OBSERVATORIO DE LA ECONOMÍA LATINOAMERICANA
revista eumednet.

ISSN: 1696-8352 - BRASIL – MARZO 2017

METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SISTEMAS. ESTUDO DE CASO DE UM SISTEMA PARA SALÃO DE BELEZA UTILIZANDO SCRUM

[1] Danilo Vinícius Roque

[2] Juliano Schimiguel

Para citar este artículo puede utilizar el siguiente formato:

Danilo Vinícius Roque y Juliano Schimiguel (2017): "Metodologias ágeis no desenvolvimento de sistemas. Estudo de caso de um sistema para salão de beleza utilizando SCRUM", Revista Observatorio de la Economía Latinoamericana, Brasil, (marzo 2017). En línea:

http://www.eumed.net/cursecon/ecolat/br/17/scrum.html

RESUMO

Este trabalho pretende verificar como o emprego das metodologias ágeis podem ser úteis no desenvolvimento de software. O objetivo é mostrar o que são e algumas metodologias e suas melhores práticas, dando foco na metodologia Scrum, mostrando seus papéis principais e práticas. O estudo de caso utilizando a metodologia Scrum, pôde evidenciar algumas vantagens e desvantagens da metodologia no processo de desenvolvimento de software.

Palavras-Chave: engenharia de software, processo de software, desenvolvimento de software, metodologias ágeis de desenvolvimento, Scrum.

[1] Assistente de Engenharia de Confiabilidade.

[2] Doutor e Mestre em Ciência da Computação(Unicamp). Professor dos Cursos de Sistemas de Informação, Análise de Sistemas e Engenharia de Produção.

ABSTRACT

This study intends to verify how the use of agile methodologies can be useful in software development. The objective is to show what they are and some methodologies and their best practices, focusing on the Scrum methodology, showing their main roles and practices. The study case using the Scrum methodology was able to show some advantages and disadvantages of the methodology in the process of software development.

Keywords: Software Engineering, software process, software development, agile development methodologies, Scrum.

1. INTRODUÇÃO

O mercado de desenvolvimento de software transformou-se em um dos mais importantes da atualidade. Contudo, o software foi conceituado como umas das propriedades do contexto intelectual de maior valor no mundo e, de fato, produto indispensável para o estabelecimento do estilo de vida contemporânea.

Com o intuito de contribuir com as empresas e suas equipes de desenvolvimento, a Engenharia de Software detectou a necessidade da criação de ferramentas que auxiliassem o processo de desenvolvimento de softwares. Para isso, criou ferramentas como modelo de processos de desenvolvimento de software, guia de melhores práticas, metodologias de desenvolvimento ágil, dentre outras ferramentas auxiliadoras.

A maior parte dos projetos de desenvolvimento de software é desenvolvida sem planejamento ou uma fase organizada de design do sistema, isso resulta uma grande quantidade de erros, que precisam ser resolvidos, em uma longa etapa que sempre estende o prazo inicialmente acordado. As metodologias de desenvolvimento proporcionam as instruções detalhadas para se desenvolver um software. Segundo Abrahamsson(2002), uma metodologia é ágil quando efetua o desenvolvimento de

software de forma incremental, colaborativa, direta e adaptativa, elas envolvem um amplo conjunto de tarefas que incluem: levantamento e análise dos requisitos, projeto, implementação e testes de software. Segundo esse conceito, inclui como metodologias ágeis: Extreme Programming (XP), Scrum, Crystal Methods, Dynamic Systems Development Method (DSM), Feature-Driven Development (FDD), Lean Development (LD), Adaptative Software Development (ASD).

Esse trabalho apresenta o estudo das metodologias de desenvolvimento de software Extreme Programming, Desenvolvimento Guiado por Funcionalidades (FDD) e Scrum, escolhidas por serem duas metodologias bastante utilizadas no âmbito acadêmico e também nas empresas e software houses, o estudo de caso será feito através do desenvolvimento de um sistema para salão de beleza, utilizando a metodologia Scrum.

2. EXTREME PROGRAMMING

O Extreme Programming é uma metodologia de desenvolvimento que tem como objetivo criar software de melhor qualidade, sendo produzidos em menos tempo e de forma mais econômica que o habitual(BASSI, 2014) . A metodologia nasceu no final dos anos 90, e tem como pai Kent Beck.

Segundo Beck(2000), Extreme Programming *"é uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e em constante mudança".*

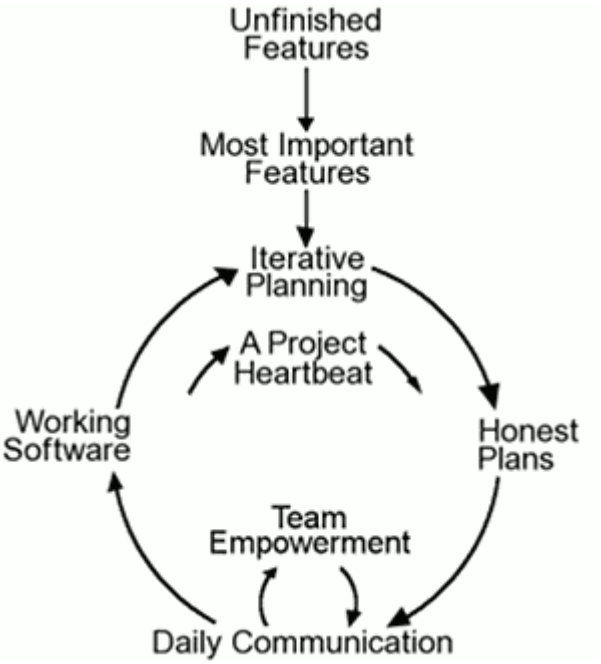
XP é bem sucedido porque enfatiza a satisfação do cliente. Ao invés de entregar todo o projeto em uma data, esse processo oferece o software que você precisa e como precisa. Extreme Programming capacita seus adeptos para responder com confiança às mudanças de necessidades dos clientes, mesmo no final do ciclo de vida de desenvolvimento. Ele enfatiza o trabalho em equipe. Gerentes, clientes e desenvolvedores são todos iguais em uma equipe colaborativa, implementando um

ambiente simples e eficaz, permitindo que as equipes se tornem altamente produtivas, o time se organiza em torno do problema, para resolvê-lo da melhor forma possível.

Extreme Programming pode melhorar um projeto de quatro maneiras essenciais: comunicação, simplicidade, feedback, e coragem(WELLS, 2009). Os desenvolvedores comunicam-se constantemente com seus clientes e colegas de equipe eles mantêm o projeto simples e limpo, eles obtêm o feedback testando o software dia a dia, entregam o sistema para os clientes assim que possível e implementam as mudanças sugeridas.

A figura 1 mostra como as regras de Extreme Programming trabalham juntas. Os clientes gostam de estar junto no processo de desenvolvimento do software, desenvolvedores contribuem ativamente, independente do nível de experiência, e os gestores se concentram em comunicação e relacionamentos. Atividades improdutivas são evitadas para redução de custos e frustração de todos os envolvidos.

Figura 1 – Extreme Programming Fluxogram



Fonte: http://www.extremeprogramming.org/

2.1 Quatro Valores de XP

Segundo Beck(2000), *"Seremos bem sucedidos quando termos um estilo que celebra um conjunto consistente de valores que atendam às necessidades humanas e comerciais: comunicação, simplicidade, feedback e coragem"*.

Em algumas ocasiões, os interesses comuns entram em divergência com os interesses pessoais. As sociedades aprenderam a conviver com este problema criando conjuntos de valores. Através destes valores, as principais ideias das sociedades são transmitidas com eficiência.

XP apresenta um conjunto de quatro valores. Estes são: comunicação, simplicidade, feedback e coragem.

