ENGENHARIA DE REQUISITOS

Prof. Renato Matroniani

EDUCAÇÃO METODISTA

Referências

- SOMMERVILLE, Ian. Engenharia de Software. 10ª ed. São Paulo: Pearson Prentice Hall, 2011.
- PRESSMAN, R. S. Engenharia de Software. Rio de Janeiro: McGraw Hill, 2002, 704p.
- RODRIGO CANTÚ POLO, Validação e teste de software. Editora Contentus, 2020, 93p.
- SBROCCO, J. H.; MACEDO, P. C., Metodologias Ágeis: Engenharia de Software sob Medida., São Paulo: Érica, 2012, 256p.
- CHIKOFSKY, Elliot. Computer-Aided Software Engineering (CASE). COMPUTER IEEE Computer Society, 1993.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software

- Softwares fazem parte de quase todas as operações de negócios.
- Novos softwares são desenvolvidos rapidamente para obterem proveito de novas oportunidades e responder às pressões competitivas.
- O desenvolvimento e entrega rápidos são, portanto, o requisito mais crítico para o desenvolvimento de sistemas de software.
- Na verdade, muitas empresas **estão dispostas a trocar a qualidade e o compromisso com requisitos do software** por uma implantação mais rápida do software de que necessitam.

BI Analytics

EDUCAÇÃO METODISTA

Texto baseado em Pressman, 2010.

Desenvolvimento ágil de software

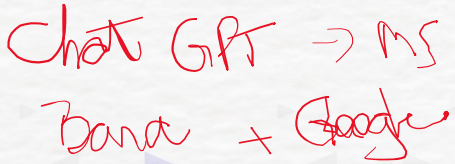
- Essas empresas operam em um ambiente de mudanças rápidas, e por isso, muitas vezes, é praticamente impossível obter um conjunto completo de requisitos de software estável.
- Os requisitos iniciais inevitavelmente serão alterados, pois os clientes acham impossível prever como um sistema afetará as práticas de trabalho, como irá interagir com outros sistemas e quais operações do usuário devem ser automatizadas.

Texto baseado em Pressman, 2010.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software

- Pode ser que os requisitos se tornem claros apenas após a entrega do sistema e à medida que os usuários ganhem experiência.
- Mesmo assim, devido a fatores externos, os requisitos são suscetíveis a mudanças rápidas e imprevisíveis.
- Por exemplo, quando for entregue, o software poderá estar desatualizado.



EDUCAÇÃO METODISTA

Texto baseado em Pressman, 2010.

Desenvolvimento ágil de software

- Processos de desenvolvimento de software que planejam especificar completamente os requisitos e, em seguida, projetar, construir e testar o sistema não estão adaptados ao desenvolvimento rápido de software.
- Com as mudanças nos requisitos ou a descoberta de problemas de requisitos, o projeto do sistema ou sua implementação precisa ser refeito ou retestado.
- Como consequência, um processo convencional em cascata ou baseado em especificações costuma ser demorado, e o software final é entregue ao cliente bem depois do prazo acordado.

Texto baseado em Pressman, 2010.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software

- Para alguns tipos de software, como sistemas críticos de controle de segurança, em que uma análise completa do sistema é essencial, uma abordagem dirigida a planos é a melhor opção.
- No entanto, em um ambiente de negócio que se caracteriza por mudanças rápidas, isso pode causar problemas reais.

Texto baseado em Pressman, 2010.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software

- Quando o software estiver disponível para uso, a razão original para sua aquisição pode ter mudado tão radicalmente que o software será efetivamente inútil.
- Portanto, para os sistemas de negócios, particularmente, os processos de desenvolvimento que se caracterizem por desenvolvimento e entrega rápidos de software são essenciais.

Dessa forma, podemos dizer em quais situações utilizar desenvolvimento ágil e quais situações utilizar desenvolvimento tradicional?

Texto baseado em Pressman, 2010.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software - Histórico

- Década de 1980: a IBM introduziu o desenvolvimento incremental.
- Década de 1980: A introdução das linguagens de quarta geração, apoiou a ideia de desenvolvimento e entrega rápidos de software.
- Década de 1990: A ideia realmente decolou no final desta década, com o desenvolvimento da noção de abordagens ágeis, como Metodologia de Desenvolvimento de Sistemas Dinâmicos (DSDM, do inglês *dynamic systems development method*), *Scrum* e *Extreme Programming*.(XP)

EDUCAÇÃO METODISTA

Texto baseado em Pressman, 2010.

