Proposta de Implantação das Práticas de Código Limpo em um Ambiente Ágil

Luiz Henrique Secco Pereira[1], Alisson Ferreira[2]

Faculdade Cidade Verde (FCV) - MBA em Gerência de Projetos em TI

Avenida Horácio Raccanello Filho , 5950 - Novo Centro - Maringá-PR

[1]luizhsp40@hotmail.com, alisson_gf@hotmail.com[2]

Abstract. Clean Code practices ensures the quality of the written code and legibility as well, making the maintenance process less complex. This article shows an approach about the main techniques of Clean Code raised by Robert C. Martin, covering the use of unit tests, Scrum agile methodology and CMMI model. Those topics are the basis for submission of an implementation proposal of the Clean Code techniques in a software development company which already uses Scrum and is certified at level 2 maturity CMMI. The main goal of this proposal is to ensure a bigger improvement in all the development process, demonstrate how deployment of the techniques introduced reach the requirements in the process areas Validation and Verification at level 3 maturity CMMI.

Resumo. As práticas de Código Limpo visam garantir a qualidade do código produzido bem como sua legibilidade tornando seu processo de manutenção menos complexo. Este artigo faz uma abordagem sobre as principais técnicas de Código Limpo levantadas por Robert C. Martin, passando pela utilização dos testes de unidade, metodologia ágil *Scrum* e o modelo *CMMI*. Todos esses tópicos servem de base para apresentação da proposta de implantação das práticas de Código Limpo em uma empresa de desenvolvimento de *software* que já faz uso da metodologia Scrum e é certificada no nível 2 de maturidade do *CMMI*. O objetivo principal da proposta é garantir uma melhoria maior de todo o processo de desenvolvimento, demonstrar como a implantação das técnicas apresentadas cumprem os requisitos das áreas de processo Validação e Verificação do nível 3 de maturidade do *CMMI*.

1. Introdução

Com fundação no ano de 1991, o local foco da proposta atua com desenvolvimento, comercialização, implantação e manutenção de *software*. É especializada no segmento de *software* de gestão de negócios (ERP do inglês *Enterprise Resource Planning*), fornecendo soluções para empresas como cooperativa de agrônomos, hospitais, editoras,transportadoras, dentre outros.

Atualmente faz uso da metodologia ágil *Scrum* para o desenvolvimento de projetos de *software* e há cerca de um ano conquistou a certificação do modelo *CMMI* (*Capability Maturity Model - Integration* ou Modelo de Maturidade em Capacitação - Integração) com nível 2 de maturidade. Para o desenvolvimento de seu ERP que representa o principal produto da empresa, utiliza a linguagem de programação C# (C Sharp) em conjunto com o *framework* de persistência de dados *Nhibernate* 3.0 e o sistema gestor de banco de dados *Microsoft SQL Server* 2008. Em seu portfólio constam os seguintes sistemas:

- ERP customizável sem módulo industrial
- PDV - Automação Comercial
- NFe- Emissor de Nota Fiscal Eletrônica
- CTe - Emissor de Conhecimento de Transporte Eletrônico

A figura 1 representa o organograma da empresa:

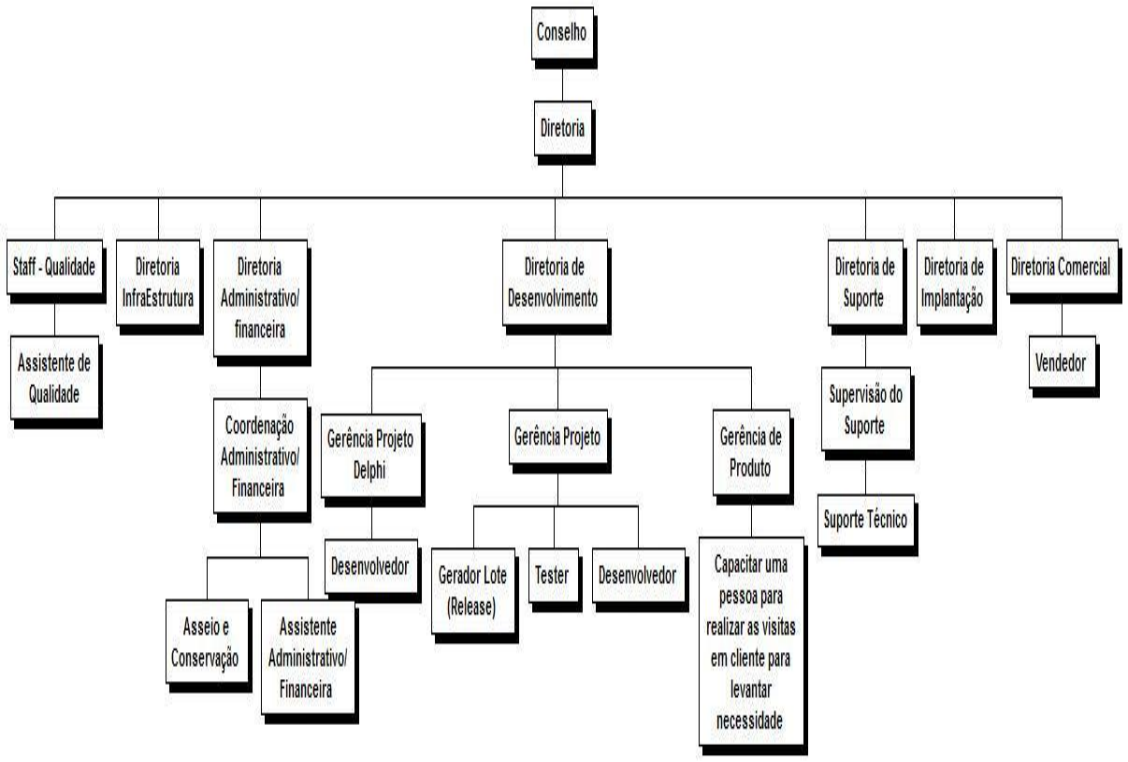

Figura 1. Organograma

Tendo em vista este cenário, esse artigo tem como objetivo apresentar uma proposta de implantação das principais técnicas do Código Limpo no ambiente ágil atual, dando ênfase para as técnicas de testes de unidade, buscando aprimorar ainda mais o processo de desenvolvimento visto que essa melhoria tende a cumprir os requisitos das áreas de processo Validação (VAL) e Verificação (VER) do nível 3 de maturidade do *CMMI*, próximo nível de implementação que a empresa pretende certificar.

2. Fundamentação Teórica

Neste item descrevemos práticas, definições sobre o Código Limpo, *Scrum*, *CMMI* e suas áreas de processo Validação, Verificação. Todos esses temas em tópicos estão relacionados a proposta de implantação das técnicas de Código Limpo.

2.1 Definições de Código Limpo

Em busca de uma definição do que seria um Código Limpo, Martin (2009) através de uma entrevista com desenvolvedores muito experientes, como Ward Cunningham cocriador da *Extreme Programming* (XP), em português Programação Extrema e participante da criação do Manifesto Ágil[1] e Bjarne Stroustrup criador da linguagem de programação C++ obteve como resposta as seguintes características:

- A codificação deve ser simples é de fácil manutenção.
- Testes de unidade devem ser implementados.
- Não pode haver repetições de código.
- Implementação de forma organizada.

Concluindo, o Código Limpo é um conjunto de técnicas de programação que quando praticadas pelos desenvolvedores tem como objetivo garantir a legibilidade e qualidade do código produzido. Segundo Almeida e Miranda (2010) a prática desse estilo de programação busca atingir os seguintes valores: expressividade, simplicidade e flexibilidade cujas características estão presentes nas respostas dos entrevistados. Tais valores possuem as seguintes definições:

- Expressividade: Representa a característica do código fonte produzido relacionada à legibilidade, o código deve possuir um nível de clareza em que outro desenvolvedor possa facilmente entendê-lo e realizar uma modificação;

- Simplicidade : Garante que o código fique livre de informações desnecessárias, e de estruturas complexas que dificultem seu entendimento;

- Flexibilidade: Auxilia o desenvolvedor na criação de uma nova funcionalidade sem a necessidade de uma grande modificação na estrutura existente.

[1]Esboço dos valores e princípios que fundamentam o desenvolvimento ágil de *software*.

2.2. Produzindo Código Limpo

Em um ambiente de desenvolvimento de *software* de uma empresa é comum o desenvolvedor se deparar com códigos que exijam horas e horas de análise para se compreender sua funcionalidade. Martin (2009) associa essa dificuldade como se o desenvolvedor estivesse caminhando por um "lamaçal de arbustos emaranhados com armadilhas ocultas". Muitas vezes o código é produzido em situações de pressa por parte do programador, atrasos no prazo de entrega, e acaba sendo entregue dessa forma sempre com promessas de futuras refatorações, que raramente ocorrem. Ter um código desorganizado e confuso normalmente custa alto para empresa, pois à medida que as alterações são realizadas ao longo dos anos o código se torna cada vez mais instável ao ponto em que influencie na produtividade de toda a equipe e no custo do projeto. A figura 2 apresenta um gráfico de produtividade com o passar do tempo.

