

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**DESENVOLVIMENTO ÁGIL PARA
GROUPWARE NA WEB 2.0**

VINÍCIUS PEREIRA

ORIENTADOR: PROF. DR. ANTONIO FRANCISCO DO PRADO

São Carlos – SP

Junho/2012

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

**DESENVOLVIMENTO ÁGIL PARA
GROUPWARE NA WEB 2.0**

VINÍCIUS PEREIRA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software

Orientador: Prof. Dr. Antonio Francisco do Prado

São Carlos – SP

Junho/2012

**Ficha catalográfica elaborada pelo DePT da
Biblioteca Comunitária da UFSCar**

P436da Pereira, Vinícius.
Desenvolvimento ágil para groupware na Web 2.0 /
Vinícius Pereira. -- São Carlos : UFSCar, 2012.
119 f.

Dissertação (Mestrado) -- Universidade Federal de São
Carlos, 2012.

1. Engenharia de software. 2. Sites da web -
desenvolvimento. 3. Validação, verificação e teste. 4.
Software - testes. I. Título.

CDD: 005.1 (20^a)

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

**“Desenvolvimento Ágil para
Groupware na Web 2.0”**

Vinícius Pereira

Dissertação de Mestrado apresentada ao
Programa de Pós-Graduação em Ciência da
Computação da Universidade Federal de São
Carlos, como parte dos requisitos para a
obtenção do título de Mestre em Ciência da
Computação

Membros da Banca:



Prof. Dr. Antonio Francisco do Prado
(Orientador - DC/UFSCar)



Prof. Dr. Marcio Eduardo Delamaro
(ICMC/USP)



Profa. Dra. Silvia Regina Vergilio
(INF/UFPR)

São Carlos
Junho/2012

A minha namorada e a minha família.

AGRADECIMENTOS

A minha namorada pelo apoio incondicional durante todas as fases do mestrado.

Aos meus pais pelo amparo nas horas de dificuldade.

A minha família pelo incentivo nessa etapa da minha vida.

Aos meus colegas do laboratório pela distração.

Ao meu orientador, Prof. Dr. Antonio Francisco do Prado, pela oportunidade e pelo apoio durante todo o mestrado.

E ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo auxílio financeiro que possibilitou a realização deste trabalho.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário”.

Albert Einstein

RESUMO

Este projeto pesquisou um Processo de Desenvolvimento Ágil para *groupware* na Web 2.0. O Processo tem suas bases em diferentes tecnologias, destacando-se as do método ágil Scrum, e é integrado com testes funcionais a nível de aceitação. Para auxiliar essa integração, estabeleceu-se uma maior intercomunicação entre as disciplinas do processo, utilizando as Histórias do Usuário como artefatos em todas as disciplinas. O processo visa à tirar proveito da agilidade do Scrum, sem perder qualidade no desenvolvimento das aplicações. Foi desenvolvido tendo como base o processo Scrum, adicionado de tecnologias que apoiam a criação e utilização de artefatos na busca para integração com testes funcionais. O processo utiliza três disciplinas (Comunicação, Modelagem e Construção), cada uma refinando as Histórias do Usuário, que ocorrem em diferentes graus de abstração ao longo do desenvolvimento. Essas histórias devem ser um retrato mais fiel possível das necessidades do usuário e são utilizadas tanto no formato de cartões (Cartões de História), para a especificação de requisitos, quanto em um formato executável para orientar a implementação e trabalhar como testes funcionais. *Mock-ups* são utilizados para complementar os requisitos especificados nos cartões, auxiliando na tarefa de representar os requisitos. Esse conjunto é a base para a criação das Histórias do Usuário no formato executável. Outros artefatos de modelagem, como o Diagrama de Classes, também podem ser especificados, para elicitare os requisitos da aplicação. O Processo é apoiado por ferramentas desenvolvidas para possibilitar a integração de testes funcionais. Essas ferramentas definem uma linguagem para especificação de testes para escrever as Histórias do Usuário, criada como uma DLL para o Visual Studio, e um conversor dessas Histórias do Usuário para testes funcionais em código C#. Assim, o processo proposto estabelece uma maior integração dos testes durante o desenvolvimento e execução de cada disciplina, visando a produção de aplicações com maior qualidade. Para testar o processo, foram desenvolvidas aplicações do domínio de *groupware* na Web 2.0. A adoção desse domínio foi motivada pela ideia de explorar o desenvolvimento de software colaborativo que apoia o trabalho em grupo de pessoas envolvidas em tarefas comuns.