Modelo tradicional x ágil – +histórico

- Na década de 1980 e início da de 1990, havia uma visão generalizada de que a melhor maneira para conseguir o melhor software era por meio de um planejamento cuidadoso do projeto, qualidade da segurança formalizada, do uso de métodos de análise e projeto apoiado por ferramentas CASE (*Computer-aided software engineering*) e do processo de desenvolvimento de software rigoroso e controlado.
- Essa percepção veio da comunidade de engenharia de software, responsável pelo desenvolvimento de sistemas de software grandes e duradouros, como sistemas aeroespaciais e de governo.

EDUCAÇÃO METODISTA

Modelo tradicional x ágil – +histórico

- Esse software foi desenvolvido por grandes equipes que trabalham para diferentes empresas.
- Geralmente, as equipes eram dispersas geograficamente e trabalhavam com o software por longos períodos.
- Um exemplo desse tipo de software é o sistema de controle de uma aeronave moderna, que pode demorar até dez anos desde a especificação inicial até a implantação.

EDUCAÇÃO METODISTA

Modelo tradicional x ágil – +histórico

- Tais abordagens dirigidas a planos envolvem um *overhead* significativo no planejamento, projeto e documentação do sistema.
- Esse *overhead* se justifica quando o trabalho de várias equipes de desenvolvimento tem de ser coordenado, quando o sistema é um sistema crítico e quando muitas pessoas diferentes estão envolvidas na manutenção do software durante sua vida.

Modelo tradicional x ágil – +histórico

- No entanto, quando essa abordagem pesada de desenvolvimento dirigido a planos é aplicada aos sistemas corporativos de pequeno e médio porte, o overhead envolvido é tão grande que domina o processo de desenvolvimento de software.
- Gasta-se mais tempo em análises de como o sistema deve ser desenvolvido do que no desenvolvimento de programas e testes.
- Como os requisitos do sistema se alteram, o retrabalho é essencial, e, pelo menos em princípio, a especificação e o projeto devem mudar com o programa.

Modelo tradicional x ágil – +histórico

- A insatisfação com essas abordagens pesadas da engenharia de software levou um grande número de desenvolvedores de software a proporem, na década de 1990, novos ‘métodos ágeis’.
- Estes permitiram que a equipe de desenvolvimento focasse no software em si, e não em sua concepção e documentação.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software

- Os processos de desenvolvimento rápido de software são concebidos para produzir, rapidamente, softwares úteis.
- O software não é desenvolvido como uma única unidade, mas como uma **série de incrementos** — cada incremento inclui uma nova funcionalidade do sistema.

Texto baseado em Pressman, 2010.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software

- Embora existam muitas abordagens para o desenvolvimento rápido de software, elas compartilham algumas características fundamentais:

1. Os processos de especificação, projeto e implementação são intercalados.

Não há especificação detalhada do sistema, e a documentação do projeto é minimizada ou gerada automaticamente pelo ambiente de programação usado para implementar o sistema.

O documento de requisitos do usuário apenas define as características mais importantes do sistema.

EDUCAÇÃO METODISTA

Texto baseado em Pressman, 2010.

Desenvolvimento ágil de software

2. O sistema é desenvolvido em uma série de versões.

Os usuários finais e outros stakeholders do sistema são envolvidos na especificação e avaliação de cada versão.

Eles podem propor alterações ao software e novos requisitos que devem ser implementados em uma versão posterior do sistema.

Texto baseado em Pressman, 2010.

Desenvolvimento ágil de software

3. Interfaces de usuário do sistema são geralmente desenvolvidas com um sistema interativo de desenvolvimento que permite a criação rápida do projeto de interface por meio de desenho e posicionamento de ícones na interface.

O sistema pode, então, gerar uma interface baseada na Web para um navegador ou uma interface para uma plataforma específica, como o Microsoft Windows.

Texto baseado em Pressman, 2010.

Desenvolvimento ágil de software

- Os métodos ágeis são métodos de desenvolvimento incremental em que os incrementos são pequenos e, normalmente, as novas versões do sistema são criadas e disponibilizadas aos clientes a cada duas ou três semanas.
- Elas envolvem os clientes no processo de desenvolvimento para obter feedback rápido sobre a evolução dos requisitos.
- Assim, minimiza-se a documentação, pois se utiliza mais a comunicação informal do que reuniões formais com documentos escritos.

Texto baseado em Pressman, 2010.