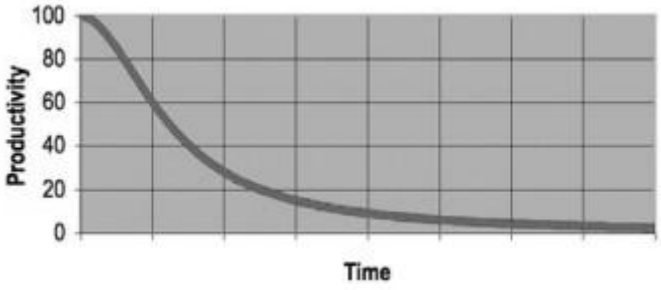

Figura 2 – Gráfico de Produtividade x Tempo

Fonte: MARTIN, 2009, p.4

Segundo Martin (2009), “dedicar tempo para limpar o código não é apenas eficaz em termos de custo, mas uma questão de sobrevivência profissional”. A "limpeza" do código é um processo que requer prática por parte dos desenvolvedores. As próximas seções apresentarão as principais técnicas para atingir um Código Limpo.

2.2.1 Nomenclatura de objetos

Um *software* é composto por diversos objetos implementados pelos programadores, como por exemplo, funções, classes ou métodos. Todos esses objetos devem estar no código fonte nomeados de forma organizada de uma maneira que o código permaneça legível, e que facilite sua localização. Segundo Martin (2009) “Escolher bons nomes leva tempo, mas economiza mais”. Sendo assim umas das primeiras boas práticas é criar nomes que expressem o propósito do objeto que está sendo criado facilitando seu entendimento e uma futura alteração. Para as classes a prática recomenda o uso de substantivos como clientes, fornecedores e para os métodos a utilização de verbos como salvar, carregar. O uso de números sequenciais, trocadilhos, gírias, nomes impronunciáveis jamais devem ser utilizados, pois geram confusão e interferem na legibilidade do código. As figuras 3 e 4 a demonstram exemplos de nomenclatura de classe e de método.

```
public static void ValidarCampoMultiSelecao(KeyPressEventArgs e)
{
    int isNum = 0;
    if (e.KeyChar == ',')
        e.Handled = false;
    else if (!int.TryParse(e.KeyChar.ToString(), out isNum) && (e.KeyChar != (char)Keys.Back))
        e.Handled = true;
}
```

Figura 3 - Exemplo de nomenclatura de método

```
public  class Combustiveis
{

    private int handle;
    private string descricao;

    public virtual int Handle
    {
        get { return handle; }
        set {handle = value; }
    }

    public virtual string Descricao
    {
        get { return descricao; }
        set {descricao = value; }
    }

}
```

Figura 4 - Exemplo de nomenclatura de classe

2.2.2 Trabalhando com funções

As funções são responsáveis por executar uma extensa parte das operações de um *software*, portanto precisam ser escritas com excelência. Segundo Martin (2009) a primeira regra que o desenvolvedor deve seguir ao escrever funções é mantê-las com um tamanho reduzido e executando somente uma operação. A nomenclatura segue os passos citados anteriormente onde o uso de nomes que descrevem suas funcionalidades é recomendado. Outro ponto que merece destaque diz respeito à passagem de parâmetros para a uma função, Martin (2009) afirma que a função ideal é aquela que não recebe nenhum parâmetro, caso não seja possível, a utilização de até três parâmetros ainda é considerada uma boa prática. A quantidade de parâmetros que as funções recebem, influenciam diretamente na complexidade dos testes de unidade, quanto maior a quantidade, maior será a dificuldade, nos casos em que parâmetros exceda a três o desenvolvedor pode optar por dividir a função em outras menores.

2.2.3 Comentários no código fonte

Comentários no código são praticados pelos programadores na maioria das vezes com a intenção de explicar quais operações uma função, método ou um bloco de instruções realizam. Martin (2009) associa a utilização de comentários a código ruim,

pois houve dificuldade do desenvolvedor em deixar o código expressivo. A utilização dos comentários deve ser avaliada, pois comentários em lugares indevidos além de poluírem o código fonte podem conter informações erradas já que não são atualizados com a mesma frequência que o código. Antes de produzir qualquer comentário, segundo Martin (2009) o desenvolvedor deve analisar o código e tentar reescrevê-lo tentando evitar ao máximo comentar, já que em muitos casos basta criar uma função ou método com o nome da mesma informação que seria adicionada como comentário. Caso o comentário seja inevitável, para ser considerado dentro das boas práticas deve estar inserido em uma das seguintes situações:

- Informações sobre direitos autorais em cada módulo, desde que não seja extenso, caso contrário uma referência externa deve ser adicionada ao invés de se listar todos os termos. A figura 5 representa um exemplo de comentário contendo direitos autorais onde o comando // na linguagem C# (C Sharp) é utilizado para comentar uma linha.

```
// Copyright (C) 2003,2004,2005 by Object Mentor, Inc. All rights reserved.
// Released under the terms of the GNU General Public License version 2 or later.
```

Figura 5 - Comentário com direitos autorais

Fonte : MARTIN, 2009, p.55

- Caráter informativo, especificando o funcionamento de uma função ou método. A figura 6 representa um comentário de caráter informativo.

```
//Validação para não permitir que as parcelas fiquem com valores negativos
if ((_listpGeraParcelas != null) && (_listpGeraParcelas.Count > 0))
{
    if (!p_Gera_ParcelasBLL.ValidaParcelas(_listpGeraParcelas))
    {
        ControlMessage.ShowMessage(ResourceMessage.ValorParcelaNegativo);
        pgcPrincipal.SelectedTabPage = tabOutros;
        return false;
    }

    // Validando se o valor das parcelas está diferente do valor da nota
    if (p_Gera_ParcelasBLL.SomaParcelas(_listpGeraParcelas) != (decimal)edtTotalNota.EditValue)
    {
        ControlMessage.ShowMessage(ResourceMessage.ValorParcelaDiferenteTotal);
        if (edtTotalNota.CanFocus)
            edtTotalNota.Focus();
        return false;
    }
}
```

Figura 6 - Exemplo de Comentário Informativo

- Nível esclarecedor, onde são especificados, por exemplo, os parâmetros de entrada de uma função, conforme figura 7.

```
//Complemento do Histórico
if ((bsNotas.DataSource as Notas).TnoHandle.HisComplemento != "")
{
    string tipoDocto = "";
    if (((bsNotas.DataSource as Notas).TnoHandle.TpdocHandle != null) &&
        ((bsNotas.DataSource as Notas).TnoHandle.TpdocHandle.DescReduzida != ""))
        tipoDocto = (bsNotas.DataSource as Notas).TnoHandle.TpdocHandle.DescReduzida;

    (bsNotas.DataSource as Notas).HisComplemento = ControlComplementoHistorico.ComplementarHistorico(
        (bsNotas.DataSource as Notas).TnoHandle.HisComplemento, //Complemento
        edtPesNome.Text,                                        //NomePessoa
        "",                                                     //ApelidoPessoa
        edtNumero.Text,                                         //NumeroNota
        edtData.Text,                                           //DataNota
        tipoDocto,                                              //TipoDocumento
        "",                                                     //NumeroCheque
        "",                                                     //NumeroDocumento
        "",                                                     //Parcela
        "",                                                     //TotalDeParcelas
        edtData.Text.Substring(3),                              //Competencia
        "",                                                     //Banco
        "",                                                     //ContaDebito
        "");                                                    //ContaCredito
}
else
    (bsNotas.DataSource as Notas).HisComplemento = "";
```

Figura 7 - Exemplo de Comentário Esclarecedor

Ao contrário dos comentários que estão presentes na regra do Código Limpo, os utilizados em maior número segundo Martin (2009) são os que fogem a regra, conforme exemplo:

- Repetitivos, são adicionados com o intuito de explicar o código, que pode ser compreendido facilmente apenas com a leitura, conforme figura 8.

```
//Converte a data passada como parâmetro para o formato 140628
public static string DateTimeAnoDuasCasas(DateTime value)
{
    return value.ToString("yyMMdd");
}
```

Figura 8 - Exemplo de Comentário Repetitivo

- Trechos de código como comentários, segundo Martin (2009) seriam a pior prática utilizada nos dias de hoje, são responsáveis por gerar dúvidas entre os desenvolvedores pois nunca se tem certeza sobre sua origem. Com as atuais ferramentas de controle de versão esse tipo de comentário não tem mais serventia.
- Comentários longos, como por exemplo, em um módulo do sistema a cada alteração o programador documentava o que havia sido alterado, nos dia de hoje os sistemas de controle de versão executam esse papel.

