UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

KEILA MENECRYS SANDRINO

ANÁLISE DA APLICAÇÃO DE METODOLOGIAS ÁGEIS: UM ESTUDO DE CASO EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2014

KEILA MENECRYS SANDRINO

ANÁLISE DA APLICAÇÃO DE METODOLOGIAS ÁGEIS: UM ESTUDO DE CASO EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. DrªMarcus Vinícius Drissen Silva

PONTA GROSSA

2014

Dedico este trabalho à minha família e empresa que proporcionaram a oportunidade de crescer e pudesse obter conhecimento.

AGRADECIMENTOS

A Deus, pois sem Ele, nada seria possível e não estaria aqui desfrutando destes momentos que são tão importantes.

Muitas pessoas contribuíram para a realização desse trabalho, sem elas teria desistido diante dos obstáculos. O seu apoio e confiança foram fundamentais para mim. E por isso deixo aqui registrado minha admiração e agradecimento.

Ao meu Marido Malcon, pelo amor, pela compreensão por estar ao meu lado nos momentos bons e ruins e por sempre me dar forças para lutar por meus objetivos. Te Amo Muito*!!*

A minha filha Thaís que acompanhou os momentos de tensão, sempre com bom humor e carinho, anciosa por ver a conclusão deste trabalho.

A minha família, pois é minha fortaleza e sem ela não seria nada.

A todos os amigos que sempre estiveram em minha vida e aqueles que conhecemos durante o período de graduação.

Aos meus colegas da universidade, que me acompanharam por estes anos,obrigado por terem crescido comigo.

Obrigado(a).

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

ANALISE QUALITATIVA DE METODOLOGIAS ÁGEIS: UM ESTUDO DE CASO EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

por

KEILA MENECRYS SANDRINO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 31 de janeiro de 2014 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Marcus Vinícius Drissen Silva
Prof. Orientador

Simone Nasser Matos
Membro titular

Thalita Scharr Rodrigues Pimenta
Membro titular

Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

Simone de Almeida
Coordenador do Curso
UTFPR - Câmpus Ponta Grossa

RESUMO

SANDRINO, Keila M. **ANÁLISE DA APLICAÇÃO DE METODOLOGIAS ÁGEIS: UM ESTUDO DE CASO EM EMPRESA DE DESENVOLVIMENTO DE SOFTWARE.** 2014. 59f. Trabalho de Conclusão de Curso de Tecnologia em Análise e Desenvolvimento de Sistemas- Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2014

Observou-se a dificuldade de trabalho de equipes de desenvolvimento de software em empresas com ausência da aplicação de padrões de desenvolvimento e metodologias de trabalho. Por meio de pesquisas realizadas na literatura identificou-se uma metodologia que melhor atenda as necessidades de organização em uma empresa de desenvolvimento. O presente trabalho mostra através de um estudo de caso em uma empresa de desenvolvimento de software adoção da metodologia ágil Scrum para proporcionar a construção de um sistema mais consistente com entregas frequentes e proporcionar a adesão de mudanças, bem com formalizar uma forma de documentação das fases que compreendem a construção desse sistema de forma organizada.

Palavras-chaves: Aplicação de Metodologia. Convenções de nomenclatura. Boas práticas de programação.

ABSTRACT

SANDRINO, Keila M. **ANALYSIS OF THE APPLICATION OF AGILE METHODS : A CASE STUDY IN SOFTWARE DEVELOPMENT COMPANY**. 2014. 59f. Conclusion Work Course Technology Analysis and Systems Development - Federal Technological University of Paraná. Ponta Grossa, 2014

Noted the difficulty of working teams of software development companies with the absence of the application of development standards and work methods. Through research conducted in the literature identified a methodology that best meets the needs of organization in a developing company. This paper shows through a case study in a company developing software for the adoption of the Scrum agile methodology to provide for the construction of a more consistent system with frequent deliveries and provide membership changes and to formalize a form of documentation of the stages comprising the construction of this system in an organized manner.

Keywords: Implementation Methodology. Naming conventions. Good Programming Practices.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

No processo de desenvolvimento de software muitas vezes a atenção se volta mais para seu funcionamento, deixando sua flexibilidade de lado. Para que equipes de desenvolvimentos possam organizar seu trabalho sugere-se o emprego de uma metodologia ágil de desenvolvimento e de medidas de boas práticas de programação para aumentar a dinamicidade de seu trabalho e otimizar a manutenibilidade do produto final.

As Metodologias Ágeis de desenvolvimento de software ganham importância em diversos segmentos da indústria de software, tendo por objetivo construir um sistema de qualidade que atendam as necessidades dos usuários, promovendo um desenvolvimento sustentável, tomando um ritmo constante de desenvolvimento e tornando-se mais eficaz (SBROCCO, MACEDO, 2012). Juntamente com as metodologias ágeis outro recurso que auxilia no processo de melhoria da qualidade de códigos, segundo Patrick (1990, p.4) é a refatoração e tem por objetivo ‘’tornar o código mais fácil de entender e de modificar, ou seja, apenas as mudanças que cumpram esses objetivos podem ser considerados refatoração’’ (PATRICK; CABRAL, 2010, p.4).

Tendo em vista que a necessidade de dinamicidade no trabalho dos desenvolvedores e que a empresa não emprega efetivamente uma metodologia ou boas práticas de programação, para obter um código limpo e flexível, busca-se alcançar padronização e documentação, agregando uma metodologia e às boas práticas de programação para construção de produtos de qualidade.

Este trabalho tem por objetivo definir o padrão de metodologia ágil e boas práticas de programação mostrando sua aplicação em uma empresa de desenvolvimento de software

1.1 JUSTIFICATIVA

A dificuldade dos desenvolvedores em produzir um sistema com código de fácil compreensão, de fácil manutenção e produzir um software com agilidade diante das mudanças, impossibilita a produção de um produto que satisfaça a necessidade do cliente e tenha flexibilidade.

Uma fator decorrente é a alta rotatividade de profissionais em uma empresa de desenvolvimento, a propriedade intelectual por muitas vezes acaba se perdendo, fazendo com que a equipe composta por acadêmicos gaste tempo para entender as regras de negócio que compõe o sistema e o que já foi desenvolvido, fazendo dom que a necessidade de empregar uma metodologia de desenvolvimento e documentar todas as fases de desenvolvimento para tornar o código de mais fácil compressão.

1.2 OBJETIVO

Este trabalho tem por objetivo melhorar a forma de trabalho de uma empresa de desenvolvimento de software, através da adoção de metodologias ágil em uma para aperfeiçoar o planejamento e desenvolvimento de softwares e facilitar ao desenvolvedor a construção de um código mais consistente. Documentando as fases que compreendem a construção desse sistema.

Aplicar boas práticas de programação que possa auxiliar no processo de desenvolvimento da empresa, definindo um padrão auxiliar no desenvolvimento de de códigos de fácil interpretação e proporcionar flexibilidade aos sistemas desenvolvidos pela empresa.

1.2.1 Objetivos Específicos

- Definir os padrões de boas práticas de programação e metodologias ágeis propostos na literatura.
- Identificar os métodos propostos na literatura e escolher uma destes para estudo de caso na empresa.
- Mostrar os pontos fortes e fracos do método aplicado na empresa com o auxilio da ferramenta TFS (*Team Foundation Server).*

2 METODOLOGIAS ÁGEIS

As metodologias ágeis nasceram da necessidade de abandonar metodologias antigas e ultrapassadas, isso ocorreu devido ao uso de hardwares mais avançados, linguagens de programação mais modernas, fatores referente ao desenvolvimento de sistemas e necessidades organizacionais. As metodologias ágeis seguem princípios para encorajar o uso dos melhores métodos para o desenvolvimento de software, entre os principais princípios citados por Macedo (2012) abrangem os seguintes:

- Satisfazer o cliente priorizandoentregas continuas e frequentes, fazendo com que o cliente se torne o principal foco das equipes de desenvolvimento.
- O projeto deve ter abertura para mudanças de requisitos, mesmo em uma fase avançada do projeto. As mudanças são aceitas nas metodologias ágeis desde que venham agregar valor ao produto e vantagens ao negócio do cliente.
- A liderança e/ou gerentes de projeto devem transmitir confiança para a equipe “Construa projeto objetivando manter uma equipe motivada, fornecendo ambiente apoio e confiança necessários para realizar o trabalho” (SBROCCO; MACEDO, 2012, p.89), prezando pelo crescimento dos seus colaboradores e pela qualidade de seu trabalho.
- A boa comunicação é primordial para que a equipe mantenha o foco no projeto promovendo o desenvolvimento sustentável. “Patrocinadores, desenvolvedores e usuários devem ser capazes de manter o ritmo constante” (SBROCCO; MACEDO, 2012, p.89), permitindo obter o ritmo de entrega das funcionalidades do sistema.

Estes entre outros critérios são essenciais para manter a simplicidade, e promover uma equipe organizada com o foco no projeto e no cliente.

A metodologia ágil propõe que as equipes trabalhem de forma organizada para adaptar-se mais facilmente as mudanças de direção que o possa tomar, pois segundo Sbrocco as equipes mais organizadas conseguem reinventar e reestruturar o negócio com criatividade e conforme a demanda do cliente. Considerando o fato de que o acompanhamento periódico do desempenho da equipe são importantes para verificar se existe algum processo falho e/ou desnecessário que precisam ser revistos e traçar um plano de melhoria, juntamente com a equipe.

Baseando-se nesses princípios os métodos ágeis propõem um conjunto de atividades que podem ser adotadas dentro do processo de desenvolvimento de software, definindo as prioridades dentro do projeto que, apesar de mostrar informalidade, as metodologias ágeis não deixam de utilizar processos, ferramentas, documentações de forma mais enxuta. A seguir serão apresentadas as características de algumas metodologias ágeis como Scrum, Extreme Programming (XP), Feature Driven Development (FDD) e Kanban.

2.1 SRUM

A metodologia ágil *Scrum* foi fortemente influenciada pelas boas práticas adotadas pelas industrias japonesas, iterações, valor, times pequenos, multifuncionais e auto organizados. A denominação dessa metodologia surgiu da associação de equipes de projetos com eventos esportivos chamado *scrum*, onde quando ocorre um incidente ou a bola sai de campo o jogo somente reinicia com reunião de todos os jogadores.

Tudo no Scrum possui uma natureza que deve ser entendida e aplicada para atingimento dos objetivos, baseando-se em seis características: flexibilidade dos resultados, flexibilidade dos prazos, times pequenos, revisões frequentes, colaboração e orientação a objetivos. O sucesso depende do correto entendimento da natureza de cada um destes pontos, os erros mais comuns são a falta de crença e colaboração, tentar eximir-se ou procurar culpados, falta ou excesso de senso de propriedade.

Sbrocco (2012) enfatiza que a metodologia *Scrum* é indicada para projetos de desenvolvimento de software com as seguintes situações:

- Requisitos mudam rápida e constantemente tornando o desenvolvimento complexo
- Gerenciamento e controle do desenvolvimento de trabalho.
- Tornar a equipe autogerenciável e funcional
- Implementar o conceito interativo e incremental no desenvolvimento de software e/ou produtos

- Identificar causas de problemas e remover impedimentos
- Valorizar os indivíduos.

Além das características citadas, o *Scrum* também é conhecido pelo fato de trabalhar com ciclos, chamados de *sprints* que representam a duração do trabalho com tempo variável de realização das tarefas, normalmente, ocorrem no período de duas a quatro semanas. A metodologia ressalta um conjunto de regras e práticas que devem ser cumpridas pelas equipes e seus respectivos papeis, entre os principais estão: *Product Owner*, o *Scrum Master* e o *Team*.

A metodologia ágil *Scrum* segundo Macedo (2012), propõe que as informações sobre o domínio do negócio necessárias sejam coletadas e detalhadas em um *Product Backlog* que se trata de conjunto de requisitos do projeto, pelo *Product Owner,* termo utilizado para referenciar o responsável pela manutenção dos requisitos do projeto. O *Product Owner* é responsável por garantir que a equipe agregue valor ao negócio, atuando como termômetro para obter um equilíbrio entre os interesses do cliente e do *Team*, trazer a visão e as funcionalidades do produto, definir as prioridades e decidir sobre as datas de entrega, mantendo o *Product Backlog*.

O *Scrum Master* representa o cliente no processo de desenvolvimento do projeto e desempenha o papel de facilitador, removendo os impedimentos que possam surgir no decorrer do projeto. O *Scrum* Master conduz o projeto, garante a produtividade da equipe e a colaboração entre diversos papeis e funções, atuando como escudo para interferências externas e aplica os valores e práticas da metodologia *Scrum* (SBROCCO; MACEDO, 2012, p.164).

O time responsável pelo desenvolvimento denominado Team, é composto por grupos de 4 a 9 pessoas, que devem possuir características multifuncionais, no desenvolvimento de software deve conter analistas, desenvolvedores, testadores, entre outros. O time deve possuir a característica de auto-organização mantendo a comunicação verbal estimulada (SBROCCO; MACEDO, 2012, p.165).

Ao iniciar cada ciclo, as informações do *Product Backlog* são priorizadas pelo *Product Owner* em *Sprint Backlog* (conjuntos de requisitos para execução no tempo determinado como um ciclo) e este deve repassá-las para os demais integrantes da

equipe, *Scrum Master*, o líder da equipe de desenvolvimento, e *Team Members* (desenvolvedores), normalmente durante uma cerimônia chamada *Sprint Planinging*, quando todos os envolvidos se reúnem para planejar as próximas tarefas a serem desenvolvidas.

Alem da cerimônia de planejamento das tarefas, realiza-se uma reunião diária entre o *Scrum Master* e os *Team Members*, onde devem ser relatadas as tarefas planejadas para serem executadas no dia e as dúvidas e os problemas identificados, chamados de impedimentos, permitidos que o *Scrum Master* busque os responsáveis para solucioná-los.

Ao final de cada ciclo e do projeto a abordagem SCRUM sugere que sejam realizadas cerimônias de revisão, para que todos possam relatar as dificuldades enfrentadas e compartilhar o conhecimento obtido durante o processo. Em poder de tais informações, cabe ao *Scrum Master* planejar as ações a serem tomadas nos próximos ciclos e projetos para otimizar o trabalho da equipe.

A abordagem de comunicação entre os membros da equipe proposta pelo *Scrum* proporciona uma discussão das regras envolvidas no projeto, porém em muitos casos os tempos determinados para tais discussões são insuficientes para o completo entendimento do conjunto de regras envolvidas, abrindo brechas para falhas de comunicação. O que a cerimônia de revisão pode contornar por possibilitar o tratamento das falhas identificadas para projetos futuros. O clico completo proposto pela metodologia é ilustrado na Figura 1.

Figura 1- Ciclo *Scrum*
Fonte: Métodos Ágeis- engenharia de Software(2012, p162)

O SCRUM não determina especificamente como deve ser a abordagem do *Product Owner* para com o especialista do domínio, durante a coleta das informações, o que pode também proporcionar falhas de comunicação, em especial no que diz respeito a importância de cada requisito para o usuário final.

2.2 EXTREAM PROGRAMING

A metodologia *Extreme Programing* surgiu na década de 90, quando Kent Beck foi chamado para analisar o desempenho do projeto de conversão da folha de pagamento da Chysler. O sistema C3, ou *Chrysler Comprehensive Compensation System* (Sistema de Compensação Abrangente da *Chrysler*), é conhecido como o berço do XP e foi onde Kent Beck utilizou pela primeira vez, em conjunto, as práticas que atualmente formam a estrutura do *Extreme Proramming* (Sato; 2007, p.14).

A metodologia XP prega a busca pela simplicidade, procurando a solução mais simples para atender as necessidades do cliente. Utiliza um modelo de mudança incremental, reduz o impacto da mudança nas organizações, mantendo o sistema sempre testado e em uso no cliente e agregando novas funcionalidades que vão sendo produzidas no decorrer do projeto. Trabalho com Qualidade, para que tanto o cliente quanto os desenvolvedores se sintam gratificados pela experiência.

A metodologia pode ser empregada em equipes pequenas e médias e que irão desenvolver software com requisitos vagos e em constante mudança, com foco explícito em escopo, recomenda-se a priorização de funcionalidades que representem maior valor possível para o negócio. Desta forma, caso seja necessário a diminuição de escopo, as funcionalidades menos valiosas serão adiadas ou canceladas.

