METODOLOGIA DE DESENVOLVIMENTO DE SISTEMAS

Uma metodologia de desenvolvimento de software é um conjunto de atividades que auxiliam a produção de software. O resultado dessas atividades é um produto que reflete a forma como todo o processo foi conduzido. Pode ser caracterizada como um *framework* de processos organizacionais que, se usados adequadamente, garantem o sucesso da área de Engenharia de Software.

Conforme CARVALHO (2001), "uma metodologia de desenvolvimento detalha as atividades do ciclo de vida, especificando um conjunto único e coerente de princípios, métodos, linguagem de representação, normas, procedimentos e documentação, que permitem ao desenvolvedor de software implementar sem ambigüidade as especificações advindas das fases do ciclo de vida do software".

Embora tenham sido criadas várias metodologias para o desenvolvimento de software, existem atividades fundamentais comuns a todas elas [SOMMERVILLE, 2003]:

- Especificação: definição das funcionalidades e demais características do produto.
- Projeto e implementação: o software é produzido de acordo com as especificações. Nesta fase são propostos modelos por meio de diagramas que serão implementados em alguma linguagem de programação.
- Validação: atividades de revisão e testes visando a assegurar que os requisitos sejam cumpridos.
- Evolução: atividades de manutenção, por exemplo, para adaptar o software a novas necessidades do cliente.

Um ambiente de desenvolvimento de software de qualidade se inicia com uma sólida definição do processo que inclui atividades usualmente definidas como fases, tarefas, passos, e o que será produzido por cada uma dessas atividades. O processo também especifica a ordenação das atividades, que podem ser seqüenciais, concorrentes ou em paralelo, e todas reunidas definem a base da execução do desenvolvimento.

Muitas organizações erradamente confundem o processo com a utilização de certas ferramentas de desenvolvimento [COSTA, 1999, p. 28-30].

Em particular, as pequenas e médias organizações não possuem recursos suficientes para adotar o uso de metodologias pesadas e, por essa razão, normalmente não utilizam nenhum processo. O resultado dessa falta de sistematização na produção de software é a baixa qualidade do produto final, além de dificultar a entrega do software nos prazos e custos predefinidos e inviabilizar a futura evolução do software.

METODOLOGIAS EXISTENTES

O número de metodologias propostas para o desenvolvimento de software atingiu um número demasiado elevado, o que torna virtualmente impossível a sua apresentação. Por

isso, enumeramos algumas metodologias, estruturadas e orientadas a objeto, conhecidas que maior relevância e divulgação tiveram. Para além destas, existiram outras contribuições importantes que não estão incluídas aqui por não apresentarem uma perspectiva integrada de todo o processo de desenvolvimento, mas apenas sugerirem anotações ou técnicas de modelagem.

1 EXTREME PROGRAMMING

A Extreme Programming (XP) é uma metodologia ágil para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que são modificados rapidamente. Entre as principais características que a diferencia das outras metodologias são:

- Feedback constante
- Abordagem incremental
- A comunicação entre as pessoas é encorajada

A finalidade da comunicação é manter o melhor relacionamento possível entre clientes, desenvolvedores e gerentes, preferindo conversas pessoais a outros meios de comunicação.

A prática do feedback constante significa que o programador terá informações constantes sobre o código e o cliente. A informação do código é dada pelos testes constantes, que indicam os erros tanto individuais quanto do software integrado. Em relação ao cliente, o feedback constante significa que ele terá frequentemente uma parte do software totalmente funcional para avaliar. Com isso, o cliente constantemente sugerirá novas características e informações aos desenvolvedores. Desta forma, a tendência é que o produto final esteja de acordo com as expectativas reais do cliente.

A XP baseia-se em 12 práticas descritas a seguir:

1 - Planejamento: baseia-se em requisitos atuais reais para desenvolvimento de software, não em possíveis requisitos futuros. A XP procura evitar os problemas de relacionamento entre a área de negócios e a área de desenvolvimento. Ambas as áreas devem cooperar para o sucesso e cada uma deve focar partes específicas do projeto. Desta forma, enquanto a área de negócios deve decidir sobre o escopo, a composição das versões e as datas de entrega, os desenvolvedores devem decidir sobre as estimativas de prazo, o processo de desenvolvimento e o cronograma detalhado para que o software seja entregue nas datas especificadas.

2 – Entregas freqüentes: visam à construção de um software simples, atualizado à medida que novos requisitos surgem. Cada versão deve conter os requisitos de maior valor para o negócio. Isto evita surpresas como a necessidade de grandes modificações após vários meses de trabalho, torna mais precisas as avaliações e aumenta a probabilidade de o software final estar de acordo com as necessidades do contratante.

3 – Metáfora: são as descrições de um software sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento de software.

4 – Projeto simples: o programa deve ser o mais simples possível e satisfazer os requisitos atuais, sem a preocupação de requisitos futuros.