EDUCAÇÃO METODISTA

Desenvolvimento ágil de software

Manifesto Ágil	Metodologias Tradicionais
Pessoas e Interação entre a equipe	Foco em processos e ferramentas
Software executável	Extensa documentação nem sempre clara
Interação e colaboração do cliente	Frequentes negociações de contratos
Respostas rápidas para mudanças	Planos previamente definidos.

Fonte: Adaptado de Sbrocco e Macedo (2012, p.13)

EDUCAÇÃO METODISTA

O “Manifesto Ágil”

- Ao todo são doze princípios desse manifesto (ANDERLE, 2015; BECK et al., 2001, SBROCCO E MACEDO (2012):

1. A prioridade é a entrega antecipada e contínua do software ao cliente, de forma a deixá-lo satisfeito.

2. Mudanças nos requisitos, mesmo que tardias, são bem-vindas, pois trazem vantagens competitivas para o cliente.

3. Entregas frequentes do software em funcionamento, em escala de semanas ou poucos meses.

4. Os desenvolvedores devem trabalhar no projeto em conjunto com pessoas de outras áreas, como a de negócios.

5. O ambiente deve ser motivador a todos os envolvidos no projeto, além de trazer aos mesmos suporte e confiança.

6. A comunicação deve ser eficiente e eficaz, e feita diretamente entre os envolvidos, pessoalmente.

EDUCAÇÃO METODISTA

O “Manifesto Ágil”

- Ao todo são doze princípios desse manifesto (ANDERLE, 2015; BECK et al., 2001, SBROCCO E MACEDO (2012):

7. O progresso é medido através de softwares que funcionam.

8.O desenvolvimento sustentável tem grande ligação com as metodologias ágeis.

9. A agilidade do método é favorecida pela excelência técnica e no design.

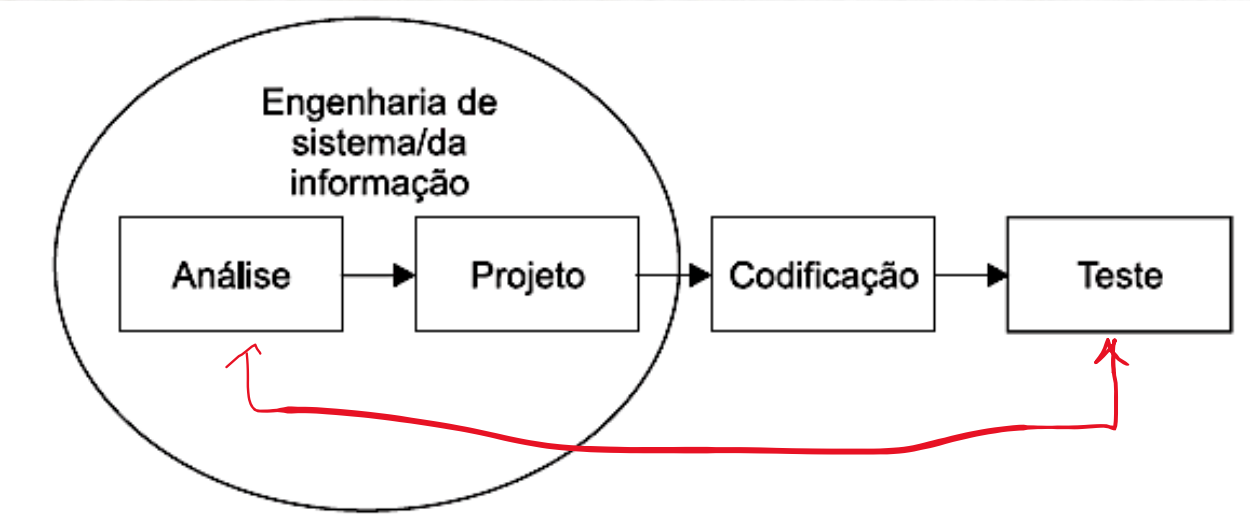
10.Simplicidade é essencial. Todo trabalho ou rotina não necessária deve ser descartada.

11.As equipes auto-organizáveis geram os melhores projetos, designs e arquiteturas.

12. A equipe busca continuamente formas de se tornarem mais eficazes e colocam em prática essas formas.

EDUCAÇÃO METODISTA

Modelo tradicional x ágil



Modelo linear de engenharia de desenvolvimento de software
Fonte: Sbrocco e Macedo (2012, p. 60) *apud* Pressman (2002).