2.2.4 Tratamento de Erros da Aplicação

O tratamento de erros da aplicação deve ser implementado de forma organizada, onde segundo Martin (2009) não deve atrapalhar a lógica existente, precisa conter mensagens informativas que facilitem sua localização e informem a fonte do problema. Um tratamento bem executado tende a auxiliar o desenvolvedor na criação dos testes.

Para se manter os tratamentos de erros limpos, segundo Martin (2009) a forma correta é disparar exceções ao invés de se lançar códigos de erro conforme eram utilizados nas linguagens de programações mais antigas. Como não havia suporte a exceções os desenvolvedores disparavam códigos de erros, nas funções ou métodos e os capturavam através do uso e muitas estruturas condicionais para se identificar as falhas. Esse tipo de abordagem sobrecarrega a aplicação e interfere na lógica, conforme ilustra a figura 9.

```
private void btnProcessar_Click(object sender, EventArgs e)
{
    if (ControlMessage.ShowMessageConfirmation(ResourceMessage.DesejaProcessarBalanco))
    {
        int handleBalanco = 0;
        handleBalanco = (int)gridViewPes.GetFocusedRowCellValue("handle");

        EntityManagerNHibernate bll = new EntityManagerNHibernate();
        try
        {
            bll.BeginTransaction();
            new BalancoBLL().GerarNotaBalanco(0, GlobalUsuario.Usuario.Login);
            bll.CommitTransaction();
        }
        catch (SqlException ex)
        {
            bll.RollBackTransaction();
            ControlMessage.ShowMessageError(ex.Message);
        }

        ControlMessage.ShowMessageInformation(ResourceMessage.ProcessoExecutadoComSucesso);
        LoadData(handleBalanco);
    }
}
```

Figura 9 - Exemplo de Tratamento de Erro utilizando uma instrução try-catch

3. Testes da aplicação

Conforme mencionado anteriormente, ter um código confuso e de difícil manutenção custa alto para empresa, pois ao longo dos anos se torna cada vez mais instável ao ponto em que interfira no funcionamento de toda a aplicação. Uma das ferramentas que pode ser utilizada em conjunto com as práticas de Código Limpo apresentadas para garantir a qualidade do código desenvolvido são os testes de unidade, conhecidos também como testes unitários.

A utilização dos testes de unidade irá facilitar o desenvolvedor a dar uma cobertura maior ao código fonte, já que uma das formas de se medir a qualidade de um software é através da porcentagem que os testes o cobrem. O conceito de teste unitário segundo Martin (2012) é originário da *Extreme Programming* e atualmente vem sendo

utilizado em outras metodologias ágeis, a utilização dessa prática visa validar partes de código, como funções, classes e métodos tendo como objetivo verificar se os retornos estão de acordo com o esperado.

O Desenvolvimento Guiado por Testes (*Test-Driven Development* ou simplesmente TDD) segundo Aniche (2012) se tornou popular entre os desenvolvedores em meados de 2002 com a publicação do livro *TDD: By Example* de Kent Beck. Ao iniciar a prática o desenvolvedor entrará em um ciclo onde a primeira regra a ser seguida é escrever primeiro o teste para uma nova funcionalidade, em seguida observar o teste falhar já que a funcionalidade ainda não foi criada, na sequência o desenvolvedor implementa a solução mais simples para que o teste compile e finalmente o código pode ser refatorado caso necessário para melhorar a legibilidade e remover duplicações. A figura 10 exemplifica o funcionamento do ciclo.

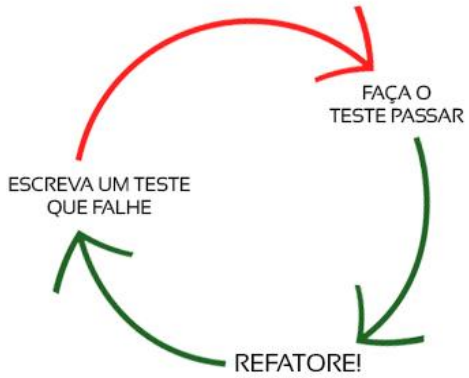

Figura 10 - Ciclo TDD

Fonte : ANICHE (2014)

Assim que o desenvolvedor inicia escrevendo primeiro os testes antes do código de produção, a prática o induz a manter o foco no design da classe, função ou método que está escrevendo, tendo como resultado final somente código que cobrirá os casos de testes, dessa forma deixando o código fonte mais limpo, livre de instruções desnecessárias e complexas. Em comparação com o modelo tradicional de desenvolvimento, onde todos os testes são executados somente após que todo o código de produção tenha sido escrito, segundo Aniche (2012) o TDD tem como destaque proporcionar ao programador que o pratica, uma quantidade maior de retorno sobre o código que está sendo produzido e em menor tempo. Ao receber os retornos em menor tempo o programador pode prever falhas com antecedência, que conforme Aniche (2012) diminui os custos de manutenção e agrega melhoria ao código fonte, ao contrário do modelo tradicional onde o tempo de retorno pode ser bem mais extenso. A figura 11 apresenta um gráfico comparativo entre o desenvolvimento utilizando o modelo tradicional e o guiado por testes.

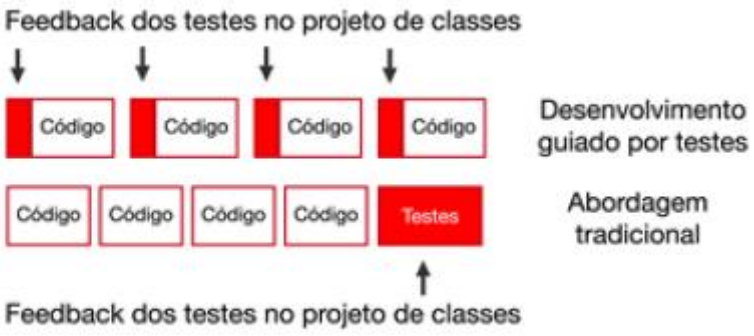

Figura 11 - Comparação da quantidade de retornos entre o TDD e a abordagem tradicional

Fonte : ANICHE, 2012, p.29

Além da quantidade de retorno obtida enquanto se está dentro do ciclo as vantagens que se destacam segundo Aniche (2012) e Martin (2012) são:

- O teste se bem escrito pode ser utilizado por outros desenvolvedores como forma de documentação, pois no teste está especificado como se utilizar cada função, método ou classe do sistema.
- O código de produção surge testado, pois ao praticar TDD seguindo o clico de maneira correta implica que existirá pelo menos um teste unitário validando seu funcionamento.
- Um código de produção coberto por testes unitários dá maior confiança ao programador para promover mudanças ao se deparar com código ruim.

As figuras seguintes exemplificam a utilização do ciclo do TDD, com o auxílio do *framework* de testes *NUnit* de código aberto, compatível com o *Microsoft Visual Studio 2010*. A figura 12 representa o primeiro passo, onde é escrito o teste para função que validará a placa de um veículo.

```
namespace WindowsFormsApplication1
{
    public class PlacaTest
    {

        [Test]
        public void PlacaInvalida()
        {
            string placa = "AAA0001";
            Assert.IsTrue(Placa.ValidatePlaca(placa));
        }

    }
}
```

Figura 12 - Criação do teste para uma função que irá validar a placa de um veículo.

Na sequência o teste é executado, na figura 13 como ainda a função não foi totalmente escrita o teste falhará.

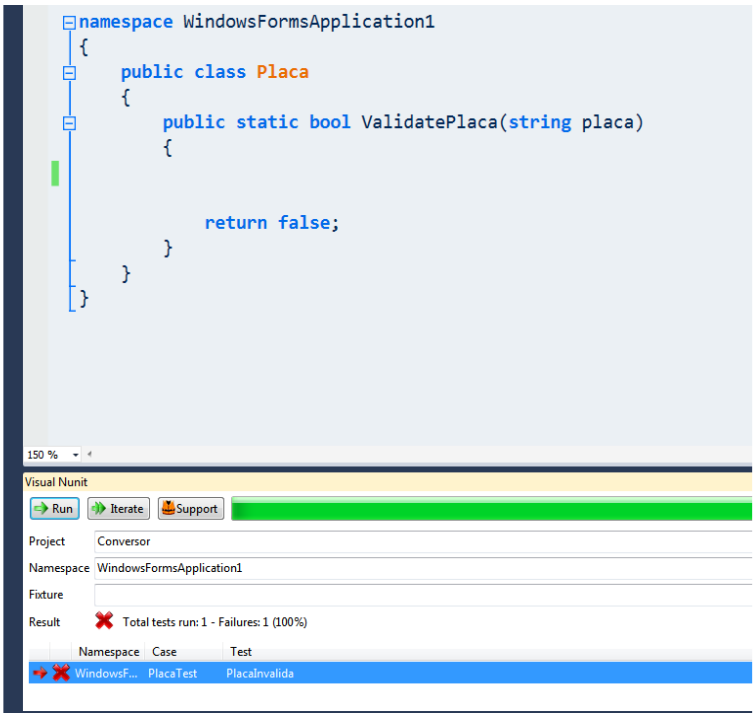

Figura 13 - O teste falha pois ainda a função precisa ser escrita em sua totalidade

A figura 14 , demonstra o momento em que a função é finalizada.

