Extreme Programming

* Extreme Programming (XP) is a method or approach to software engineering and a precursor of several agile software development methodologies.
* Formulated by Kent Beck, Ward Cunningham, and Ron Jeffries. Kent Beck wrote the first book on the topic, Extreme programming explained: Embrace change, published in 1999.
* The second edition of the book, which appeared in 2005, delves more into the philosophy of Extreme Programming and describes it as being:
  + a mechanism for social change
  + a style of development
  + a path to improvement
  + an attempt to reconcile humanity and productivity
  + a software development discipline
* The twelve key features of XP, outlined below in Beck’s words, are:
  + **The Planning Game:** Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes, update the plan.
  + **Small Releases:** Put a simple system into production quickly, then release new versions on a very short cycle.
  + **System Metaphor:** Guide all development with a simple shared story of how the whole system works.
  + **Simple Design:** The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.
  + **Testing:** Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write user stories for desired features that the system must demonstrate to expose.
  + **Refactoring:** Programmers restructure the system without changing its behaviour to remove duplication, improve communication, simplify, or add flexibility.
  + **Pair Programming:** All production code is written with two programmers at one workstation.
  + **Collective Ownership:** Anyone can change code anywhere in the system at any time.
  + **Continuous Integration:** Integrate and build the system many times a day, every time a task is completed.
  + **Sustainable Pace:** Work no more than 40 hours a week as a rule. Never allow overtime for the second week in a row.
  + **On-site Customer:** Include a real, live customer on the team, available full-time to answer questions.
  + **Coding Standards:** Programmers write all code in accordance with rules emphasizing communication throughout the code.
  + These ideas are not new. They have been tried before and there have been many reports of failure. The point of XP is that, taken together, these techniques do constitute a workable methodology.

12 Key features of Extreme Programming

1. The Planning Game

* Description
  + - The plan tells you what to do during the next few days. It is not detailed, and it can be changed as required. It can also be changed by either the customer or developers, depending on the situation.
* Potential drawbacks
  + - A rough plan is not a sufficient basis for detailed development. Constantly updating the plan may be inefficient and may confuse customers and developers.
* Why it works in XP
  + - The overall plan is sufficient to give the clients a vision of what you can achieve.
    - At the start of every build, create a detailed plan for the build and update it as the build is created.
    - Small releases will reveal defects in the overall plan. A client who is working with the team can spot potential for changes and improvements.
* Project
  + - Make a plan for every build, assign tasks.
    - Meet regularly and update the plan according to the latest developments.
    - Make the updated plan available to everybody in the team.

2. Small Releases

* Description
  + - A release is a working version of the software. Between releases, the software may be in an inconsistent state. “Small” releases mean obtaining a working version every week, or every month, rather than every six months, or every year.
* Potential drawbacks 缺点
  + - Small releases mean that time is spent on getting the releases to work perfectly.
    - May not be necessary if the client does not need/want the intermediate builds to be delivered and used operationally.
* Why it works in XP
  + - Planning focuses attention on the most important parts of the system, so even small releases are useful to customers.
    - With continuous integration, assembling a release does not take much effort.
    - Frequent testing reduces the defect rate and release testing time.
    - The design is simple but may be elaborated later.
    - Any XP project starts with two programmers working on the first release. This pair generates enough work for a second pair to start, and so on.
* Project
  + - Each predefined build is a small release.

3. System Metaphor

* Description
  + - The system metaphor is a “story” about the system. It provides a framework for discussing the system and deciding whether features are appropriate. A well-known simple example of a metaphor is the Xerox “desktop” metaphor for user-interface design. Another is the “spreadsheet” metaphor for accounting. Games are their own metaphor: knowledge of the game helps to define the program.
* Potential drawbacks
  + - A metaphor may not have enough detail. It might be misleading or even become wrong if it is not updated.
* Why it works in XP
  + - Small releases provide quick feedback from real code to support the metaphor. Clients know the metaphor and can use it as a basis for discussion.
    - Frequent refactoring 重构 are made within the practical implications of the metaphor.
* Project
  + - The initial project description is the system metaphor.

4. Simple Design

* Description
  + - A simple design is an outline for a small portion of the software.
    - Has the smallest number of features that meet the requirements of current phase and does not incorporate solutions to the requirements of the upcoming phases.
    - Generally, overly complicated designs end up having unused features that become hindrance妨碍.
* Potential drawbacks
  + - A simple design may have faults and omissions. 太简单了，有些需要做的没做，有错误或遗漏
    - Implementing an overly simple design might bring unanticipated problems to light.
    - Components with simple designs might not integrate correctly into the system.
* Why it works in XP
  + - Refactoring 重构 allows you to correct design errors and omissions. 有补救措施：重构
    - The metaphor helps to keep the design process on track. 如果有全局想法，
    - Pair programming helps to avoid silly mistakes and to anticipate design problems.
* Project 在项目中如何做到这一点
  + - Every time a solution is proposed, it should be debated as to whether it is the simplest solution that can meet the required features. 每次都要找出最simple的解决方案
    - Overly complex designs should be avoided as a team principle. 全团队的人都要挑剔的去避免复杂的设计。Openminded