2.1.1 Comunicação

O primeiro valor de XP é a comunicação. As adversidades encontradas nos projetos podem ser invariavelmente rastreadas até um ponto onde uma pessoa não falou alguma coisa para outra, sobre alguma coisa importante. O programador pode não ter dito a alguém sobre uma mudança fundamental no design, às vezes o programador não faz a pergunta certa ao cliente, fazendo com que uma decisão importante seja prejudicada, às vezes o gestor não faz ao programador a pergunta certa, e o progresso do projeto é mal reportado.

Má comunicação não acontece por acaso. Há várias circunstâncias que conduzem a um problema de comunicação, como por exemplo, um cliente ao ser ignorado ao informar algo importante ao programador.

XP visa estimular a comunicação nos projetos através da adesão de práticas que não podem ser realizadas sem comunicação.

2.1.2 Simplicidade

Simplicidade em XP refere-se a desenvolver apenas o necessário para atender as necessidades do cliente, deixando de lado qualquer funcionalidade não essencial.

Ainda que tal funcionalidade esteja ligada com o avanço do produto, ela deve ser descartada até que se tenha certeza da sua necessidade. Assume-se assim a seguinte estratégia: desenvolver algo simples hoje e fazer modificações futuramente, ao invés de desenvolver algo completo hoje e correr o risco de não ser utilizado no futuro (SOARES, 2004).

Comunicação e simplicidade possuem um relacionamento correspondente. Quanto mais se comunica, mais explícito fica o que precisa ser feito, e consequentemente, mais explícito fica o que não precisa ser feito. Quanto mais simples o software, menos comunicação ele requer. Quanto mais se comunica, mais simples o software se apresenta (BECK, 2000).

2.1.3 Feedback

Segundo Sensangent(2009), feedback é: *"Um mecanismo de comunicação dentro de um sistema em que o sinal de entrada gera uma resposta de saída que retorna para influenciar a atividade continuada ou a produtividade desse sistema."*

No desenvolvimento de software, feedback é o processo de troca de informações que ocorre entre o cliente e a equipe de desenvolvimento durante a produção de um software.

O feedback no projeto, é o que permite a orientação do projeto na melhor direção em que ele pode andar.. Não é possível acompanhar um processo sem ter como avalia-lo(BECK, 2000). Por isso em XP, várias práticas proporcionam vários tipos de feedback diferentes, procurando tornar o desenvolvimento o mais transparente possível a gestores, desenvolvedores e clientes.

O feedback concreto trabalho junto com a comunicação e a simplicidade. Não tem como obter feedback sem comunicação. Porém, a comunicação eficaz faz o feedback acontecer de forma natural. Sistemas simples são mais simples de testar, e consequentemente de avaliar, por isso contribuem para um feedback mais eficaz.

2.1.4 Coragem

De acordo com Soares (2004) é preciso coragem para implantar comunicação entre desenvolvedores e clientes, pois não são todas as pessoas que possuem facilidade de comunicação e conseguem manter um bom relacionamento. É necessário também coragem para aplicar *refactoring* em um código que está funcionando corretamente somente para torna-lo mais simples.

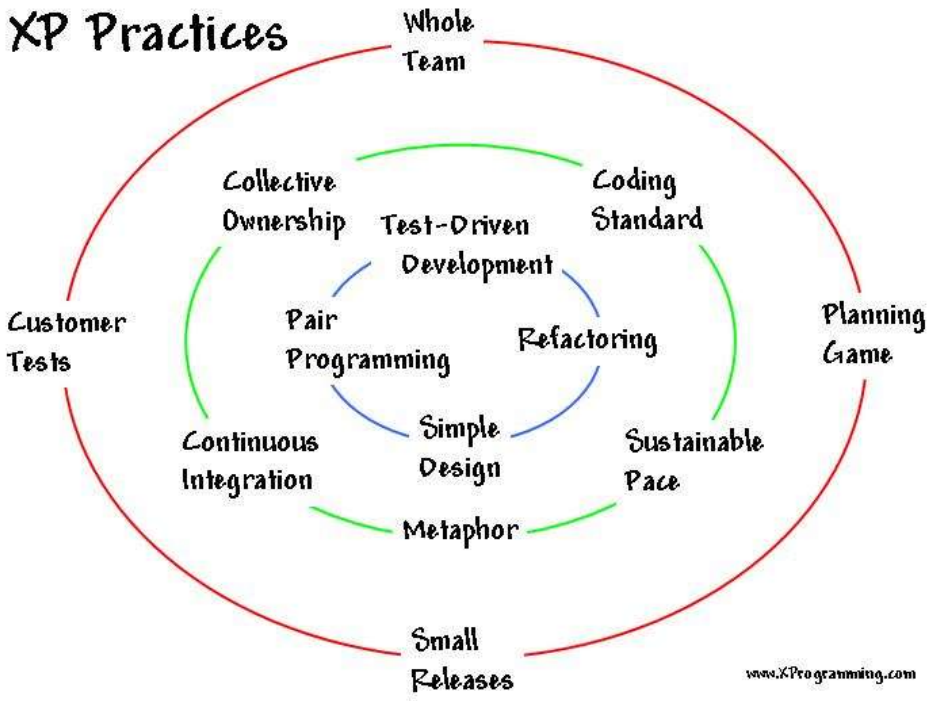
A coragem, quando combinada com os outros valores, é necessária para que o processo seja executado de forma aprazível, porém, ter coragem sem ter comunicação, simplicidade e feedback, pode ser um tipo de ignorância. A comunicação dá suporte à coragem, pois se comunicando melhor pode-se conhecer melhor os riscos envolvidos. A simplicidade também está ligada, porque você pode se dar ao luxo de ser muito mais corajoso em um sistema simples. Coragem suporta simplicidade porque assim que você vê a possibilidade de simplificar o sistema, irá fazê-lo. O feedback coopera para uma avaliação rápida das hipóteses, e até mesmo de decisões tomadas.

2.2 Práticas do XP

"As práticas se apoiam mutuamente. A fraqueza de um é coberta pelos pontos fortes de outros" (BECK, 2000).

A partir dos valores de XP, que foram citados acima, temos o conjunto de práticas do XP: *The Planning Game* (Planejamento), *Releases* pequenas, Metáfora, Desenho simples, Testes automatizados, *Refactoring,* Programação em pares, Propriedade Coletiva, Integração Contínua, Semana de 40 horas, Cliente *On-site* e Padrões de Programação. As práticas reforçam os valores (BECK, 2000).

Figura 1 – Práticas do XP.



Fonte: www.XProgramming.com

The Planning Game (Planejamento)

O planejamento em XP aborda duas questões-chave no desenvolvimento de software: prever o que será realizado até a data de vencimento e determinar o que será feito a seguir. Para fazer o planejamento é necessário conciliar decisões técnicas e gerenciais, de forma que sejam tomadas decisões que satisfazem as necessidades do projeto como um todo.

As responsabilidades no planejamento podem ser divididas entre responsabilidades técnicas e gerenciais (BECK, 2000). Entre as responsabilidades técnicas, podemos destacar estimativas de tempos para as solicitações, as consequências de algumas decisões gerenciais, organização da equipe de desenvolvimento e cronograma detalhado. Entre as responsabilidades gerenciais, podemos destacar o escopo das funcionalidades, datas importantes do projeto.

Releases Pequenas

As equipes que utilizam XP utilizam releases pequenos de duas maneiras importantes: primeiro, a equipe disponibiliza o software testado, oferecendo valores de

negócios escolhido pelo cliente, a cada iteração. O cliente pode usar esse software para qualquer finalidade, seja de avaliação ou mesmo de liberação para usuários finais (recomendado). O aspecto mais importante é que o software é visível, e entregado ao cliente, ao fim de cada iteração. Isso mantém tudo aberto e tangível. Em segundo lugar, as equipes de XP também lançam seus usuários finais com frequência.