EDUCAÇÃO METODISTA

Modelo tradicional x ágil

- Na metodologia tradicional, os testes de software sucedem a geração do código, que é a interpretação do programa para a linguagem de máquina.
- Com o código gerado, o programa então é testado.
- Percebe-se, dessa forma, uma necessidade latente em processos de desenvolvimento e teste de softwares que sejam mais rápidos e que gerem resultados com maior assertividade, excelência e qualidade.

Modelo tradicional x ágil

- Dentre as metodologias ágeis, o SCRUM se destaca.
- O nome SCRUM vem do método utilizado no jogo de *rugby* quando a bola sai do campo e é necessário reunir todos os jogadores, ou seja, no SCRUM, o atua em conjunto, de forma integrada, visando um objetivo comum.

EDUCAÇÃO METODISTA

Modelo tradicional x ágil

- A metodologia SCRUM segue as doze premissas do manifesto ágil, além de possuir seis características (SBROCCO; MACEDO, 2012, p.161) “flexibilidade de resultados, flexibilidade de prazos, times pequenos, revisões frequentes, colaboração e orientação a objetos”.

EDUCAÇÃO METODISTA

Modelo tradicional x ágil

- As aplicações do SCRUM atendem muitas demandas de agilidade de projetos, inclusive de desenvolvimento e teste de softwares, pois sua utilização abrange:
- Projetos complexos e com mudanças frequentes – o que costuma ser comum em encomendas de software, onde o cliente pode solicitar mudanças no projeto, ao longo do seu desenvolvimento;
- Gerenciamento de tarefas de trabalho;
- Formação de equipes autogerenciáveis;

EDUCAÇÃO METODISTA

Modelo tradicional x ágil

- Implementação de processos iterativos e incrementais, onde o produto é construído em partes;
- Análise das causas de problemas para eliminar impedimentos;
- Valorização de cada indivíduo da equipe.

Princípios dos métodos ágeis

ncípios	Descrição
olvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
rega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
soas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
itar as mudanças	Deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
nter a simplicidade	Focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

Métodos ágeis e Desenvolvimento de Sistemas

Métodos ágeis têm sido muito bem-sucedidos para alguns tipos de desenvolvimento de sistemas:

1.O desenvolvimento de produtos, em que uma empresa de software está desenvolvendo um produto pequeno ou médio para venda.

2.Desenvolvimento de sistema personalizado dentro de uma organização, em que existe um compromisso claro do cliente de se envolver no processo de desenvolvimento, e em que não há muitas regras e regulamentos externos que afetam o software.

Extreme Programming

- Extreme Programming (XP) é talvez o mais conhecido e mais utilizado dos métodos ágeis.
- O nome foi cunhado por Beck (2000), pois a abordagem foi desenvolvida para impulsionar práticas reconhecidamente boas, como o desenvolvimento iterativo, a níveis ‘extremos’.
- Por exemplo, em XP, várias novas versões de um sistema podem ser desenvolvidas, integradas e testadas em um único dia por programadores diferentes.