5. Testing

* Description
  + - Write large numbers of simple tests. Provide a fully automatic testing process.
    - Unit tests are automated tests that test the functionality of methods.
    - Unit tests are written before the eventual code is coded. This approach stimulates the programmer to think about conditions in which their code could fail. The programmer is finished with a certain piece of code when they cannot come up with any further condition on which the code may fail.
* Potential drawbacks
  + - Writing tests is time consuming. Time spent on testing must be justified. 花时间多
    - In larger projects, programmers don’t write tests — testing teams do. 大项目中，程序员不测试，而是由专门的测试部门负责
* Why it works in XP
  + - Simple design implies that the tests should be simple too. 设计越简单，测试越简单。如果发现测试很麻烦，就说明软件设计出了问题
    - With pair programming, one partner can think of tests while the other is coding. Seeing tests work is good for morale士气.
    - Clients like seeing tests working. 向顾客展示测试能通过
    - There are many tests and most of them are run automatically.
* Project
  + - Unit tests must be delivered with each build.

6. Refactoring 在软件能运行后，再仔细检查结构，找出如何能使其更完善。不同于debugging

* Description
  + - Refactoring is making changes to the software to achieve better design, conforming to coding standards. Refactoring is not debugging.
    - Because XP advocates 提倡 productivity. At times this may result in a system that becomes increasingly difficult to increment upon.
    - Symptoms: one functional change requires changes to multiple copies of the same (or similar) code; changes in one part of the code affect lots of other parts.
* Potential drawbacks
  + - Refactoring takes time, is hard to control, and is error-prone易出错的.
* Why it works in XP
  + - Collective ownership makes refactoring easier.
    - Coding standards reduce the task of refactoring.
    - Pair programming makes refactoring less risky and adventurous.
    - You have a set of tests that you can run at any time during the refactoring process.
    - Continuous integration gives rapid feedback about refactoring problems.
* Project
  + - After each build is delivered, have a meeting and decide what parts need to be cleaned up before development continues on the next build.

7. Pair Programming

* Description
  + - Pair Programming means that all code is produced by two people programming on one task on one workstation.
    - One programmer has control over the workstation and is thinking mostly about the coding in detail. 一个人负责细节
    - The other programmer is more focused on the big picture, continually reviewing the code that is being produced, as well as researching solutions. 一个人负责掌控全局
    - The pairs are not fixed: it's recommended that programmers try to mix as much as possible, so that everybody can become familiar with the whole system.
* Potential drawbacks
  + - Pair programming if not done properly may be inefficient. 如果不慎，会效率低下
* Why it works in XP
  + - Coding standards avoid trivial arguments. No one is over-tired or over-worked.
    - Simple design and writing tests together helps to avoid misunderstanding.
    - Both members of the pair are familiar with the metaphor.
    - If one partner knows a lot more than the other, the second person learns quickly.
* Project
  + - Work in pairs and make sure both individuals know their responsibilities.

8. Collective Ownership

* Description
  + - Anyone can make changes to any part of the system.
    - This contrasts with traditional processes, in which each piece of code is owned by an individual or a small team who has complete control over it and access to it.
    - Speeds up the development process, because if an error occurs in the code, any programmer may fix it.
* Potential drawbacks
  + - Can become problematic if change are applied without caution. Applying changes may result in other parts of the code to fail.
* Why it works in XP
  + - Continuous integration 持续整合 avoids breakdowns.
    - Continuously writing and running tests also avoids breakdowns.
    - Pair programmers are less likely to break code than individual programmers.
    - Coding standards avoid trivial arguments.
    - Knowing that other people are reading your code makes you work better.
    - Complex components are simplified as people understand them better.
* Project
  + - Setup a software repository and enforce that it is used as frequently as possible.

9. Continuous Integration

* Description
  + - The system is rebuilt very frequently, perhaps several times a day.
    - Not to be confused with short releases, in which a new version with new features is built and delivered.
    - The newly integrated system is compiled and tested.
* Potential drawbacks
  + - Each integration can be difficult if different programmers are going in different directions or changing existing code without consulting other programmers.
* Why it works in XP
  + - Tests are run automatically and quickly, so that errors introduced by integration are detected quickly.
    - Pair programming reduces the number of changed modules that have to be integrated.
    - Refactoring maintains good structure and reduces the chance of conflicts in integration.
    - Simple designs can be integrated quickly.
* Project
  + - Enforce that any code committed actually compiles, and that it passes all tests.

