ISTITUTO FEDERAL GOIANO CAMPUS MORRINHOS

ANA CAROLINA OLIVEIRA SILVA, BIANCA CRISTINA DE SOUSA SAMPAIO BELA, BRUNA ALVES DE MENEZES,
CECÍLIA ANDRADE PEREIRA, JEFERSON CESAR LANDI

EXTREME PROGRAMMING

MORRINHOS
2017

EXTREME PROGRAMMING

trabablo apresentada ao Curso de Desenvol. de Projetos de Sistemas, do curso tecnico integrado em informatica, Istituto Federal Goiano campus Morrinhos, como requisito parcial para a obtenção do título de (Tecnico) em informatica.

Professora: Leila
Coordenador: Odilon

CIDADE

20XX

SUMÁRIO

RESUMO

NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO NOME DO TRABALHO

Objetivo: Avaliar a diferença do comportamento mecânico de tendões solidarizados e não solidarizados para verificar se a solidarização tem função efetiva para reconstrução do ligamento cruzado anterior. **Material e Método:** Vinte tendões digitais bovinos frescos foram usados. Para determinar a área da secção transversal utilizou-se alginato. Dez tendões foram solidarizados seguindo as orientações do fabricante; outros 10 pares não. Foram desenvolvidas garras para fixação dos enxertos à máquina universal para simulação da fixação, sendo a superior bipartida e de passagem controlada dos pinos e a inferior com dentes alternados. **Resultados:** A carga máxima dos corpos de prova não solidarizados foi de 849,4N ± 386,8 a área 30,4 mm^2± 7,7, tensão de 29 ± 17Mpa. Os solidarizados obtiveram carga máxima de 871,8N ± 484,9 área 35 mm^2± 5,8, tensão de 24 ± 10Mpa. Não houve diferença estatística entre os dois grupos ($p>0,05$). **Conclusão**: A distribuição de probabilidade mostra que para 400 N os tendões não solidarizados apresentam confiabilidade de 83,8% e os solidarizados de 78,5%%.

Descritores: Ligamento Cruzado Anterior, Fêmur, Dispositivos de Fixação Ortopédica, Mecânica, Tendões

INTRODUÇÃO

Neste trabalho será descrito uma metodologia ágil, o Extreme Programming. Sobre o uso, as características, vantagens e desvantagens do Extreme Programming. E na sala de aula terá apresentação sobre o mesmo.

E para isso, primeiro é necessário entender o que é uma metodologia ágil.

1 DESENVOLVIMENTO

1.1 METODOLOGIAS ÁGEIS.

Metodologias ágeis existem desde a década de 80, mas algumas informações passam por distorções, fato que dificultou no início a utilização das metodologias. Por consequência, desenvolvedores passaram a entender a metodologia ágil como algo que tudo se pode, ou seja, podemos desenvolver sem documentação, sem padrão e sem cuidado. Isto não é verdade, as metodologias ágeis podem trazer sucesso ao projeto, e são utilizadas inclusive na indústria. Apesar das metodologias existirem, foi em 2001 que um grupo formado por Kent Beck e mais dezesseis renomados desenvolvedores assinaram o MANIFESTO PARA O DESENVOLVIMENTO ÁGIL DE SOFTWARE e o grupo foi batizado de aliança dos ágeis. O manifesto tem a seguinte base:

- Os indivíduos e as interações são mais importantes do que os processos e as ferramentas;
- O software funcionando é mais importante do que uma documentação completa;
- A colaboração com e dos clientes acima de apenas negociações de contratos e;
- Respostas a mudanças acima de seguir um plano.

Isso não significa que a documentação não é importante e que os processos e as ferramentas sejam inúteis, quer dizer que o software funcionando é mais valorizado.

http://www.devmedia.com.br/uma-visao-geral-sobre-metodologia-agil/27944“A engenharia de software ágil combina filosofia com um conjunto de princípios de desenvolvimento. A filosofia defende a satisfação do cliente e a entrega de incremental prévio; equipes de projetos pequenas e altamente motivadas; métodos informais; artefatos de engenharia de software mínimos e, acima de tudo, simplicidade no desenvolvimento geral. Os princípios de desenvolvimento priorizam a entrega mais que a análise e projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e contínua entre desenvolvedores e clientes”. (Pressman, 2011)

Mas, qual o cerne de ser ágil?