A metodologia XP preza por alguns valores que são fundamentais e devem ser encaradas como prioridade pela equipe de desenvolvimento como: A metodologia XP promete reduzir o risco do projeto, melhorar a capacidade de resposta às mudanças do negócio, aumentar a produtividade ao longo da vida de um sistema e adicionar diversão para a construção de software em equipes, tudo ao mesmo tempo. O XP possui algumas características como: *Feedback* constante, abordagem incremental e encorajar a comunicação entre as pessoas envolvidas.

Feedback - aproxima o desenvolvedor da parte interessada do sistema, o cliente. Após a criação de parte do sistema o cliente passa a utilizá-lo, opinando sobre o que foi desenvolvido, assim como o desenvolvedor pode opinar mostrando ao cliente que algumas alterações podem afetar a regra do negócio. Esta interatividade faz com que ambos possam entender melhor sobre a regra do negócio e estabelecer um acordo entre cliente e desenvolvedor, permitindo que o sistema seja constantemente avaliado auxiliando a estabelecer as prioridades no desenvolvimento.

Comunicação – comunicar-se com o cliente, fazendo com que desempenhe um papel importante no do projeto. A comunicação de forma direta faz com que os membros da equipe de desenvolvimento possam tratar de assuntos relacionados ao projeto sem que haja mal entendidos ou qualquer tipo de dúvida relacionada ao projeto. O gerente de projetos deve agir como intermediador estabelecendo contato entre cliente-desenvolvedor para estabelecer um grau de confiança, e não comprometer a qualidade do projeto.

Simplicidade - trata de não fazer uso de recursos desnecessários para o desenvolvimento do sistema, porém, não privar o cliente de funcionalidades necessárias ao seu negócio atendendo todos os requisitos do projeto. A equipe deve extrair as informações do cliente para que o sistema funcione da forma esperada,

evitando fazer uso de recursos conhecidos como “perfumaria” (SBROCCO, MACEDO, 2012).

Coragem - a equipe deve entra em discussão logo após o recebimento dos requisitos do sistema e ter a intrepidez para abrir questionamentos perante o cliente a respeito, propondo sugestões para melhor resolver o problema. Após realizar a análise devem ser propostas pela equipe as alterações necessárias, mesmo que estejam fora do scopo proposto pelo cliente inicialmente. Madedo (2012, p.146) propõe alguns tópicos a serem abordados como: Adotar novos processos, contrato com escopo, simplicidade do sistema, desenvolvimento incremental, ritmo sustentável, assumir atrasos e problemas para cumprir cronograma, exposição de código disponibilizando à todos os membros da equipe, *refactoring* contínuo, relação cliente x desenvolvedor, documentação apenas em caso de necessidade, permitir que o cliente opine, dividir em pares, assegurar um tempo para desenvolver testes e automatiza-los para o projeto em execução e projetos futuro e delegar tarefas a membros da equipe para obter homogeneidade na equipe.

A metodologia XP propõe o uso de alguns passos importantes a serem seguidos, primeiramente os requisitos devem ser separados e discutidos pela equipe, definindo as ferramentas, a linguagem de programação e banco de dados que devem ser utilizados, esta definição deve abranger protocolos de comunicação, os prazos e custos iniciais do projeto. Estas definições devem ser devidamente documentadas e revisadas para atestar a viabilidade do uso das ferramentas propostas.

O segundo passo é realizar uma reunião juntamente com o cliente para realizar um mapeamento das prioridades de desenvolvimento dos requisitos propostos. Após o cliente especificar as prioridades, cabe ao gerente de projetos realizar as *releases* que serão implementados de acordo com as histórias (funcionalidades), possibilitando que o cliente possa utilizar as funcionalidades prontas enquanto as demais estão sendo desenvolvidas.

O terceiro passo realiza-se o primeiro *Standup Meeting,* utilizando as histórias aprovadas e com suas prioridades especificadas pelo cliente. Esta reunião deve ocorrer na empresa do cliente e ser conduzida pelo gerente de projeto para que sejam extraídas as informações necessárias. Neste passo é que devem ser criados todos os

diagramas como caso de uso, classe, atividades ou sequência, para que os desenvolvedores e o cliente possam entender graficamente o sistema em questão. O desenvolvimento do sistema pode ser iniciado, paralelamente o analista de teste realiza a criação dos testes junto com o cliente.

O quarto passo organiza uma nova reunião para que o cliente possa ter maior contato com o que foi desenvolvido, permitindo que os requisitos sejam validados e incluídos novos requisitos que não foram levantados anteriormente.

Em todos os passos descritos um redator deve formalizar uma ata documentando as decisões tomadas e realize uma discussão para determinar novos prazos e custos. No final do projeto é recomendado elaborar um contrato contendo forma de manutenção e garantias.

2.3 FEATURE DRIVEN DEVELOPMENT(FDD)

O Desenvolvimento Guiado por Características da metodologia *Feature Driven Development*, criada por Jeff de Luca e Peter Coad em Singapura(SBROCCO, 2012,p.99) nos anos 1997/1998, é considerada robusta oferecendo outra opção para organizações para quem gosta de ter os benefícios de uma metodologia iterativa e incremental, mas precisam manter algum nível mínimo de processo definido.

A metodologia FDD pode ser aplicada tanto para pequenas equipes quanto para grandes equipes, trazendo algumas características como a rastreabilidade e relatórios bastante detalhados contando com monitoramento permitindo visualizar o andamento do desenvolvimento, tudo em termos de negócios.

Realiza-se uma modelagem do sistema de maneira que possa demonstrar o que será desenvolvido pela equipe. A partir dos requisitos levantados constrói-se uma lista de funcionalidades que em sua composição traga o funcionamento do negócio do cliente. Sbrocco (2012) sugere que sejam abordadas ter três níveis: Áreas de negócios, Atividades de Negócios e Passos da Atividade de Negócios.

Sbrocco (2012) apresenta um conjunto de práticas que recomenda serem seguidas no desenvolvimento de software utilizando FDD, como Modela de objetos do domínio, Desenvolvimento por funcionalidades, Entregas regulares, Formação de equipe e Posse individual do Código.

Na fase de modelagem de domínio devem ser realizadas reuniões para definir a modelagem do projeto em uma visão macro para ficar claro o que será produzido pela equipe.

Na fase de desenvolvimento por funcionalidade recomenda-se que, a partir dos requisitos, sejam listadas as funcionalidades que devem compor o sistema.

As entregas de parte do sistema desenvolvido devem ser concretas, com frequência e funcionais, ou seja, sugere que sempre que uma funcionalidade estiver concluída, deve ser disponibilizada aos usuários, permitindo realizar o acompanhamento do processo de desenvolvimento. O modelo deve trazer uma descrição macro do projeto, demonstrando uma visão do que deve ser desenvolvido guiando a equipe.

Cabe ao gerente de projeto formar equipes por funcionalidades a serem desenvolvidas, realizando um estudo mais aprofundado do sistema a ser desenvolvido juntamente com um especialista sobre as regras do negócio, para manter a equipe sempre atualizada, permitindo a equipe de desenvolvedores escreverem os métodos e classes necessárias para a concepção do projeto.

Cada lista de funcionalidades deve ter seu proprietário que fica responsável por desenvolvê-la e depois possa responder por ela em qualquer problema de implementação. O código deve ser inspecionado para constatar de que a nova funcionalidade foi implementada corretamente.

Segundo Sbrocco (2012), a FDD sugere um modelo gráfico para visualizar e medir todo o processo de desenvolvimento, devendo mostrar cada funcionalidade e a porcentagem de completude que ela se encontra. A metodologia indica também um relatório de toda implementação para documentação histórica da seguinte forma: organizado por funcionalidade e datas, desde a criação, implementação e possíveis modificação.

“A utilização da FDD num projeto, torna recomendável o uso de uma ferramenta que permita organizar todas as implementações que se deseja criar sendo possível a inclusão e discriminação de todas as componentes necessárias para as novas funcionalidades. Uma ferramenta deste tipo deve ter alguns requisitos necessários para facilitar a utilização de FDD” (BARBOSA; 2008, p.10).

Pode-se observar que os requisitos são base para que o gerente de projetos modele o objeto de domínio. O objeto de domínio pode ser feito utilizando um diagrama que mostra a visão macro do negócio. Estes requisitos devem ser oriundos de uma visita do gerente do projeto, através de uma ou mais entrevistas realizadas com as pessoas diretamente envolvidas no projeto.

Para estas entrevistas o gerente de projetos deve fazer uso de questionários de apoio que podem ser aplicados a todos os envolvidos. A partir disso o gerente de projetos consegue modelar os objetos do domínio utilizando um diagrama, proporcionando uma visão macro do sistema semelhante a Figura 2.

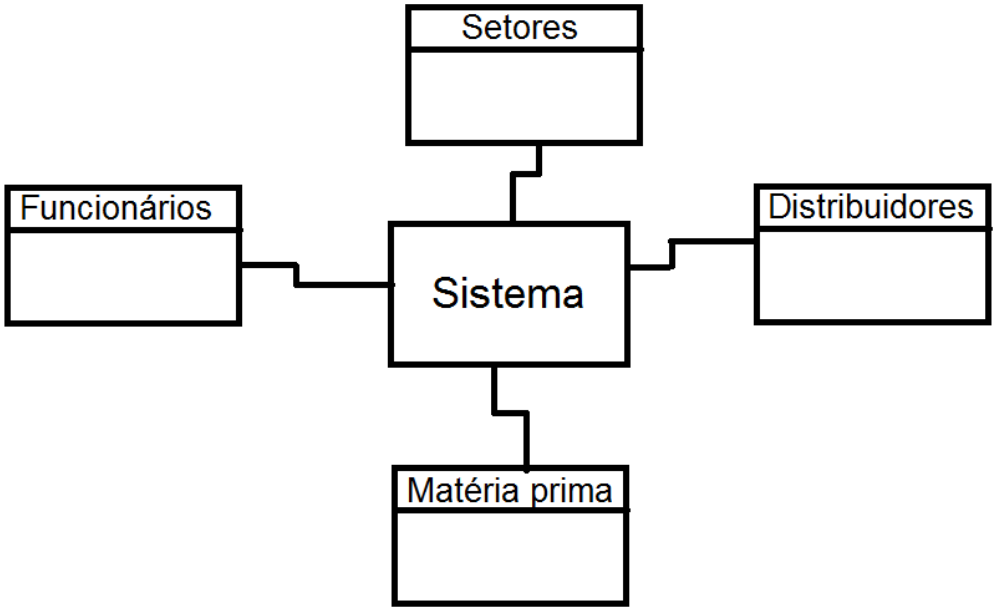


Figura 2 - Exemplo de visão macro do sistema
Fonte: Métodos ágeis –Engenharia de software sob medida(2012)

Na modelagem o gerente de projeto consegue mensurar preliminarmente todos os componentes do sistema, porém esta visão é uma visão macro sem detalhar funcionalidades específicas. A próximo passo é o de detalhamento, proposto que seja através de reuniões em grupo para realizar as definições do detalhamento, e

determinado pelo gerente de projetos quem fará parte da equipe que irá desenvolver o projeto e suas atribuições.

Após a Modelagem de domínio estar formulada, as regras de negócios devem ser criadas por um especialista na área, documentando todos os processos de funcionamento da empresa. As Regras de negócios são compostas por regras utilizadas na prática, estas devidamente documentadas formalmente mediante aceite de todos os envolvidos. Neste momento, deve ser definido e estabelecido o documento de requisito do projeto para que a equipe de modelagem possa criar a lista de funcionalidades (*features*), contendo informações dos detalhes do desenvolvimento, incluindo as prioridades levando em consideração que devem ser realizados os testes e as entregas para o cliente.

Esses requisitos devem ser apresentados em forma de funcionalidades obedecendo a três categorias: Áreas de negócios, Atividades de Negócios e Passos da Atividade de Negócios.

O Quadro 1 mostra a lista de funcionalidades criada a partir dos requisitos do sistema que foram levantados, deve possibilitar o rastreamento, o refinamento dos requisitos, dando a possibilidade de enxergar os detalhes do que deve ser produzido no projeto.

Lista de Feature			**Empresa: X**
			10/01/12
	Responsáveis: João Silva; Roberto Manoel; Marina Lima		
Área de negócios			
Requisito	Prioridade	Data da Entrega	Responsável
RNF-001 Sistema bilíngue	4	imediata	Rafael Dias
RNF-002 Plataforma Linux	4	imediata	Guilherme Rosa
...			
Atividade de Negócio			
Requisito	Prioridade	Data da Entrega	Responsável
RNF-001 Cadastro Cliente	4	06/01/12	Sergio Santos

Passos da Atividade de Negócios			
Requisito	Prioridade	Data da Entrega	Responsável
RNF-001 Validação de CNPJ	4	06/02/2012	Sergio Santos

Quadro 1 - Lista de *Featuras*
Fontes: Métodos ágeis- Engenharia de Software

A partir da lista de funcionalidades, fazer o refinamento das funcionalidades e realizar sua separação em pequenos pacotes que possam ser implementados e utilizados pelo cliente enquanto outros pacotes estão sendo desenvolvidos. O programador-chefe deve conferir da funcionalidades, fazer testes unitários das funcionalidades, testes estruturais e de desempenho. Sbrocco (2012) aconselha que seja gerado um documento contendo todos os minipacotes listados, quem são os responsáveis, bem como um mecanismo de controle de versão, as datas de compilação e os testes realizados com seus resultados.

Recomenda-se criar uma forma de acompanhamento para medir o andamento do desenvolvimento, permitindo gerar um relatório do processo e observar as funcionalidades que merecem maior atenção, as pendentes e a concluídas, facilitando dar um retorno a respeito do andamento do projeto. Este acompanhamento permite ao gerente de projeto avaliar a equipe e com relação ao rendimento do desenvolvimento fazendo com que os prazos sejam estipulados com maior precisão.

2.4 KANBAN: MEDINDO O FLUXO

Kanban é um termo de origem japonesa e significa literalmente “cartão” ou “sinalização”. É um conceito relacionado com a utilização de cartões (*post-it* e outros) para indicar o andamento dos fluxos de produção em empresas de fabricação em série desenvolvido pela *Toyota Motor Company*. Nesses cartões são colocadas indicações sobre uma determinada tarefa, por exemplo, “para executar”, “em andamento” ou “finalizado”. Quem realizou as adaptações foi Anderson, na Microsoft, para engenharia de software em 2004. Anderson (2004) o apresentou na conferência Agile 2007 de Washington e, então começou o entusiasmo pelo método, pois apresentou resultados muito animadores (BORIA; RUBINSTEISN; RUBINSTEIN, 2013, p.35).

Com a utilização do método *Kanban* a equipe pode alcançar um ritmo de produção sustentável e introduzir mudanças nos processos com baixíssima resistência. Por isso, recomenda-se preferencialmente naquelas organizações de baixa maturidade institucional.

Uma característica da metodologia *Kanban* é permitir a compreensão do tempo de circulação de cada tarefa, desde a inserção no sistema até a finalização. Após ajustar os parâmetros de produção, a equipe alcança um ritmo de produção que pode ser mantido indefinidamente, consegue-se ter uma previsão que outros métodos demoram para alcançar.

Segundo Boria (2013, p.36) “uma das vantagens do *Kanban* é que, ao produzir com qualidade e tempo, gera-se confiança nos clientes, que recebem produtos com regularidade e com a qualidade esperada”. Outra vantagem é que, ao fazer com que constantemente a equipe busque melhorar seus processos evitando demoras, fazendo entregas mais frequentes e que cada entrega contenha o maior número de funcionalidades possível.

O diferencial do *Kanban* com relação aos outros métodos ágeis é sua flexibilidade; considerando a limitação do volume de trabalho em cada etapa. Focando nas seguintes características: Foco na Qualidade, Redução do Trabalho em Desenvolvimento, Entregas Frequentes, Equilíbrio da Demanda contra a Produção, Fixação de Prioridades e Ataque às Fontes de Variação para Melhorar a Previsibilidade.

Anderson(2011) identificou três cinco princípios principais do método Kanban.

1. “Kanban não é um método de desenvolvimento ágil de software. Também não é um projeto de gerenciamento ágil. Não é um método para desenvolvimento de software. O Kanban é algo que você aplica a uma forma de trabalho pré-existente. Aplica-se em algo que já existe por si mesmo”.