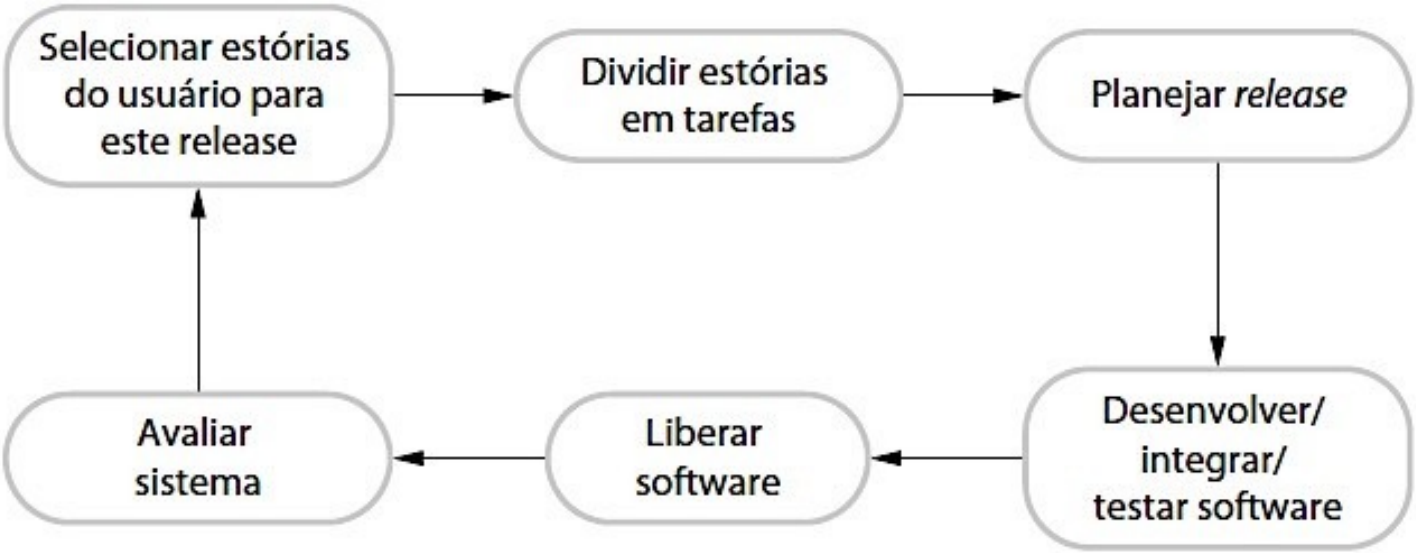
EDUCAÇÃO METODISTA

Extreme Programming

- Em Extreme Programming, os requisitos são expressos como cenários (chamados de estórias do usuário), que são implementados diretamente como uma série de tarefas.
- Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes de escreverem o código.
- Quando o novo código é integrado ao sistema, todos os testes devem ser executados com sucesso.
- Há um curto intervalo entre os releases do sistema.

Extreme Programming

O ciclo de um *release* em Extreme Programming



EDUCAÇÃO METODISTA

Extreme Programming – Práticas

1.O desenvolvimento incremental é sustentado por meio de pequenos e frequentes releases do sistema. Os requisitos são baseados em cenários ou em simples de clientes, usadas como base para decidir a funcionalidade que deve ser incluída em um incremento do sistema.

2.O envolvimento do cliente é sustentado por meio do engajamento contínuo do cliente com a equipe de desenvolvimento. O representante do cliente participa do desenvolvimento e é responsável por definir os testes de aceitação para o sistema.

3.Pessoas — não processos — são sustentadas por meio de programação em pares, propriedade coletiva do código do sistema e um processo de desenvolvimento sustentável que não envolve horas de trabalho excessivamente longas.

EDUCAÇÃO METODISTA

Extreme Programming – Práticas

4.As mudanças são aceitas por meio de releases contínuos para os clientes, do desenvolvimento test-first, da refatoração para evitar a degeneração do código e integração contínua de nova funcionalidade.

5.A manutenção da simplicidade é feita por meio da refatoração constante que melhora a qualidade do código,bem como por meio de projetos simples que não antecipam desnecessariamente futuras mudanças no sistema.

EDUCAÇÃO METODISTA

Extreme Programming - Práticas

rincípio ou prática	Descrição
anejamento incremental	Os requisitos são gravados em cartões de estória e as estórias que serão incluídas em um release são determinadas pelo tempo disponível e sua relativa prioridade. Os desenvolvedores dividem essas estórias em 'Tarefas'. Veja os quadros 3.1 e 3.2.
equenos *releases*	Em primeiro lugar, desenvolve-se um conjunto mínimo de funcionalidades útil, que fornece o valor do negócio. *Releases* do sistema são frequentes e gradualmente adicionam funcionalidade ao primeiro *release*.
ojeto simples	Cada projeto é realizado para atender às necessidades atuais, e nada mais.
esenvolvimento *test-first*	Um *framework* de testes iniciais automatizados é usado para escrever os testes para uma nova funcionalidade antes que a funcionalidade em si seja implementada.
efatoração	Todos os desenvolvedores devem refatorar o código continuamente assim que encontrarem melhorias de código. Isso mantém o código simples e manutenível.
ogramação em pares	Os desenvolvedores trabalham em pares, verificando o trabalho dos outros e prestando apoio para um bom trabalho sempre.

Extreme Programming - Práticas

rincípio ou prática	Descrição
opriedade coletiva	Os pares de desenvolvedores trabalham em todas as áreas do sistema, de modo que não se desenvolvam ilhas de *expertise*. Todos os conhecimentos e todos os desenvolvedores assumem responsabilidade por todo o código. Qualquer um pode mudar qualquer coisa.
tegração contínua	Assim que o trabalho em uma tarefa é concluído, ele é integrado ao sistema como um todo. Após essa integração, todos os testes de unidade do sistema devem passar.
tmo sustentável	Grandes quantidades de horas-extra não são consideradas aceitáveis, pois o resultado final, muitas vezes, é a redução da qualidade do código e da produtividade a médio prazo.
iente no local	Um representante do usuário final do sistema (o cliente) deve estar disponível todo o tempo à equipe de XP. Em um processo de Extreme Programming, o cliente é um membro da equipe de desenvolvimento e é responsável por levar a ela os requisitos de sistema para implementação.

