**UNIVERSIDADE PRESBITERIANA MACKENZIE**

RICARDO BERNARDELI COSTA DA SILVA

Titulo a ser melhor definido

São Paulo

2011

RICARDO BERNARDELI COSTA DA SILVA

**Titulo a ser melhor definido**

Trabalho Graduação Interdisciplinar apresentado à Faculdade de Computação e Informática da Universidade Presbiteriana Mackenzie do Estado de São Paulo como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação

Orientadora: Profa. Ms. Ana Cláudia Rossi

São Paulo

2011

RICARDO BERNARDELI COSTA DA SILVA

Titulo a ser melhor definido

Trabalho Graduação Interdisciplinar apresentado à Faculdade de Computação e Informática da Universidade Presbiteriana Mackenzie do Estado de São Paulo como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação

Aprovado em

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profª. Ms. Ana Cláudia Rossi – Orientadora

Universidade Presbiteriana Mackenzie

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Examinador 1

Universidade Presbiteriana Mackenzie

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Examinador 2

Universidade Presbiteriana Mackenzie

Dedicado à João Odair Costa da Silva e Benedita Rosa Bernardeli Costa da Silva

**RESUMO**

Resumo completo do TGI ao finalizar.

**ABSTRACT**

This is a typical abstract. Will be done asap.

**LISTA DE ILUSTRAÇÔES**

**SUMÁRIO**

[1. Introdução 12](#_Toc247045439)

[1.1. Justificativa 12](#_Toc247045440)

[1.2. Objetivo 13](#_Toc247045441)

[1.3. Estrutura do Trabalho 14](#_Toc247045442)

[2. Fundamentação Teórica 14](#_Toc247045443)

[2.1. técnicas de desenvolvimento de testes 16](#_Toc247045445)

[2.1.1. Test Driven Development - TDD 16](#_Toc247045446)

[2.1.2. Behaviour Driven Development - BDD 18](#_Toc247045447)

[2.1.3. Integration Test 20](#_Toc247045448)

[2.2. Estratégias de desenvolvimento de software 16](#_Toc247045445)

[2.2.1. Abordagem tradicional – Watterfall 16](#_Toc247045446)

[2.2.2. Iterativo e incremental 17](#_Toc247045447)

[2.2.3. Metodologias Ágil 20](#_Toc247045448)

[3. A DEFINIR 26](#_Toc247045452)

4. A DEFINIR 26

[5. Considerações Finais 60](#_Toc247045481)

[6. Referências Bibliográficas 61](#_Toc247045482)

# Introdução

Testes aliado com o paradigma ágil têm se tornado cada vez mais imprescindíveis no atual mercado de desenvolvimento de software. Além de demonstrar ser uma maneira de diminuirmos a possível quantidade de problemas que o software pode ter, ainda prevalece a principal característica dos testes, sendo sem dúvida aumentar a qualidade do código, diminuindo o acoplamento e obtendo assim um software de fácil manutenção e extensão. Devido a sua existência, uma equipe se sente tranqüila para efetuar refatorações, manutenções ou até mesmo fazer a adição de novas funcionalidades (Bertolino, 2007).

Testes não substituem princípios e decisões sobre o design ou então seus diagramas UML, o foco é validar e melhorar as decisões de design, visto que é uma técnica que ajuda na criação e evolução do mesmo.

## Justificativa

Segundo Beck (2003), o crescimento do uso de técnicas de desenvolvimento ágil trouxe consigo grandes avanços na velocidade e qualidade do desenvolvimento de software. No entanto a grande variedade de escolhas buscando a melhor implementação em uma tarefa na hora de realizá-la também ocasionou confusões, maus práticas e desentendimento. Ainda, com a crescente do trabalho colaborativo a distância, o uso de tais práticas ágeis trouxe, em alguns casos, mais problemas do que os métodos antigos.

Obviamente este não é o intuito e nem a realidade das aplicações coerentes de métodos ágeis. Exemplos de sucesso e fracassos são vistos diariamente e busca por melhorias são sempre bem vistas. Assim sendo, há uma necessidade de esclarecer enganos e estabelecer diretrizes para o desenvolvimento ágil, focado na aplicação desenvolvida em cima dos próprios testes. No entanto, no que diz respeito a situações menos convencionais, com equipes colaborando a distância, com novas ferramentas sendo inclusas no projeto a toda hora, com especificações de qualidade distintas, não existe uma extensa bibliografia.

