with varying skill levels can create a more inclusive and productive work environment.

9. Collective Code Ownership

“Any pair of programmers can improve any code at any time”

How was this applied in your team?

To implement collective code ownership during eXtreme Programming, the following strategies were applied during our development.

- We have weekly team meetings to help ensure that everyone is on the same page and provide an opportunity for team members to discuss code changes, ask questions, and address concerns.
- We ask for code reviews each time we upload the code through Github, Any pair programmers need to request for review to the group members who were responsible for this part before.
- We establish clear ownership boundaries to avoid confusion and ensure accountability.

Benefits

- Increased collaboration: CCO encourages developers to work together to solve problems and make improvements, leading to a more collaborative work environment.
- Higher code quality: Since all team members have equal ownership, they are more likely to take pride in the codebase and work to improve its quality.
- Better knowledge sharing: CCO promotes the sharing of knowledge and skills across the team, reducing the risk of knowledge silos and creating a more well-rounded team.
- Faster development: With everyone contributing, the development process can be more efficient and faster.
- Reduced risk of bottlenecks: CCO reduces the risk of bottlenecks caused by one person being the sole owner of critical code areas.

Drawbacks

- Lack of accountability: Without clear ownership, it can be difficult to hold individuals accountable for their work and ensure they are meeting deadlines.
- Potential for conflicts: If team members have different ideas about how to approach a particular problem, it can lead to conflicts that need to be resolved.

- Lower motivation: With everyone responsible for the codebase, some team mem bers may feel less motivated to contribute, assuming that someone else will take care of it.
- Increased overhead: CCO requires more coordination and communication to ens ure that everyone is on the same page, which can lead to increased overhead.
- Difficult to manage: Managing CCO can be challenging, especially in larger t eams, since it requires a high level of coordination and communication to b e effective.

Lessons learned

After implement collective code ownership during XP in our project, we have learnt the following lessons:

- In weekly meetings, it is necessary to summarize the progress of the previo us week, so that personnel can quickly understand the work content after ch anging.
- Before uploading the code to github, a self-test is required, and the uploa d can only be completed after the self-test is completed. When merging bran ches on Github, other team members should be requested to code reviews

10. Coding Standard

“All the code in the systems looks as if it was written by a single – very compe tent – individual”

How was this applied in your team?

In our team, we have a set of coding standards that we follow to ensure that all c ode in the system looks consistent and is easy to understand. These standards cove r a range of areas including naming conventions, code formatting, and documentatio n.

To apply these standards, we have a code review process in place by using GitHub w here each code change is reviewed by at least one other team member. During the re view, the reviewer checks that the code follows the coding standards and provides feedback if any issues are found.

We also use tools like linters and style checkers to enforce our coding standards automatically. For example, we use ESLint to ensure that our JavaScript code follows our coding standards, and Prettier to automatically format our code to a consistent style.

In addition to these tools, we also have regular team discussions and training sessions to ensure that everyone is aware of the coding standards and how to apply them effectively. This includes sharing examples of good and bad code, and discussing the reasoning behind certain coding standards.

Overall, our approach to applying coding standards is focused on ensuring that all code in the system is consistent, easy to understand, and maintainable. By following these standards, we are able to reduce the amount of time spent on code reviews and debugging, and ensure that our code is of high quality.

Benefits

- Consistency: Applying coding standards ensures that all code in the system looks consistent, which can make it easier to read and understand. This can be particularly important for large and complex projects, where consistency can help to reduce cognitive load and improve maintainability.

- Improved collaboration: By using consistent coding standards, team members can work more effectively together, as they will be able to understand each other's code more easily. This can help to reduce the amount of time spent on code reviews and debugging, and ensure that the team can work efficiently.

- Higher quality code: Coding standards can help to ensure that code is written in a clear and concise manner, reducing the potential for bugs and making it easier to maintain and modify in the future.

Drawbacks

- Learning curve: Applying coding standards can require a shift in mindset and may take some time for team members to learn and apply effectively. This can be particularly challenging for team members who are used to working in a less structured environment.