EDUCAÇÃO METODISTA

Extreme Programming

- Em um processo XP, os clientes estão intimamente envolvidos na especificação e priorização dos requisitos do sistema.
- Os requisitos não estão especificados como uma lista de funções requeridas do sistema.
- Pelo contrário, o cliente do sistema é parte da equipe de desenvolvimento e discute cenários com outros membros da equipe.
- Juntos, eles desenvolvem um ‘cartão de estória’, englobando as necessidades do cliente. A equipe de desenvolvimento, então, tenta implementar esse cenário em um release futuro do software.

Extreme Programming

- Na prática, muitas empresas que adotaram XP não usam todas as práticas da Extreme Programming listadas nas tabelas.
- Elas escolhem de acordo com sua organização.
- Por exemplo, algumas empresas consideram útil a programação em pares, outras preferem a programação individual e revisões.
- Para acomodar os diferentes níveis de habilidade, alguns programadores não fazem refatoração em partes do sistema que não desenvolveram, e podem ser usados os requisitos convencionais em vez de estórias de usuários.
- No entanto, a maioria das empresas que adotaram uma variante de XP usa releases de pequeno porte, desenvolvimento do test-first eintegração contínua.

Teste em Extreme Programming

- Uma das diferenças importantes entre o desenvolvimento incremental e o desenvolvimento dirigido a planos está na forma como o sistema é testado.
- Com o desenvolvimento incremental, não há especificação do sistema que possa ser usada por uma equipe de teste externa para desenvolvimento de testes do sistema.
- Como consequência, algumas abordagens para o desenvolvimento incremental têm um processo de testes muito informal em comparação com os testes dirigidos a planos.

Teste em Extreme Programming

- Para evitar alguns dos problemas de teste e validação do sistema, a abordagem XP enfatiza a importância dos testes do programa.
- Extreme Programming inclui uma abordagem de testes que reduz as chances de erros desconhecidos na versão atual do sistema.

Teste em Extreme Programming

- As principais características dos testes em XP são:

1. desenvolvimento test-first*;
2. desenvolvimento de teste incremental a partir de cenários;
3. envolvimento dos usuários no desenvolvimento de testes e validação;
4. uso de frameworks de testes automatizados.

Teste em Extreme Programming

- O desenvolvimento *test-first é uma das mais importantes inovações no XP.
- Em vez de escrever algum código e, em seguida, escrever testes para esse código, você escreve os testes antes de escrever o código.
- Isso significa que você pode executar o teste enquanto o código está sendo escrito e pode encontrar problemas durante o desenvolvimento.

Teste em Extreme Programming

- Em XP, os requisitos do usuário são expressos como cenários ou estórias, e o usuário os prioriza para o desenvolvimento.
- A equipe de desenvolvimento avalia cada cenário e divide-o em tarefas.
- No processo de testes, o papel do cliente é ajudar a desenvolver testes de aceitação para as estórias que serão implementadas no próximo release do sistema.
- Em XP, o teste de aceitação, assim como o desenvolvimento, é incremental. O cliente, que faz parte da equipe, escreve os testes enquanto o desenvolvimento avança. Portanto, todos os novos códigos são validados para garantir que realmente é o que o cliente necessita.