Metáfora

Em XP, as equipes desenvolvem uma visão comum de como o sistema funciona, chamamos isso de “metáfora”. A metáfora é uma descrição simples de como o sistema funciona, ou ele como um todo. Um exemplo de uma metáfora é ilustrar um editor de textos como uma máquina de escrever.

A metáfora é uma ideia que permite às pessoas entenderem os principais objetivos das funcionalidades que compõe o sistema, é uma linguagem comum que todos devem possuir. O uso da metáfora pode permitir uma boa e fácil comunicação entre as pessoas dentro do projeto.

Design simples

As equipes que utilizam XP constroem software para um design simples, mas sempre adequado. Eles começam simples, e por meio de testes dos desenvolvedores e melhoria de design, eles mantêm dessa forma. Uma equipe de XP mantém o design exatamente adequado para a atual funcionalidade do sistema.

O melhor design para qualquer sistema deve atender aos seguintes requisitos:

- Deve funcionar em todos os testes;
- Não possuir uma lógica ambígua;
- Deve demonstrar o objetivo do desenvolvedor;
- Deve ter o mínimo de classes possíveis.

Testes automatizados

Extreme programming é obcecado pelo feedback, e no desenvolvimento de software, o bom feedback requer um bom teste. As principais equipes de XP praticam o “desenvolvimento orientado a testes”, trabalhando em ciclos muito curtos de adição de um teste, e em seguida, fazendo com que ele funcione. Quase sem esforço, as equipes produzem código com quase 100% de cobertura de teste, o que é um grande passo. Com isso o sistema vai ficando bem mais confiável a medida que evolui.

Refactoring

Ao implementar uma característica em um software, o desenvolvedor procura a forma mais simples de fazer as modificações necessárias. Depois de feita a modificação, o desenvolvedor deve confirmar se é possível simplificar o sistema de forma que todos os testes continuem rodando, isso é *refactoring* (BECK, 2000).

Se um desenvolvedor vê uma maneira fácil de incluir uma função ao software em um minuto e uma maneira em que ele tenha que refazer uma parte do software em dez minutos de forma que ele fique simples e atenda aos testes, ele deve gastar os dez minutos para que o sistema permaneça simples.

Programação em Pares

Em XP, todos os softwares são desenvolvidos por dois desenvolvedores, sentados lado a lado, na mesma máquina. Essa prática garante que todo o código é revisado pelo menos por outro desenvolvedor e resultam em melhor design, melhor teste e código melhor.

Existem dois papéis em cada par. Enquanto o que está com o teclado e mouse vai pensando na melhor maneira de implementar os métodos, o outro está pensando de forma estratégica: Toda essa abordagem vai funcionar? Existe alguma forma de simplificar o sistema de forma que o problema atual apenas desapareça?

Os pares também devem revezar, entre si (quem fica no comando), e também revezar entre o time de desenvolvedores. Se um desenvolvedor precisa desenvolvedor uma função que ele não tem experiência, ele precisa encontrar alguém que conheça e tenha experiência com essa função, para programarem em dupla. Com isso os dois irão adquirir conhecimento nessa parte do sistema.

Propriedade Coletiva

Em XP, qualquer programador pode melhorar o código a qualquer momento. Isso significa que todo o código recebe atenção de muitas pessoas, o que aumenta a qualidade do código e reduz os defeitos.

A propriedade coletiva poderia ser um problema se as pessoas trabalhassem cegamente em um código que não entendessem. O XP evita esses problemas através de duas técnicas-chave: o programador encontra determinados erros, e a programação em pares diz que a melhor maneira de trabalhar com código desconhecido é se juntar com o especialista. Além de garantir boas modificações, quando necessário, esta prática dissemina o conhecimento com toda a equipe.

Integração contínua

Novas versões são integradas e testadas com pequenos intervalos (às vezes poucas horas). Uma forma simples de implementar, isto é, ter uma máquina de integração que não pertença especificamente a nenhum programador (BECK, 2000).

O benefício desta prática pode ser visto pensando em projetos que você pode ter ouvido falar (ou mesmo ter feito parte), onde o processo de construção foi semanal ou com menos frequência, onde tudo quebrou e ninguém sabia o porquê.

Na integração contínua, quando um par termina de escrever uma funcionalidade, ele deve aguarda até que a máquina onde é feita a integração esteja livre, e quando ela estiver livre, eles carregam a última versão do código fonte na máquina, colocam suas alterações e executam os testes, para garantir que não haja erros. Se houver erros,

eles devem ser corrigidos para que a versão que foi colocada se torne a versão corrente. Se não for possível remover os erros, a versão anterior deve ser recolocada na máquina de integração.

Semana de 40 horas

Trabalho extra indica que podem existir problemas no projeto. Em XP, a regra é a seguinte: não se deve fazer hora extra por mais de uma semana, esse regime pode ser útil para alcançar objetivos e reparar possíveis erros. Mas mais do que isto, indica que existem problemas no projeto que não podem ser resolvidos simplesmente por trabalhar mais horas que o normal.

Cliente On-Site

Um real cliente (um futuro usuário) do sistema, deve se sentar com a equipe para tirar dúvidas e definir prioridades de pequena escala.

A grande objeção a essa prática, que esse usuário do sistema pode gerar gastos, os gestores devem avaliar o que vale mais a pena: ter o software funcionando de maneira correta em menos tempo, ou não ter esse usuário e correr o risco de demorar mais para ficar pronto.

Padrões de Programação

As equipes que utilizam XP seguem um padrão de codificação comum, de modo que todo o código no sistema parece ter sido escrito por um único indivíduo. As especificações do padrão não são importantes, o que é importante é que o código parece familiar a todos, em apoio à propriedade coletiva.

Práticas separadas por categoria

As práticas do XP, podem ser separadas por categoria na tabela 1, é mostrado como pode ser feita essa divisão:

Tabela 1 – Práticas do XP separadas por categoria.

Categoria	Pratica
Desenvolvedores	Design Simples, Testes automatizados, Padrões de Programação.
Equipe	Propriedade coletiva, Integração contínua, Metáfora, Padrões de Programação, Semana de 40 horas, Programação em pares, Releases pequenas
Processos	Cliente On-Site, Testes automatizados, Releases pequenas, Planning Game

Fonte: Extreme Programming-Explored, by William Wake.

3. DESENVOLVIMENTO GUIADO POR FUNCIONALIDADES(FDD)

FDD é um método de desenvolvimento ágil e altamente adaptável, ele é:

- Curto e interativo: com resultados a cada duas semanas ou menos;
- Enfatiza a qualidade em todas as etapas;
- Proporciona resultados de trabalhos frequentes e tangíveis em todas as etapas;
- Fornece informações precisas e significativas de progresso e status, com o mínimo de sobrecarga e interrupção para os desenvolvedores;
- É apreciado por clientes, gerentes e desenvolvedores.

O FDD surgiu em 1997, quando Jeff De Luca foi gerente de projeto de um grande projeto de desenvolvimento em Singapura. O problema era tão complexo que Jeff percebeu que a tarefa em mãos não poderia ser concluída no prazo, com os recursos disponíveis, usando estratégias tradicionais de desenvolvimento de software. Ele, portanto, com a ajuda de Peter Coad, descobriu a modelagem em técnica de cor e o conceito de desenvolvimento guiado por funcionalidades. A versão impressa foi publicada inicialmente em 1999 no livro “Java Modeling in Color with UML”, de Peter Coad(COAD, 1999).

3.1 Práticas do FDD

FDD combina uma série de práticas que são fundamentais e definem o FDD:

Modelagem em Objetos de Domínio

Consiste na construção de diagramas de classes, que descrevem os tipos significativos de objetos dentro de um domínio e suas relações. É uma forma de análise de objetos. O problema é dividido em objetos significativos que estão envolvidos. O design e a implementação de cada objeto ou classe identificada no modelo, é um problema menor a ser resolvido. Quando as classes completas são combinadas, elas formam a solução para um problema maior. “Modelagem em cores” é a melhor técnica para modelagem de objetos de domínio.