5 – Testes: focaliza a validação do projeto durante todo o processo de desenvolvimento. Os programadores desenvolvem o software criando primeiramente os casos de testes.

6 – Programação em pares: a implementação do código é feita em dupla, ou seja, dois desenvolvedores trabalham em um único computador. Uma grande vantagem da programação em dupla é a possibilidade de os desenvolvedores aprenderem um com o outro. O código possui maior probabilidade de estar correto, uma vez que duas pessoas estão preocupadas em sua implementação. A programação em pares possibilita que revisões sejam feitas já durante a implementação do software.

7 – Refatoração: focaliza o aperfeiçoamento do projeto do software e está presente em todo o desenvolvimento. A refatoração deve ser feita sempre que for possível simplificar uma parte do software, sem que seja perdida nenhuma funcionalidade.

8 – Propriedade coletiva: o código pertence a todos os membros da equipe. Uma grande vantagem desta prática é que, caso um membro da equipe deixe o projeto antes de concluí-lo, a equipe conseguirá terminá-lo com puçás dificuldades, visto que todos conhecem o software, mesmo que não seja de forma detalhada.
Um benefício adicional é a menor dependência de um autor específico, como no caso de um programador "herói".

9 – Integração contínua: uma vez testado e validade, o código produzido por uma equipe deve ser integrado ao sistema e este, por sua vez, também deve ser testado. Dessa maneira, o software é construído e verificado gradativamente, possivelmente sendo mais fácil isolar erros e suas causas. Esta prática é facilitada com o uso de apenas uma máquina de integração, que deve ser de livre acesso a todos os membros da equipe.

10 – Trabalho semanal de 40 horas: a XP assume que não se deve fazer horas extras constantemente. Caso seja necessário trabalhar mais de 40horas pela segunda semana consecutiva, há um problema sério no projeto que deve ser resolvido não com o aumento de horas trabalhadas, mas com melhor planejamento, por exemplo. Esta prática procura ratificar o foco nas pessoas e não em processos e planejamentos. Preferencialmente, os planos devem ser alterados, em vez de sobrecarregar as pessoas.

11 – Cliente presente: o cliente deve estar sempre disponível para sanar todas as dúvidas de requisitos. Isto evita atrasos e até mesmo construções errôneas. Uma idéia interessante é manter o cliente como parte integrante da equipe de desenvolvimento.

12 – Código-padrão: a padronização favorece o trabalho em equipe e a propriedade coletiva do código.

A XP é ideal para ser usada em projetos em que os stakeholders não sabem exatamente o que desejam e podem mudar muito de opinião durante o desenvolvimento.

A prática do feedback constante permite, então, adaptar rapidamente o produto. Outro ponto positivo são as entregas freqüentes de software executável: o cliente não aguarda um longo período antes de conhecer o produto, testá-lo e apontar novas mudanças. A interação e teste contínuos também contribuem para a melhora na qualidade. A fase de integração clássica é assim substituída por várias pequenas integrações de novos componentes, em que eventuais problemas são detectados e resolvidos constantemente.

A XP apresenta também alguns problemas [SOARES, 2004]. Muitos acreditam que a metodologia seja uma volta ao processo caótico de desenvolvimento de software, conhecido também como “codifica-remenda”. Esse modelo existe principalmente em pequenas e médias organizações que não podem suportar os altos custos de desenvolvimento das metodologias tradicionais. Além disso, o uso errôneo da XP pode inibir certas práticas positivas de desenvolvimento, como, por exemplo, a análise do problema por meio de diagramas. Obviamente, não se devem projetar diagramas que nunca serão consultados, mas é importante projetar alguns modelos que ajudarão no entendimento do problema.

A informalidade no levantamento de requisitos pode não ser bem vista pelos clientes, que podem sentir-se inseguros. Situação semelhante pode ocorrer com a refatoração de código, que pode ser interpretada pelos clientes como amadorismo e incompetência.

Segundo BECK (2001), o criador da XP, há ainda outros fatores que podem tornar a metodologia inadequada. Por exemplo, profissionais que não se adaptam bem a práticas de equipe, como a programação em duplas, podem ter muita dificuldade em aceitar a XP. A exigência de que a equipe não esteja geograficamente separada cria sérias dificuldades, sobretudo em grandes empresas onde isso é mais comum.

As 12 práticas da XP apóiam-se mutuamente e, portanto, em princípio se deve aplicá-las em conjunto. Exemplificando, não usar a programação em dupla pode comprometer a propriedade coletiva de código. Contudo, a implantação de todas as práticas simultaneamente pode ser percebida de forma confusa pela equipe. Beck sugere, então, que as práticas sejam implantadas uma a uma, o que permite à equipe uma adaptação gradual e segura, evitando desentendimento e pressões.

Fonte: “UMA METODOLOGIA DE DESENVOLVIMENTO DE SISTEMAS PARA UMA EMPRESA DE PLANO ODONTOLÓGICO” - UNIVERSIDADE FEDERAL DA BAHIA - DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO