Extreme Programming (XP)

Engenharia de Software II

Profa. Andréa Sabedra Bordin

O SURGIMENTO

- Em meados de 1990, ***Kent Beck*** procurou formas mais simples e eficientes de desenvolver software.
  - Identificou o que tornava simples e o que dificultava o desenvolvimento de software.
  - **Software com requisitos vagos e sujeitos a mudanças.**
- Em Março de 1996, ele iniciou um projeto com novos conceitos que resultaram na metodologia XP *–e**X**treme **P**rogramming.*

O QUE É *EXTREME PROGRAMMING?*

- “Trata-se de uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e em constante mudança”

Kent Beck

PROGRAMANDO AO EXTREMO

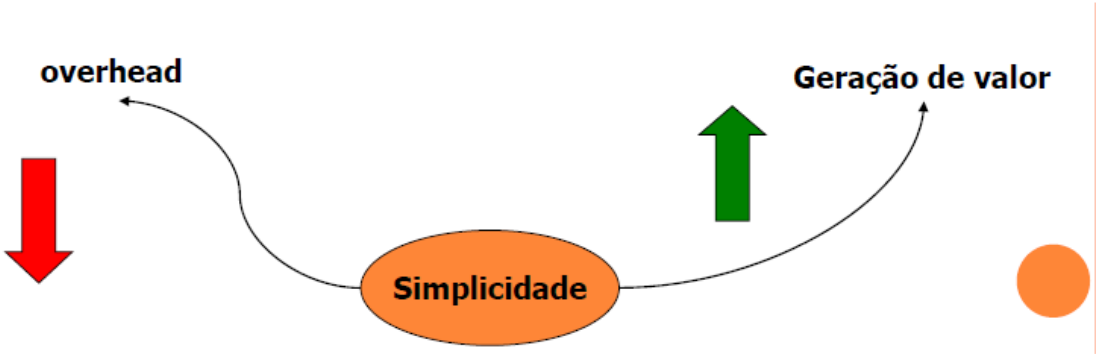
- Levar todas as boas práticas ao extremo
  - Se testar é bom, vamos testar toda hora!!
  - Se projetar é bom, vamos fazer disso parte do trabalho diário de cada pessoa!
  - Se integrar é bom, vamos integrar a maior quantidade de vezes possível!
  - Se iterações curtas é bom, vamos deixar as iterações realmente curtas!

XP

- XP não é um método prescritivo, que define um passo a passo detalhado para construção de software.
- Em vez disso, XP é definido por meio de um conjunto de **valores**, **princípios** e **práticas de desenvolvimento**.
  - XP é inicialmente definido de forma abstrata, usando-se de valores e princípios que devem fazer parte da cultura e dos hábitos de times de desenvolvimento de software.
- Depois, esses valores e princípios são concretizados em uma lista de **práticas de desenvolvimento**.
- Frequentemente, quando decidem adotar XP, desenvolvedores e organizações concentram-se nas práticas.
  - Porém, os valores e princípios são componentes chaves do método, pois são eles que dão sentido às práticas propostas em XP.

Valores do XP

- Simplicidade
  - Muitas funcionalidades de sistemas não são utilizadas.
  - A equipe deve se concentrar nas funcionalidades efetivamente necessárias.
  - Simplicidade normalmente gera mais valor.



- Comunicação
  - Essencial para o cliente dizer aquilo que realmente precisa.
  - Comunicação face a face é a maneira mais rápida de “espalhar” conhecimento.

Valores do XP

- *Feedback*
  - Buscar *feedback* constante para que eventuais falhas de comunicação sejam corrigidas o mais rapidamente possível, antes que os danos se alastrem e o custo da correção seja alto.
  - *Feedback* aumenta aprendizado e produtividade.
- Coragem
  - *Extreme Programming* é novo e diferente.
  - Atitude é mais importante que processo.
  - Coragem pra dizer a verdade, para recomeçar do zero, para inovar.
  - A única constante em um projeto de software é a necessidade de mudança.
- Respeito
  - Coragem, Feedback, Simplicidade, Comunicação...
  - Respeito é necessário para tirar o máximo desses valores.
  - Se não houver respeito, a comunicação falha e o projeto afunda.

Princípios do XP

- Os valores são abstratos e universais.
- Por outro lado, as práticas que vamos mencionar mais adiante são procedimentos concretos e pragmáticos.
- Assim, para unir esses dois extremos, XP defende que projetos de software devem seguir um conjunto de **princípios**.

Princípios do XP

- **Humanidade** (*humanity*). Software é uma atividade intensiva no uso de capital humano. O principal recurso de uma empresa de software não são seus bens físicos — computadores, prédios, móveis ou conexões de Internet, por exemplo — mas sim seus colaboradores.
- Um termo que reflete bem esse princípio é *peopleware,* que foi cunhado por Tom DeMarco, em um livro com o mesmo título (link).
  - A ideia é que a gestão de pessoas — incluindo fatores como expectativas, crescimento, motivação, transparência e responsabilidade — é fundamental para o sucesso de projetos de software.

Princípios do XP

- **Economicidade** (*economics*). Se por um lado, *peopleware* é fundamental, por outro lado software é uma atividade cara, que demanda a alocação de recursos financeiros consideráveis.
- Logo, tem-se que ter consciência de quem está pagando as contas do projeto, espera resultados econômicos e financeiros.
- Por isso, na grande maioria dos casos, software não pode ser desenvolvido apenas para satisfazer a vaidade intelectual de seus desenvolvedores.
- Software não é uma obra de arte, mas algo que tem que gerar resultados econômicos, como defendido por esse princípio de XP.

Princípios do XP

- **Benefícios Mútuos**. XP defende que as decisões tomadas em um projeto de software têm que beneficiar múltiplos *stakeholders*.
- Por exemplo: *refactoring* é uma atividade que torna o código mais limpo e fácil de entender, tanto para quem o escreveu, como para quem futuramente terá que mantê-lo.

Princípios do XP

- **Melhorias Contínuas** (*improvements*): Nenhum processo de desenvolvimento de software é perfeito.
- Por isso, é mais seguro trabalhar com um sistema que vai sendo continuamente aprimorado, a cada iteração, com o feedback dos clientes e de todos os membros do time.
- Pelo mesmo motivo, XP não recomenda investir um grande montante de tempo em um *design* inicial e completo.
- Em vez disso, o *design* do sistema também é incremental, melhorando a cada iteração.

Princípios do XP

- **Falhas Acontecem**. Desenvolvimento de software não é uma atividade livre de riscos. Logo, falhas são esperadas em projetos de desenvolvimento de software.
- No contexto desse princípio, falhas incluem *bugs*, funcionalidades que não se mostraram interessantes para os usuários finais e requisitos não-funcionais que não estão sendo plenamente atendidos, como desempenho, usabilidade, privacidade, disponibilidade, etc.
- Evidentemente, XP não advoga que essas falhas devem ser acobertadas. Porém, elas não devem ser usadas para punir membros de um time. Pelo contrário, falhas fazem parte do jogo, se um time pretende entregar software com rapidez.

Princípios do XP

- ***Baby Steps***. É melhor um progresso seguro, testado e validado, mesmo que pequeno, do que grandes implementações com riscos de serem descartadas pelos usuários.
- O mesmo vale para testes (que são úteis mesmo quando as unidades testadas são de menor granularidade), integração de código (é melhor integrar diariamente, do que passar pelo stress de fazer uma grande integração após semanas de trabalho) e refatorações (que devem ocorrer em pequenos passos, quando é mais fácil verificar que o comportamento do sistema está sendo preservado).
- Em resumo, o importante é garantir melhorias contínuas, não importando que sejam pequenas, desde que na direção correta.
- Essas pequenas melhorias são melhores do que grandes revoluções, as quais costumam não apresentar resultados positivos, pelo menos quando se trata de desenvolvimento de software.

Princípios do XP

Responsabilidade Pessoal (*accepted responsibility*). De acordo com esse princípio, desenvolvedores devem ter uma ideia clara de seu papel e responsabilidade na equipe.

O motivo é que responsabilidade não pode ser transferida, sem que a outra parte a aceite. Por isso, XP defende que o engenheiro de software que implementa uma *história* — termo que o método usa para requisitos — deve ser também aquele que vai testá-la e mantê-la.