Teste em Extreme Programming

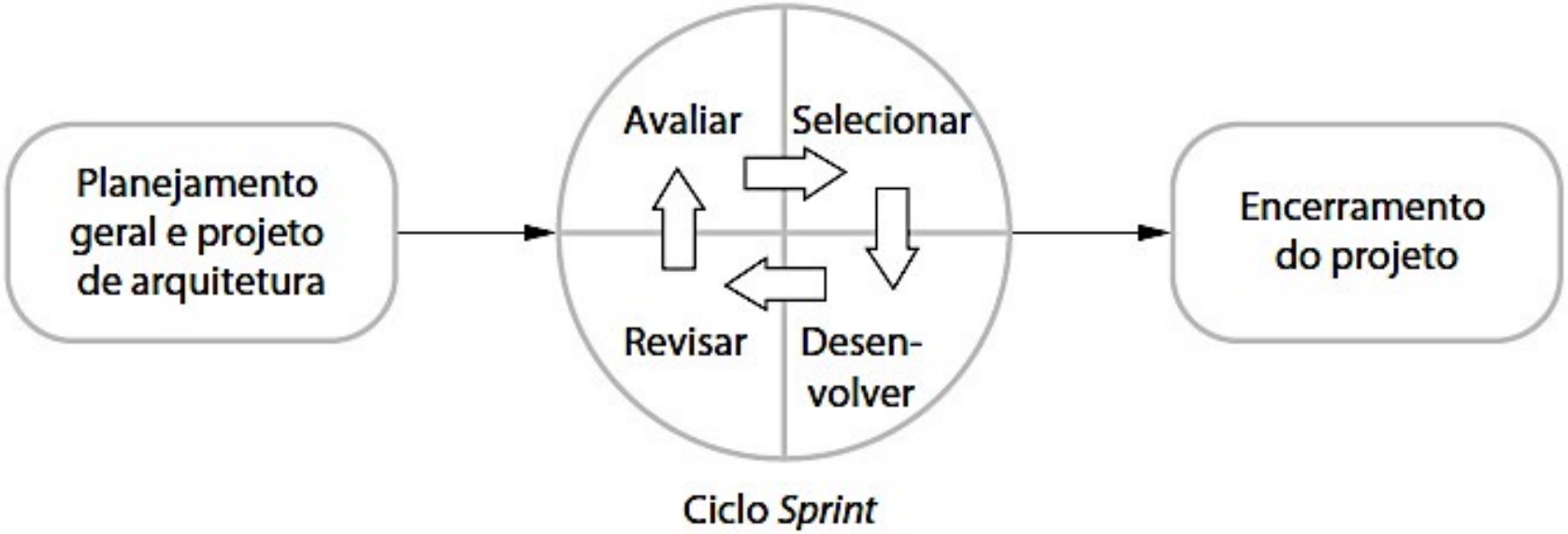
- Automação de testes é essencial para o desenvolvimento test-first.
- Os testes são escritos como componentes executáveis antes que a tarefa seja implementada.
- Esses componentes de teste devem ser autônomos, devem simular a submissão de entrada a ser testada e devem verificar se o resultado atende à especificação de saída.
- Um framework de testes automatizados é um sistema que torna mais fácil escrever os testes executáveis e submeter um conjunto de testes para execução.
- **Junit** é um exemplo amplamente usado de framework de testes automatizados

Scrum

- A abordagem Scrum é um método ágil geral, mas seu foco está no gerenciamento do desenvolvimento iterativo, ao invés das abordagens técnicas específicas da engenharia de software ágil.
- Scrum não prescreve o uso de práticas de programação, como programação em pares e desenvolvimento test-first.
- Portanto, pode ser usado com abordagens ágeis mais técnicas, como XP, para fornecer um framework de gerenciamento do projeto.

Scrum

rocesso Scrum



Scrum

- No Scrum, existem três fases:

1. A primeira é uma fase de planejamento geral, em que se estabelecem os objetivos gerais do projeto e da arquitetura do software.
2. Em seguida, ocorre uma série de ciclos de sprint, sendo que cada ciclo desenvolve um incremento do sistema.
3. Finalmente, a última fase do projeto encerra o projeto, completa a documentação exigida, como quadros de ajuda do sistema e manuais do usuário, e avalia as lições aprendidas com o projeto.

Scrum – *Sprint*

- A característica inovadora do Scrum é sua fase central, chamada ciclos de sprint.
- Um sprint do Scrum é uma unidade de planejamento na qual o trabalho a ser feito é avaliado, os recursos para o desenvolvimento são selecionados e o software é implementado.
- No fim de um sprint, a funcionalidade completa é entregue aos stakeholders.

EDUCAÇÃO METODISTA

Scrum – *Sprint* - Características

1.Sprints são de comprimento fixo, normalmente duas a quatro semanas. Eles correspondem ao desenvolvimento de um release do sistema em XP.

2.O ponto de partida para o planejamento é o *backlog* do produto, que é a lista do trabalho a ser feito no projeto.

Durante a fase de avaliação do sprint, este é revisto, e as prioridades e os riscos são identificados.

O cliente está intimamente envolvido nesse processo e, no início de cada sprint, pode introduzir novos requisitos ou tarefas.

EDUCAÇÃO METODISTA

Scrum – *Sprint* - Características

3. A fase de seleção envolve todos da equipe do projeto que trabalham com o cliente para selecionar os recursos e a funcionalidade a ser desenvolvida durante o sprint.

EDUCAÇÃO METODISTA

Scrum – *Sprint* - Características

4. Uma vez que todos estejam de acordo, a equipe se organiza para desenvolver o software.

Reuniões diárias rápidas, envolvendo todos os membros da equipe, são realizadas para analisar os progressos e, se necessário, repriorizar o trabalho.