2. “Mudança incremental evolutiva. Mudanças devem ocorrer de forma é incremental, ou seja, uma coisa de cada vez. Mas, como evolutiva entende que o cenário mudou: resenhar o quadro e incluir uma coluna, por exemplo, ou apagar algo que não serve mais e voltar à versão anterior. A abordagem evolutiva é boa para

resolução de problemas complexos, já que permite um planejamento mais apurado das coisas".

3. "Inicialmente, é preciso respeitar papéis, responsabilidades e cargos atuais. As pessoas tendem a associar imagem e identidade pessoais aos papéis desempenhados no trabalho. Se você disser a elas que isso irá mudar, elas ficam com medo e acabam tendo uma reação emocional, pois simplesmente dizer a elas que o trabalho irá mudar é um ataque à identidade. E esse é também o cerne da resistência delas. É justamente isso que queremos evitar".

3 PADRÕES DE DESENVOLVIMENTO E ARQUITETURA

Nas empresas de desenvolvimento de software a preocupação é que o sistema esteja funcionando e se esquecem de outra característica importante que é a Flexibilidade. As metodologias ágeis vêm se destacando por oferecerem respostas rápidas para casos em que os requisitos são mutáveis ou não estão completamente claros (Ortoncelli, 2010, p.2).

Quando o cliente por alguma razão decide alterar as regras ou requisitos do software, a dificuldade de realizações das alterações é maior do que o esperado. Por este motivo, objetiva-se descrever as convenções de nomenclatura e arquitetura que devem ser aplicadas para padronização de código, facilitando eventuais manutenções e atualizações do sistema.

Definir um padrão é algo que deve ser realizado pela equipe de desenvolvimento antes do início de um projeto. Após definido, o projeto é iniciado e todos devem seguir o padrão anteriormente definido. Referente ao estudo de proposto será apresentado um padrão e boas práticas de programação que foram definidas pela equipe de desenvolvimento da empresa.

3.1 TEST DRIVEN DEVELOPEMENT - TDD

Considera-se que a prática do *Test Driven Development* tenha origem da metodologia ágil *Extreme Programming* (XP), sendo esta uma das metodologias mais utilizadas, A prática envolve a implementação de um sistema começando pelos casos de teste de um objeto. Escrever casos de teste não se trata somente de uma estratégia de testes, seu principal conceito é permitir que testes modelem guiando o desenvolvimento do sistema(SCHISSATO, 2010)

O TDD trás alguns benefícios se comparado com os testes tradicionais, como aumento da qualidade do código e produtividade do desenvolvedor, não sendo utilizado apenas para validação do código, mas também como documentação que auxilia o desenvolvedor a codificar o que foi solicitado pelo cliente.

Segundo Borges (2006) o TDD pode ser representado por interações, na Figura 3 podemos observar que cada uma possui os seguintes passos:

- Escolher História: selecionar o requisito ou área da tarefa que melhor oriente o desenvolvimento.
- Escrever Testes: desenvolver testes concretos o mais simples possível para orientar o desenvolvedor também testar o sistema
- Executar Testes: verificar se o sistema satisfaz a um ou vários testes.
- Refinar o programa e refatorar: editar o sistema sem quebrar nenhum teste

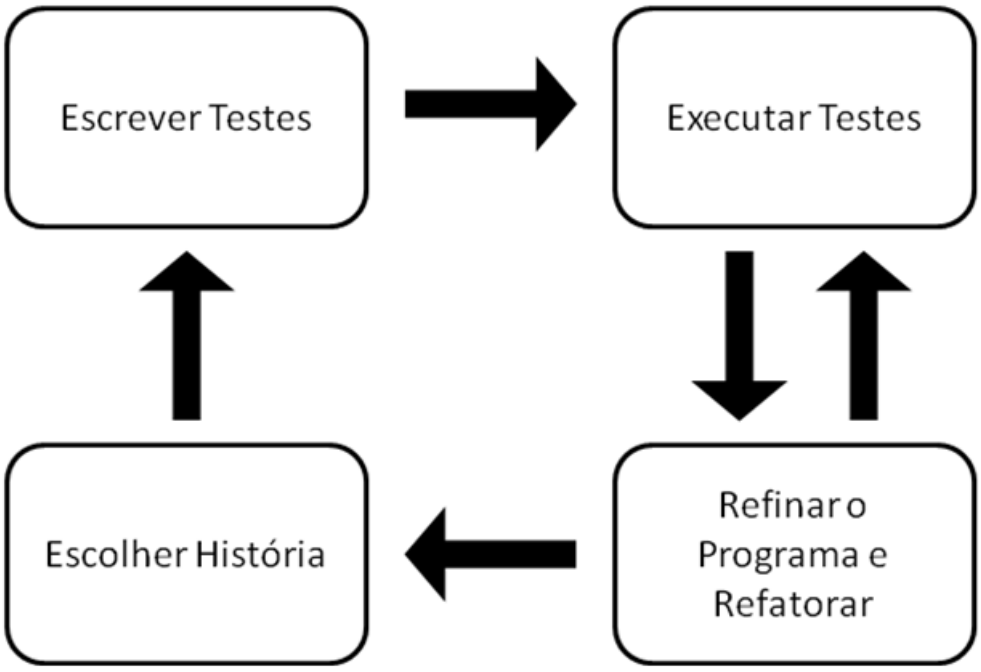


Figura 3 - Exemplo de Modelo TDD
Fonte: Conceitos e Benefícios do Test Driven Development, p.3

O TDD não extingue a necessidade de realizar a fase de teste após a implementação do código fonte, pois o desenvolvimento dirigido a testes diminui a ocorrência de erros, mas não é capaz de evitar completamente que eles venham a ocorrer.

3.2 DOMAIN-DRIVEN- DESIGN- DDD

Segundo Evans (2010) o *Domain- Drive Design* trata-se de uma abordagem do projeto de software de forma disciplinada, que engloba uma série de conceitos e

técnicas sem perder o foco no domínio do software. O foco no Domínio do Software é a automatização de um processo de negócio, atendendo completamente um determinado negócio. Evans(2010, p.77) diz que o modelo aborda cinco itens principais: Entidades, Objetos de Valor, *Factories*, Serviços, Repositórios.

A definição de Entidade refere-se a tudo o que representa valor ao domínio ou regra de negócio. Deve-se escrever em uma determinada camada todas as entidades ligadas ao domínio de forma simples, implementando sem algum tipo de interface.

A definição de objeto de valor segundo Evans(2010, p.92) representa de forma descritiva o domínio sem nenhuma entidade conceitual. O objeto de valor é instanciado para representar elementos do projeto pelos quais dá-se importância pelo que são estes objetos, não quem ou quais são.

Um exemplo dado por Evans(2010, p.93) é o de uma criança que utiliza lápis de cor para pintar, se um lápis for perdido pode-se substituir por outro de mesma cor e continuar seu desenho, não sendo importante os demais atributos que compõe o lápis ou de qual marca ele é, como o seu tamanho, para ela o que importa mesmo é a cor.

Assim pode-se entender que um “objeto que representa um aspecto descritivo do domínio sem nenhuma identidade conceitual é chamado de Objeto de Valor” (EVANS, 2010, p.94). Para melhorar a identificação o Objeto de valor pode ser observado quando em uma aplicação o mais relevante são os atributos de um elemento do modelo, expressando o significado dos atributos e relacionando uma funcionalidade a ele, tratando o objeto de valor como imitável.

Evans(2010, p.94) descreve que os atributos devem formar um Objeto de valor formam um todo conceitual, por exemplo, rua, cidade e código postal não devem ser atributos separados do objeto Pessoa, pois fazem parte de um único endereço, formando um único objeto mais coerente conforme Figura 4. O Objeto trás informações sobre uma entidade, ou seja, os atributos significativos.

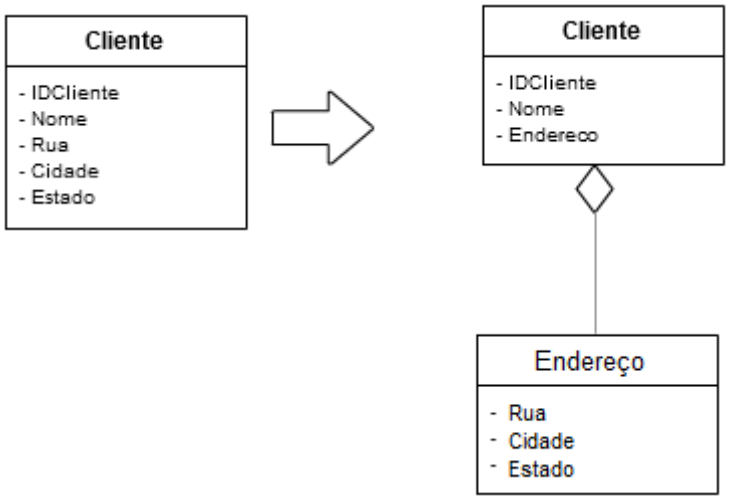


Figura 4 - Exemplo de Objeto de Valor
Fonte: Donain-Drive Desegn, 2010

3.3 BEHAVIOR DRIVEN DESIGN - BDD

A técnica de desenvolvimento ágil Desenvolvimento Guiado por Comportamento encoraja a interção e colaboração entre desenvolvedores, setores de qualidade e pessoas não técnicas ou de negócios num projeto de software. Equipe de desenvolvimento se utiliza de dois tipos de linguagem: narrativa e linguagem ubíqua[1], que permitem focarem nos motivos pelos quais o código esta sendo desenvolvido e não tecnicamente.

A técnica BDD se utiliza dos conceitos do TDD focando seu desenvolvimento visando testes, as especificações usam uma linguagem que pode ser entendida em vários contextos, beneficiando os usuários técnicos e os usuários de negócio (Oenning). A técnica é guiada pelos valores de negócios, comumente única maneira dos benefícios serem percebidos é através de interfaces de usuário para a aplicação.

Cada trecho de código pode ser considerado como sendo um cliente de outros módulos de código no qual a interface é utilizada. Cada elemento de código tem algum tipo comportamento, portanto, vários elementos determinam o comportamento da aplicação.

---

[1] Uma linguagem que pode ser entendida claramente por todos os envolvidos em um projeto.

3.4 MODEL-VIEW-CONTROLER - MVC

O padrão arquitetural MVC divide o sistema em três camadas o *Model*, *View* e *Controller*. Na camada *Model* contém as informações relacionadas ao domínio e os objetos implementadores das principais funcionalidades do sistema. Na camada View contém os objetos que representam a interface gráfica com o usuário. Na camada *Controller* define como a interface gráfica deve proceder com as informações fornecidas pelo usuário e atualizar as informações e estado dos objetos da camada *model* (Durelli; Viana; Penteado, 2008, p.4).

A atuação da camada *Controller* é de intermediação, tendo acesso as classes da camada *Model* para realizar as tarefas do sistema a partir das informações fornecidas pelo usuário na interface gráfica (*View*) conforme exemplo da figura 5.

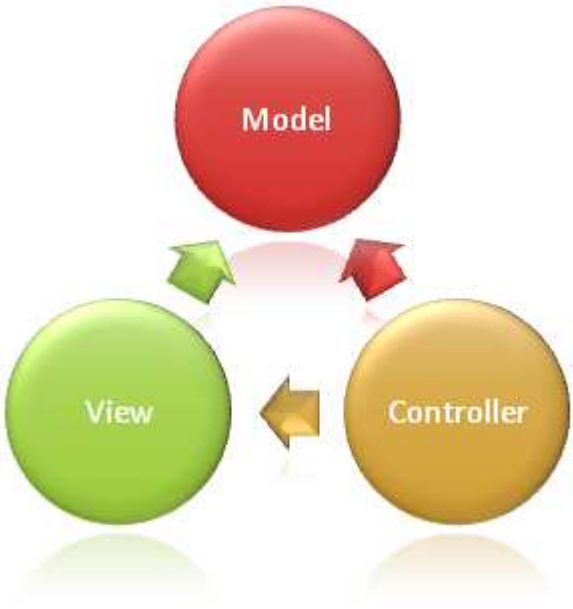


Figura 5 - Exemplo de modelo de Arquitetura MVC

Fonte: Uma Proposta de Reuso de Interface Gráfica como Usuário Baseada no Padrão Arquitetural MVC

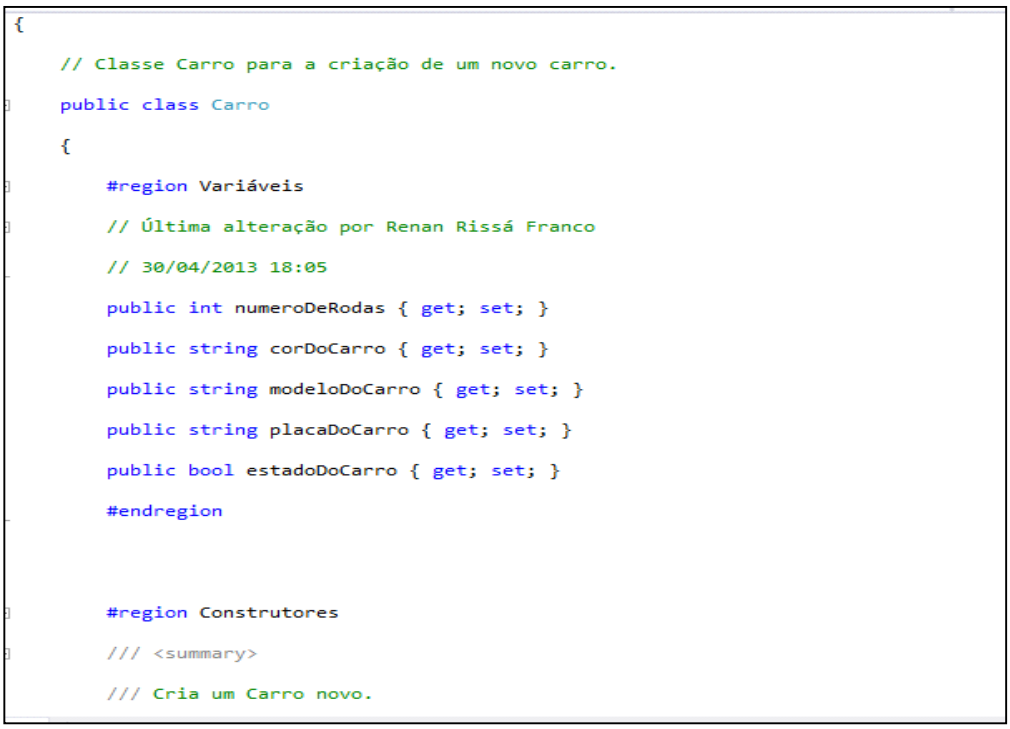
3.5 PADRÕES E BOAS PRÁTICAS DE DESENVOLVIMENTO

Existem normas e padrões de desenvolvimento de software que permitem produzir sistemas visando reduzir a ocorrência de vulnerabilidades, para tanto alguns conceitos muito importantes para o desenvolvimento de softwares, independente da linguagem, foram reunidos para utilização no desenvolvimento de sistemas estabelecidos pela equipe de desenvolvimento da empresa proposta para o estudo de caso. Após uma reunião realizada para estabelecimento das regras foi gerado um documento formalizando as regras gerais a serem seguida. Permitindo que caso necessário sofra adaptações.

- Todo código deve ser comentado e atualizado conforme possíveis alterações a que se refere.
- Todas as classes, funções e métodos devem conter as seguintes especificações:
    - descrição do que contém e/ou funcionalidade
    - informar responsável pela criação ou alteração
    - registrar a data da criação ou última alteração
    - informar as entradas e saídas
- Toda classe deve ter tratamento de exceção consistente.Códigos que não estão sendo utilizados e permanecem comentados somente para fins de utilização de teste com as devida descrição e finalidade.
- Tenha como prática declarar todas as variáveis que um programa irá utilizar logo após o título da rotina, simplificando uma possível manutenção.
- Endentação facilita tanto a construção de um programa como sua manutenção.
- Utilizando o próprio tab[2] para que seja fácil manter todas as linhas iniciando-se na mesma coluna. Deixando as chaves {e} sempre em linhas independentes, para que seja fácil identificar tudo que está entre os dois.
- Mesmo que seja apenas um comando, é uma boa prática colocar chaves {e}
- Transformar comentários em documentação de código utilizando o recurso próprio da linguagem (três barras ///)

[2] Tecla do tab é utilizada para tabulação ou espaçamento inicial de um parágrafo em um texto.

