**XP - Extreme Programming é dinâmica e flexível, porém é necessária muita disciplina para usá-la em um projeto**. Para demonstrar isso, abaixo temos um conjunto sugerido de "boas práticas" em projetos usando XP.

**14 DICAS DE BOAS PRÁTICAS COM XP**

1. [The Customer is Always Available](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Available)
2. [Metaphor](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Metaphor)
3. [Planning Game](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Planning)
4. [Small Releases](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Releases)
5. [Acceptance Tests](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Acceptance)
6. [Test First Design](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Design)
7. [Continuous Integration](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Integration)
8. [Simple Design](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Simple)
9. [Refactoring](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Refactoring)
10. [Pair Programming](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Programming)
11. [Move People Around](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Move)
12. [Collective Code Ownership](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Collective)
13. [Coding Standards](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Coding)
14. [40 Hour Week](https://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#Week)

**The Customer is Always Available (O cliente sempre disponível)**

Constante disponibilidade do cliente para colaborar em dúvidas, alterações, e prioridades em um escopo, ou seja, dando um dinamismo ativo ao projeto.

**Metaphor (Uso de metáforas no projeto)**

Visando facilitar a comunicação da equipe, caso seja possível, é estabelecido o uso de metáforas em pontos chaves ao projeto como, por exemplo, a definição de um nome que seja comum à equipe e simbolize algo de fácil assimilação como, por exemplo: "Vamos chamar nosso projeto de "cartão de ponto", para um sistema que gerencie as batidas de ponto de funcionários, gerando o provisionamento financeiro e mensal para módulo de folha de pagamento".

**Planning Game (Planejando o jogo)**

Entre o cliente e os técnicos são estimuladas reuniões usando quadros brancos, com o objetivo de captar e definir as "user stories" (estórias, que são textos claros ou diagramas com notação UML com as especificações de regras de negócios inerentes ao sistema) e também para poder estimar o tempo ideal das interações, o projeto como um todo, elaborar estratégias e tentar prever as contingências para projeto.

Essa prática é fundamental para elaborar a estratégia das interações, que é a forma como se trabalha o "cronograma" de um projeto com XP, onde basicamente define-se um tempo padrão para as interações e especifica-se quais e quantas estórias podem ser implementadas em uma interação. Exemplo:

Digamos que seja definido o tempo padrão de 2 semanas para cada interação e que temos 60 estórias a serem implementadas. Em seguida, iremos analisar os requisitos (estórias) e priorizá-las junto ao cliente. Após esse processo, definimos que iremos implementar 4 estórias por interação, fazendo com que as 60 estórias sejam implementadas em 15 interações (60/4), chegando a um total estimado de 30 semanas (15\*2) para que se implemente todas as estórias.

Claro que esse exemplo é bem genérico, pois nem sempre é possível estabelecer uma organização tão exata na implementação das estórias, até mesmo porque existe uma variação na necessidade de esforço que cada estória exige para ser implementada. É comum que uma única estória necessite de mais uma interação ou que existam estórias tão pequenas que devam ser agrupas em uma só interação.

**Small Releases (Pequenas versões)**

Conforme as interações são concluídas, o cliente recebe pequenas versões/releases do sistema, visando com que seja colocado em prática e validado aquilo que está sendo implementado. Isto também permite que mais cedo possam ser detectadas necessidades de alterações de requisitos no software.

**Acceptance Tests (Testes de Aceitação)**

São definidos pelo usuário na fase inicial do projeto e são os critérios de aceitação do software conforme a estratégia de entrega e representa exatamente a métrica de aderência do software desenvolvido/implantado ao universo do cliente.

**Test First Design (Primeiro os testes)**

Aplicados a partir de testes unitários do código produzido, além de serem preparados utilizando os critérios de aceitação definidos previamente pelo cliente. Garante também a redução de erros de programação e aumenta a fidelidade do código produzido ao padrão estabelecido para o projeto. Através da prática de testes unitários, definimos antes da codificação os testes dos métodos críticos do software ou métodos simples que podem apresentar alguma exceção de processamento.

**Continuous Integration (Integração Contínua)**

Os diversos módulos do software são integrados diversas vezes por dia e todos os testes unitários são executados. O código não passa até obter sucesso em 100% dos testes unitários, facilitando, dessa forma, o trabalho de implementação da solução.

**Simple Design (Simplicidade de Projeto)**

O código está, a qualquer momento, na forma mais simples e mais clara, conforme os padrões definidos pela equipe de desenvolvimento, facilitando a compreensão e possível continuidade por qualquer um de seus membros.

**Refactoring (Refatoração - melhoria constante do código)**

A cada nova funcionalidade adicionada, é trabalhado o design do código até ficar na sua forma mais simples, mesmo que isso implique em "mexer" em um código que esteja em funcionamento. Claro que a prática de refatoração nem sempre é aceita, pois envolve questões como prazo e custo. Além disso, e essa prática em si pode ser minimizada caso o projeto esteja usando 100% de orientação a objeto, onde podemos criar códigos os mais genéricos e reutilizáveis possíveis, diminuindo o trabalho em caso de uma possível refatoração.