## Objetivo

As metodologias ágeis sofreram com o seu próprio sucesso. A crescente busca por melhor eficiência e qualidade trouxe muita atenção à estas. Mas no caminho para o Graal sagrado do desenvolvimento de software ficaram aqueles que falharam em aplicar tais idéias coerentemente. Nem sempre esta falha é responsabilidade total do gerente da equipe. Problemas contratuais podem causar a relutância em quebrar paradigmas ou as vezes impossibilitar a troca de ferramentas. Ainda existe a dificuldade em demonstrar o valor adicional que a adoção de metodologias ágeis trará para a empresa.

Com tais dificuldades em mente uma pergunta se faz evidente: é possível evitar os acidentes de percurso na implementação de metodologias ágeis para situações específicas? Levando em consideração as particularidades de cada projeto tal tarefa parece difícil.

Felizmente existem muitas maneiras de se conduzir um projeto ágil, cada qual com suas vantagens e desvantagens para diferentes projetos, vendo a flexibilidade agregada às metodologias ágeis. Este trabalho tem a intenção de responder à pergunta supracitada criando um workflow baseado em metodologias ágeis, com ênfase no Scrum, e utilizando técnicas de garantia de qualidade e design não obstrusivo, o Test Driven Development e o Behavior Driven Development (BDD).

## Estrutura do Trabalho

Este trabalho consiste em 3 grandes capítulos principais. O primeiro deles, referente a fundamentação teórica, contextualiza o panorama atual sobre testes na engenharia de software no Brasil e no mundo.

Já no segundo capítulo, será fornecido uma base sobre diversas metodologias disponíveis no mercado atual, tendo uma grande ênfase ao paradigma ágil.

O capitulo seguinte descreve uma proposta de workflow voltado à implementação de testes nas metodologias ágeis afim de ter um melhor design da aplicação, envolvendo com testes unitários, funcionais e integração, provando a eficácia de seus pontos positivos e justificando o porquê o teste deve ser encarado como parte do desenvolvimento como um todo.

# Metodologia de pesquisa

A metodologia de pesquisa a ser utilizada para o desenvolvimento deste trabalho compõe as seguintes fases:

* Elaborar o referencial teórico: será realizada uma pesquisa bibliográfica para estabelecer a fundamentação teórica sobre os tópicos que compõe este trabalho: metodologias ágeis, desenvolvimento orientado a testes, tipos de testes e outros.
* Refinar o objetivo da pesquisa: definir e estabelecer com maior clareza o escopo do trabalho a ser desenvolvido, uma vez que, através da revisão bibliográfica, o objetivo da pesquisa estará endereçando os problemas em aberto na área de pesquisa.
* Definir o *framework* orientado a testes: será proposto como resultado deste trabalho um *framework* baseado no desenvolvimento orientado a testes com o cruzamento das metodologias ágeis e um *workflow,* onde viabiliza a velocidade e a entrega com valor ao cliente.
* Elaborar a conclusão: elaborar a conclusão da tese a partir da análise dos resultados obtidos com a proposta elaborada no passo anterior.

# Fundamentação teórica

## 3.1 Panorama atual no desenvolvimento de softwares

O desenvolvimento de software vem passando por constantes mudanças. Em 1979, estatísticas foram apresentadas demonstrando que 50% do tempo de um projeto eram gastos em seu desenvolvimento e os outros 50% eram gastos em testes.

Hoje, mais de um quarto de século depois, as estatísticas ainda são verdadeiras mesmo com novas ferramentas de apoio e novos sistemas de desenvolvimento (MYERS, 2004).

## 3.2 Test Driven Development (TDD)

*Test Driven Development* (TDD) foi criado afim de encorajar o design de código simples e insparar confiança no momento de uma possível refatoração, em padrões aceitáveis. Seu principal objetivo é escrever código limpo e funcional. Existem algumas razões para isso (BECK, 2003):

* É um método previsível de desenvolver. O desenvolvedor sabe quando termina, sem ter que preocupar-se por uma trilha de bugs
* Melhora a qualidade do software
* Elimina duplicações

Segundo Deivid Chalinsky (2008) em The RSpec Book, *Test Driven Development* é uma prática que envolve escrever testes antes do código que está sendo testado.

*Test Driven Development* é baseada em um ciclo curto de repetições. Em uma visão ampla, primeiramente o desenvolvedor escreve um caso de teste automatizado que define uma nova funcionalidade ou uma melhoria desejada. A partir deste teste, o código é produzido e o resultado é validado pelo mesmo.

O teste contém asserções, que podem ser verdadeiras ou falsas, confirmando o comportamento correto. Isso permite que o desenvolvedor evolua prosseguindo e garante a confiabilidade em uma refatoração, uma vez que com a suíte de testes feita, garante a não mudança do comportamento esperado.