Segundo Ivar Jacobson “Atualmente, agilidade tornou-se a palavra da moda quando se descreve um moderno processo de software. Todo mundo é ágil. Uma equipe ágil é aquela rápida e capaz de responder apropriadamente a mudanças. Mudanças têm muito a ver com desenvolvimento de software. Mudanças no software que está sendo criado, mudanças nos membros da equipe, mudanças devido a novas tecnologias, mudanças de todos os tipos que poderão ter um impacto no produto que está em construção ou no projeto que cria o produto. Suporte para mudanças deve ser incorporado em tudo o que fazemos em software, algo que abraçamos porque é o coração e a alma do software. Uma equipe ágil reconhece que o software é desenvolvido por indivíduos trabalhando em equipes e que as habilidades dessas pessoas,

suas capacidades em colaborar, estão no cerne do sucesso do projeto.”

O desenvolvimento ágil é incremental, ou seja, não se faz um plano completo com tudo que devemos fazer para depois iniciar o desenvolvimento, muito menos, desenvolvemos o produto sem contato com o cliente, ao invés disso, desenvolvemos incrementalmente, ou seja, o produto é feito aos poucos e entregue constantemente, desta forma, toda mudança é bem vinda, pois o projeto está em desenvolvimento e não foi concluído por completo.

Segundo Sommerville, os incrementos iniciais do sistema podem fornecer uma funcionalidade de alta prioridade, de forma que os clientes logo poderão obter valor do sistema durante seu desenvolvimento. Os clientes podem assim ver os requisitos na prática e especificar mudanças para serem incorporadas nos releases posteriores do sistema.

1.2 EXTREME PROGRAMMING (XP):

A Extreme Programming (XP) é uma Metodologia Ágil para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente. Criada em 1997, nos Estados Unidos, por Kent Back e Ward Cunningham.

O XP é um método de desenvolvimento de software, leve, não é prescritivo, e procura fundamentar as suas práticas por um conjunto de valores que serão vistos posteriormente no artigo. O XP, diferentemente do que muito pensam, também pode ser adotar por desenvolvedores médios e não apenas por desenvolvedores experientes.

O objetivo principal do XP é levar ao extremo um conjunto de práticas que são ditas como boas na engenharia de software. Entre elas podemos citar o teste, visto que procurar defeitos é perda de tempo, nós temos que constantemente testar. Mas o XP possui mais práticas do que apenas testar, entre as práticas, o XP diz que:

- Já que testar é bom, que todos testem o tempo todo;
- Já que revisão é bom, que se revise o tempo todo;
- Se projetar é bom, então refatorar o tempo todo;
- Se teste de integração é bom, então que se integre o tempo todo;
- Se simplicidade é bom, desenvolva uma solução não apenas que funcione, mas que seja a mais simples possível;
- Se iterações curtas é bom, então mantenha-as realmente curtas.

O XP muda o paradigma, onde não temos o medo da mudança, pois o errar é feito com um baixo custo. Diferente do tradicional em que se diz que quanto mais tarde a mudança, maiores são os custos, e assim sendo nunca devemos fazer mudanças o XP diz que devemos sim estar constantemente fazendo mudanças e não devemos teme-las, principalmente quando seguimos os seus valores e as suas práticas. Outra situação desafiada pelo XP é a engenharia de software que afirma sempre projetarmos para mudança, ou seja, vale despender tempo e esforço antecipando mudanças quando isso reduz custos posteriores no ciclo de vida. No entanto, novamente o XP assume que este esforço não vale a pena quando as mudanças não podem ser confiavelmente previstas, ou seja, não vale a pena empregarmos um grande esforço que pode nem mesmo ser utilizada no agora, no futuro ou nunca.

Para conseguirmos se adaptar as mudanças o XP preconiza ciclos curtos que nos dá previsibilidade e redução de incertezas/riscos, Simplicidade e melhorias constantes de código (refactoring) para facilitar a mudança e Testes Automatizados e Integração Contínua para aumentar a confiança.

2 BOAS PRÁTICAS

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.

2.1 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".

2.2 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.

2.3 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.

2.4 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.

2.5 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.

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.

2.6 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.

2.7 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.

2.8 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.

2.9 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.

2.10 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.

2.11 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.

2.12 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 quem bem entender para equilibrar o trabalho mental e físico.

REFERÊNCIAS

Cohn, Mike. Desenvolvimento de Software com Scrum: Aplicando métodos ágeis com sucesso, Bookman, Porto Alegre, 2011.

Pressman, Roger S. Engenharia de Software: Uma abordagem profissional, Bookman, Porto Alegre, 2011;

Sommerville, Ian. Engenharia de Software, Person, São Paulo, 2010.

http://www.devmedia.com.br, acessado em abril de 2017.