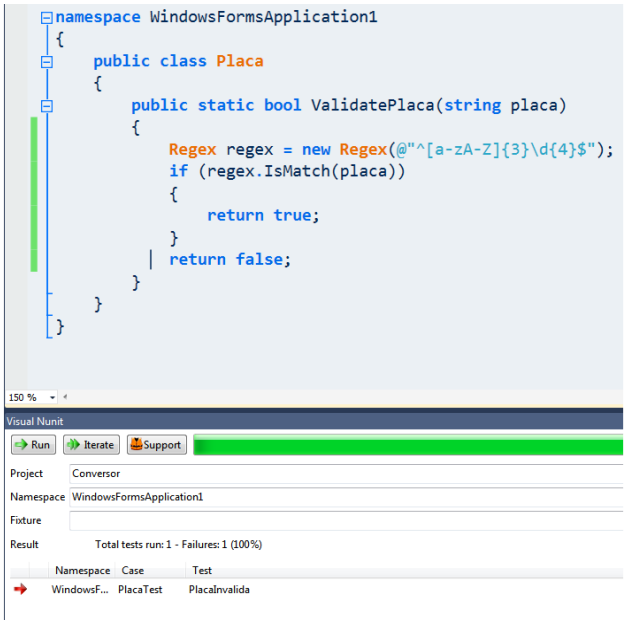

Figura 14 - A função ValidatePlaca é concluída.

Ao executar o teste novamente conforme a figura 15, é possível visualizar que o teste passa.

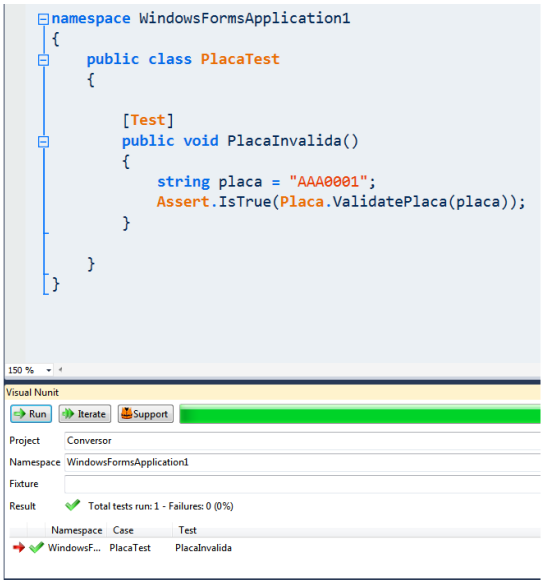

Figura 15 - O Teste é executado novamente e passa

4. Scrum

O termo *Scrum* segundo Pham Phuong-Van; Pham Andrew (2012) foi citado pela primeira vez no ano de 1986 com a publicação do artigo "Um novo jogo para do desenvolvimento de produtos" de autoria de Hirotaka Takeuchi e Ikujiro Nonaka no qual foi apresentado um modelo de desenvolvimento mais veloz e flexível trabalhando

com pequenas equipes visando um objetivo em comum. Os autores do artigo associaram o desempenho das equipes a jogada de mesmo nome do esporte rugby, que consiste em recuperar a posse de bola, nessa jogada os jogadores se agrupam formando uma espécie de muralha, onde o trabalho em equipe e esforço de cada um é de extrema importância para o sucesso da jogada. Em 1995, com a publicação do artigo "*Scrum and the Perfect Storm"* dos autores Ken Schwaber e Jeff Sutherland que o método se tornou conhecido por "*Scrum*".

Segundo (Schwaber e Sutherland, 2011) o *Scrum* é um *framework* utilizado para dar suporte ao desenvolvimento de produtos complexos e é composto por papéis, eventos, artefatos.

4.1 Papéis presentes no Scrum

Os três papéis que compõem o ciclo do *Scrum* segundo seus criadores (Schwaber e Sutherland, 2011), são o *Product Owner*, *o Scrum Team* (ou simplesmente time) e o *Scrum Master*.

- *Product Owner(P.O)*: Responsável direto pelo produto e conhecedor de todos os seus requisitos. Esse papel pode ser atribuído ao cliente ou a quem represente os envolvidos no projeto, como por exemplo o gerente de projetos. É responsável por definir e priorizar os requisitos do produto a serem realizados pelo time, esclarecer dúvidas a respeito dos requisitos e tem o poder de aceitar ou rejeitar as entregas do time.
- *Scrum Team :* Equipe de desenvolvimento, geralmente composta de 5 a 9 desenvolvedores. Selecionam os itens priorizados pelo P.O para serem desenvolvidos, auxiliam na avaliação da complexidade das atividades, entregam tudo o comprometido e realizam uma demonstração de tudo que foi desenvolvido ao P.O.
- *Scrum Master:* Segundo Schwaber e Sutherland (2011) é o responsável por garantir que todas as práticas sugeridas pelo *Scrum* estejam sendo seguidas pela equipe, trabalha em conjunto com P.O na definição e no gerenciamento do projeto, participando dos eventos e atuando como facilitador. Garante que o time fique focado nas atividades não permitindo que interferências externas afetem a produtividade do time.

4.2 Eventos

Os eventos são todas as reuniões, que dentro da metodologia possuem uma duração fixa de tempo. Cabe ao *Scrum Master* guiar os eventos para que sejam produtivos e não se perca o foco. São eles: *Sprint Planning*, *Daily Scrum*, *Sprint Review*, *Planning Poker* e *Sprint Retrospective.* "No *Scrum* o progresso do projeto é baseado em uma série de iterações definidas chamadas de *Sprints*" (Marçal, Torreão e Pereira ,2007) onde é recomendado que não ultrapassar o período de quatro semanas de duração.

4.2.1 *Sprint Planning*

É a reunião de planejamento onde o P.O expõe para o time as atividades previamente priorizadas a serem executadas. Os membros do time fazem uma análise

das atividades que conseguirão realizar e em seguida segundo Marçal, Torreão e Pereira (2007) uma técnica de estimativa conhecida como *Planning Poker* é utilizada para definir o esforço para cada atividade , onde pode ser medido em horas/tamanho.

4.2.2 *Planning Poker*

Consiste em uma técnica de estimativa em grupo (Marçal; Torreão; Pereira, 2007), onde cada participante da reunião recebe um conjunto de cartas contendo a sequência (1, 2, 3, 5, 8, 13, 20,40) que representarão a estimativa de tempo para cada atividade a ser votada. Esse evento conta com a participação de todo o time, incluindo o P.O, e têm seu inicio assim que a primeira atividade priorizada é apresentada, nesse momento cada membro seleciona uma carta que se acredita ser o tamanho que a atividade representa e aguarda até que todos tenham escolhido.

Após todos terem feito suas escolhas as cartas são reveladas, se há um consenso entre todos a pontuação é atribuída a atividade, caso contrario é necessário que os participantes expliquem o motivo de suas escolhas para que sejam analisadas em conjunto. Ao término de todas as discussões é iniciado uma nova votação para que todos reavaliem suas escolhas,as atividades que receberem pontuações altas devem ser quebradas em outras menores para facilitar as estimativas. A figura 16 demonstra um exemplo de cartas utilizadas durante o *planning poker*.

Figura 16 - Cartas utilizadas no *planning poker*.

Fonte : MARÇAL, TORREÃO, PEREIRA,2007,p.68

4.2.3 *Daily Scrum*

Representa a reunião diária feita de pé com duração de 15 minutos, realizada pelo time para avaliar o andamento das atividades e mediada pelo *Scrum Master* para que não ultrapasse o tempo estimado. Cada participante deve responder a três perguntas que de acordo com Marçal, Torreão e Pereira (2007) são:

1. O que foi feito desde ontem ?
2. O que planeja fazer para amanhã ?
3. Você tem algum impedimento ?

Se houverem outros problemas identificados durante a reunião diária, eles devem ser tratados separadamente em outra ocasião.

4.2.4 *Sprint Review*

Reunião onde o time apresenta todas as atividades desenvolvidas que passarão pela avaliação e aceite do *Product Owner*.

4.2.5 *Sprint Retrospective*

Essa reunião ocorre após a *sprint review*, onde o time faz uma auto avaliação apontando os pontos positivos e negativos ocorridos durante o período em que trabalharam nas atividades priorizadas. De acordo com Schwaber e Sutherland (2011) é um evento de lições aprendidas, que pode gerar um plano de melhorias para ser aplicado no próximo planejamento.

4.3 Artefatos

Dentro da metodologia os artefatos servem como ferramentas de apoio e são: *Product Backlog*, *Sprint Backlog* e o gráfico *Burndown Chart*.

4.3.1 *Product Backlog*

O *Product Backlog* representa a lista de todas atividades priorizadas a serem executadas, onde o P.O é responsável por seu conteúdo. De acordo com Kniberg (2007), se trata de uma lista de requisitos, que o cliente tem necessidade, descritas utilizando a terminologia do cliente. Os itens que se encontram no topo da lista são os primeiros itens a serem estimados e priorizados.

4.3.2 Sprint Backlog

São as atividades selecionadas do *Product Backlog* que serão desenvolvidas pelo time naquela *Sprint*.

4.3.3 *Burndown Chart*

É o gráfico que tem como objetivo demonstrar a evolução das atividades que estão sendo executadas, dia a dia. Nesse gráfico de acordo com Marçal,Torreão e Pereira (2007) é exibido o progresso diário produzido pelo time em função da soma total das horas dos itens priorizados do *Product Backlog*. A figura 17 representa o gráfico de *burndown.*

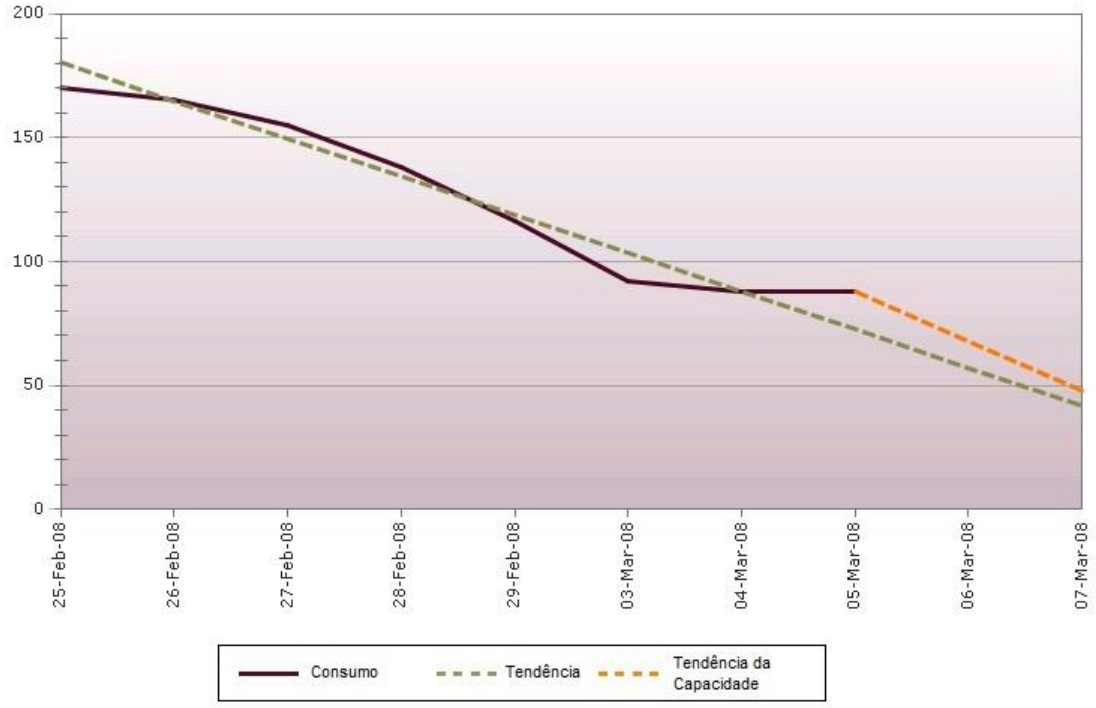

Figura 17 - Gráfico *burndown*

MARÇAL, 2009, p.45

4.4 Funcionamento do *Scrum*

A metodologia *Scrum* assim como o TDD, funciona em um ciclo de iterações, onde no *Scrum* possui tempo definido que pode variar de 2 até 4 semanas, conhecida como *Sprint.* Uma nova *Sprint* é iniciada assim que a anterior é finalizada, havendo sempre entrega de produto em cada encerramento.

O primeiro passo antes do início da *Sprint* é realizar o *Sprint Planning* (reunião de planejamento) onde o *Product Owner* demonstra e prioriza todas as atividades que precisam ser realizadas. Após a fase de planejamento, o ciclo é iniciado, e seu controle é realizado pelo time através das reuniões diárias e pelo acompanhamento das horas apontadas no gráfico *burndown chart*. Assim que a *Sprint* se encerra, o time precisa apresentar tudo que o foi produzido para receber o aceite ou não do P.O, esse procedimento é executado durante a reunião de revisão (*Sprint Review*). Com o fim da *Sprint Review*, o próximo e último evento a ser executado antes de iniciar uma nova *Sprint*, é a reunião de retrospectiva realizada com o time, onde são levantados todos os pontos positivos e negativos que ocorreram durante o *Sprint*, no caso de pontos negativos são discutidas as possíveis soluções para que isso não volte a ocorrer. Esse evento tem como objetivo principal levantar as lições aprendidas, que podem ser aplicadas para melhorar todo o ciclo nas próximas *Sprints*, a figura 18 exemplifica o ciclo do *Scrum*.

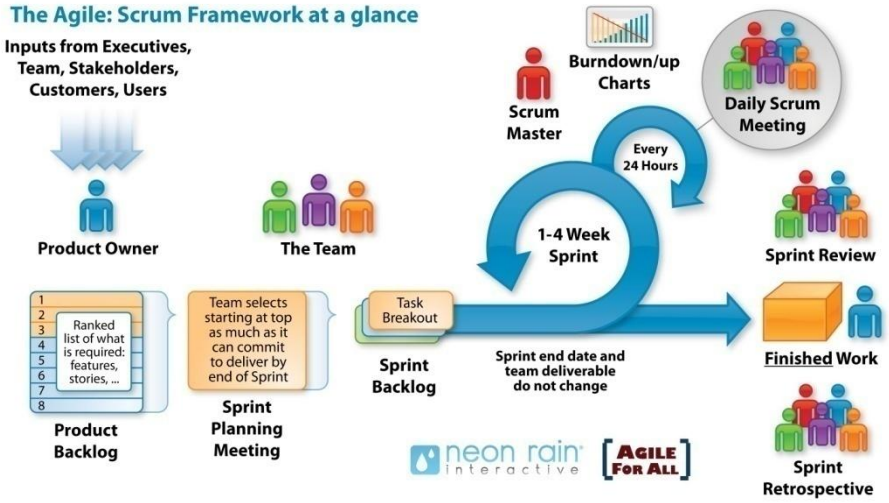

Figura 18 - Clico do *Scrum*

Fonte : Agile For All

5. Apresentação do modelo CMMI

Desenvolvido pelo SEI (*Software Engineering Institute*), o *CMMI* trata de melhoria de processos organizacionais e fornece as melhores práticas para levar a organização à atingir o processo de melhoria contínua para o desenvolvimento de software, segundo Marçal (2009) cobre todo o ciclo de vida de um produto desde o momento em que foi criado até a fase de sua entrega e manutenção. Em suma serve como um guia para melhorar os processos existentes na empresa e elevar suas habilidades para o gerenciamento, aquisição e a manutenção de produtos e serviços.

Os principais benefícios recebidos que se destacam com a implantação do *CMMI* pelas organizações segundo Rouiller, Magalhães e Vasconcelos (2005) são:

- Uma visão comum e integrada da melhoria de todos os elementos de uma organização
- Redução de custos de treinamento e avaliações
- Avaliação eficiente e efetiva de múltiplas disciplinas de processos ao longo de toda a organização.

5.1 Estrutura

O *CMMI*, atualmente é composto de 22 áreas de processos divididas em duas representações: uma por estágios e outra contínua. Uma área de processos (do inglês *Process Area* ou simplesmente PA) pode ser definida como "um conjunto de práticas relacionadas a uma área que ao serem executadas em conjunto, satisfazem os objetivos definidos como importantes para a melhoria desta área" (MARÇAL; ANA SOFIA CYSNEIROS, 2009, p.30). A representação contínua possui em sua definição níveis de capacidade por área de processo enquanto a representação por estágios é organizada em cinco níveis de maturidade.

Na representação contínua é possível que a empresa selecione uma única área que deseja buscar melhoria ou várias áreas que representem seus objetivos estratégicos. Segundo *CMMI Product Team* (2006) a utilização da representação contínua permite que a empresa promova melhoria em processos distintos com diversos objetivos ao longo de sua utilização. Sua utilização é recomendada quando se tem total conhecimento dos processos que serão passíveis de melhoria e ainda se as dependências entre as áreas do modelo *CMMI* estão bem resolvidas.

A representação por estágios onde cada estágio representa um nível de maturidade a ser conquistado pelas empresas, segue um modelo formal para a melhoria de processos focando um estágio por vez. Cada nível conquistado garante uma estrutura de processos que servirão de base para o nível seguinte. De acordo com Rouiller, Magalhães e Vasconcelos (2005) quando uma empresa conquista um nível de maturidade pode se dizer que seus processos agora possuem mecanismos que garantem a repetição sucessiva de bons resultados. A adoção do modelo por estágios é recomendada quando não se conhece todos os processos que necessitam de melhoria. As figuras 19 e 20 representam na devida ordem, a representação contínua e a representação por estágios.

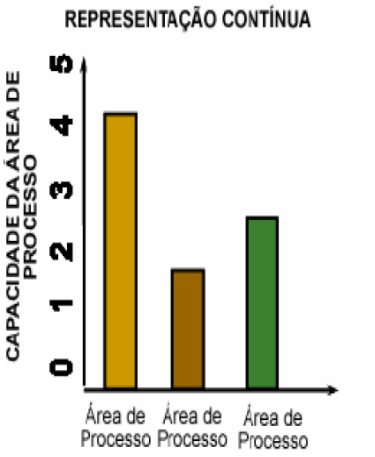

Figura 19 - Representação contínua

Fonte : MARÇAL, 2009, p.31

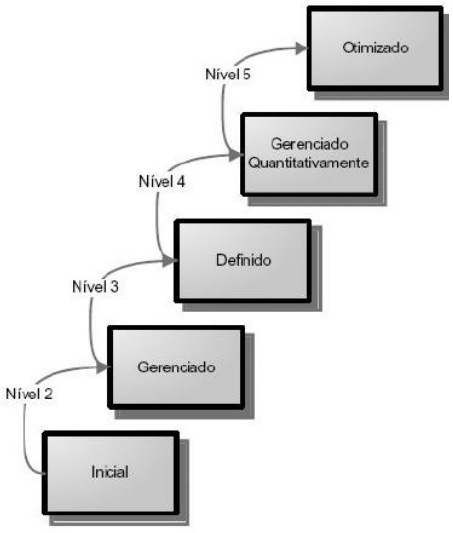

Figura 20 - Representação por estágios

Fonte : DIAS, 2005, p.55

A tabela I apresenta um comparativo das vantagens entre as duas representações que de acordo com o *CMMI Product Team* (2006) pode servir de ajuda na escolha da representação que mais se adéqua a empresa.

Representação Contínua	*Representação por Estágios*
Permite livre escolha da sequência de melhorias, de forma a melhor satisfazer aos objetivos estratégicos e mitigar as áreas de risco da organização.	Permite que as organizações tenham um caminho de melhoria predefinido e testado.
Permite visibilidade crescente da capacidade alcançada em cada área de processo.	Foca em um conjunto de processos que fornece à organização uma capacidade específica caracterizada por cada nível de maturidade.
Permite que melhorias em diferentes processos sejam realizadas em diferentes níveis.	Resume os resultados de melhoria de processo em uma forma simples: um único número que representa o nível de maturidade.
Reflete uma abordagem mais recente que ainda não dispõe de dados para demonstrar seu retorno do investimento.	Baseia-se em uma história relativamente longa de utilização, com estudos de casos e dados que demonstram o retorno do investimento.

Tabela I - Comparativo das vantagens entre as representações contínua e por estágios

Fonte : SEI CMMI-DEV. V1.2, 2006, p.10

Para que a empresa obtenha a melhoria contínua em seus processos, a qual é o resultado da aplicação das práticas do modelo, é necessário seguir os passos evolutivos entre os cinco níveis de maturidade numerados de 1 a 5 (MARÇAL, 2009) que estão representados na tabela II.

Nível de Maturidade	Descrição
1: Inicial	Os processos são informais e caóticos. A organização normalmente não possui um ambiente estável. O sucesso destas organizações depende da competência e heroísmo das pessoas e não do uso de processos comprovados. Apesar deste ambiente informal e caótico, organizações muitas vezes produzem produtos e serviços que funcionam; entretanto, elas frequentemente excedem o orçamento e o cronograma de seus projetos.
2: Gerenciado	Os requisitos, processos, produtos de trabalho e serviços são gerenciados. A situação dos produtos de trabalho e a entrega dos serviços são visíveis para o gerenciamento em marcos definidos. Compromissos são estabelecidos entre os stakeholders relevantes e são revistos conforme necessário. Os produtos de trabalho são revisados com os stakeholders e controlados. Os produtos de trabalho e serviços satisfazem seus requisitos, padrões e objetivos definidos.
3: Definido	O conjunto de processos padrão da organização é estabelecido e melhorado ao longo do tempo. Os projetos estabelecem seus processos definidos adaptando o conjunto de processos padrão da organização. A alta gestão da organização estabelece os objetivos dos processos e assegura que estes objetivos estão sendo tratados de forma adequada. Neste nível, os processos são gerenciados de forma mais pró-ativa, utilizando um entendimento dos inter-relacionamentos das atividades de processos e medidas detalhadas do processo, seus produtos de trabalho e seus serviços.
4: Gerenciado Quantitativamente	São selecionados os sub processos que contribuem significativamente para o desempenho geral do processo. A qualidade e o desempenho do processo são entendidos em termos estatísticos e são gerenciados durante toda a vida dos processos. Medidas de qualidade e desempenho de processos são incorporadas ao repositório de medições da organização para dar suporte a futuras decisões baseadas em fatos ocorridos.
5: Otimizado	Os processos são continuamente melhorados com base em um entendimento quantitativo das causas comuns de variações inerentes aos processos.

Tabela II - Níveis de maturidade do modelo CMMI

Fonte : MARÇAL, 2009, p.32

As áreas de processo que compõem o modelo *CMMI*, segundo Rouiller, Magalhães e Vasconcelos (2005) podem ser agrupadas em quatro categorias:

- Gerenciamento de Projeto: são as atividades que correspondem manter o planejamento do projeto, como monitoramento do progresso das atividades, manutenções.
- Engenharia: representa as atividades de engenharia que definem o desenvolvimento e a manutenção do produto em um cenário de melhoria de processo.
- Suporte: correspondem as atividades que fornecem apoio ao desenvolvimento, manutenção e aquisição de produtos para toda empresa, e não só para um projeto em específico.
- Gerenciamento de Processo: agrupa as ações que possuem o objetivo de instanciar processos e padrões organizacionais para as necessidades especificas do projeto.

A tabela III demonstra as 22 áreas de processo que compõem o modelo *CMMI*, presentes em cada nível de maturidade com as abreviações das áreas de processo em inglês mantendo a definição original.

Nível	**Área de Processo**	**Sigla**	**Categoria**
2	Gerenciamento de Requisitos	REQM	Engenharia
	Planejamento do Projeto	PP	Gerenciamento de Projetos
	Controle e Monitoramento do Projeto	PMC	Gerenciamento de Projetos
	Gerenciamento de Acordos com Fornecedores	SAM	Gerenciamento de Projetos
	Medição e Análise	MA	Suporte
	Garantia da Qualidade do Processo e do Produto	PPQA	Suporte
	Gerenciamento de Configuração	CM	Suporte
3	Desenvolvimento de Requisitos	RD	Engenharia
	Solução Técnica	TS	Engenharia
	Integração de Produtos	PI	Engenharia
	Verificação	VER	Engenharia
	Validação	VAL	Engenharia
	Foco no Processo Organizacional	OPF	Gerenciamento de Processos
	Definição do Processo Organizacional + IPPD	OPD	Gerenciamento de Processos
	Treinamento Organizacional	OT	Gerenciamento de Processos
	Gerenciamento Integrado de Projetos Desenvolvimento Integrado do Produto e do Processo	IPM + IPPD	Gerenciamento de Projetos
	Gerenciamento de Riscos	RSK	Gerenciamento de Projetos
	Análise de Decisões e Resoluções	DAR	Suporte
4	Desempenho do Processo Organizacional	OPP	Gerenciamento de Processos
	Gerenciamento Quantitativo do Projeto	QPM	Gerenciamento de Projetos
5	Inovação e Desenvolvimento Organizacional	OID	Gerenciamento de Processos
	Análise de Causas e Resoluções	CAR	Suporte

Tabela III - Áreas de Processo do *CMMI*

Fonte : MARÇAL, 2009, p.34

5.2 Áreas de Processo com Foco em Testes de Produtos

De acordo com Dias (2005), as áreas presentes no nível 3 de maturidade do modelo *CMMI* que estão vinculadas a testes de produto são: Validação (VAL) e Verificação (VER).