As repetições consistem no seguinte fluxo (BECK, 2003):

* Adicionar um teste
* Executar todos os testes aguardando que falhe o teste adicionado
* Escrever o código para que o teste passe
* Executar novamente os testes esperando sucesso em todos
* Refatorar o código

## 3.2.1.Adicionar um teste

Cada nova funcionalidade inicia com a criação de um teste, o qual precisa inevitavelmente falhar, já que é escrito antes da funcionalidade a ser implementada. Para escrever um teste, o desenvolvedor precisa claramente entender as especificações e requisitos da funcionalidade, o que pode ocorrer através de casos de uso ou *user stories* que cubram os requisitos e exceções condicionais.

## 3.2.2.Executar todos os testes e ver se algum falha

Esse passo validará se todos os testes estão funcionando corretamente e se o novo teste não traz nenhum equivoco, sem que seja requerido nenhum código novo. O novo teste deve então falhar pela razão esperada: a funcionalidade não foi implementada. Isto aumenta a confiança que será testado a funcionalidade certa, e que o teste somente irá ter sucesso nos casos intencionados.

## 3.2.3.Escrever o código

Esse passo é escrever o código que irá ocasionar ao teste passar. O novo código escrito até esse ponto poderá não ser perfeito e pode passar no teste de uma forma não elegante. Isso é aceito devido à refatoração posteriormente feita. O importante nesta etapa é fazer que o código seja feito apenas o suficiente para o teste ter sucesso.

## 3.2.4.Executar novamente os testes

Se todos os testes respondem com sucesso, o desenvolvedor pode ter a confiança de que todos os requisitos estarão testados, seja a nova funcionalidade ou então as antigas, partindo do pré-suposto que foram todas previamente também testadas.

## 3.2.5.Refatoração

Neste ponto, o código pode ser limpo como necessário, seja removendo duplicações ou . Ao re-executar os testes, o desenvolvedor pode confiar que a refatoracao não é um processo danoso a qualquer funcionalidade existente.

## 3.2.6.Repetição

Após a fase de refatoração, todo o processo é repetido.

## 3.3 Behaviour Driven Development (BDD)

*Behaviour Driven Development* (BDD), assim como o *Test Driven Development*, é uma abordagem de desenvolvimento ágil afim de encorajar a colaboração entre desenvolvedores, setores de qualidade e pessoas não técnicas ou de negócios em um projeto de software (North, 2003).

*Behaviour Driven Development* tem como objetivo usar a linguagem natural com a linguagem ubíqua utilizada no projeto de software, aplicado com o foco em por quê o código deve ser criado, ao invés de detalhes técnicos, e minimiza traduções entre linguagem técnica na qual o código é escrito e outras linguagens de domínio, usuários, clientes, gerentes, entre outros.

Segundo Dan North (2003), as práticas de Behaviour Driven Development incluem:

* Envolver as partes interessadas no processo através de *Outside-in Development* (Desenvolvimento de fora para dentro)
* Usar exemplos para descrever o comportamento de uma aplicação ou unidades de código
* Automatizar os exemplos para prover um *feedback* rápido e evitar os testes de regressão
* Usar a palavra “deve” na hora de descrever o comportamento de software para ajudar a esclarecer responsabilidades e permitir que funcionalidades do software sejam questionadas
* Usar dublês de teste, como *mocks, stubs, fakes, dummies* e *spies* para auxiliar na colaboração entre módulos e códigos que ainda não foram escritos

## 3.3.1.Desenvolvimento *Outside-In*

*Behaviour Driven Development* é guiado pelos valores de negócios, vendo o software como benefício trazido para o negócio no qual a aplicação será produzida. Segundo Albertino (2004), a única maneira na qual o benefício do *Outside-In* pode ser percebido na visão do cliente é através de interfaces de usuário para a aplicação. Cada elemento de código provê algum comportamento, o qual em colaboração com outros elementos provê o comportamento geral da aplicação.

# Referências Bibliográficas

MYERS, Glenford J. The Art of Software Testing. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004  
  
CHELIMSKY, David. The RSpec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends. Pragmatic Bookshelf, 2010  
  
BECK, Kent. Test Driven Development: By Example. Bonston: Addison-Wesley Professional, 2002  
  
SCHWABER, Ken, BEEDLE, Mike. Agile Project Management with Scrum (Microsoft Professional). Prentice Hall, 2001  
  
SHORE, James, WARDEN, Shane. The Art of Agile Development. O'Reilly Media, 2007

SOMMERVILE, Ian; SoftwareEngineering. Boston: Addison Wesley – 2007

KETTUNEN, Vesa el at. A study on agility and testing processes in software organizations, 2010

PULEIO, Michael. How Not to Do Agile Testing, Agile 2006 (AGILE’06), 2006

BERTOLINO, Antonia. Software Testing Research: Achievements, Challenges, Dreams. Minneapolis, 2007