Mundo Real

- Um dos primeiros sistemas a adotar XP foi um sistema de folha de pagamentos da fabricante de automóveis Chrysler, chamado *Chrysler Comprehensive Compensation (*C3) (link). O projeto desse sistema começou no início de 1995 e, como não apresentou resultados concretos, ele foi reiniciado no ano seguinte, sob a liderança de **Kent Beck**. No desenvolvimento do sistema C3, foram usadas e testadas diversas ideias do método que poucos anos depois receberia o nome de XP.

Práticas sobre o processo de desenvolvimento

- XP recomenda o envolvimento dos clientes com o projeto.
  - Ou seja, além de desenvolvedores, os times incluem pelo menos um **representante dos clientes**, que deve entender do domínio do sistema que será construído.
- Uma das funções desse representante é escrever as **histórias de usuário** (*user stories*), que é o nome que XP dá para os documentos que descrevem os requisitos do sistema a ser implementado.
- **Histórias são documentos resumidos, com apenas duas ou três sentenças, com as quais o representante dos clientes define o que ele deseja que o sistema faça, usando sua própria linguagem**.

Práticas sobre o processo de desenvolvimento

- Depois de escritas pelo representante dos clientes, as histórias são **estimadas** pelos desenvolvedores.
  - São os desenvolvedores que definem, mesmo que preliminarmente, quanto **tempo** será necessário para implementar as histórias escritas pelo representante dos clientes.
  - Frequentemente, a duração de uma história é estimada em ***story points***, em vez de horas ou homens/hora.
  - Nesses casos, usa-se uma escala inteira para classificar histórias como possuindo um certo número de *story points*.
  - O objetivo é definir uma ordem relativa entre as histórias.
  - As histórias mais simples são estimadas como tendo tamanho igual a 1 *story point*; histórias que são cerca de duas vezes mais complexas do que as primeiras são estimadas como tendo 2 *story points* e assim por diante.

Práticas sobre o processo de desenvolvimento

- A implementação das histórias ocorre em **iterações**, as quais têm uma duração fixa e bem definida, variando de **uma a três semanas**, por exemplo.
- As iterações, por sua vez, formam ciclos mais longos, chamados de **releases**, de dois a três meses, por exemplo.
- A **velocidade** de um time é o número de *story points* que ele consegue implementar em uma iteração.
- Sugere-se que o representante dos clientes escreva histórias que requeiram pelo menos uma release para serem implementadas.
  - Em XP, o horizonte de planejamento é uma release, isto é, alguns meses.

Não necessariamente a versão do sistema ao final de uma release de XP precisa entrar em produção.

Práticas sobre o processo de desenvolvimento

- O representante do cliente deve priorizar as histórias.
  - Para isso, ele deve definir quais histórias serão implementadas nas iterações da primeira *release*.
  - Nessa priorização, deve-se respeitar a velocidade do time de desenvolvimento.
  - Por exemplo, suponha que a velocidade de um time seja de 25 story points por iteração.
  - Nesse caso, o representante do cliente não pode alocar histórias para uma iteração cujo somatório de story points ultrapasse esse limite.
- A tarefa de alocar histórias a iterações e releases é chamada de **planejamento de releases** (*planning game*)

Nessa tabela, estamos assumindo que o representante dos clientes escreveu 8 histórias, que cada release possui duas iterações e que a velocidade do time é de 21 story points por iteração (veja que o somatório dos *story points* de cada iteração é exatamente igual a 21).

	Story point	**Interação**	**release**
Cadastrar usuário	8	1	1
Postar perguntas	5	1	1
Postar respostas	3	1	1
Tela de abertura	5	1	1
Gamificar perguntas/respostas	5	2	1
Pesquisar perguntas/respostas	8	2	1
Adicionar tags	5	2	1
Comentar perguntas/respostas	3	2	1

Práticas sobre o processo de desenvolvimento

- Uma vez realizado o planejamento de uma release, começam as iterações.
- Antes de mais nada, o time de desenvolvimento deve se reunir para realizar o **planejamento da iteração.**
- O objetivo desse planejamento é **decompor as histórias de uma iteração em tarefas**, as quais devem corresponder a atividades de programação que possam ser alocadas para um dos desenvolvedores do time.

Práticas sobre o processo de desenvolvimento

- Por exemplo, a seguinte lista mostra as tarefas para a história **Postar Perguntas**:
  - Projetar e testar a interface Web, incluindo leiaute, CSS templates, etc.
  - Instalar banco de dados, projetar e criar tabelas.
  - Implementar a camada de acesso a dados.
  - Instalar servidor e testar framework web.
  - Implementar camada de controle, com operações para cadastrar, remover e atualizar perguntas.
  - Implementar interface Web.

Práticas de Programação

- O nome *Extreme Programming* foi escolhido porque XP propõe um conjunto de práticas de programação inovadoras, principalmente para a época na qual foram propostas, no final da década de 90.
- O método também propôs um novo conjunto de práticas de programação, incluindo **programação em pares, testes automatizados, desenvolvimento dirigido por testes (TDD), builds automatizados, integração contínua**, etc.
- A maioria dessas práticas passou a ser largamente adotada pela indústria de software.

Práticas de Programação

- **Design Incremental**. No XP não há uma fase de design e análise detalhados, conhecida como *Big Design Up Front* (BDUF), a qual é uma das principais fases de processos do tipo Waterfall.
- A ideia é que o time deve reservar tempo para definir o design do sistema que está sendo desenvolvido.
- Porém, isso deve ser uma atividade contínua e incremental, em vez de estar concentrada no início do projeto, antes de qualquer codificação.
- Quando o design é realizado no início do projeto, correm-se diversos riscos, pois os requisitos ainda não estão totalmente claros para o time, e nem mesmo para o representante dos clientes.
- Design incremental somente é possível caso seja adotado em conjunto com as demais práticas de XP, principalmente **refactoring.**

Práticas de Programação

- **Programação em Pares**. Toda tarefa de codificação deve ser realizada por dois desenvolvedores trabalhando juntos, compartilhando o mesmo teclado e monitor. Um dos desenvolvedores é o **líder** (ou *driver*) da sessão, ficando com o teclado e o mouse. Ao segundo desenvolvedor cabe a função de revisor e questionador do trabalho do líder. Esse segundo desenvolvedor é chamado de **navegador**.
- Com programação em pares espera-se melhorar a qualidade do código e do *design*, pois **duas cabeças pensam melhor do que uma**.
- Além disso, programação em pares contribui para disseminar o conhecimento sobre o código, que não fica nas mãos e na cabeça de apenas um desenvolvedor.

Mundo Real

Em 2008, dois pesquisadores da Microsoft Research, Andrew Begel e Nachiappan Nagappan, realizaram um survey com 106 desenvolvedores da empresa, para capturar a percepção deles sobre programação em pares (link).
Quase 65% dos desenvolvedores responderam positivamente a uma primeira pergunta sobre se programação em pares estaria funcionando bem para eles (*pair programming is working well for me*). Quando perguntados sobre os benefícios de programação em pares,
as respostas foram as seguintes: redução no número de bugs (62%), produção de código de melhor qualidade (45%), disseminação de conhecimento sobre o código (40%) e oportunidade de aprendizado com os pares (40%).
Por outro lado, os custos da prática foram apontados como sendo seu principal problema (75%). Sobre as características do par ideal, a resposta mais comum foi complementaridade de habilidades (38%). Ou seja, desenvolvedores preferem parear com uma pessoa que o ajude a superar seus pontos fracos.

Práticas de Programação

- **Propriedade Coletiva do Código**. A ideia é que qualquer desenvolvedor — ou par de desenvolvedores trabalhando junto — pode modificar qualquer parte do código, seja para implementar uma nova *feature*, para corrigir um bug ou para aplicar um *refactoring*.

Práticas de Programação

- **Testes Automatizados**. Essa é uma das práticas de XP que alcançou maior sucesso. A ideia é que testes manuais — um ser humano executando o programa, fornecendo entradas e checando as saídas produzidas — é um procedimento custoso e que não pode ser reproduzido a todo momento.
- Logo, XP propõe a implementação de programas — chamados de testes — para executar pequenas unidades de um sistema, como métodos, e verificar se as saídas produzidas são aquelas esperadas.
- Essa prática prosperou porque, mais ou menos na mesma época de sua proposição, foram desenvolvidos os primeiros frameworks de testes de unidade — como o JUnit, cuja primeira versão, implementada por Kent Beck e Erich Gamma, é de 1997.

Práticas de Programação

- **Desenvolvimento Dirigido por Testes (TDD).** Se em XP todo método deve possuir testes, por que não escrevê-los primeiro?
- Isto é, implementa-se o teste de um método e, só então, o seu código.
- TDD, que também é conhecido como *test-first programming*, possui duas motivações principais:
- (1) evitar que a escrita de testes seja sempre deixada para amanhã, pois eles são a primeira coisa que se deve implementar;
- (2) ao escrever um teste, o desenvolvedor se coloca no papel de cliente do método testado, isto é, ele primeiro pensa na sua interface, em como os clientes devem usá-lo, para então pensar na implementação.

Práticas de Programação

- **Build Automatizado**. Build é o nome que se dá para a geração de uma versão de um sistema que seja executável e que possa ser colocada em produção. Logo, inclui não apenas a compilação do código, mas a execução de outras ferramentas como empacotadores de código em arquivos WAR, JAR, etc.
- No caso de XP, a execução dos testes é outra etapa fundamental do processo de build.
- Para automatizar esse processo, são usadas ferramentas, como Ant, Maven, etc.
- Primeiro, XP defende que o processo de build seja automatizado, sem nenhuma intervenção dos desenvolvedores.
- O objetivo é liberá-los das tarefas de rodar scripts, informar parâmetros de linhas de comando, configurar ferramentas, etc. Assim, eles podem focar apenas na implementação de histórias.
- Segundo, XP defende que o processo de build seja o mais rápido possível, para que os desenvolvedores recebam rapidamente feedback sobre possíveis problemas, como um erro de compilação.
- Recomenda-se um limite de 10 minutos para a conclusão de um build. No entanto, dependendo do tamanho do sistema e de sua linguagem de programação, pode ser difícil atender a esse limite.

Práticas de Programação

- **Integração Contínua**. Sistemas de software são desenvolvidos com o apoio de sistemas de controle de versões (VCS, ou *Version Control System*), que armazenam o código fonte do sistema e de arquivos relacionados, como arquivos de configuração, documentação, etc.
- Hoje o sistema de controle de versão mais usado é o git, por exemplo.
- Quando se usa um VCS, desenvolvedores têm que primeiro baixar (*pull*) o código fonte para sua máquina local, antes de começar a trabalhar em uma tarefa.
- Feito isso, eles devem subir o código modificado (*push*).
- Esse último passo é chamado de **integração** da modificação no código principal, armazenado no VCS.
- Porém, entre um *pull* e um *push*, outro desenvolvedor pode ter modificado o mesmo trecho de código e realizado a sua integração.

Mundo Real

Em 2010, Laurie Williams, professora da Universidade da Carolina do Norte, nos EUA, pediu que 326 desenvolvedores respondessem a um questionário sobre a experiência deles com métodos ágeis (link). Em uma das questões, pedia-se aos participantes para ranquear a importância de práticas ágeis, usando uma escala de 1 a 5, na qual o score 5 deveria ser dado apenas para práticas essenciais em desenvolvimento ágil. Três práticas ficaram empatadas em primeiro lugar, com score médio 4.5 e desvio padrão de 0.8. São elas: integração contínua, iterações curtas (menos de 30 dias) e definição de critérios para tarefas concluídas (*done criteria*). Por outro lado, dentre as práticas nas últimas posições podemos citar planning poker (score médio 3.1) e programação em pares (score médio 3.3).

Referências

- Valente, Marco Tulio. Engenharia de Software Moderna: Princípios e práticas para desenvolvimento de software com produtividade. Belo Horizonte, 2020.
- Wazlawick, R.S. Engenharia de Software. Conceitos e Práticas. Rio de Janeiro: Elsevier, 2013.
- Slides de aula do Prof. Alexandre Monteiro –UFPE.
- http://www.extremeprogramming.org/