UML em Cores

A UML em cores é UML habitual com classes codificadas por cores. Todas as classes são divididas em categorias diferentes, cada categoria tem sua própria cor. As classes e interfaces auxiliares, são incolores.

A figura 3 mostra um pedaço de um diagrama UML colorido.

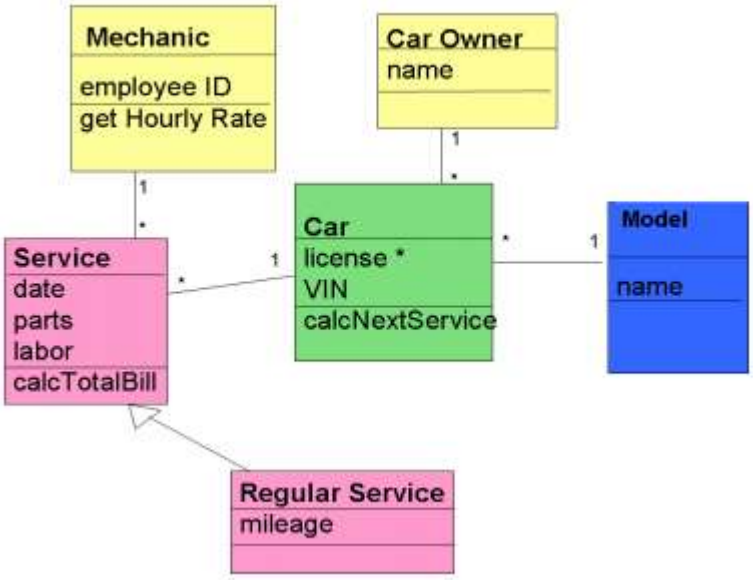
Amarelo: um papel a ser desempenhado, normalmente por uma pessoa ou organização. Por exemplo, o usuário de um site de leilão on-line pode desempenhar papéis diferentes: pode ser um comprador, um vendedor ou um administrador do sistema.

Azul: uma descrição semelhante a um catálogo. Por exemplo, uma loja on-line pode ter descrições dos CD players que vende. A descrição dá todas as características do tocador, mas não é o tocador.

Verde: uma festa, lugar ou coisa. No exemplo anterior, o leitor de CD em estoque seria modelado como verde. A cor verde normalmente tem alguns atributos de identificação, como número de série, nome de pessoa, etc.

Rosa: um intervalo de tempo normalmente associado a algum processo de negócios. Por exemplo, a compra pode ser mostrado na cor rosa, uma vez que tem um tempo de venda, que é seguido pela loja on-line.

Figura 3 – Exemplo de UML colorida.



Fonte: FELSING, A Pratical Guide to Feature-Driven Development, 2002.

Desenvolvendo por Característica

Qualquer função que seja complexa demais para ser implantada dentro de duas semanas é decomposta para funções menores até que cada subproblema seja pequeno o suficiente para ser chamado uma característica.

Propriedade Individual

A propriedade de código em um processo de desenvolvimento denota quem(pessoa ou função) é em última instância responsável pelo conteúdo de uma classe (pedaço de código). FDD usa propriedade individual – os desenvolvedores são atribuídos a propriedade de um conjunto de classes de objetos de domínio.

Times da Característica

A implementação de uma característica pode envolver mais de uma classe, portanto, mais de um proprietário. Assim, o chefe programador recruta os proprietários das classes que serão utilizadas. Esse grupo de proprietários é chamado de time da característica.

Inspeções

Inspeções é a forma de verificação de qualidade, do código e do projeto, aplicando as técnicas de detecção de defeitos mais conhecidas e incitando as oportunidades para propagar boas práticas, convenções e cultura de desenvolvimento.

Builds Regulares

Em um determinado período de tempo fixo devem ser compiladas as características já terminadas, permitindo a verificação de erros e também disponibilizando uma versão atual ao cliente. Isso assegura que sempre há um sistema demonstrável disponível.

Gerenciamento de Configurações

Serve para identificar as versões mais recentes dos códigos-fonte concluídos, e fornece o histórico de todas as informações no projeto.

Reportar resultados

Ao longo do projeto são feitos relatórios de progresso, que permitem o conhecimento do progresso do projeto.

4. SCRUM

Segundo Schwaber (2011), “*Scrum* é um *framework* dentro do qual pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível”.

A metodologia Scrum não é um processo ou uma técnica para a construção de produtos, mas sim um framework em que é possível empregar vários processos e técnicas (SCHWABER & BEEDLE, 2002).

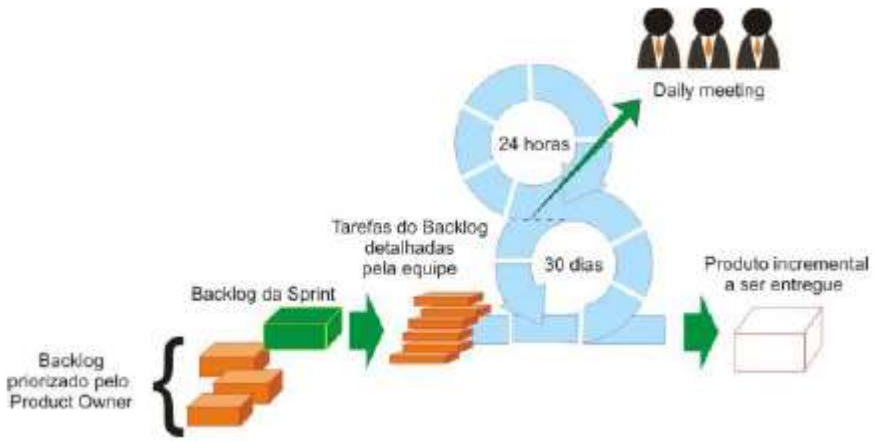
4.1 Metodologia

O Scrum é constituído por algumas características: flexibilidade dos resultados, flexibilidade dos prazos, times pequenos, revisões constantes, colaboração, orientação a objetos. Esse método não exige ou propicia qualquer técnica particular para a fase de desenvolvimento, apenas determina conjuntos de regras e práticas que devem ser adotadas para o sucesso de um projeto (CARVALHO & MELLO, 2009).

Scrum não é um método previsível, ele não especifica o que fazer em toda situação. É usado em trabalhos complexos, nos quais não é possível presumir tudo o que pode acontecer, e oferece um *framework* e uma série de hábitos que torna tudo exposto. Isso permite aos praticantes de Scrum saber rigorosamente o que está acontecendo ao longo do projeto e fazer os devidos ajustes para manter o projeto se movendo ao longo do tempo visando alcançar seus objetivos (SCHWABER, 2004).

Em Scrum, todo o desenvolvimento é feito em iterações. Todo o trabalho é orientado de forma que seja apresentado um novo conjunto de funções ao final de cada iteração. Cada iteração tem um período de tempo definido.

Figura 4 – Ciclo de desenvolvimento do Scrum



Fonte: PEREIRA, Ciclo do Scrum

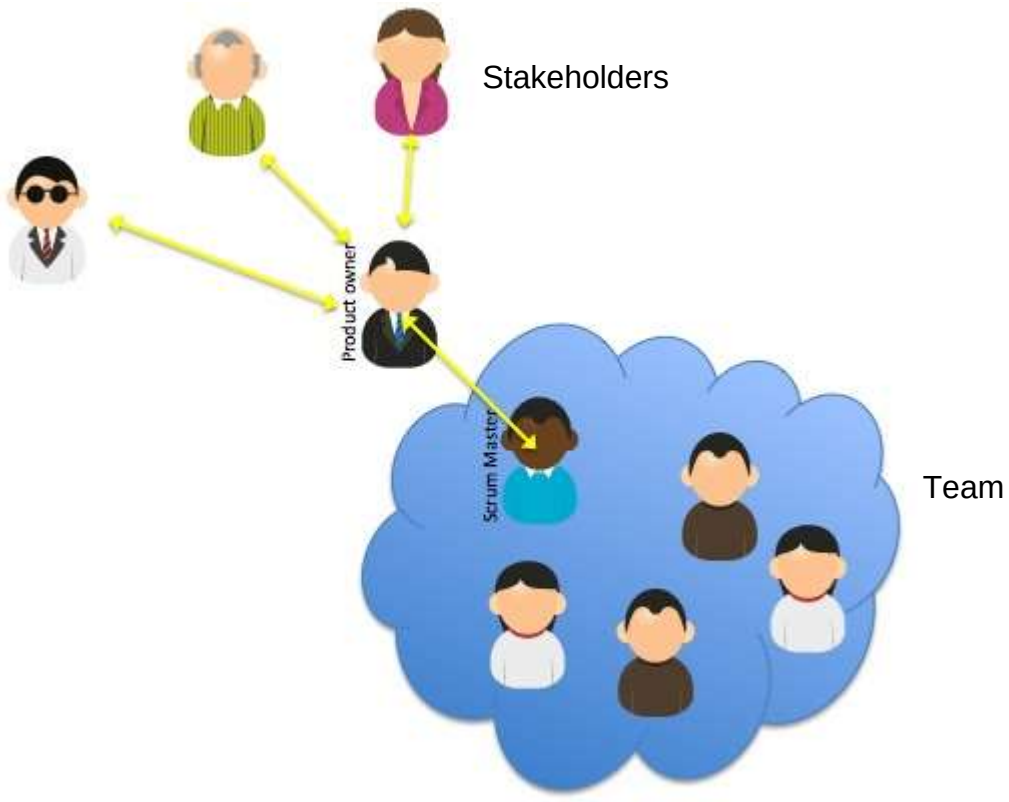
A figura 4 apresenta o ciclo de desenvolvimento de um projeto que utiliza Scrum.

O ciclo do Scrum tem sua evolução fundamentada em algumas iterações predeterminadas, cada uma com duração de 2 a 4 semanas, chamadas de *Sprints.* Antes de cada *Sprint*, é feita uma reunião de planejamento onde a equipe de desenvolvedores tem contato com o cliente para definir o trabalho que irá ser realizado, selecionar e avaliar as tarefas que o time irá fazer dentro da *Sprint* (PEREIRA, 2007).

4.2 Papéis no Scrum

No Scrum existem três papéis com tarefas e responsabilidades durante o processo e suas práticas, são eles: *Product Owner, Team,* e *Scrum Master.* Juntos, são conhecidos como *Scrum Team*, um exemplo do *Scrum Team* e seus papéis, pode ser visto na figura 5.

Figura 5 – Scrum Roles & Stakeholders.



Fonte: http://www.scrum-institute.org/Scrum_Roles_The_Scrum_Team.php

Product Owner

O *Product Owner* é o responsável por representar os interesses dos *stakeholders* no projeto, é responsável por maximizar o retorno do investimento, identificando as características do produto, traduzindo-os em uma lista com prioridades, definindo qual deve estar no topo da lista para o próximo Sprint e continuamente redefinindo e refinando a lista. O *Product Owner* tem responsabilidade de lucro e perda do produto, assumindo que é um produto comercial. No caso de uma aplicação interna, o *Product Owner* não é o responsável pelo retorno do investimento, como no caso de um produto comercial, mas eles são responsáveis pelo retorno no sentido da escolha dos itens de maior valor.

Team

O *Team*(equipe de desenvolvimento) é quem faz o produto que o *Product Owner* indica: uma aplicação ou website, por exemplo. A equipe em Scrum tem normalmente de 6 a 10 pessoas auto organizáveis, auto gerenciáveis e multifuncionais (DEEMER, BENEFIELD, LARMAN, VODDE, 2012). Eles são quem decidem quantos itens(a partir da lista feita pelo *Product Owner*) irão fazer em um Sprint, e a melhor maneira de atingir o objetivo.

Na equipe não há uma divisão através de papéis tradicionais como analista de negócios, DBA, arquiteto de software, analista de testes ou programador. Eles trabalham juntos e são responsáveis por completar cada Sprint.

Scrum Master

O Scrum Master é o responsável pelo processo Scrum, ele deve garantir que o projeto seja realizado conforme as regras e práticas do Scrum, e deve ensinar todas as pessoas envolvidas no projeto (DEEMER, BENEFIELD, LARMAN, VODDE, 2012).

O Scrum Master não é o gerente dos membros da equipe, nem um gerente de projetos, líder da equipe ou representante da equipe, o Scrum Master serve a equipe.

Ele ajuda a remover empecilhos, protege o time de interferências externas e ajuda a equipe a adotar práticas modernas de desenvolvimento, e também deve ajudar a equipe para que ela não se comprometa excessivamente com relação ao que é capaz de fazer durante um Sprint.

A função de Scrum Master pode ser exercida por qualquer membro do time, mas normalmente é exercida por um gerente de Projetos.

4.3 Práticas no Scrum

A seguir, serão detalhadas alguns detalhes sobre as principais práticas de Scrum.

Sprint

Sprints são as iterações de um ciclo de desenvolvimento em Scrum. Cadas Sprint renderá um incremento do produto à medida em que a equipe trabalha para a versão final. Um Sprint é definido de uma semana até um mês, e a equipe tem essa quantia de tempo para implementar as funções no *Sprint Backlog.*

Product Backlog

O *Product Backlog* é uma lista de itens contendo tudo o que é necessário para que o produto seja considerado como completo. Um *Product Backlog* é dinâmico, isso significa que, à medida que o produto é desenvolvido e uma melhor compreensão dele é adquirida, o número de itens na lista aumenta ou diminui, de acordo com sua prioridade.

O proprietário do *Product Backlog* é o *Product Owner*, o Scrum Master, a equipe e outros *stakeholders* contribuem para que ela tenha uma completa lista de tarefas pendentes. A figura 6 mostra um exemplo de como é o *Product Backlog*.

Figura 6 – Exemplo de Product Backlog.

ToDo List

ID	Story	Estimation	Priority
7	As an unauthorized User I want to create a new account	3	1
1	As an unauthorized User I want to login	1	2
10	As an authorized User I want to logout	1	3
9	Create script to purge database	1	4
2	As an authorized User I want to see the list of items so that I can select one	2	5
4	As an authorized User I want to add a new item so that it appears in the list	5	6
3	As an authorized User I want to delete the selected item	2	7
5	As an authorized User I want to edit the selected item	5	8
6	As an authorized User I want to set a reminder for a selected item so that I am reminded when item is due	8	9
8	As an administrator I want to see the list of accounts on login	2	10
Total		**30**	

Fonte: http://www.scrum-institute.org/The_Scrum_Product_Backlog.php

Sprint Backlog

O *Sprint Backlog* é uma lista de atividades identificadas pelo time do Scrum, que deve ser concluída durante o Sprint. Durante a reunião de planejamento(*Sprint Planning Meeting)*, e equipe seleciona alguns itens do *Product Backlog*, e identifica as tarefas necessárias para completar cada funcionalidade. As equipes também estimam quanto tempo irá demorar para cada tarefa ser concluída por algum membro da equipe.