5.2.1 Validação

A principal meta que a validação visa é prover todos os recursos necessários para garantir que um produto ou um componente de produto satisfaça seu uso esperado quando inserido em seu ambiente esperado. Em resumo, de acordo com *CMMI Team* (2006) a validação garante que o produto correto foi construído. A tabela IV demonstra as práticas e sub-práticas presentes na área de processo de Validação.

Metas Específicas / Genéricas	Práticas Específicas / Genéricas	Sub - Práticas
SG 1 Preparar para a validação	SP 1.1 Selecionar os produtos para validação	1.Identificar os princípios chave, características e fases para a validação dos produtos ou componentes por meio da vida do projeto.
		2. Determinar quais categoria de necessidades do usuário (operacionais, de manutenção, de treinamento ou suporte) deverão ser validadas.
		3. Selecionar os produtos e componentes que deverão ser validados.
		4. Selecionar os métodos de validação para os produtos e componentes.
		5. Revisar a seleção de validações, restrições e métodos relevantes com os stakeholders
	SP 1.2 Estabelecer o ambiente de validação	1. Identificar os requisitos do ambiente de validação.
		2. Identificar os produtos fornecidos pelos clientes.
		3. Identificar ops itens de reuso.
		4. Identificar os equipamentos e ferramentas de teste.
		5. Identificar os recursos de validação que estarão disponíveis para reuso e modificação.
		6. Planejar a disponibilidade de recursos detalhadamente
	SP 1.3 Estabelecer os procedimentos e critérios de validação.	1. Revisar os requisitos do produto para garantir que as questões que afetam a validação do produto e seus componentes foram identificadas e resolvidas.
		2. Documentar o ambiente, o cenário operacional, os procedimentos a entrada, a saída e os critérios para a validação do produto ou componente selecionado.
		3. Avalie o projeto durante sua maturação no contexto do ambiente de validação para identificar as questões de validação.
SG 2 Validar Produtos ou Componentes do Produto	SP 2.1 Realizar a validação	1. Relatórios de validação.
		2. Resultados de validação.
		3. Matriz de referência cruzada da validação.
		4. Log de procedimento "como executar".
		5. Demonstrações operacionais.
	SP 2.2 Analisar os resultados da validação	1. Comparar resultados obtidos com os esperados.
		2. Baseando-se nos critérios de validação estabelecidos, identificar os produtos e componentes que não tiveram a performance desejada no seu ambiente de operação, ou identificar problemas nos métodos, critérios e ou ambiente.
		3. Analisar os dados de validação sobre os defeitos.
		4. Registrar os resultados das analises e identificar os casos a serem tratados.
		5. Usar os resultados da validação para comparar as medições feitas e performance obtida com o uso pretendido ou necessidades operacionais.

Tabela IV - Metas, práticas e sub-práticas da área de processo Validação

Fonte : DIAS,2005,p.66

5.2.2 Verificação

O processo de verificação tem como principal objetivo fornecer os recursos para garantir que os produtos selecionados estejam de acordo com os requisitos especificados, é incremental pois ocorre durante a fase de desenvolvimento do produto, iniciando com a verificação de requisitos, em seguida pela verificação em desenvolvimento e finalmente com a verificação dos produtos finalizados.

Segundo definição do *CMMI Team* (2006) a verificação valida se o produto foi construído de forma correta. As práticas, metas e sub-práticas da área de processo Verificação são apresentadas na tabela V.

Metas Específicas / Genéricas	**Praticas Específicas / Genéricas**	**Sub - Práticas**
SG 1 Preparar para a Verificação	SP 1.1 Selecionar os produtos para verificação.	1. Identificar os produtos para verificação
		2. Identificar os requisitos a serem satisfeitos por cada produto.
		3. Identificar os métodos de verificação que estão disponíveis para uso.
		4. Definir os métodos de verificação que serão usados para cada produto.
		5. Mandar os produtos a serem verificados, os requisitos a serem satisfeitos e os métodos a serem usados para integração com o plano de projeto.
	SP 1.2 Estabelecer o ambiente de verificação	1. Identificar os requisitos do ambiente de verificação.
		2. Identificar os recursos de verificação disponíveis para reuso e modificação.
		3. Identificar os equipamento e ferramentas de verificação.
		4. Adquirir equipamento de suporte e ambiente de verificação, como equipamentos de teste e softwares.
	SP 1.3 Estabelecer os procedimentos e critérios de verificação	1. Gerar um conjunto de procedimentos de verificação compreensível e integrado para os produtos e para qualquer produto comercial, conforme a necessidade.
		2. Desenvolver e refinar os critérios de verificação quando necessário.
		3. Identificar os resultados esperados, quaisquer tolerâncias permitidas na observação e outros critérios para cumprir os requisitos.
		4. Identificar qualquer equipamento e componentes do ambiente necessários para a verificação.
SG 2 Revisão pelos participantes	SP 2.1 Preparar revisão pelos participantes	1. Determinar o tipo de revisão que será conduzida.
		2. Definir os requisitos para a coleta de dados durante a revisão.
		3. Estabelecer e manter critérios de entrada e saída para a revisão.
		4. Estabelecer e manter critérios para a requisição de uma nova revisão.
		5. Estabelecer e manter checklists para garantir que os produtos sejam revisados constantemente.
		6. Desenvolver uma agenda de revisão detalhada, incluindo as datas para o treinamento de como fazer revisão e de quando o material para revisão estará disponível.

		7. Certificar-se de que o produto cumpre os critérios de entrada para revisão antes da distribuição.
		8. Distribuir o produto para revisão e as informações relacionadas entre os participantes com uma antecedência suficiente para que eles se preparem adequadamente para a revisão .
		9. Designar papéis para a revisão apropriadamente.
		10. Preparar-se para a revisão dos participantes fazendo uma revisão do produto antes de conduzir a revisão dos participantes.
	SP 2.2 Conduzir a revisão pelos participantes	1. Desempenhar os papéis da revisão.
		2. Identificar quaisquer defeitos nos documentos e outros aspectos do produto.
		3. Guardar os resultados da revisão, incluindo os itens de ação.
		4. Coletar os dados da revisão pelos participantes.
		5. Identificar os itens de ação e levar a questão aos stakeholders relevantes.
		6. Conduzir quaisquer revisões necessárias se os critérios definidos indicarem a necessidade
		7. Certificar-se de que os critérios de saída da revisão foram satisfeitos.
	SP 2.3 Analisar os dados da revisão pelos participantes	1. Guardar os dados relacionados a preparação, condução e resultados das revisões.
		2. Guardar os dados para futura referência e análise.
		3. Proteger os dados para garantir que não sejam usados de inapropriadamente.
		4. Analisar os dados da revisão.
SG 3 Verificar os produtos selecionados	SP 3.1 Realizar a verificação	1. Verificar se os produtos estão de acordo com seus requisitos.
		2. Guardar os dados das atividades de verificação.
		3. Identificar os itens de ação resultantes da verificação dos produtos.
		4. Documentar o método de verificação "as-run" e os desvios dos métodos e procedimentos disponíveis descobertos durante a sua realização.
	SP 3.2 Analisar os resultados da verificação e identificar as ações de correção	1. Comparar os resultados reais aos resultados esperados.
		2. Baseando-se nos critérios de verificação estabelecidos,identificar os produtos que não cumprem seus requisitos ou identificar os problemas nos métodos, procedimentos, critérios e ambiente de verificação
		3. Analisar os dados da verificação sobre os defeitos.
		4. Guardar todos os resultados da análise num relatório.
		5. Use os resultados da verificação para comparar a performance real com os parâmetros técnicos de performance.
		6. Prover informação de como os defeitos deverão ser resolvidos (incluindo os métodos de verificação, critérios e ambiente de verificação) e formalize num plano.