- Utilizar o recurso *Regions* que permite especificar um bloco de código que pode expandir ou recolher ao usar o recurso de traçar o editor de código da ferramenta utilizada, como por exemplo, a Figura 6 criada para demonstrar o recurso utilizando a ferramenta Microsoft Visual Studio.



```
{
    // Classe Carro para a criação de um novo carro.
    public class Carro
    {
        #region Variáveis
        // Última alteração por Renan Rissá Franco
        // 30/04/2013 18:05
        public int numeroDeRodas { get; set; }
        public string corDoCarro { get; set; }
        public string modeloDoCarro { get; set; }
        public string placaDoCarro { get; set; }
        public bool estadoDoCarro { get; set; }
        #endregion


        #region Construtores
        /// <summary>
        /// Cria um Carro novo.
```

Figura 6 - Exemplo de utilização do recurso *Regions*
Fonte: Autoria própria

3.6 PADÕES DE NOMENCLATURAS

- As nomenclaturas em geral devem estar na língua portuguesa.
- Nomes de variáveis devem ter um significado lógico, de forma que alguém apenas lendo o nome consiga identificar o que deve ser armazenado na variável.
- Para as variáveis e parâmetros do método utilizar do padrão de nomenclatura *Camel Case*, primeira letra em minúsculo e cada palavra concatenada em maiúsculo (Manjaly, 2004).

3.6.1 Nomenclatura Do Banco De Dados

3.6.1.1 Tabela

3.6.1.2Procedures, Views e Functions

VW	_	ListarTransacoesDeCliente
2 letras da view		Verbo ou Tabela(singular)

Quadro 2 - Exemplo de nomenclatura de view
Fonte: Autoria Própria

3.6.1.3 Parâmetros de Procedures

- Idêntico ao campo da tabela inclusive o tipo e o tamanho conforme exemplifica Tarifa.

  Ex. Campo - cliCodigo char (10)

  Declare @cliCodigo as char(10)

  Caso não seja campo de tabela deve-se seguir o padrão de atributos.

3.6.2 Classe Com Seus Atributos E Métodos

- As classe com seus atributos, métodos, por convenção, toda classe deve começar com uma letra maiúscula e, de preferência, não pode conter qualquer caractere especial (@, #, $, %, &, *, _, etc...) ou número.

- Caso o nome de uma classe seja composto por mais de uma palavra, a primeira letra de cada palavra deve ser em maiúscula.
- O nome da classe deve ser exatamente o mesmo nome de seu arquivo fonte (Xavier).
- Fazer uso do recurso *Regions,* conforme Figura 7.

```
namespace Carro
{
    // Classe Carro para a criação de um novo carro.
    public class Carro
    {
        #region Variáveis
        // Última alteração por Renan Rissá Franco
        // 30/04/2013 18:05
        public int numeroDeRodas { get; set; }
        public string corDoCarro { get; set; }
        public string modeloDoCarro { get; set; }
        public string placaDoCarro { get; set; }
        public bool estadoDoCarro { get; set; }
        #endregion

        #region Construtores
        /// <summary>
        /// Cria um Carro novo.
        /// </summary>
        /// Saída= Um objeto Carro completo.
        /// Última alteração por Renan Rissá Franco
        /// 30/04/2013 18:10
        /// <param name="numeroDeRodas">Numero de Rodas do Carro</param>
        /// <param name="corDoCarro">Cor do Carro</param>
        /// <param name="modeloDoCarro">Modelo do Carro</param>
        /// <param name="placaDoCarro">Placa do Carro</param>
        /// <param name="estadoDoCarro">Informe true para o Carro em movimento e false para o Carro parado.</param>
        public Carro(int numeroDeRodas, string corDoCarro, string modeloDoCarro, string placaDoCarro, bool estadoDoCarro)
        {
            this.numeroDeRodas = numeroDeRodas;
            this.corDoCarro = corDoCarro;
            this.modeloDoCarro = modeloDoCarro;
            this.placaDoCarro = placaDoCarro;
            this.estadoDoCarro = estadoDoCarro;
```

Figura 7- Exemplo de Classe
Fontes: Autoria Própria

3.7 COMENTÁRIOS

3.7.1 Banco de Dados

Os comentários atribuídos ao banco de dados devem conter informações sobre a empresa, a que sistemas estão ligados, qual objetivo, autor, data de criação e possíveis alterações conforme exemplifica a Figura 8.

```
/*
||----------------------------------------------------------------------------
|| EMPRESA      :    DELOITTE TOUCHE TOHMATSU
|| SISTEMA      :    SISTEMA DE FATURAMENTO
|| OBJETIVO     :    ALTERAÇÃO DE AVISOS
|| AUTOR        :    ALEXANDRE TARIFA
|| CRIACAO      :    06-03-2003
|| MANUTENCAO   :
|| OBSERV.      :
||----------------------------------------------------------------------------
*/
```

Figura 8 - Exemplo de Informações de Banco de Dados
Fonte: Autoria Própria

3.7.2 Sistemas

Os comentários atribuídos ao sistema devem conter as informações sobre a empresa, objetivo do sistema, autor, data da criação e possíveis manutenções. A Figura 9 exemplifica o comentário referente ao sistema (Tarifa).

```
'*********************************************************************
' EMPRESA           :
' OBJETIVO          :
'
'---------------------------------------------------------------------
' AUTOR             :
' DATA CRIAÇÃO      :
'      MANUTENÇÃO   :
' OBSERVAÇÃO        :
'*********************************************************************
```

Figura 9 - Exemplo de Comentário do Sistema
Fonte: Autoria Própria

3.7.3 Comentário da Classe

Os comentários atribuídos ao sistema devem conter as informações sobre saída, autor, data da criação ou possíveis manutenções e explicação do cada parâmetro representa, conforme exemplo na figura 10.

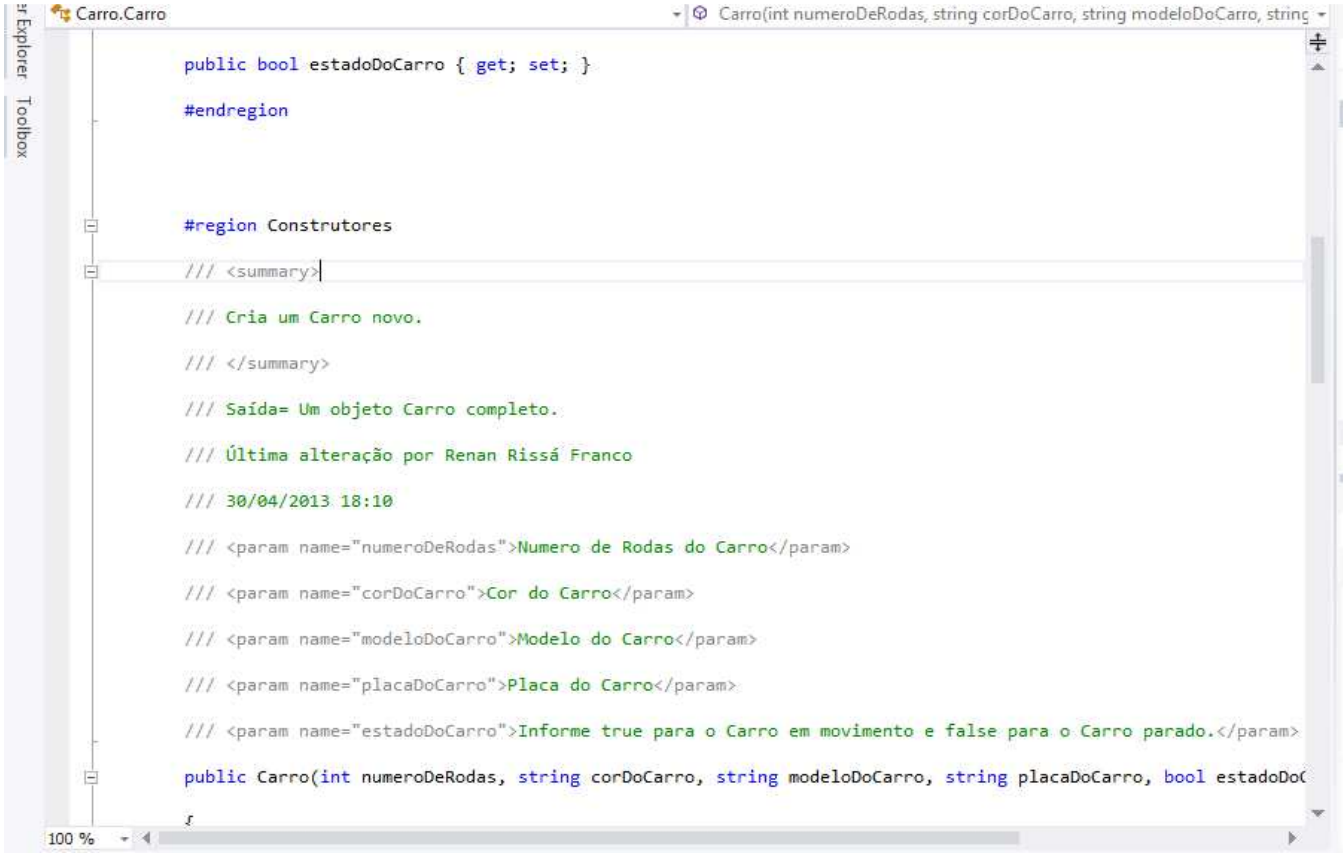


Figura 10 - Comentário da Classe
Fonte: Autoria Própria

Apesar de definidos alguns padrões de desenvolvimento composto pelas boas práticas de programação, o desafio concentra-se em encontrar a metodologia que melhor se enquadra no estudo de caso proposto, considerando seu aspecto inicial. Para este cenário, problemas de comunicação interna devem ser estudados e resolvidos, para que a equipe possa atingir uma interação entre os colaboradores.

4 ESTUDO DE CASO

4.1 DESCRIÇÃO

O presente trabalho consiste no estudo de caso em uma empresa onde há incidência de rotatividade de colaborares, profissionais com pouca experiência e o conhecimento é perdido frequentemente, assim, os desenvolvedores encontram dificuldade para continuar e realizar atualizações nos sistemas.

Identificou-se a necessidade da criação de um conjunto de regras intitulado Boas Práticas de Programação, para auxiliar o desenvolvimento de códigos mais fáceis de interpretar e proporcionar a flexibilidade aos sistemas desenvolvidos pela empresa, adaptando os critérios de desenvolvimento e fazendo estudo das Metodologias Ágeis para definir a metodologia que melhor se aplica ao estudo de caso proposto.

Com base no estudo realizado, foram relacionadas vantagens e as dificuldades na adaptação da equipe no uso das boas práticas de programação e metodologia.

4.2 DEFINIÇÃO DO PROBLEMA

A empresa X está no mercado de desenvolvimento de software para empresas de segurança há 11 anos. Essa surgiu da necessidade atender as empresas brasileiras realizando adaptações de um sistema de origem internacional utilizado para monitoramento de alarme. O sistema nasceu com seu desenvolvedor, que pertence a uma família proprietária de uma empresa prestadora de serviço de segurança e monitoramento de alarme, a qual identificou a necessidade desta adaptação e criação de um sistema que atendesse o mercado nacional. Ao passar dos anos o sistema, criado por este desenvolvedor, foi abrangendo o mercado e crescendo em funcionalidades, assim, demandando mão de obra para o desenvolvimento dando origem a empresa X.

Em se tratando de uma empresa que conta com poucos recursos, proporciona a oportunidade de seus colaboradores estudarem as ferramentas e sobre o ramo de negócios que o sistema atende até ter conhecimento necessário para atuar efetivamente no desenvolvimento do sistema, porém há rotatividade de profissionais

na empresa, que levam consigo toda propriedade intelectual dos projetos que por ele foram desenvolvidos, fazendo com que a equipe esteja em constante aprendizagem, trazendo à tona a necessidade de implementar um plano estratégico e documentar todas as fases de desenvolvimento, utilizando regras de boas práticas e metodologias.

A equipe de desenvolvimento não empregava uma metodologia específica e o detentor do conhecimento com relação às regras do negócio é quem realizava as distribuições, informalmente, de tarefas a serem executadas, sem documentação e prazos específicos ou algum tipo de padrão a ser seguido.

Com o passar dos anos a empresa começou a trabalhar e desenvolver sistemas para outros ramos. Mas, a falta de uma equipe preparada, metodologia de trabalho ser empregada, definições padrões e boas práticas de desenvolvimento, a dificuldade da equipe de se organizar e trabalhar dando continuidade ao trabalho de outros desenvolvedores é persistente. Continuando a dificuldade dos desenvolvedores em produzir um sistema com código de fácil compreensão, ou seja, códigos de fácil manutenção, que sejam “simples” de refatorar, que satisfaça a necessidade do cliente e tenha flexibilidade.

Identificaram-se as necessidades de utilizar convenções de nomenclatura e boas práticas de programação, que devem ser aplicadas para padronização de código, adoção de metodologias para proporcionar ao desenvolvedor a construção de um código mais consistente e a documentação das fases que compreendem a construção desse sistema de forma organizada, facilitando eventuais manutenções e atualizações do sistema.

4.3 ANÁLISE DAS METODOLOGIAS

As organizações que sobrevivem são aquelas que melhor se adaptam a mudanças. A melhoria de processos não é a modificação dos documentos que os descrevem, mas sim a conduta das pessoas que devem seguí-los (BORIA; RUBINSTEIN, 2013, p.37).

Para que a empresa tenha a possibilidade de organizar a forma de trabalho de sua equipe de desenvolvimento realizou-se primeiramente um trabalho de motivação e conscientização com todas as pessoas junto com os demais setores. Isto

é uma tarefa muito difícil de conseguir e requer profunda atenção. Quando a proposta é a modificação do comportamento da equipe, a mudança não é fácil de adotar. (BORIA, 2013, p.38). O que a empresa X procura é evidenciando as vantagens da mudança decorrente da aplicação da metodologia ágil. Mesmo quando as pessoas estão de acordo com a necessidade de mudança, não quer dizer que estão de acordo com a direção que queremos dar à mudança.

O desconhecido causa temor ou, pelo menos, ressentimento. Esperar que todos entendam a mudança desde o início e a adotem por suas vantagens é ilusório, a maioria ignorará ou resistirá à mudança (BORIA; RUBINSTEIN, 2013, p.38).

Duas são as estratégias sugeridas por Boria (2013, p.58) como principais para reduzir o impacto da mudança: uma é dividir a mudança em etapas tão pequenas quanto forem sustentáveis. A outra é dar o apoio para realizar as mudanças de condutas nas equipes que precisam realizá-las. Só quando as equipes entenderem que devem modificar o seu comportamento é que os processos se institucionalizam na dentro da empresa. Para fazer isso de forma consciente, é necessário um diagnóstico e o planejamento das atividades, pois as inclusões de métodos para controle de tarefas e documentação fazem com que a equipe encare de forma resistente, deixando de cumprir o que foi definido.

Para a aplicação da metodologia ágil na empresa foi observada as características que a mesma deve atender alguns critérios como: delimitação para distribuição, determinação de prazo para realização das tarefas e um controle para que a equipe tivesse o conhecimento do andamento das tarefas em processo de desenvolvimento. A equipe contando com apenas três desenvolvedores e uma analista deve ser capaz de projetar, desenvolver, testar e comunicar, se adequado a mudanças necessárias no decorrer do projeto.

Realizou-se uma breve análise das principais características das metodologias ágeis Scrum, Feature Driven Development e Programação Extrema apresentadas no Quadro 3.

	Scrum	XP	FDD

Foco	Foco em gerenciamento de projeto onde é difícil planejar à frente	Foco especifico no escopo, recomenda-se a priorização de funcionalidades que representem maior valor para o negócio	Fdd atua no processo de desenvolvimento tanto para a gestão de projetos quanto para a engenharia de software
Equipes	Destinado para gerenciamento e software, mas pode ser utilizado para manutenção de software Equipe multidisciplinar, auto-organizada	Destinado a equipes pequenas e medias que irão atuar no desenvolvimento, O código é comunitário	Equipes pequenas, destinadas a desenvolver recursos necessários ao projeto, o desenvolvedor como único responsável pelo módulo
CARACTERÍSTICAS			
Cliente	Cliente se torna parte da equipe de desenvolvimento.	Reunião semanal com cliente para priorizar funcionalidades	Integração continua permite que a equipe esteja em contato constante com o cliente, tornando o processo ágil e com entregas constantes
Entregas	Entregas frequentes e intermediarias de funcionalidades 100% desenvolvidas.	O cliente recebe novas funcionalidades semanais.	Entregas regulares, mantendo o cliente atualizado, ativando as novas funcionalidades
Procedimento	Discussão diária na qual cada membro da equipe responde á algumas perguntas: o que fiz desde ontem? o que planejo fazer amanha? existe algo impedindo de atingir minha meta?	Como o escopo é reavaliado semanalmente, o projeto é regido por um contrato de escopo negociável, que difere significativamente das formas tradicionais de contratação de projetos de software	O planejamento é feito com base na lista de funcionalidades(backlog). As equipes são formadas por funcionalidades. medição gráfica deve mostrar as porcentagens de conclusão de cada funcionalidade
Comunicação	Reuniões frequentes com todos os envolvidos no projeto	Produção em pares pra melhorar a comunicação da equipe. Reuniões em pé, apenas	Implementação de relatórios, sugere ferramenta para organizar as implementações

		abordando tarefas realizadas e tarefas futuras	
Valores	Flexibilidade de resultados, flexibilidade de prazos, times pequenos, revisões frequentes, colaboração e orientação a objetivos	Feedback rápido, presumir simplicidade, mudanças incrementais, trabalho de alta qualidade, programação pareada	Testes unitários, refatoração, programação em pares, integração contínua, inspeção formal (de desenho e de código) e posse individual/situacional de código/classe
Organização	Trabalha com product backlogs(funcionalidades)que podem ser dividdas em sprints(ciclos) que representa o trabalho em tempos variáveis de realização de tarefas	O planejamento é feito com histórias escritas em pequenos cartões, Escrito pelo cliente e deve descrever uma Requisito funcional. Evidencia o valor de negócio	Captação de requisitos(lista de features), formação de equipes conforme funcionalidade. Realiza modelagem do projeto (uml) Transformara as funcionalidades Em modelo de objetos
Processos	Desenvolve por conjunto de prioridades de requisitos. Construção ocorre por ciclos de duas semanas. Não descreve o que fazer em cada situação Organização sem planejmentr à frente	Desenvolvimento feito em interações semanais, Liberações de pequenas versões Propriedade coletiva Desenvolvimento orientado a testes	Desenvolvimento de modelo abrangente Construção de lista de funcionalidades Planejar por funcionalidade Detalhe por funcionalidade Construção por funcionalidade

Quadro 3 - Comparativo das Metodologias ágeis

Fonte: Autoria Própria

4.4 APLICAÇÃO DE METODOLOGIA E PADRÕES DE ARQUITETURA

O cenário atual aponta para uma demanda por software de qualidade com prazo e custos cada vez mais reduzidos. Portanto, as empresas que desenvolvem sistema necessitam de recursos que auxiliem a oferecer o que o mercado atual deseja para não perder clientes.

Usar padrões facilita o reuso, os testes e permite que futuros desenvolvedores, ao olharem o código em uma manutenção, consigam facilmente identificar e assim trabalhar em cima do problema e não reaprender tudo o que foi desenvolvido (Aéce; 2007).

A empresa contando com contato anterior de parte da equipe com a metodologia ágil *Scrum* e com o aplicativo *Team Foundation Server* capaz de gerenciar um projeto e permitir integração entre todos os envolvidos. O TFS oferece uma série de recursos, alguns exemplos são: colaboração em equipe, controle de versões, gerenciamento de mudanças e de compilações, e relatórios. Permitindo a utilização do *template* de processo 100% baseado no *framework* do *Scrum*, possibilitando usar os principais artefatos, utilizando a mesma nomenclatura e funcionalidade, mantendo os mesmos padrões até para os mais puristas no *framework.*

Para o estudo de caso iniciou-se um levantamento das ações para organização adotando alguns procedimentos, documentando os requisitos apontados pelo cliente, ou seja, requisitos funcionais que dão origem aos *Product Backlogs Itens* a serem realizadas pela equipe de desenvolvimento.

Primeiramente observou-se que a empresa possuía algumas ferramentas para auxiliar a equipe na organização inicial e em decisões de como o trabalho seria iniciado.

Para tanto se iniciou um trabalho com a equipe aplicando a metodologia ágil *Scrum,* por se tratar de um time pequeno para equipe é necessário ter iterações, multifuncionais e auto-organizado. A equipe possui profissionais com habilidades distintas, porem complementares, esta característica atende a multifuncionalidade que

a equipe necessita. A auto-organização da equipe direciona responsabilidades e com isso pode fazer com que a equipe permaneça motivada (SBROCCO, 2012 p.164).

Procurou-se realizar as práticas sugeridas pela metodologia *Scrum* como:

Em se tratando do produto CP o cliente está representado por um grupo de três empresários que são os idealizadores do projeto. Portanto, analista da equipe realiza o trabalho de coleta as informações a respeito do negócio, criando um conjunto de requisitos do projeto, considerando o tamanho da equipe o responsável pelos requisitos (*Product Owner*) e o *Scrum Master,* que conduzem o projeto, está representados pela mesma pessoa.

O conjunto de requisitos a serem executados em tempo determinado por ciclos (*Sprint Backlog*) foram revisados e repassados na reunião de revisão das tarefas realizadas informalmente. Estas reuniões servem para retratar as dificuldades enfrentadas e compartilhar conhecimento adquirido no decorrer da execução das tarefas.

Para realizar o processo de desenvolvimento conforme metodologia criou-se uma forma de documentação de levantamento de requisitos de forma simples e aplicável à realidade da empresa. As informações que compõe o documento são informações para que se possa criar o *Products Backlog* que originarão as tarefas a serem realizadas e registr-lás na ferramenta que auxilia no gerenciamento do ciclo de vida do projeto o TFS[3] (*Team Foundation Server*).

O *Team Foundation Server* permite criação de um projeto e para cada projeto *Work Itens que são todos os itens criados: Product Backlog, task, user story* e etc*.* Equipes capturam e atualizar informações entre formulários de item de trabalho usando página de *fallback* de produto e de tarefas. *Products Backlog Itens* define as histórias, os requisitos e os elementos que a equipe precisará criar. Os proprietários do produto geralmente definem, oferece a prioridade, e atribua a um valor de negócio para cada PBI (*Products Backlog Itens*). A equipe armazena na ferramenta o esforço

[3] O Visual Studio Team Foundation Server 2012 (TFS) é a plataforma de colaboração na essência da solução de gerenciamento do ciclo de vida de aplicativos (ALM) da Microsoft.

de trabalho para consulta posterior e o trabalho para fornecer os itens de prioridade mais alta.

A Figura 11 exemplifica os *Products Backlog Itens* criados para esta aplicação, ou seja, o que foi solicitado pelo cliente.

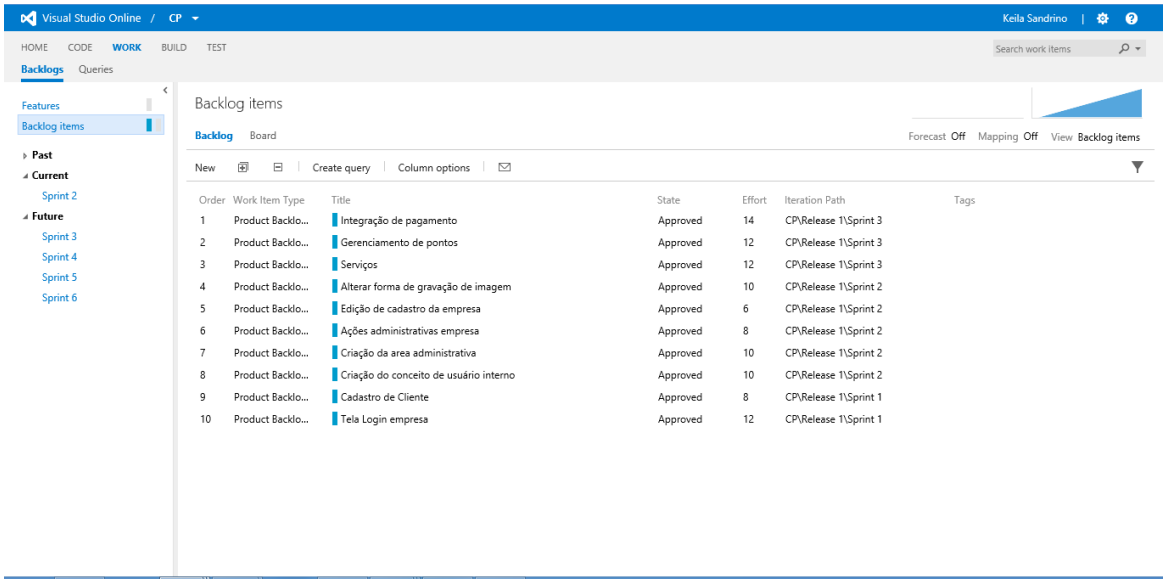


Figura 11 - Lista de *Product Backlog*
Fonte: Autoria própria

A Figura 12 exemplifica em detalhes das informações contidas no *Product Backlog Itens* criado com Tela de Login empresa com as informações de Interações, Status, esforço, responsável, relacionando as *Tasks* criadas para este *Product Backlog.* Possibilita anexar documentos relacionados ao projeto.

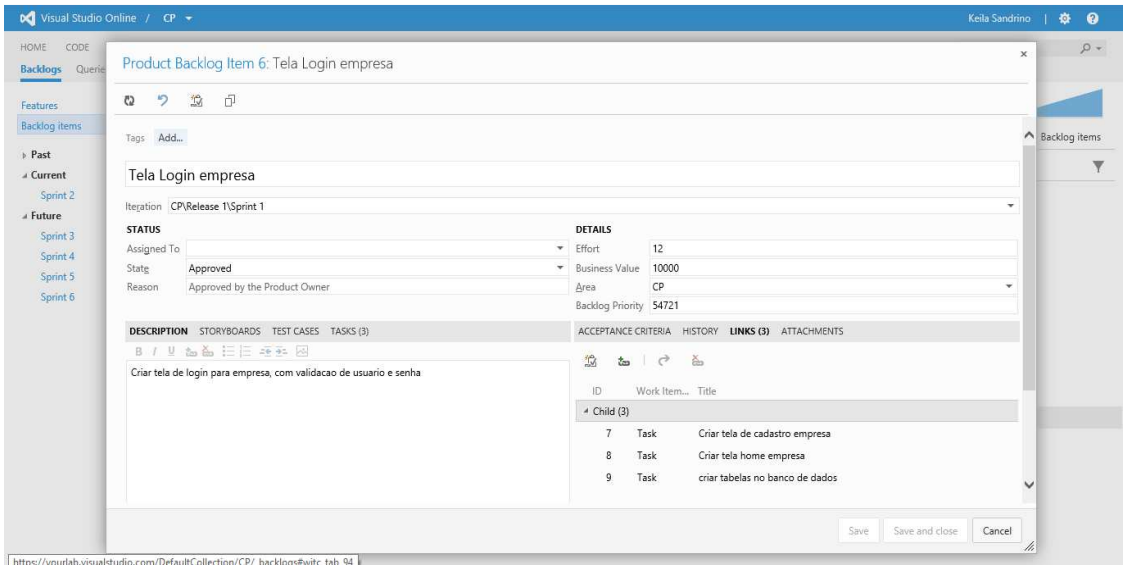


Figura 12 - Descriçãp do *Product Backlog*
Fonte: Autoria Proópria

A Figura 13 exemplifica em detalhes da *Task* nomeada como Criar tela de Cadastro empresa criada para o *Product Backlog* nomeado comoTela Login empresa, contendo a descrição com as informações de situação(*status*), prioridade, atividade e responsável.

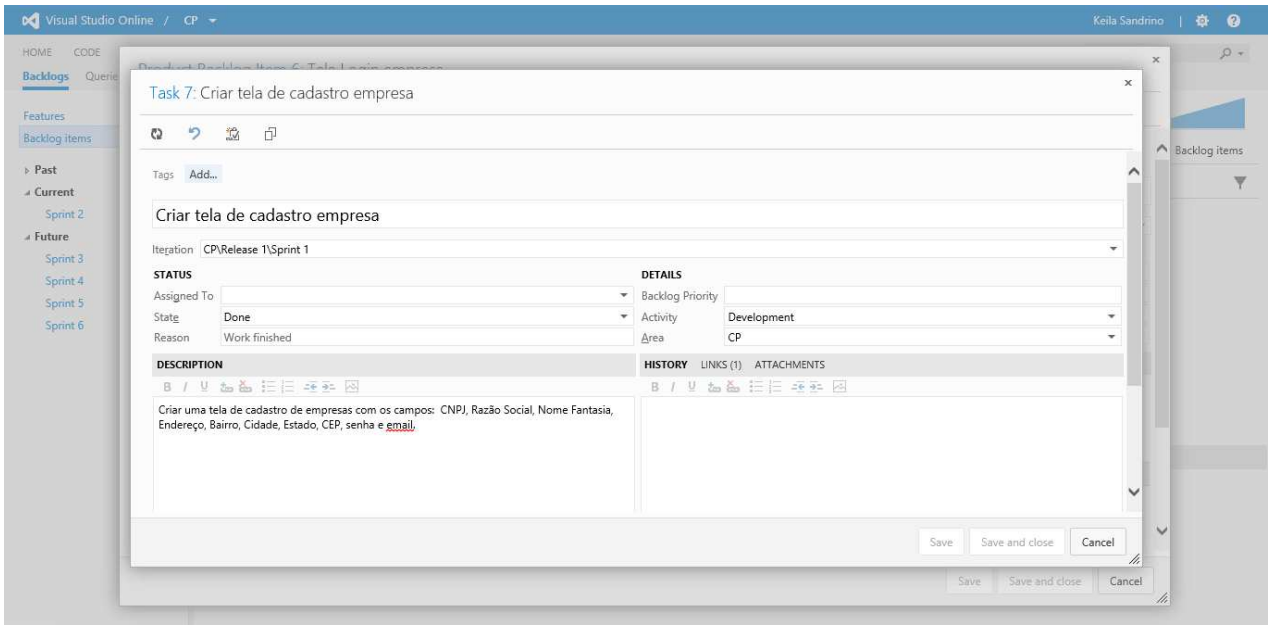


Figura 13 - Exemplo de *Tasks*
Fonte: Autoria Própria

O *Team Foundation Server* permite que após a criação do *Product Backlog* para geração dos ciclos (Sprints) com suas tarefas, atribuir esforço e situação de cada *Task* criada, agrupando as tarefas a serem desenvolvidas em ciclos.

O uso da ferramenta esta sendo feita de forma progressiva, a Figura 14 mostra o *Team Sprint* 2 criado, porém as informações de controle de tempo de conclusão ainda não estão sendo consideradas, pois o time esta em fase de adaptação com o uso da ferramenta e com a nova forma de trabalho se utilizando da metodologia ágil.