O *Sprint Backlog* é normalmente mantido como uma tabela. Na tabela 2 temos um exemplo de *Sprint Backlog:*

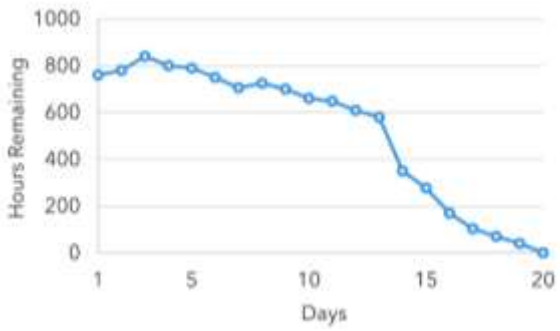
Tabela 2 – Exemplo de Sprint Backlog.

Tarefas	Responsável	Estimativa	Status	Dia 1	Dia 2	Dia 3	Dia 4
Codificar interface	João	5	Complete	5	2	0	0
Codificar camada de integração	Marcos	2	In Progress	2	2	1	0
Testar camada de integração	Mateus	1	Started	1	1	0	0
Criar classe reserva		2	Not Started	2	2	2	2
Criar serviço de busca		4	Not	4	4	4	4

			Started				

Durante um Sprint, é esperado que os membros da equipe atualizem o *Sprint Backlog* conforme as tarefas são completadas e quanto tempo a equipe acredita que será necessário para completar as tarefas que não estão prontas. Uma vez por dia, o Scrum Master faz uma estimativa do trabalho que ainda falta ser feito no Sprint, e gera um gráfico *Burndown*, como o da figura 7.

Figura 7 – Exemplo de Burndown chart.



Fonte: https://www.mountaingoatsoftware.com/agile/scrum/sprint-backlog

4.4 Reuniões

O desenvolvimento de software é um processo complexo que requer constantemente um alto nível de comunicação. O Scrum utiliza reuniões diárias e semanais para garantir que haja comunicação entre o time.

Sprint Planning Meeting

No Scrum, a *Sprint Planning Meeting* (reunião de planejamento do Sprint) é assistida pelo *Product Owner*, Scrum Master e toda a equipe do Scrum. Outros participantes podem participar por convite da equipe. Durante a reunião, o *Product Owner* descreve as funcionalidades de maior prioridade para a equipe, e a equipe faz perguntas suficientes para priorizar o trabalho que precisa ser feito, selecionar e estimar as tarefas que o time pode realizar dentro da Sprint (PEREIRA, 2007).

Daily Scrum Meeting

No Scrum, a cada dia de um Sprint, o Scrum Master realiza uma reunião chamada de *Daily Scrum Meeting.* Elas duram até 15 minutos e tem como objetivo informar a todos os componentes da equipe o que cada um está fazendo. Preferencialmente, essas reuniões são feitas todos os dias, na parte da manhã, para ajudar a definir as prioridades do dia de trabalho.

Durante a reunião, cada membro da equipe responde às seguintes perguntas:

- O que você fez ontem?
- O que você fará hoje?
- Há algo que possa te impedir de fazer?

Através dessa reunião, o Scrum Master e o time, podem entender o status do projeto e suas perspectivas. Os problemas informados pelo time devem ser atacados pelo Scrum Master para diminuir as situações em que o time fica impedido de produzir.

5. ESTUDO DE CASO

Neste capítulo serão apresentados os resultados obtidos no desenvolvimento de um software utilizando Scrum.

5.1 Sistema para gerenciamento de Salão de Beleza

Não existe um sistema consolidado e conhecido no mercado que faça o gerenciamento de um salão de beleza, vendo essa necessidade e com o surgimento da demanda foi obtida a ideia de desenvolver um sistema que atendesse as necessidades de um profissional da área. O sistema deverá realizar todo o gerenciamento dos clientes, funcionários, gastos e agendamento de serviços.

5.2 Por que Scrum?

Existem diversas metodologias diferentes para o desenvolvimento ágil de software, cada uma com sua particularidade. Foi escolhido Scrum para o projeto, pois é

uma metodologia que eu já tenho uma certa prática, por já trabalhar com ela e que considero eficiente.

5.3 Desenvolvimento do Sistema

O software é uma aplicação desktop para Windows, desenvolvida em C#, utilizando o Visual Studio como IDE. O banco de dados utilizado foi SQL Server.

Para desenvolvimento do sistema, foi criado o *Product Backlog,* conforme a tabela 3, onde o “ID” é o identificador único do item de backlog, “Atividade” é o nome da atividade, “Descrição” é a descrição do que deve ser feito e “Prioridade” determina a prioridade dessa atividade. Na tabela 3 temos o *Product Backlog* feito no início do projeto.

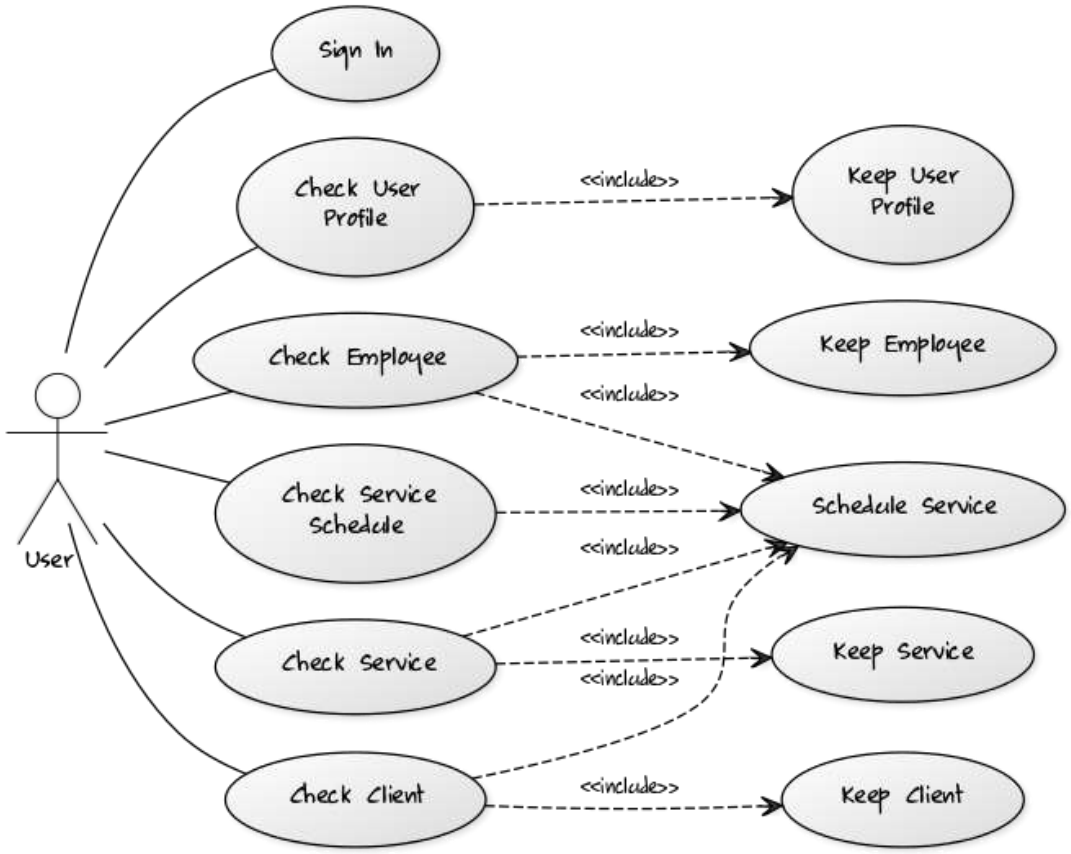
Tabela 3 – Product Backlog.