Nessa etapa, a equipe está isolada do cliente e da organização, com todas as comunicações canalizadas por meio do chamado '**Scrum Master**'.

EDUCAÇÃO METODISTA

Scrum – *Sprint* - Características

O papel do Scrum Master é proteger a equipe de desenvolvimento de distrações externas.

A maneira como o trabalho é desenvolvido depende do problema e da equipe.

Diferentemente do XP, a abordagem Scrum não faz sugestões específicas sobre como escrever os requisitos ou sobre o desenvolvimento *test-first* etc.

No entanto, essas práticas de XP podem ser usadas se a equipe achar que são adequadas.

EDUCAÇÃO METODISTA

Scrum

- A ideia por trás do Scrum é que toda a equipe deve ter poderes para tomar decisões, de modo que o termo ‘gerente de projeto’ tem sido deliberadamente evitado.
- Pelo contrário, o ‘Scrum Master’ é um facilitador, que organiza reuniões diárias, controla o backlog de trabalho, registra decisões, mede o progresso comparado ao backlog e se comunica com os clientes e a gerência externa à equipe.

Scrum

- Toda a equipe participa das reuniões diárias; às vezes, estas são feitas com os participantes em pé (‘stand-up’), muito rápidas, para a manutenção do foco da equipe.
- Durante a reunião, todos os membros da equipe compartilham informações, descrevem seu progresso desde a última reunião, os problemas que têm surgido e o que está planejado para o dia seguinte.
- Isso garante que todos na equipe saibam o que está acontecendo e, se surgirem problemas, poderão replanejar o trabalho de curto prazo para lidar com eles.
- Todos participam desse planejamento de curto prazo; não existe uma hierarquia top-down a partir do Scrum Master.

Scrum – Caso de dev. SW Telecomunicações

1.O produto é decomposto em um conjunto de partes gerenciáveis e compreensíveis.

2. Requisitos instáveis não atrasam o progresso.

3.Toda a equipe tem visão de tudo, e, consequentemente, a comunicação da equipe é melhorada.

4.Os clientes veem a entrega de incrementos dentro do prazo e recebem feedback sobre como o produto funciona.

5.Estabelece-se confiança entre clientes e desenvolvedores e cria-se uma cultura positiva, na qual todo mundo espera que o projeto tenha êxito.

EDUCAÇÃO METODISTA

Ferramentas CASE

- Ramos (2011, p.27) descreve a ferramenta CASE como “como um conjunto de técnicas e ferramentas informatizadas que auxiliam o engenheiro de software no desenvolvimento de aplicações”.
- As aplicações aqui mencionadas são os sistemas e os softwares.
- As ferramentas CASE têm como objetivo melhorar a eficiência no desenvolvimento de softwares, para a obtenção de softwares com melhor qualidade.

Ferramentas CASE

- Ela permite, durante o desenvolvimento de software, automatizar as atividades, como os próprios testes de execução de software e identificação de problemas.
- Mais especificamente, as ferramentas CASE são muito importantes para empresas que trabalham com Engenharia de Software, pois tratam de todo o ciclo de desenvolvimento, que inclui a análise do projeto, o projeto propriamente dito, a implementação e a realização de testes (RAMOS, 2011).

Ferramentas CASE

Vantagens na utilização das ferramentas CASE:

- Padronização dos processos;
- Reutilização de ferramentas ao longo do projeto, com consequente aumento de produtividade;
- Automatização das atividades, que inclusive é a principal característica dessas ferramentas.
- Tempos de desenvolvimento otimizados;
- Integração entre as diversas etapas e equipes de desenvolvimento;
- Foco em testes e correção do software;
- Qualidade superior para o produto.

Ferramentas CASE - Categorias

As ferramentas CASE são classificadas em três categorias: Lower CASE (back-end), Upper CASE (front-end) e Integrated CASE, união das duas ferramentas anteriores.

- As ferramentas da categoria *lower* CASE trabalham em ambientes mais simples, auxiliando na criação dos códigos dos softwares, dos seus testes, da depuração e da manutenção do mesmo.
- As ferramentas da categoria *upper* CASE trabalham em ambientes mais complexos, as tarefas de análise, de projetos e de geração de código são mais automatizadas do que na categoria *lower* CASE.
- As ferramentas da categoria integrated CASE trabalham integradas em ambientes que relacionam entradas e saídas. Isso permite o controle dos dados de forma consistente.