- Over-reliance on standards: It's important to ensure that coding standards are used as a tool to improve code quality, rather than a set of rules that must be followed without question. Over-reliance on standards can lead to rigid and inflexible code, which may not meet the specific needs of the project or organization.

- Time-consuming: Applying coding standards can be time-consuming, particularly in the early stages of a project where standards are being established. This can be a challenge for teams who are working under tight deadlines or with limited resources.

Lessons learned

Early on in a project, we tried to apply a very rigid set of coding standards to all code changes. We required that all code followed a specific formatting style, used specific naming conventions, and included detailed comments explaining each section of code.

While this approach did result in consistent and high-quality code, it also led to frustration and confusion among team members. Some team members found the standards to be overly prescriptive and time-consuming, and others felt that they were inhibiting creativity and innovation.

To address these concerns, we had a team discussion about the coding standards and how they were being applied. We realized that we needed to be more flexible and adaptable in our approach, and that we needed to apply the coding standards in a way that made sense for each specific project or task.

As a result, we revised our approach to coding standards to be more flexible and adaptable. We still had a set of standards that we followed, but we allowed for more variation in formatting and naming conventions, and we encouraged team members to use their own judgment when it came to commenting and documentation.

This approach allowed us to strike a better balance between consistency and flexibility, and it ultimately led to more satisfied team members and higher-quality code. We learned that it’s important to be flexible and adaptable when it comes to applying coding standards, and to apply them in a way that makes sense for each specific project or task.

11. Metaphor

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

How was this applied in your team?

Make the program's goals and purpose clear. Define the program's remit. Engage in brainstorming discussions. Meetings for brainstorming are a terrific method to collect opinions and ideas from everyone. Construct a program plan. It’s time to crea te a program plan when everyone has had a chance to voice their opinions. This plan should include a description of the program's goals, objectives, scope, budget, and resource requirements. Make certain that the creation of the plan was a collaborative effort. Describe the vision. It's crucial to share the vision with the entire team once the program strategy is in place.

Benefits

Developing a common vision of how the program works has several benefits:

- Increases understanding: When the team has a shared understanding of how the program works, it helps to ensure that everyone is on the same page. This can reduce confusion and misunderstandings, and improve communication between team members.
- Facilitates collaboration: A common vision of how the program works helps to facilitate collaboration between team members. It provides a shared framework for discussing ideas and making decisions, which can lead to more effective problem-solving and decision-making.
- Reduces errors and issues: When the team has a common vision of how the program works, it can help to identify potential issues and errors early on in the development process. This can reduce the risk of defects and improve the overall quality of the program.
- Improves productivity: When the team has a shared understanding of how the program works, it can help to streamline the development process. This can lead to increased productivity and efficiency, as team members can work more effectively together.
- Enhances customer satisfaction: A common vision of how the program works can help to ensure that the program meets the needs of the customers. This can lead to increased customer satisfaction and loyalty, as the program is more likely to meet their expectations and requirements.

Drawbacks

Developing a common vision of how the program works can have some drawbacks:

- Time-consuming: Developing a common vision of how the program works can be time-consuming, especially if team members have different perspectives or ideas. This can delay the development process and increase the time required to complete the project.
- Difficult to achieve consensus: Achieving a common vision of how the program works can be challenging, especially if team members have different backgrounds or perspectives. It may be difficult to achieve consensus on key des

ign decisions or features, which can result in delays or conflicts within the team.

- May lead to groupthink: Developing a common vision of how the program works can lead to groupthink, where team members conform to the majority view rather than expressing their own opinions. This can result in a lack of diversity of thought and potentially lead to suboptimal design decisions.
- May restrict creativity: A common vision of how the program works may restrict creativity and limit the exploration of new ideas. Team members may feel constrained by the shared vision and may be less likely to propose alternative approaches or solutions.
- May become outdated: A common vision of how the program works may become outdated as the project progresses. Changes in requirements or new insights may require revisions to the vision, which can be difficult to implement and communicate to the team.