10. Sustainable Pace

* Description
  + - Many software companies require large amounts of overtime: programmers work late in the evening and during weekends.
    - They get over-tired, make silly mistakes, get irritable, and waste time in petty arguments.
    - This XP policy ensures that no one works too hard.
    - If a developer works overtime this week, they are not allowed to work overtime next week.
* Potential drawbacks
  + - 40 hours a week is often not enough to obtain the productivity required for competitive software development.
* Why it works in XP
  + - Good planning increases the value per hour of the work performed; there is less wasted time.
    - Planning and testing reduces the frequency of unexpected surprises.
    - XP as a whole helps you to work rapidly and efficiently.
* Project
  + - Distribute work evenly. Do not wait until the last few days to work day and night.

11. On-site Customer

* Description
  + - A representative of the client’s company works at the developer’s site all the time.
    - The client is available all the time to consult with developers and monitor the development of the software.
* Potential drawbacks
  + - The representative would be more valuable working at the client’s company.
* Why it works in XP
  + - Clients can contribute, e.g. by writing user stories and contributing/commenting on tests.
    - Rapid feedback for programmer questions is valuable.
* Project
  + - Discussions about the project during lectures. Contact the instructor for clarifications.

12. Coding Standards

* Description
  + - All code written must follow defined conventions for layout, variable names, file structure, documentation, etc.
    - Note that this is a local standard, not a standard defined by XP.
* Potential drawbacks
  + - Programmers can be individualists and refuse to be told how to write their code.
    - Can be overdone and thus be wasteful of time.
* Why it works in XP
  + - Coding standards lead to more understandable code, which is required for pair programming, continuous integration, testing, and productivity in general.
    - Refactoring can be used to enforce conformance to coding standards between builds.
* Project
  + - Use a predefined set of coding conventions
    - Use a documentation generation software (e.g. Javadoc)

简单讲一下，讲很快：

6 Values of Extreme Programming

1. Communication

* Building software systems requires:
  + communicating system requirements to the developers of the system,
  + communicating the software interfaces (APIs) to fellow developers.
* In formal software development methodologies, this task is accomplished through precise and standard documentation.
* Extreme programming techniques can be viewed as methods for efficiently building and disseminating institutional knowledge among members of a development team.
* The goal is to give all developers a shared view of the system which matches the view held by the users of the system.
* To this end, extreme programming favors simple design, common metaphors, collaboration of users and programmers, frequent verbal communication, and feedback.
* During coding, automated code documentation generation tools (e.g. Doxygen, Javadoc) can be used to facilitate communication between developers.

2. Simplicity

* Extreme programming encourages starting with the simplest solution. Extra functionality can then be added later.
* The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month.
* Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed, while perhaps delaying crucial features that are needed now.
* Often summed up as the "You aren't gonna need it" (YAGNI) approach.
* A simple design with very simple and neat code could be easily understood by most programmers in the team.
* When many simple steps are made, the customer and the developers have more control and feedback over the development process and the system that is being developed.

3. Feedback

* **Feedback from the system**: by writing unit tests, or running tests during continuous integration, the programmers have direct feedback from the state of the system after implementing new code or changes to existing code.
* **Feedback from the customer**: The functional tests are provided by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks during the delivery of each build so the customer can easily steer the development.
* **Feedback from the team**: When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.
* Feedback is closely related to communication and simplicity.
* Flaws in the system are easily communicated by writing a unit test that proves a certain piece of code will not misbehave in a specific case.
* The direct feedback from the system tells programmers to fix this part.
* A customer is able to test the system periodically according to the functional requirements, simplified as user stories.

4. Courage

* Several XP practices require courage.
* One is to always design and code for today and not for tomorrow. This is an effort to avoid getting bogged down in overly complicated design and concentrate on what is required now.
* Courage enables developers to feel comfortable with refactoring their code when necessary. This means reviewing the existing system and modifying it so that future changes can be implemented more easily.
* Continuous integration forces all individuals to confront their own code with the main body of code, which might uncover design flaws or omissions.
* Pair programming forces individuals to uncover their lack of knowledge or erroneous code to tier peers as they are working in pairs.
* Courage is required when code needs to be thrown away: courage to remove source code that is obsolete, no matter how much effort was used to write it.

5. Respect

* The respect value includes respect for others as well as self-respect.
* Programmers should never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers.
* Members respect their own work by always striving for high quality and seeking for the best design for the solution at hand through refactoring, and to follow coding standards.
* Adopting good values leads to respect gained from others in the team. Nobody on the team should feel unappreciated or ignored if they embrace the values common to the team.
* This ensures a high level of motivation and encourages loyalty toward the team and toward the goal of the project.
* This value is very dependent upon the other values, and is very much oriented toward people in a team.

6. Embracing change

* The principle of embracing change is about not working against changes but embracing them.
* For instance, if at one of the iterative meetings it appears that the customer's requirements have changed dramatically, programmers are to embrace this and plan the new requirements for the next iteration.
* If the design of the system shows significant flaws that are hampering further development, its change should be embraced through redesign and refactoring.
* When encountering unit testing failures or integration problems, one should see this as an opportunity to improve the system.