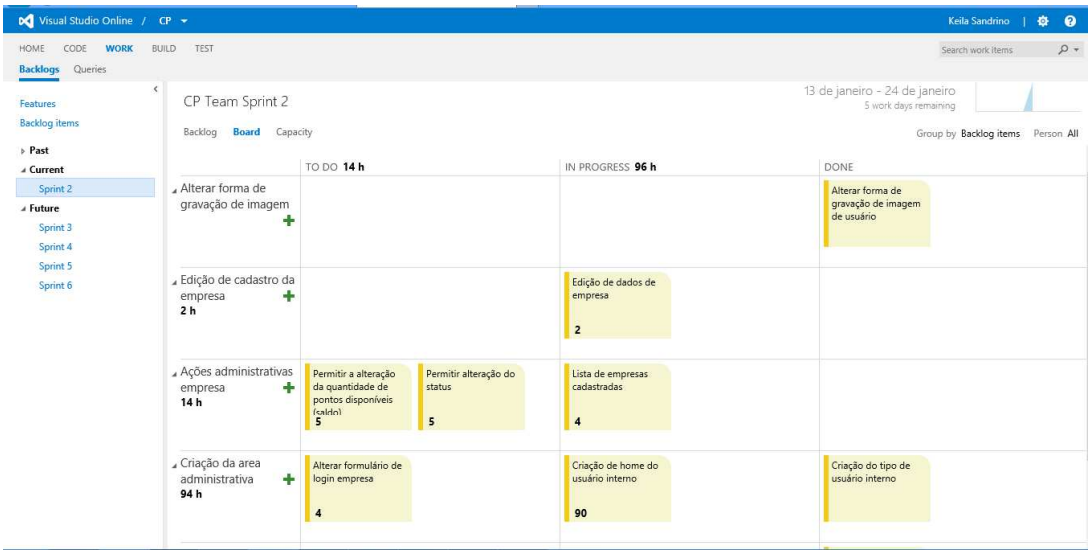


Figura 14 - *Team Sprint*
Fonte: Autoria Própria

Além da ferramenta TFS foi criado um repositório de documentos utilizando a plataforma de aplicação *web* para gestão bastante abrangente. No caso da subpágina do desenvolvimento está sendo utilizada para realizar a gestão documental.

Primeiramente cria-se uma solicitação de desenvolvimento com as informações a respeito do sistema na visão do cliente. Na Figura 15 observa-se a disposição do repositório de documentos no menu Solicitação de desenvolvimento.

Figura 15 -Repositório de documentos- solicitação
Fonte: Autoria Própria

Cada solicitação de desenvolvimento tem sua descrição e um documento anexado, a ferramenta permite que seja em vários formatos de documento como: .doc, pdf, jpg, entre outros. Na figura 16 observa-se a presença do documento no repositório de documentos Solicitação de Desenvolvimento disponível verificar no “ANEXO A”.

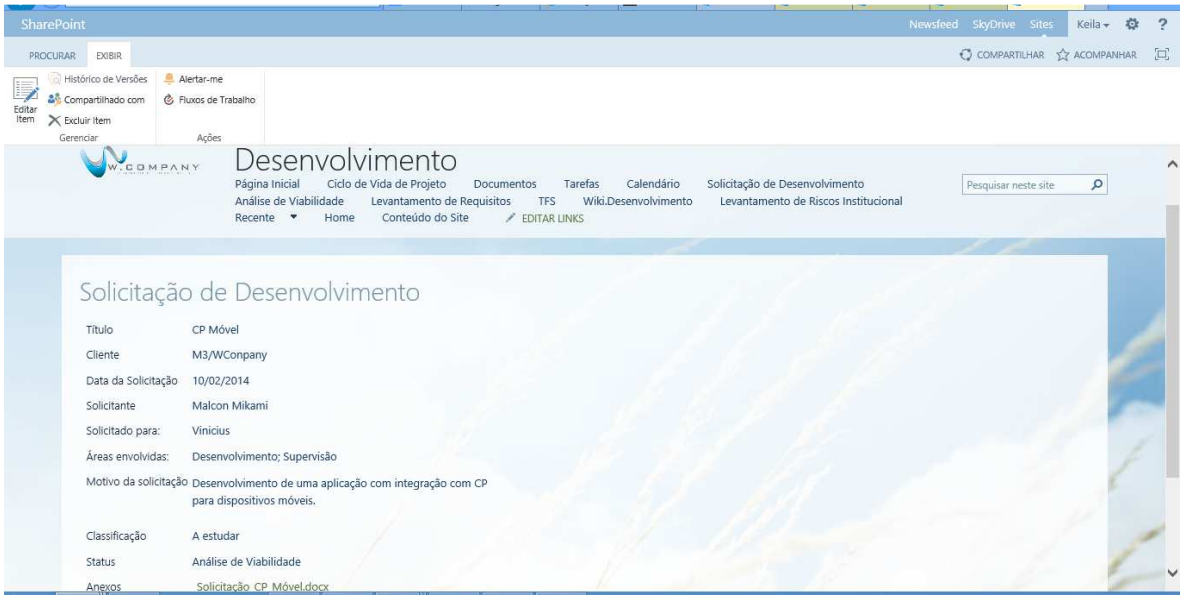


Figura 16 - Repositório de documentos- Soliciatação CP Móvel
Fonte: Autoria Própria

Após a criação da solicitação de desenvolvimento cria-se uma análise de viabilidade para cada solicitação a qual está vinculada conforme figura 17.

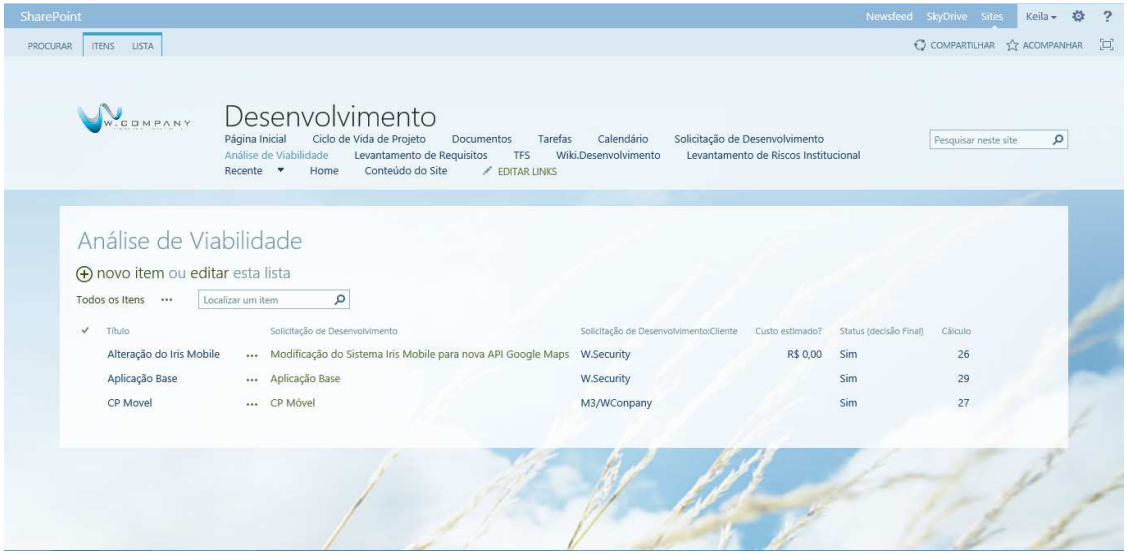


Figura 17 - Repositório de documentos- Análise de viabilidade
Fonte: Autoria Própria

Esta análise possui vários itens que devem ser analisados conforme figura 18.

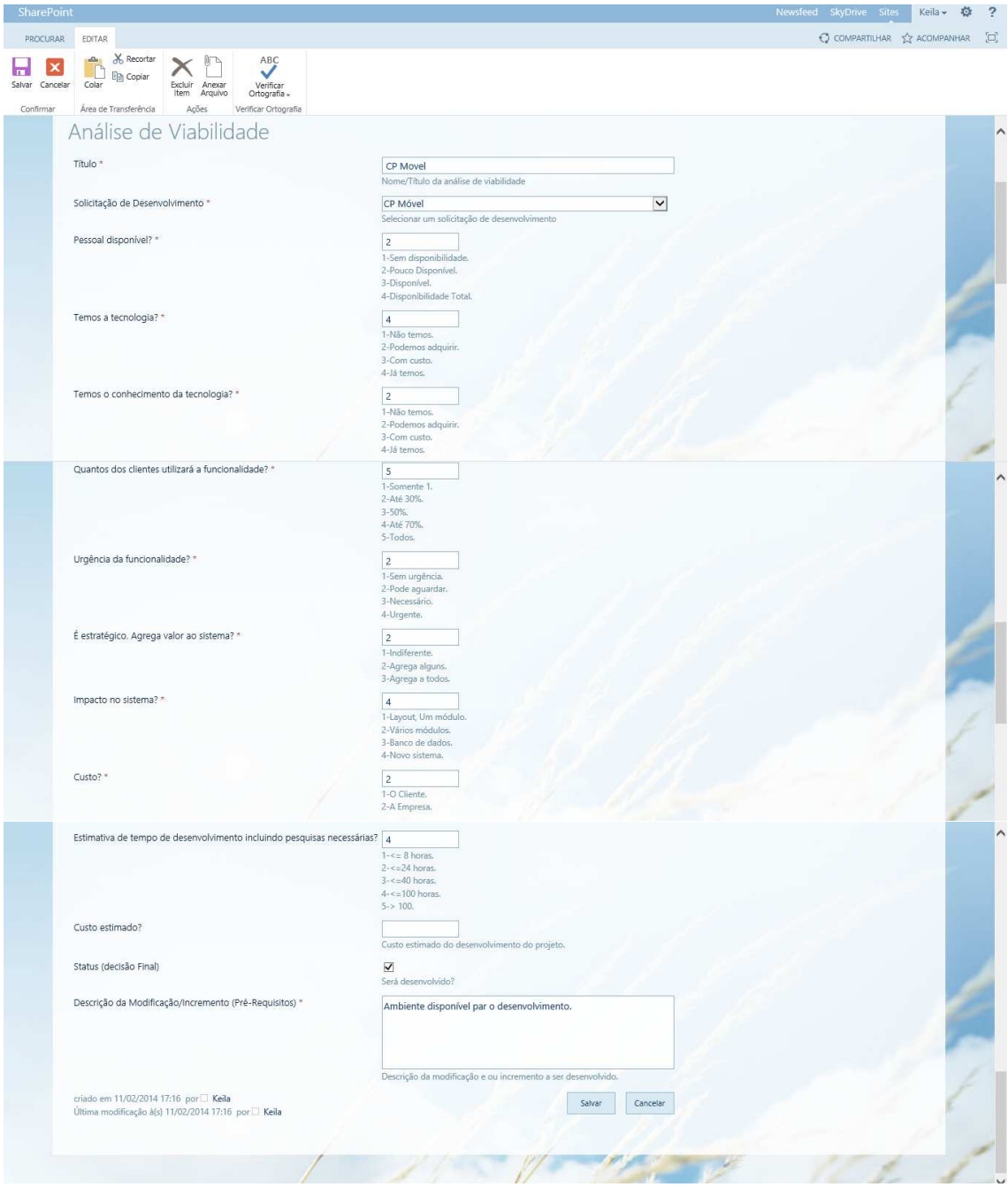


Figura 18 - Análise de viabilidade
Fonte: Autoria Própria

No repositório de documentos criou-se menu para o levantamento de requisito, com as informações a respeito do sistema na visão de negócios. Na Figura 19 observa-se a disposição do repositório de documentos e a forma de armazenamento do documento de Requisitos “ANEXO B”

Figura 19 - Repositório de Documentos- Levantamento de Requisitos
Fonte: Autoria Própria

Em reuniões, a equipe relatou que foram percebidas melhorias com a nova abordagem, em especial, na organização das tarefas repassadas para a equipe. Pois, antes da adoção da metodologia, os desenvolvedores enfrentavam dificuldades na priorização de novas tarefas e para resolver problemas encontrados no decorrer dos trabalhos, como dúvidas em regras de negócio.

A disposição dos documentos organizados no repositório de documentos utilizando a plataforma de aplicações Web da Microsoft trouxe a equipe melhor acesso as informações de forma prática disponibilizando para toda a equipe, permitindo rastrear várias fases iniciais do projeto anteriormente restrita ao analista.

A utilização das boas práticas de programação e padrões de nomenclatura não possibilitou a equipe perceber benefícios de imediato. Acredita-se que possibilidade de observa-se as vantagens ocorrerão no momento em novos colaboradores venham somar à equipe.

4.5 OUTRAS PRÁTICAS DE DESENVOLVIMENTO

Pode-se observar a seguir algumas práticas de desenvolvimento utilizadas pela equipe de desenvolvimento designadas para o projeto especifico CP. Estas práticas

não se aplicam a outros sistemas já existentes na empresa, porém, o conhecimento pode abranger outros projetos futuros.

Conforme o estudo realizado, a técnica de desenvolvimento ágil Behavior-Driven Design (BDD) é guiado pelos valores de negócios, trazendo benefícios para o negócio no qual a aplicação está sendo produzida. Para tal foram criadas as *User Stories* que focam nos objetivos do usuário e como o sistema alcança esses objetivos.

Pode-se observar na Figura 20 o cenário que a funcionalidade descreve. É recomendável que sejam escritas segundo o ponto de vista do usuário, especificando ator, a ação e a funcionalidade desejada.

Exemplo:

```
Cenário: Verificar se usuário está inativo
Dado um usuário inativo
Quando for realizada sua autenticação
Então o sistema deve informar que o usuário está inativo
```

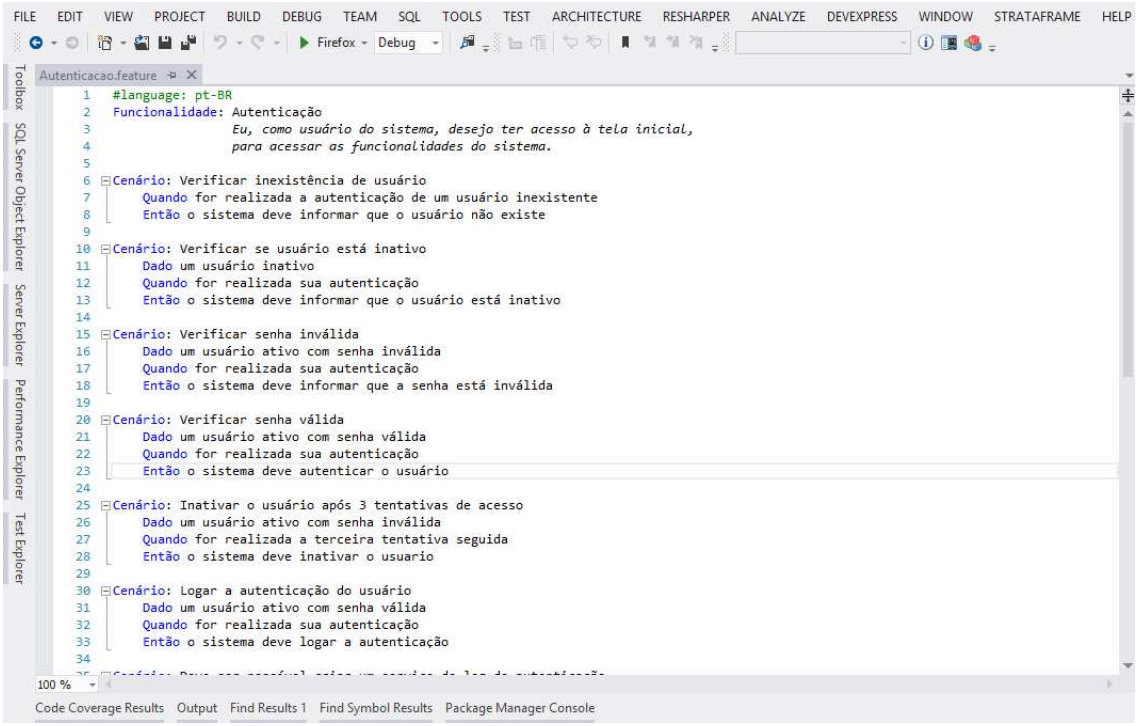


Figura 20 - Exemplo da técnica BDD
Fonte: Autoria Própria

Outra metodologia de desenvolvimento ágil estudada foi o TDD (*Test-driven Development*). O seu principal conceito os seus testes modelem a forma do

desenvolvimento do sistema. Os exemplos retratam a integração entre as metodologias BDD e TDD, pois permitem a criação do teste a partir do cenário.