Lessons learned

Developing a common vision of how the program works has several important lessons:

- Communication is key: Developing a common vision of how the program works requires effective communication between team members. It is important to establish clear lines of communication and ensure that all team members have the opportunity to share their ideas and perspectives.
- Collaboration is essential: Developing a common vision of how the program works requires collaboration between team members. It is important to encourage collaboration and create an environment where team members feel comfortable sharing their ideas and working together to achieve a shared vision.
- Flexibility is important: A common vision of how the program works should be flexible enough to accommodate changes in requirements or new insights. It is important to be open to revising the vision as the project progresses and to communicate any changes to the team.
- Consensus is not always necessary: Achieving consensus on every design decision or feature may not always be necessary. It is important to prioritize key design decisions and features and focus on achieving consensus on those.
- Documentation is crucial: It is important to document the common vision of how the program works to ensure that all team members have a clear understanding of it. This can help to reduce confusion and misunderstandings and ensure that the project stays on track.

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?

In our team, we have adopted a sustainable pace approach to software development. We recognize that the project is more like a marathon than a sprint, and we ensure that team members work at a comfortable, steady pace. This involves setting realistic deadlines, avoiding excessive overtime. Additionally, we emphasize the importance of regular breaks and time off to rejuvenate and maintain productivity.

Benefits

Adhering to a sustainable pace offers several advantages for our team. First, it helps prevent burnout, ensuring that team members remain motivated and engaged throughout the project. Second, it fosters a positive work environment where team members feel valued and supported, leading to higher job satisfaction and retention. Third, it contributes to more consistent productivity levels, as team members are less likely to experience fatigue or reduced performance due to overwork.

Drawbacks

Despite the benefits of a sustainable pace, there are also potential drawbacks. One challenge is that external pressures or tight deadlines may sometimes require faster progress, making it difficult to maintain a sustainable pace. Additionally, some team members may prefer a more intense, sprint-like work style, and may feel restrained by the slower pace. Lastly, achieving a sustainable pace may require additional time for project planning and coordination, which could delay the start of development work.

Lessons learned

From our experience with sustainable pace, we have learned several key lessons. First, it is crucial to establish clear expectations and guidelines for work hours and time off to ensure that everyone is on the same page. Second, we discovered that open communication and transparency about workload and potential challenges can help the team adjust and adapt more effectively. Third, we recognized the importance of empowering team members to set their own pace and voice concerns when they feel overwhelmed. Lastly, we learned that continually evaluating and adjusting our approach to sustainable pace allows us to maintain a positive and productive work environment in the long run.

13. Overall Project

Benefits

Through XP programming, the team's feedback and iteration efficiency have been improved. At the same time, XP programming emphasizes timely communication and negotiation among team members. XP programming can better meet customer needs because of its fast iteration and communication efficiency, and the ability to make modifications based on customer needs.

Drawbacks

XP programming contains limited documentation and relies on verbal communication, making it difficult to implement complex projects. Simultaneously relying on team communication. In addition, XP programming projects lack predictability and flexibility.

Having said that, XP might not be the ideal choice for all projects or teams. As well as a culture of constant growth and adaptability, it calls for a high level of discipline, teamwork, and trust. The effectiveness of any technique depends on how well it is applied and tailored to the unique requirements and circumstances of the project and team.

Lessons learned

Following our collaboration on this XP project, our group has discovered the following lessons:

Working in small steps, receiving quick feedback, and enhancing the code regularly are all encouraged by the XP methodology.

In the context of XP, when two developers collaborate on the same code, pair programming is crucial. This can facilitate knowledge transfer, enhance the quality of our code, and lower the possibility of mistakes.

XP places a strong emphasis on the value of cooperation and communication among our team members. This can make it easier to make sure that everyone is on the same page and pursuing the same objectives. For XP, testing and test-driven development are crucial to ensuring that the code is dependable, resilient, and compliant.