**Pair Programming (Programação em dupla)**

Todo código de produção é desenvolvido por duas pessoas trabalhando com o mesmo teclado, o mesmo mouse e o mesmo monitor, somando forças para a implementação do código. À primeira vista pode parecer loucura, pois se imagina estar gastando dois recursos humanos ao mesmo tempo para fazer a mesma tarefa e sem possibilidade de avanço substancial no projeto. Mas na verdade, essa prática tem pontos positivos como:

* Compartilhamento de conhecimento sobre das regras de negócio do projeto por todos da equipe de desenvolvimento;
* Fortalece a prática de Propriedade Coletiva do Código;
* Nivelação de conhecimento técnico dos programadores;

Elevação dos níveis de atenção ao código produzido, pois um “supervisiona” e orienta o trabalho do outro. Dessa forma, minimiza-se a possibilidade de erros no código, erros de lógica e produção de um código fora dos padrões estabelecidos pela equipe.

**Move People Around (Rodízio de pessoas)**

As duplas de programação são revezadas periodicamente, com o objetivo de uniformizar os códigos produzidos, deixar todos os módulos do sistema com mesmo padrão de código/pensamento e compartilhar o código com todos da equipe.

**Collective Code Ownership (Propriedade coletiva - O código é de todos da equipe)**

Uma vez aplicados a Programação em Dupla e o Rodízio de Pessoas, a equipe como um todo é responsável por cada arquivo de código. Não é preciso pedir autorização para alterar qualquer arquivo, mantendo claro, um padrão prático de comunicação da equipe.

**Coding Standards (Padronização do código)**

Todo código é desenvolvido seguindo um padrão, qualquer que seja, mas toda equipe deve seguir o mesmo padrão. Dessa forma, todos da equipe terão a mesma visão do código.

**40 Hour Week (Otimizando as jornadas de trabalho)**

Trabalhar por longos períodos é contraproducente. Portanto, sempre que possível, deve-se evitar a sobrecarga de trabalho de todos da equipe, criando condições favoráveis ao uso da carga normal de trabalho. É necessário deixar a equipe livre para relaxar, brincar, ou fazer o que bem entender para equilibrar o trabalho mental e físico. Existe até uma frase que diz: trabalhe a 100% durante as 40 horas e descanse a 100% no resto. Se algum deles não for feito com 100%, um afetará o outro.

**CONCLUSÕES**

Como disse no início do artigo, essas são sugestões de boas práticas, pois XP não é uma metodologia estática, ela é dinâmica, dando liberdade inclusive para cada um modelar sua própria forma de trabalho com XP. É comum que não se consiga aplicar todas essas práticas em um projeto e sim que se faça uma espécie de "mix" de práticas XP, com práticas pessoais mais intuitivas e geradas a partir de experiências anteriores em projetos, ou mesmo oriundas de práticas de outras metodologias como RUP, CMM, PMI , etc.

Hoje, XP está sendo aplicada em larga escala em vários projetos no mundo todo, porém ainda temos muito a evoluir em sua compreensão e aplicação. Nota-se isso principalmente em pontos polêmicos como testes unitários, programação em dupla, rodízio de pessoas, propriedade coletiva do código e otimização de jornadas, que são práticas que se mal utilizadas podem realmente trazer aumentos no custo e no prazo de projetos. Ou seja, é de extrema importância que se entenda bem a essência de XP e principalmente que se tenha disciplina e criatividade, duas qualidades básicas em quem pretende usá-la em projetos.

Somente a partir de uma visão criativa sobre a metodologia e uma disciplina equilibrada para cumpri-la , é que todos poderão usar e ter **benefícios através de Extreme Programming**. Caso alguém esteja interessado em conhecer melhor XP ou compartilhar informações sobre ela, acesse br.groups.yahoo.com/group/xpnorte/, onde nós, do grupo de usuários XPnorte, temos um espaço aberto a dicas, dúvidas e informações diversas sobre XP.

Vale lembrar, que ao contrário do que se pensa, XP pode ser aplicada em projetos de vários portes, pois seu dinamismo é tão latente, que permite seu uso por equipes criativas em qualquer projeto.

É importante lembrar também que os valores citados acima, alicerçam a metodologia, pelos seguintes motivos:

* A simplicidade é necessária desde a forma como se levanta requisitos até a codificação e os testes da solução desenvolvida;
* A comunicação é obrigatória para que não haja lacunas em processos e problemas entre equipe, cliente e fornecedor;
* O feedback é a pratica fundamentada em retornar informações entre os membros da equipe e também na relação com o cliente, desde responder e-mails, telefonemas bips e demais meios. Devido a isso, é um mecanismo para melhorar a prática de comunicação explanada acima;
* E a coragem para saber dizer NÃO quando necessário, ou então para dizer que o projeto vai demorar além do estimado, pois os novos requisitos precisam ser codificados ou o código já em funcionamento precisa ser refatorado.