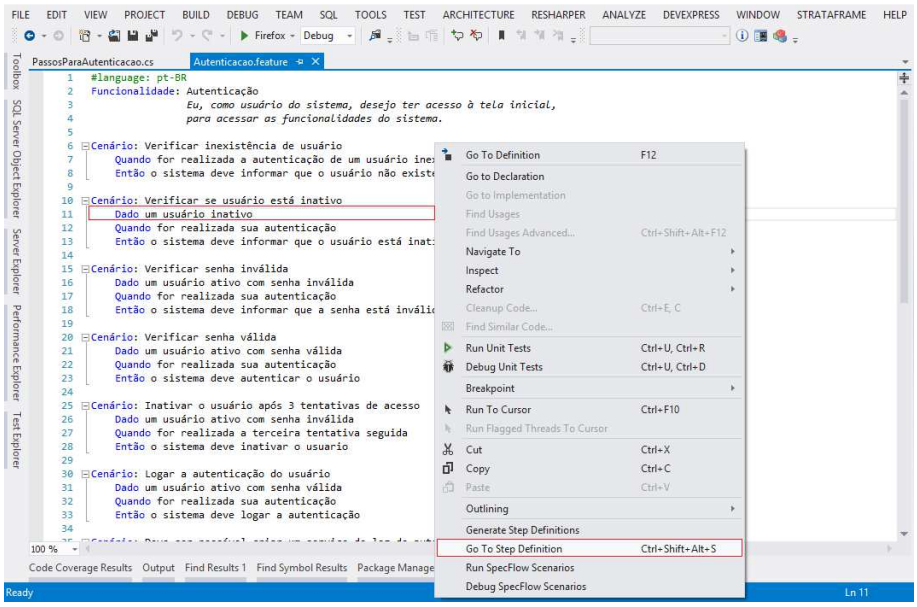


Figura 21 - Integração BDD e TDD
Fonte: Autoria Própria

A Figura 22 exemplifica a implementação da metodologia de desenvolvimento TDD com a construção de teste unitário. A própria ferramenta cria os métodos baseado no cenário para que o desenvolvedor implementar o teste unitário para cada de cenário.

Implementação de Teste Unitario:

```
[Given(@"um usuário inativo")]
        public void DadoUmUsuarioInativo()
        {
            this._usuario = ProvedorDeUsuario.ObterUsuarioInativo();
        }

        /// <summary>
        /// [When(@"for realizada sua autenticação")]
        /// </summary>
        [When(@"for realizada sua autenticação")]
        public void QuandoForRealizadaSuaAutenticacao()
        {
            this._dadosDeLogDeAutenticacao = new DadosDeLogDeAutenticacao();
            var mockDaConsultaDeUsuario = new Mock<IConsultaDeUsuario>();
            mockDaConsultaDeUsuario
                .Setup(m => m.ObterPorNomeDeUsuario(It.IsAny<string>()))
                .Returns(this._usuario);

            var mockServicoDeInfraestrutura = new Mock<IServicoDeComparacaoDeSenha>();
            mockServicoDeInfraestrutura
                .Setup(m => m.CompararSenha(It.IsAny<string>(), It.IsAny<byte[]>()))
                .Returns(this._senhaValida);

            var mockServicoDeLogDeAutenticacao = new Mock<IServicoDeLogDeAutenticacao>();
            mockServicoDeLogDeAutenticacao
                .Setup(m => m.GravarLog(It.IsAny<DadosDeLogDeAutenticacao>()))
                .Callback((DadosDeLogDeAutenticacao dados) => this._dadosDeLogDeAutenticacao.Id =
dados.Id);

            var mockDeRepositorioDeUsuario = new Mock<IRepositorioDeUsuario>();
            mockDeRepositorioDeUsuario.Setup(m => m.Alterar(It.IsAny<Usuario>()));

            var servicoDeAutenticacao = new ServicoDeAutenticacao(mockDaConsultaDeUsuario.Object,
mockServicoDeInfraestrutura.Object,                                mockServicoDeLogDeAutenticacao.Object,
mockDeRepositorioDeUsuario.Object);

            ScenarioContext.Current.Add("Exceção de Autenticação", null);

            try
            {
                servicoDeAutenticacao.Autenticar(this._login, string.Empty, string.Empty, string.Empty);
            }
            catch (Exception e)
            {
                ScenarioContext.Current["Exceção de Autenticação"] = e;
            }
        }

        /// <summary>
        /// [Then(@"o sistema deve informar que o usuário está inativo")]
        /// </summary>
        [Then(@"o sistema deve informar que o usuário está inativo")]
        public void EntaoOSistemaDeveInformarQueOUsuarioEstaInativo()
        {
            Assert.IsTrue(ScenarioContext.Current["Exceção de Autenticação"] is UsuarioInativo);
        }
```

Figura 22 - Implementação do TDD
Fonte: Autoria Própria

Em se tratando de TDD, para escrever um teste o desenvolvedor precisa entender com clareza as especificações e requisitos da funcionalidade. O desenvolvedor pode fazer isso através de casos de uso ou *user stories* que cubram os requisitos e exceções condicionais. A Figura 23 nos mostra exemplo do módulo de testes criado.

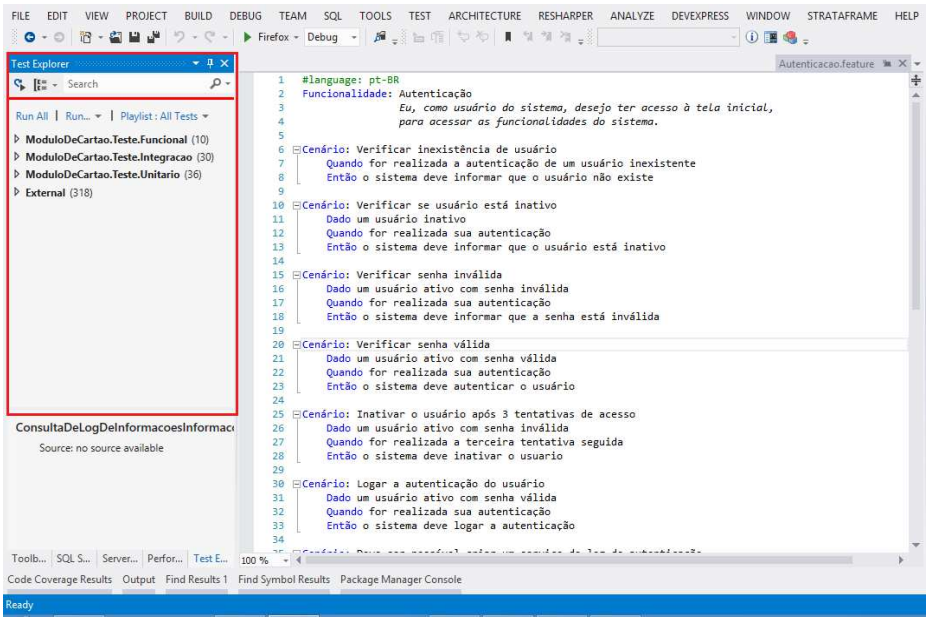


Figura 23 - Módulo de testes Fonte: Sistema Desenvolvido- Estudo de caso
Fonte: Autoria Própria

A figura 24 mostra o teste de cobertura de código, o código escrito até esse ponto poderá não ser perfeito porque posteriormente ele será melhorado. O teste de cobertura executado verifica que porcentagem do código esta sendo verificado ou testado.

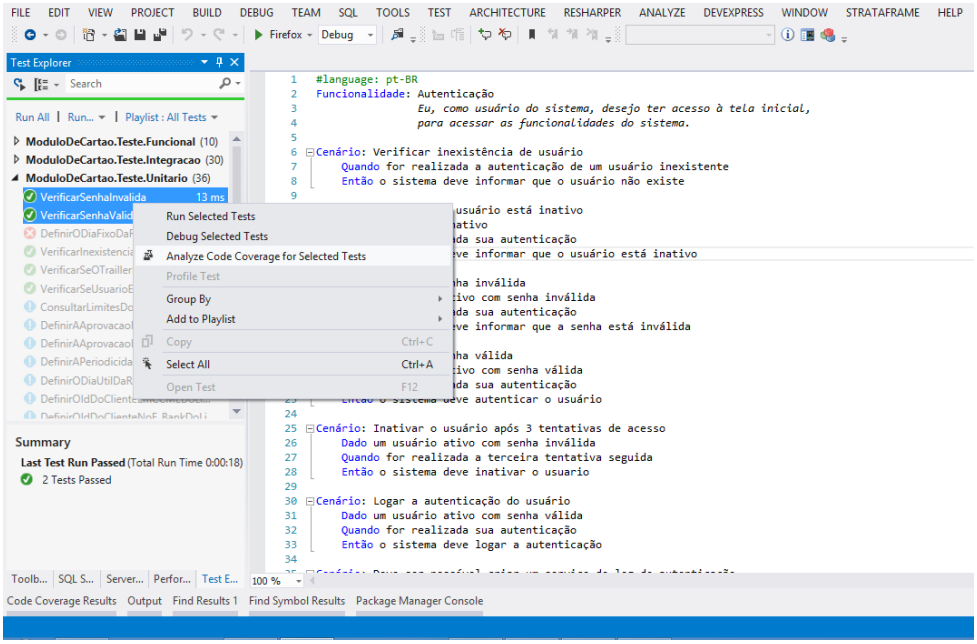


Figura 24 - Exemplo de teste de código

Fonte: Sistema Desenvolvido- Estudo de Caso

A ferramenta utilizada permite verifica quais pontos no sistema os testes não passaram. Na figura 25 observa-se que os testes não passaram pela clausula *thows*.

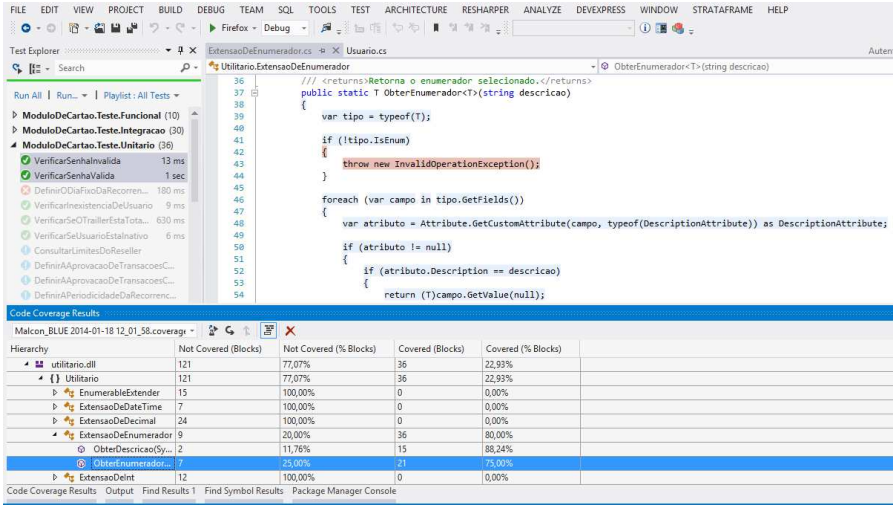


Figura 25 - Resultado dos testes

Fonte: Autoria Própria

4.5.1 Arquitetura

A solução foi inteiramente desenvolvida utilizando o *Visual Studio*, ferramenta de desenvolvimento da *Microsoft*. Inicialmente foi criada uma *solution*, a qual contem projetos separados, separando fisicamente as camadas. Esta foi utilizada pela equipe envolvida no projeto para o desenvolvimento da aplicação Cp para *web*, podendo ser aprimorada para outros projetos, como por exemplo, o desenvolvimento do CP Móvel que se encontra em fase de documentação, conforme a figura 26.

A arquitetura esta dividida da seguinte forma:

- Utiliza o padrão arquitetural *Model-Viel-Controller*
- Realizada modelagem de domínio do sistema e sistemas agregados como, por exemplo, o *facebook* que integra para realizar l*ogin* no sistema CP.
- Criada uma Infraestrutura responsável por fazer o acesso aos dados e controle de transações
- Camada de serviços de aplicação que orquestra as chamadas das camadas de aplicação entre os métodos das camadas de domínio e infraestrutura.
- *Cross-Cutting* camada diretamente responsáveis pelo embaraçamento, ou inter-dependências, consiste inteiramente de chamadas de procedimento

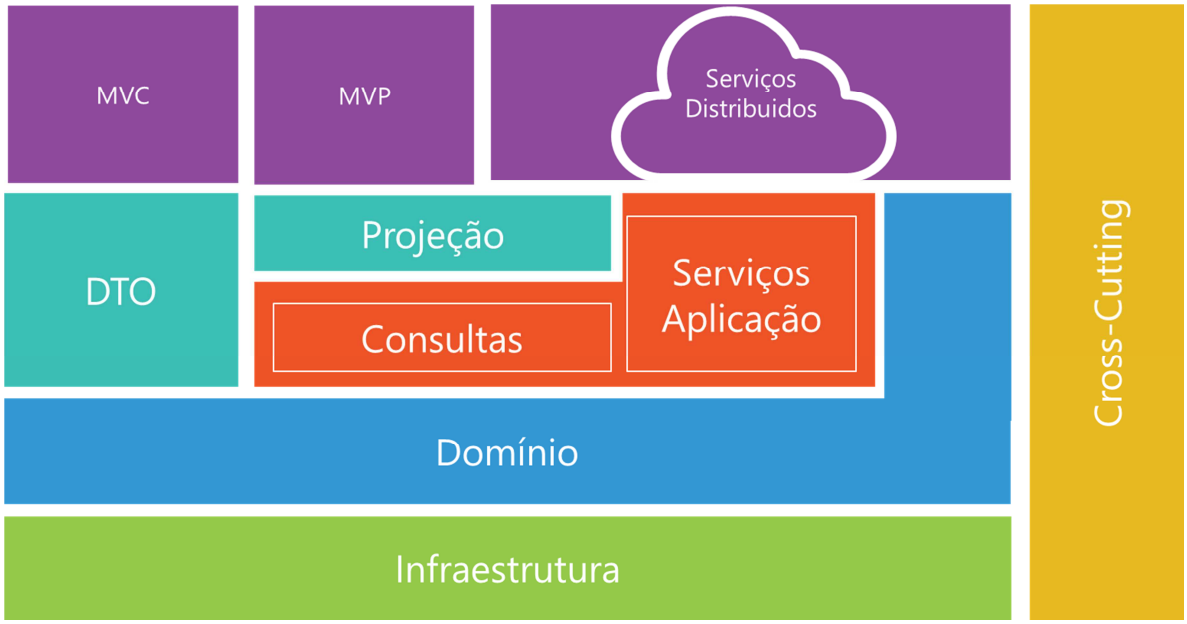


Figura 26 - Modelo Arquitetural
Fonte: Autoria Própria

No projeto CP conforme figura 27, traz a interface de interação com o usuário, a apresentação construída desta forma possibilita a criação mais projetos aplicados em outras plataformas, por exemplo, dos tipos: Windows *forms*[4], *mobile*[5], *silverlight*[6], etc.

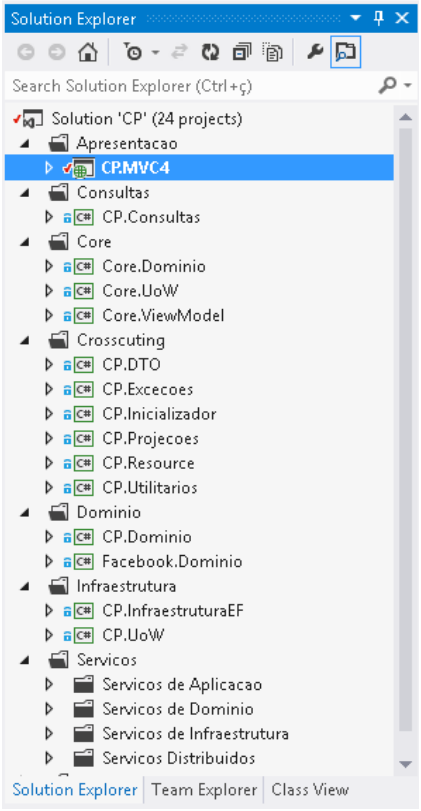


Figura 27 - Separação física das camadas
Fonte: Autoria Própria

A Camada Cp.mvc4 a plataforma web que tem a função de apresentação para o usuário. Para criação do projeto CP foi utilizado asp.net mvc4. A figura 28 mostra no modelo MVC as camadas *controller* e *views*

---

[4] O *Windows* Form é uma versão mais avançada dos formulários que temos usado até então em nossos projetos VB.
[5] Desenvolvimento de aplicações e sistemas para dispositivos móveis.
[6] O *Microsoft Silverlight* é um *plug-in* para vários navegadores e várias plataformas de mídia baseadas em .NET e aplicativos interativos e sofisticados para a *Web*.

Na arquitetura do projeto utiliza-se o padrão arquitetural MVC (*Model View Controller*) mostrando como foram criadas as camadas do modelo. A camada *model* esta representada pelo domínio.