Tabela V - Metas, práticas e sub-práticas da área de processo Validação

Fonte : DIAS, 2005, p.63

6.Resultados Decorrentes da Implantação

Os requisitos cumpridos para a aderência à algumas das práticas e metas das áreas de processo Validação e Verificação do modelo *CMMI* são os seguintes:

1. **Validação - VAL**
   - **1.1. SP 1.1 Selecionar Produtos para Validação:** Essa prática envolve a seleção dos produtos que farão parte processo de validação, bem como a definição dos métodos de validação que serão utilizados. Atualmente, é durante a *Sprint Review* em que é selecionada a versão liberada do sistema pelos desenvolvedores para ser demonstrada ao *Product Owner* e o método utilizado é a apresentação dos recursos liberados em um banco de dados de testes.

   - **1.2. SP 1.2 Estabelecer Ambiente de Validação:** Um ambiente deve ser criado para que os produtos sejam validados. O ambiente selecionado para a realização dos testes é um banco de dados criado especificamente com essa finalidade que é atualizado sempre a cada final de *Sprint* para que fique disponível para a realização da *Sprint Review*. Com a implantação da proposta será incluso no ambiente de teste atual, a utilização do *framework* de testes *NUnit* que será responsável em auxiliar na execução dos testes unitários escritos.

   - **1.3. SP 1.3 Estabelecer Procedimentos e Critérios de Validação:** O objetivo dessa meta é garantir que o produto tenha seu uso esperado quando colocado no ambiente esperado.Sua aderência é atingida através dos testes de aceitação, que são especificados nas *User Stories*[2] e são conferidos na *Sprint Review* pelo *Product Owner* para que a atividade receba o aceite ou não.

2. **Verificação - VER**
   - **2.1. SP 1.3 Estabelecer Procedimentos e Critérios de Verificação:** Com o objetivo de garantir que os produtos atendam os requisitos solicitados, como exemplo de critérios de validação a meta exige a existência de padrões e tipos de teste. A padronização é atingida com a utilização das principais técnicas de Código Limpo apresentadas, garantindo a legibilidade e qualidade do código produzido e quanto aos testes se aplica o TDD trazendo uma cobertura maior ao código fonte com a criação de testes unitários ao ser utilizado.

   - **2.2. SP 3.1 Realizar Verificação:** Segundo o manual de implantação do *CMMI* SEI (2006) a verificação de produtos de forma incremental que é realizada para as áreas Val e Ver as quais caminham juntas, faz com que seja possível detectar problemas de forma mais prematura assim antecipando suas soluções, economizando tempo com possíveis retrabalhos. Todas essas características estão presentes nos benefícios da aplicação do TDD, quando o desenvolvedor

[2] Artefato do *Scrum* que representa os itens do *Sprint Backlog*, segundo Kniberg (2007) são "coisas que o cliente deseja, descritas utilizando a terminologia do cliente".

começa a utilizá-lo tem uma quantidade maior de retorno sobre o código que está produzindo, dessa forma há a possibilidade de o desenvolvedor prever falhas com antecedência e deixar o teste preparado para elas não ocorram. Outro ponto importante a destacar a respeito do TDD é que o código criado surge testado estando implícito que existirá ao menos um teste unitário garantindo seu funcionamento.

2.3. GP 2.3 Fornecer Recursos: Os processos de Validação e Verificação podem precisar de recursos para serem verificados. Conforme mencionado na prática SP 1.2 Estabelecer Ambiente de Validação, a apresentação do *Sprint Review* é realizada em um banco de dados exclusivo para os testes de versão liberados separado da base de desenvolvimento, e com a aplicação da proposta será utilizado o framework de testes *NUnit* que será responsável pela execução dos testes unitários.

6.1 Considerações Finais

Atualmente o local foco da proposta em seu processo de desenvolvimento, faz uso da metodologia ágil Scrum desde o ano de 2009, e está certificada no nível 2 de maturidade do modelo *CMMI*.

O processo para certificação para o nível 2 do *CMMI* teve início no ano de 2009 em conjunto com outras empresas de desenvolvimento de *software* da cidade de Maringá - PR com o apoio do SEBRAE e da APL de Software de Maringá e em parceria com a empresa SWQuality Consultoria e Sistemas.

O presente artigo teve o propósito de apresentar uma proposta de implantação das práticas de Código Limpo, no processo de desenvolvimento atual, voltando à atenção para conceitos do Desenvolvimento Guiado a Testes (TDD), visto que o local alvo da proposta almeja certificar o nível 3 de maturidade do modelo *CMMI* ,e para que isso ocorra a parte de padronização e testes de produto devem estar consolidadas.

Diante de técnicas e conceitos apresentados, conclui se que a implantação das práticas de Código Limpo no ambiente atual foco do trabalho, se torna viável à partir do momento em que os desenvolvedores comecem colocar em uso os conceitos de Código Limpo aqui apresentados até que se atinja seu total domínio, já que não é um processo simples e de curto prazo e que se realmente implantadas em conjunto com o cenário atual auxiliarão em parte a aderência dos requisitos das áreas VAL e VER do nível 3 do *CMMI* apontadas no estudo. Um conselho que Martin (2009) sugere aos desenvolvedores e que vale a pena ser destacado e a regra do escoteiro: "Deixe a área do acampamento mais limpa do que como você a encontrou". Se todo programador tentasse promover um pouco de "limpeza" no código com certeza se reduziria a quantidade de código ruim.

Segundo Martin (2009) o primeiro passo a ser dado a caminho da "limpeza" é iniciar com pequenas alterações, como eliminar repetições aos poucos, reduzir tamanho de funções ou métodos, alterações simples que estimulem a utilização das técnicas.

Referências

Agile For All.Disponível em < http://www.agileforall.com/intro-to-agile>.Acesso em: 12 ago.2014

Almeida, Lucianna Thomaz e Miranda, João Machini (2010), Código Limpo e Seu Mapeamento Para Métricas de Código Fonte. Disponível em : <http://ccsl.ime.usp.br/pt-br/system/files/relatorio-codigo-limpo.pdf>.Acesso em : 10 abr.2014.

Aniche, Mauricio, Test Driven Development: Teste e Design no Mundo Real com .Net, Casa do Código.

Aniche, Mauricio, Test Driven Development, Disponível em : < http://tdd.caelum.com.br/>. Acesso em : 15 mai.2014.

Dias, Roni Queiroz (2005), Adoção Das Práticas de Testes do Modelo CMMI na Melhoria da Qualidade e Serviços de Testes de Software em Instituições Financeiras, 241 p, Dissertação (Mestrado em Sistemas de Gestão) - Universidade Federal Fluminense ,Rio de Janeiro.Disponível em <http://www.bdtd.ndc.uff.br/tde_arquivos/14/TDE-2008-08-14T120806Z-1587/Publico/DISSERTACAO%20RONI%20DIAS.pdf>.Acesso em : 29 jun.2014.

Kniberg, Henrik (2007) , Scrum e XP direto das Trincheiras , C4Media Inc. Disponível em <http://www.infoq.com/br/minibooks/scrum-xp-from-the-trenches>. Acesso em : 03 ago.2014

Martin, Robert C. (2009), Código Limpo : Habilidades Práticas do Agile Software, Alta Books,Edição Revisada.

Martin,Robert C. e Martin, Micah (2011) , Padrões e Práticas Ágeis em C#, Bookman, 1ª Edição.

Martin, Robert C. (2012), O Codificador Limpo : Um Código de Conduta para Programadores Profissionais.Alta Books, 2ª Reimpressão.

Marçal, Ana Sofia Cysneiros (2009) , SCRUMMI : Um processo de gestão ágil baseado no SCRUM e aderente ao CMMI , 205 p , Dissertação (Mestrado em Informática Aplicada da Universidade de Fortaleza UNIFOR), Fortaleza. Disponível em < http://www.cin.ufpe.br/~if720/downloads/SCRUMMI%20-%20AnaSofia.pdf>. Acesso em : 28 set.2014.

Pereira , Paulo ; Torreão, Paula e Marçal, Ana Sofia (2007) , Entendendo Scrum para Gerenciar Projetos de Forma Ágil , 8 p, Revista Mundo PM.

Pham,Andrew e Pham,Phuong-Van (2012) , Scrum em Ação, Novatec, 1ª Reimpressão.

Schwaber, Ken e Sutherland, Jeff (2011) , Guia do Scrum , Disponível em : <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em : 20 mai.2014.

SEI - Software Engineering Institute (2006) , CMMI para Desenvolvimento - Versão 1.2. Disponível em : < http://www.sei.cmu.edu/library/assets/whitepapers/cmmi-dev_1-2_portuguese.pdf>.Acesso em : 05 out.2014.

Vasconcelos, Ivo M. Michalick ; Magalhães, Ana Liddy C. de Castro e Rouiller, Ana Cristina (2005) , CMMI e Gerenciamento de Projetos de Software, 6 p, Revista Mundo PM.