Product Backlog			
ID	Atividade	Descrição	Prioridade
1	Criação do Banco de dados	Criação do banco de dados, suas tabelas e respectivas relações	9
2	Consultas SQL	Criação do Scripts SQL que o sistema irá utilizar	7
3	Sistema de Autenticação	Desenvolvimento do sistema de autenticação e cadastramento dos usuários	4
4	Incluir usuário	Inclusão de usuários e/ou clientes no sistema	6
5	Alterar usuário	Alteração de usuários e/ou clientes no sistema	6
6	Excluir usuário	Exclusão de usuários e/ou clientes no sistema	6
7	Modificar usuário	Modificação de usuários e/ou clientes no sistema	6
8	Configuração da Agenda	Configuração da agenda para inserção e manutenção dos dados	8
9	Financeiro	Criação de módulo financeiro para o salão, incluindo informações de gastos e lucros	8

Após a conclusão do *Product Backlog*, foram definidos 4 *Sprints*, para o primeiro foram escolhidos os itens 1 e 2, “Criação do Banco de Dados” e “Consultas SQL”, para o segundo *Sprint* foi definido o item 3 “Sistema de Autenticação”, para o terceiro *Sprint* foi definido as configurações de manter usuário, itens 4, 5, 6, e 7, e para o quarto e último *Sprint o* item 8 “Configuração da Agenda”.

As iterações(Sprints) foram definidas em quinzenas. Para o levantamento de requisitos e desenvolvimento do diagrama UML, foram necessárias 3 iterações, iniciando na 1ª quinzena de setembro e finalizando no final da 1ª quinzena de outubro. Para a implementação houve um atraso, por conta de alguns motivos externos, ela foi iniciada na 1ª quinzena de novembro.

Figura 8 – Diagrama UML de casos de uso



Houve algumas mudanças em relação as suas funções, o objetivo inicial era o controle total de um salão de beleza, incluindo os gastos e lucros, com o andamento do projeto o foco foi voltado para o gerenciamento dos clientes, desenvolvendo a manutenção de clientes e usuários, e controle da agenda de cada um. Com isso houve alteração no *Product Backlog*, de acordo com essas mudanças, conforme a tabela 4.

Tabela 4 – Product Backlog após as alterações.

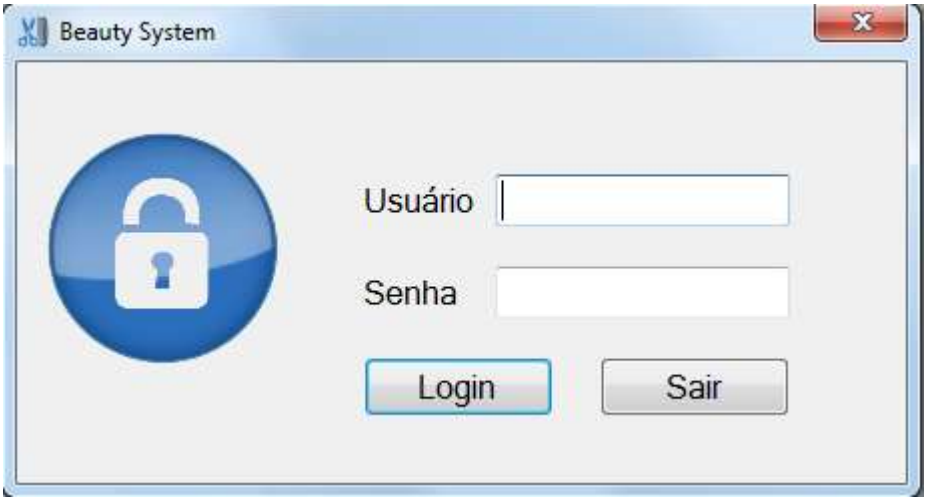
Product Backlog			
ID	Atividade	Descrição	Prioridade
1	Criação do Banco de dados	Criação do banco de dados, suas tabelas e respectivas relações	9
2	Consultas SQL	Criação do Scripts SQL que o sistema irá utilizar	7
3	Sistema de Autenticação	Desenvolvimento do sistema de autenticação e cadastramento dos usuários	4
4	Incluir usuário	Inclusão de usuários e/ou clientes no sistema	6
5	Alterar usuário	Alteração de usuários e/ou clientes no sistema	6
6	Excluir usuário	Exclusão de usuários e/ou clientes no sistema	6
7	Modificar usuário	Modificação de usuários e/ou clientes no sistema	6
8	Configuração da Agenda	Configuração da agenda para inserção e manutenção dos dados	8

O desenvolvimento foi finalizado na 2ª quinzena de novembro, onde começaram os testes, que ainda não foram finalizados.

5.4 Telas do Sistema

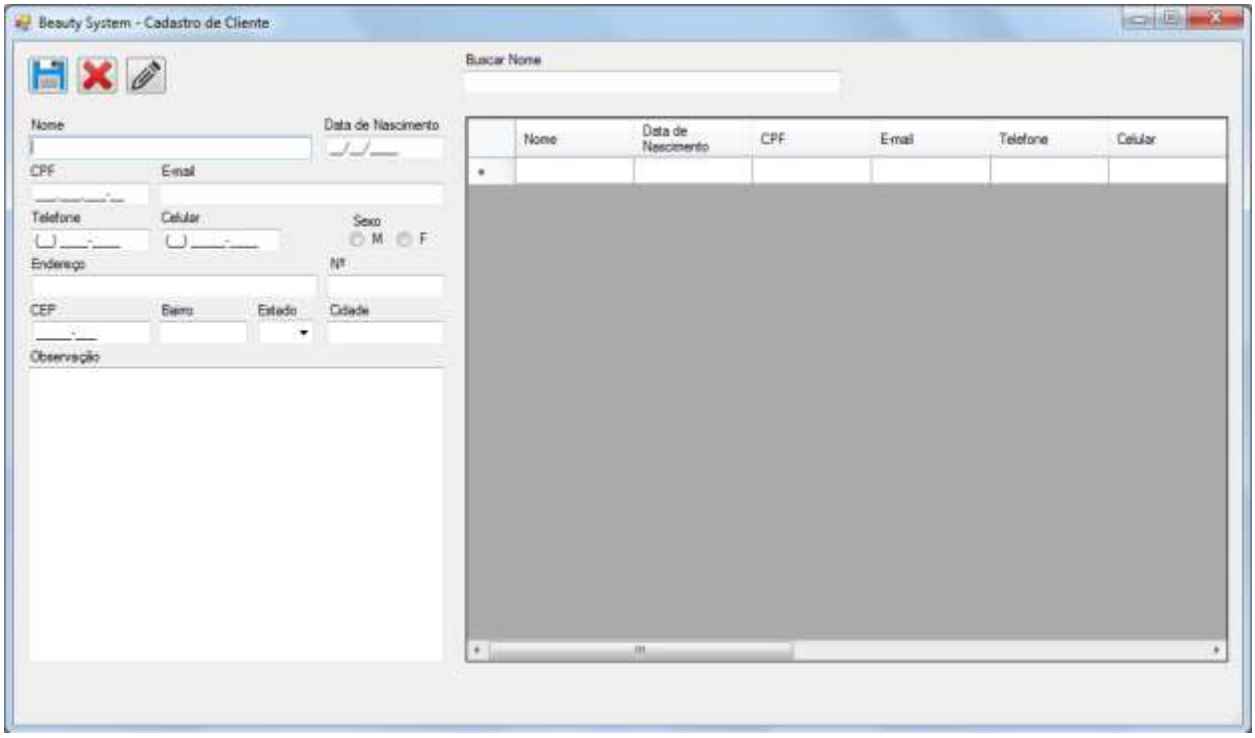
As telas do sistema foram feitas utilizando uma das práticas de XP, o *Design Simples*, onde o design deve ser feito de forma a ser entendido, sem um conhecimento aprofundado do sistema para que se possa utilizar, e também foram utilizados conceitos de IHC. As figuras 9, 10 e 11 mostram um pouco disso.