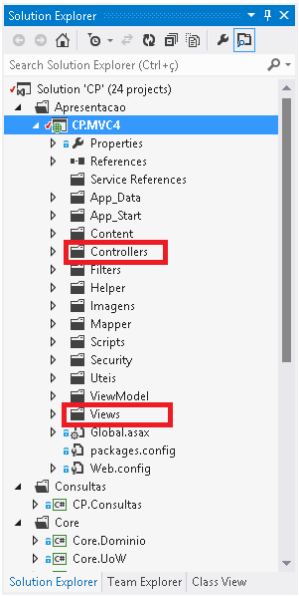


Figura 28 - Camadas MVC
Fonte Autoria Própria

A figura 29 mostra a implementação da camada *Viewmod*el, um objeto que contem os atributos que serão utilizados nas *view* não carregando informações adicionais, podendo conter a junção de 2 ou mais objetos entidades e atributos necessários para comunicação entre a *view* e os *controllers*. A *viewmodel* ainda contem *data annotations* utilizadas para fazer as validações *client-side.*

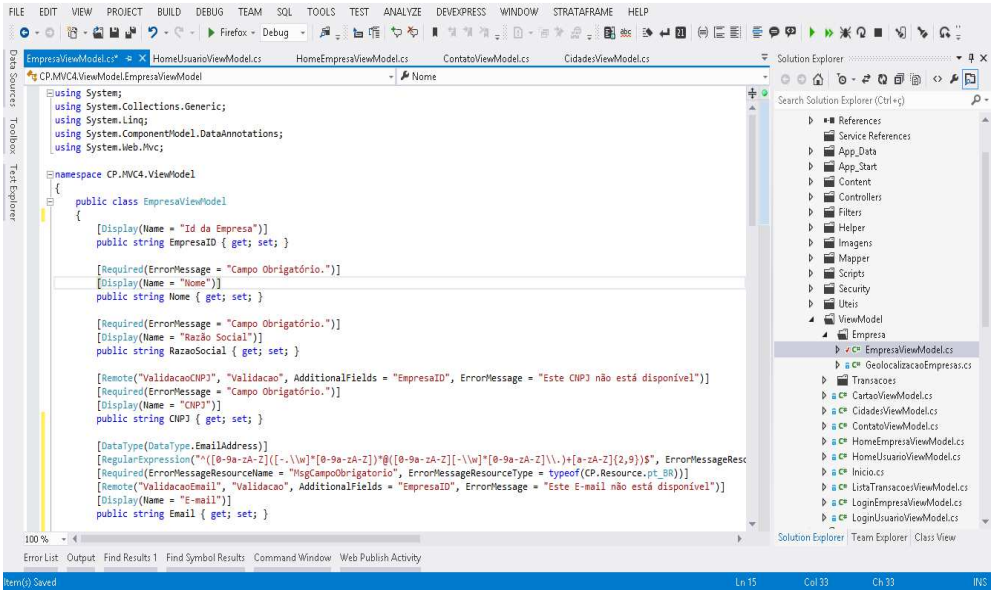


Figura 29 - Viewmodel
Fonte: Autoria Própria

4.6 A DOCUMENTAÇÃO

O estudo mostrou a boa aceitação da equipe à metodologia. Por ter sido rapidamente assimilada pela equipe, a queda de produtividade inicial com a utilização parcial do método *Scrum* e na organização das tarefas repassadas para a equipe não teve alto impacto. Pois, antes da adoção da metodologia, os desenvolvedores enfrentavam dificuldades na priorização de novas tarefas e para resolver problemas encontrados no decorrer dos trabalhos, como dúvidas em regras de negócio.

A documentação de requisitos apresentada, estando integrada a plataforma de aplicações Web utilizada para criação do repositório de documentos, e ferramenta de desenvolvimento, diminuiu os problemas que ocorriam a cada Sprint, pois o repositório auxilia o entendimento do projeto.

A integração com o sistema de controle de projeto e suas tarefas, onde partes do código estavam diretamente relacionados a uma tarefa, e está a uma solicitação de *Product Backlog*, ainda não pode ser avaliada.

Algumas dificuldades foram encontradas na implementação de uma ferramenta de gerenciamento e controle TFS, foi um ponto que a equipe teve de se adaptar, pois esta ainda não tinha sido aprendida em sua totalidade.

A implantação de uma nova arquitetura, apenas conhecida por um dos membros, causou um atraso no inicio, porem este foi contornado pela produtividade nos demais;

Outra dificuldade encontrada no processo de implantação das mudanças, se deu por conta dos atrasos e interferências ocorridas devido a definições dos modelos de documentação e disposição no repositório e documentos.

5 CONCLUSÃO

Analisando o cenário exposto da produção de software, percebeu-se que os maiores problemas enfrentados pela empresa de desenvolvimento de software – atrasos na entrega do projeto, produtos de baixa qualidade e aumento significativo dos custos, por exemplo – são causados principalmente pela falta organização e padronização.

Isso justifica a necessidade da adoção de metodologias e ferramentas que utilizem práticas ágeis que visam uma redução de desperdício e que sobrevivam diante de um ambiente em que as mudanças ocorrem com grande frequência.

Pôde-se observar que a metodologia ágil *Scrum* se mostrou adequada para o uso em ambientes de desenvolvimento de softwares para a realidade da empresa, e que sua aplicação engloba as principais etapas do desenvolvimento, através de pequenas e médias adaptações.

Apesar da adesão à metodologia ser parcial, reduzindo as reuniões, removendo a geração vasta de documentação produzindo somente o necessário, escolhido pelo *Scrum Master* ou *Product Owner*, como foi o caso da geração de um documento de requisitos macro do projeto justificado pela dinâmica de desenvolvimento do negócio. Objetiva-se tornar o desenvolvimento mais pratico, possibilitando a equipe melhor aproveitamento do tempo da equipe no desenvolvimento dos projetos.

5.1 TRABALHOS FUTUROS

Para trabalho futuro, é sugerido implantar o processo de Gerenciamento de Qualidade, no qual o controle da qualidade será feito através de avaliação periódica do desempenho geral do projeto, sendo baseada nas informações coletadas na ferramenta de desenvolvimento.

Além disso, desenvolver uma forma de indicação de dependências entre as tarefas no *Scrum Board*, quando implantado, ajudará a equipe na ordem de execução de suas próprias tarefas.

Por fim, é esperado também, que depois de feitas essas alterações, o *Scrum* seja implantado em todas as equipes da empresa, bem como todas as etapas que a metodologia sugere, aliando assim a documentação com o desenvolvimento e testes integrados.

REFERÊNCIAS

AÉCE, I., **Boas Práticas de Programação- Para termos um código bem enxuto e com boa qualidade não basta apenas termos convenção de nomes, classes, variáveis**, 2007. Publicado em: Linas de Código, Disponível em: <http://www.linhadecodigo.com.br/artigo/1310/boas-praticas-de-programacao.aspx #ixzz2VocdUCwV > Acessado em: 14 jun 2013

ANDRADE, W.; **Entiry Framework Code Firts – Primeiros Passos**, Publicado em: **Development**, 14 out 2013, Disponível em: <http://sloblog.io/+dev/nIvR5FdWsr0/ entity-framework-code-first-primeiros-passos> Acessado em: 20 dez 2013.

BARBOSA, A. AZEVEDO, B.; PEREIRA, B.; CAMPOS, P.; SANTOS, P. **Metodologia Ágil: Feature Driven Development**, Publicado em: Faculdade de Engenharia da Universidade do Porto/ 2008, Disponível em: < http://paginas.fe.up.pt/~aaguiar/es/ artigos%20finais/es_final_22.pdf>, Acessado em: 18 set 2013

BORGES, E. N. **Conceitos e Benefícios do Test Driven Development;** Universidade federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brasil; Disponivel em: http://www.inf.ufrgs.br/~cesantin/TDD-Eduardo.pdf. Acessado em: 30 dez 3013.

BORIA, J.;RUBINSTEIN, V.; RUBINSTEIN, A. **A História da Tahini-Tahini Melhoria de Processos de Software com Métodos Ágeis e Modelo MPS**, PBQP Software, Julho 2013.

EVANS, E. **Domain-Driven Design- Atacando as complexidades no Coração do Software;** Rio de Janeiro, 2ª Edição, 2010.

MANJALY, C. **C# Coding Standards and Best Programming Practices**, Code Project, 5 Dec 2004, Disponível em: <http://www.codeproject.com/Articles/8971/C-Coding-Standards-and-Best-Programming-Practices> Acesso em: 08 jun 2013.

PATRICK, M.; CABRAL, L**, RP: Refactored Programming- Conjunto de boas práticas para uma programação limpa e produtiva**, Publicado em: Revista Cientifica Tecnologus, 30 dez 2010, Disponível em: <http://www.unibratec.edu.br/ tecnologus/wp-content/uploads/2010/12/patrick_cabral.pdf>, Acessado em: 10 jun 2013.

ORTONCELLI, A. R., MACHADO, S. N., MENESES, B. R., JUNIOR, J. V., & GASPAR, M. S., **Uso de Scrum e TDD em um Ambiente Acadêmico de Desenvolvimento de Software**, 2010. Disponível em: Universidade Estadual do Norte do Paraná (UENP), Acesso em 30 dez. 2013

OENING, G., **Behaviour-Driven Development**, Publicado em: Artigo .net Magazine84, Disponível em: http://www.devmedia.com.br/behaviour-driven-development-artigo-net-magazine-84/20413#ixzz2th7bUoII, Acesso em 22 dez. 2013.

SANTANA, A,; TIMÓTEO, A., L.;.VASCONCELOS, A. M. L., **Mapeamento do modelo de Melhoria do Processo de Software Brasileiro (MPS.Br) para empresas que utilizam Extreme Programming (XP) como metodologia de desenvolvimento**, Centro de Informática – Universidade Federal de Pernambuco (UFPE), Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/sbqs/2006/009.pdf, Acesso em: 19 dez. 2013.

SATO, Danilo Toshiaki, **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**,2007, 155f, Dissertação(Mestrado em Ciências) – Instituto de Matemática e Estatística da Universidade de São Paulo, 2007.

SBROCCO, J.H. ; MACEDO, P. C. **Metodologias Ágeis Engenharia de Software sob Medida**, São Paulo, 1ª Edição, 2012.

SCHISSATO Jéssica, PEREIRA Rodolfo, **TDD, DDD e BDD-Práticas de desenvolvimento**, Publicado em 15 jun. 2012, Disponível em: http://www.princiweb.com.br/blog/programação/tdd/tdd-ddd-e-bdd-praticas-de-desenvolvimento.html, Acesso em: 18 dez 2013.

SOARES, M. S. (2004) **"Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software**", Revista Eletrônica de Sistemas de Informação, Vol. 3, p.8-13.

SOUZA, C**. Padrões de Projetos (Design Patterns) para Web com PHP**, Publicado em: Revista Interdisciplinar, 09 dez 2009, Disponível em: < http://www.univar.edu.br/ revista/downloads/pattern.pdf>. Acesso em: 08 jun. 2013.

TARIFA, A., **Padrões de nomenclatura – Guia de consulta rápida**, Publicado em: Linha de Código, Disponível em: http://www.linhadecodigo.com.br/artigo/253/padroes-de-nomenclaturas-guia-de-consulta-rapida.aspx#ixzz2RPfp6UnL, Acessado em: 19 jun. 2013.

XAVIER, Denys William, **Regras e convenções de nomenclatura**, 2010. Publicado em: TI Expert, Disponível em: http://www.tiexpert.net/programacao/java/ nomenclatura.php, Acesso em: 15 jan. 2014.

Kanban por David Anderson: como equilibrar agilidade versus demanda, 2011. Publicado em: Blog Gonow, Disponível em: http://www.gonow.com.br/blog/2011/11/28/ kanban-por-david-anderson-como-equilibrar-agilidade-versus-demanda/, Acesso em: 02 fev 2014

ANEXO A

A- Solicitação de Desenvolvimento

Solicitação - Centralize Pontos Móvel

Data Solicitação:	05/02/2014
Sistema:	Centralize Pontos Móvel
Módulo:	Único
Cliente:	**Ronald Felipe Wolochn/Malcon Mikami**
Analista:	**Keila Sandrino**

A aplicação deve conter uma tela de login de usuário através do facebook. Após a confirmação de login e senha o usuário deve ser redirecionado para uma tela inicial contendo a visualização do cartão virtual (frente e verso).

Caso o usuário não possua cartão a tela inicial do sistema deve conter a opção de solicitação de um cartão virtual, visualização do menu com busca por empresa no mapa e visualização de ultimas transações.

O sistema deve permitir que o usuário realize a visualização das ultimas transações de debito e credito realizadas em seu cartão.

A aplicação deve possibilitar ao usuário cliente realizar busca por empresas credenciadas próxima de sua localização através do mapa.

A aplicação deve possibilitar ao usuário cliente realizar busca por empresas através da cidade em que o usuário esta localizado.

ANEXO B

B- Levantamento de Requisitos

Levantamento de Requisitos

Cliente:	
Analista:	
Data Levantamento:	06/01/2014
Localização:	
Sistema:	Centralize Pontos
Módulo:	Único

1. Solicitação do cliente

   Justificativa

Para identificar o que o cliente efetivamente espera, é preciso, antes de tudo, saber quem são os clientes. A construção de uma aplicação web para fidelização de clientes e marketing vinculado a rede social conseguem-se dados sobre as necessidades, preferências, hábitos de consumo, exigências particulares.

Solicitação

Produzir um sistema que permita fidelizar o cliente à um(s) estabelecimento(s) através de pontos adquiridos de mediante consumo de diversos realizado no estabelecimento. O sistema trás soluções de marketing através de publicações realizadas pela rede social e acesso aos dados mediante autorização do usuário.

A aplicação integrada com a rede social facebook permite ao usuário fazer login através do facebook, redirecionando para a página principal do sistema. Ao

entrar no sistema o usuário deve ter a possibilidade de vincular um cartão fornecido pelo estabelecimento. Ao entrar na página inicial do sistema usuário deve ter acesso às transações de crédito e debito.

A aplicação deve conter login de empresa, permitindo que empresas sem cadastro possam realizar cadastro de empresa e permitir acesso para realizar edição de cadastro da empresa. À empresa é permitido realizar/criar transações de crédito(“C”) e débito(“D”) para um cartão e acesso aos últimos lançamentos realizados.

2. Fornecedor de requisitos

3. Requisitos

3.1 - Tela de login de usuário

A aplicação deve conter uma tela de login permitindo ao usuário entrar no sistema através do facebook, buscando os dados mediante confirmação do mesmo, redirecionando para página principal;

3.2 - Tela de login de empresa

Criar uma tela de login de empresa com validação de usuário e senha

3.3 – Criar tela de cadastro de nova empresa

A tela de cadastros de empresas deve conter os dados cadastrais como: Razão Social, Nome fantasia, CNPJ, Endereço, Telefone, Ramo de negócios.

3.4 - Vinculação de cartão

Quando o usuário entrar no sistema deve ser possível vincular um cartão, fornecido pelo estabelecimento (empresa), ao usuário.

3.5 - Home de usuário

Ao entrar no sistema, na página principal deve listar as informações consolidadas do usuário.

3.6 - Home de empresa

Quando a empresa acessar o sistema, na página principal, deve ter acesso informações consolidadas de empresa.

3.7 - Edição de dados cadastrais de empresa

Como empresa deve ser possível alterar meus dados cadastrais.

3.8 - Transações de empresa

O Sistema deve permitir à empresa criar transações de credito “c” e debito “d” para um cartão que esteja vinculado a um usuário;

3.9 - Mapa

O sistema permite que o usuário tenha acesso as localizações de empresas que estão vinculadas ao sistema.

3.10Detalhamento de transações por empresa

O sistema permite que a empresa tenha acesso as transações de crédito(C) e débito(D) do usuário especifico com detalhes de cada transação.

3.11Detalhamento de transações por usuário

O sistema permite que o usuário tenha acesso as transações de crédito(C) e débito(D) consolidadas em um estabelecimento específico.

4. Recursos adicionais

Não são necessários recursos adicionais para este projeto.

5. Riscos

Referente ao plano de riscos estabelecido pela empresa (Apêndice III).

6. Protótipo

6.1 Tela de login de empresa

Tela de login de empresa com validação de usuário e senha

6.2 Tela vincular cartão