Figura 9 – Tela de login



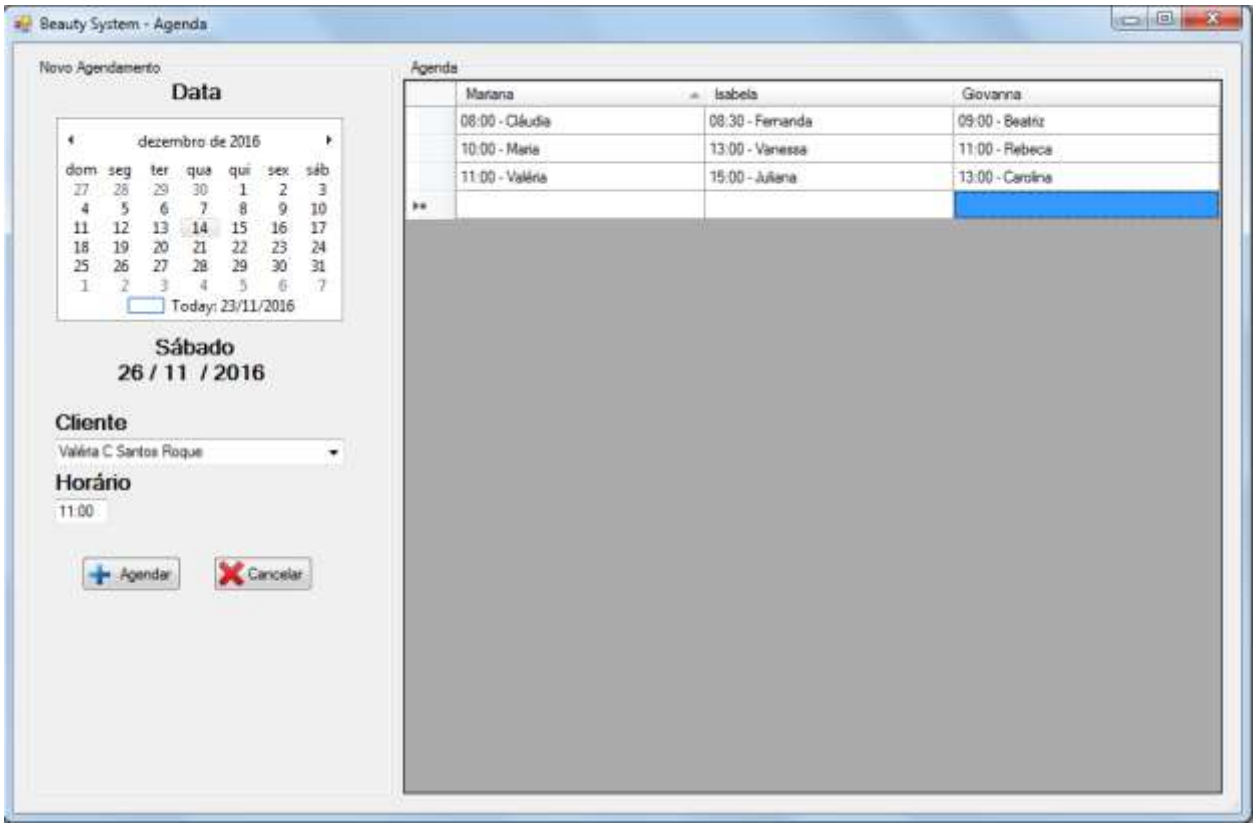
O sistema de autenticação é baseado nos usuários cadastrados no banco.

Figura 10 – Tela de cadastro de clientes



O cadastro segue esse padrão, para clientes e funcionários

Figura 11 – Tela de agendamento



5.5 Práticas Implementadas

O primeiro passo utilizando as práticas de Scrum, foi o planejamento do projeto, onde foram definidas 4 *sprints* para a conclusão do mesmo. Em seguida foram definidas as atividades a serem realizadas, e foram enquadradas no *Backlog.* Após isso, foi apresentado ao cliente o plano do projeto.

Para cada *Sprint* foi realizado um *Sprint Planning Meeting,* reunião para planejamento das atividades na *Sprint*, de modo que pudesse definir entre as atividades do *Backlog* as que seriam executadas na *Sprint*. A reunião diária não era feita, pois como não havia uma equipe, era possível identificar as atividades finalizadas, em andamento, e a identificação dos impedimentos ocorridos.

Ao fim de cada *Sprint*, é feita uma revisão (*Sprint Review)* para apresentação do que foi feito durante a *Sprint.*

5.6 Resultados Obtidos

O gerenciamento de projetos é importante para determinar o sucesso de um projeto que tem como objetivo a otimização custos e prazo, levando em conta o contentamento dos integrantes, controlando o escopo e a qualidade do produto.

Através da utilização de Scrum, é possível desenvolver o software mais rápido, e sem perder sua qualidade, ganhando tempo em relação. A iteração constante com o cliente possibilita seu envolvimento no projeto, com isso há uma satisfação maior, pois ele pode ver o que está acontecendo e já dizer o que precisa ou não mudar.

6.CONCLUSÕES

Através dos resultados exibidos neste trabalho, e dentro de suas limitações, podemos concluir que as práticas apontadas pelas metodologias ágeis são eficientes no desenvolvimento de sistemas, desde que utilizadas de forma correta.

Através do estudo de caso, podemos perceber que o Scrum descreve melhor a organização de como o projeto deve ser feito, sem definir os documentos de

especificação técnica do componente. Scrum não define que temos que criar um modelo lógico de dados, mas não impede que seja criado para facilitar a implementação do sistema, sendo que esse modelo pode ser registrado como um item do *Product Backlog.*

Não é possível dizer qual metodologia é pior ou melhor, todas tem suas vantagens e desvantagens. Para a escolha da metodologia a ser utilizada é necessário avaliar o projeto e ver qual atente melhor suas necessidades. Também é preciso saber as habilidades da equipe, pois uma metodologia que a equipe já conheça e tenha prática em trabalhar é melhor do que uma que eles não conheçam e precisem aprender. Uma boa forma de se trabalhar, é combinando metodologias, por exemplo, uma boa combinação é o uso de *Scrum* com FDD, *Scrum* pode auxiliar na parte da documentação do projeto, organizando tarefas, prazos, atividades em cada iteração, entre outras coisas, e para o desenvolvimento em si, FDD pode ajudar com as práticas de programação.

7.REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHAMSSON, Pekka; SALO, Outi; RONKAINEN, Jussi; WARSTA, Juhani. *Agile Software Development Methods: Review and analysis*, 2002. Disponível em: http://www.pss-europe.com/P478.pdf, recuperado em 19/09/2016.

Agile Manifesto. *Manifesto for Agile Software Development*, 2011. Disponível em: http://agilemanifesto.org/, recuperado em: 14/11/2016.

BECK, Kent. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

CARVALHO, B.V.; MELLO, C.H.P. *Revisão, Análise e Classificação da Literatura sobre o Método de Desenvolvimento de Produtos Ágil Scrum,* 2009.

COAD, Peter. *Java Modeling in Color with UML,* 1999.

DEEMER, Pete; BENEFIELD, Gabrielle; LARMAN, Craig; VODDE, Bad. *The Scrum Primer. Disponível em: http://www.scrumprimer.com/, recuperado em: 17/10/2016.*

FEATURE DRIVEN DEVELOPMENT, disponível em: http://www.featuredrivendevelopment.com/, recuperado em: 01/11/2016.

PALMER, SR.; FELSING, JM. *A Pratical Guide to Feature Driven Development,* 2002.

SCHWABER, Ken; BEEDLE, Mike. *Agile Software Development with Scrum,* 2001.

SENSAGENT. Disponível em: http://dicionario.sensagent.com/feedback/en-en/, recuperado em: 02/10/2016.

SOARES, Michel dos Santos. *Comparação Entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software, 2004. Disponível em*: http://infocomp.dcc.ufla.br/index.php/INFOCOMP/article/view/68, recuperado em 25/09/2016.

SOMMERVILLE, Ian. *Engenharia de Software.* Addison-Wesley, 2003.