Palavras-chave: Processo Ágil, Desenvolvimento Web, Web 2.0, *Groupware*, História do Usuário, Desenvolvimento Orientado a Testes, Testes Funcionais.

ABSTRACT

This project investigated an Agile Development Process for groupware on the Web 2.0. The process has its based in different technologies, especially those of the Scrum agile method, integrated with functional testing at the acceptance level. To assist this integration, we established a greater intercommunication between the process disciplines, using the User Stories as artifacts in all disciplines. The process aims to take advantage of the agility of the Scrum without losing quality in the development of applications. It was developed based on the Scrum process, with the addition of technologies that support the creation and use of artifacts in the search for integration with functional testing. The process uses three disciplines, each one refining the User Stories, which occur in different degrees of abstraction throughout the development. These stories should be a most faithful portrait possible of user needs and are used both in the form of cards (Story Cards), for requirements specification, and in an executable format to guide the implementation and work as functional testing. Mockups are used to complement the requirements specified in the cards, assisting in the task of representing the requirements. This set is the basis for the creation of User Stories in executable format. Other modeling artifacts, like Class Diagram, can also be specified in order to elicit requirements of the application. The procedure is supported by tools developed to enable the integration of the functional tests. These tools define a tests specification language to write User Stories, created as a DLL for Visual Studio, and a converter of these User Stories for functional testing in C# code. Thus, the proposed process provides a greater integration of tests during the development and execution of each discipline in order to produce higher quality applications. To test the process, applications were developed in the domain of groupware in Web 2.0. The adoption of this area was motivated by the idea of exploring the development of collaborative software that supports group work of people involved in common tasks.

Keywords: Agile Process, Web Developmnet, Web 2.0, Groupware, User Story, Test Driven Development, Functional Testing.

LISTA DE FIGURAS

2.1	Web tradicional X Web 2.0.	21
2.2	Princípios da Web 2.0 (adaptado de (GASPAR; YAGUINUMA; PRADO, 2009)). . .	21
2.3	Groupware auxiliando na interação entre pessoas em locais diferentes.	23
2.4	Abordagem para utilização das Histórias do Usuário em cartões, retirado de (COHN, 2004).	26
2.5	Visão geral do TDD, adaptado de (BECK, 2003).	27
2.6	O método Scrum, adaptado de (SCHWABER, 2004).	29
2.7	As principais práticas do XP, adaptado de (JEFFRIES; ANDERSON; HENDRICKSON, 2000).	31
2.8	Os cinco processos do FDD.	32
2.9	Ciclo de vida do AUP, retirado de (SLIGER; BRODERICK, 2008).	33
3.1	Documento FIT com sucessos e falhas, mostrando o que era esperado e qual o valor obtido, retirado de (MUGRIDGE; CUNNINGHAM, 2005).	36
3.2	A abordagem proposta por (TALBY et al., 2005).	37
3.3	Resultado do teste apresentado dentro da tabela, na última coluna. Retirado de (TALBY et al., 2005).	37
3.4	Processo colaborativo e a plataforma CFT, retirado de (YU et al., 2009).	38
3.5	Processo AWDWF, retirado de (RAN et al., 2008).	39
3.6	Processo WebML, retirado de (CERI; FRATERNALI; BONGIO, 2000).	40
3.7	Exemplo da notação do UWE, retirado de (KOCH et al., 2008).	42
4.1	Diferença entre não usar (a) e usar (b) o conceito de Interface Fluente.	45

4.2	Diferença entre utilizar os códigos da Figura 4.1.	46
4.3	O processo de geração do código C#.	47
4.4	Trecho do código escrito em DOT.	47
4.5	Trecho do <i>template</i> T4 (a) e o código C# gerado por ele (b).	48
4.6	Exemplo de uma História do Usuário escrita na LET.	49
4.7	Descrições na LET convertidas para Teste Funcional no C#.	50
4.8	Opções do Story2CSharp Converter.	51
4.9	SADT das etapas para utilização das ferramentas.	52
4.10	Diferenças entre as descrições das Histórias do Usuário.	53
4.11	História do Usuário, da Figura 4.10 (b), sendo convertida.	54
4.12	História convertida com o maior nível de detalhamento possível.	54
4.13	Resultado inconclusivo ao executar a História, da Figura 4.12, recém convertida.	55
4.14	Resultado apresentando uma falha.	56
4.15	Resultado mostrando que a História da Figura 4.12 foi aprovada.	56
4.16	Utilizando a “Abordagem para Integração de Testes Funcionais” com o processo Scrum.	57
4.17	Uma Expressão Regular no Cucumber.	58
5.1	Ciclo de vida do processo RAMBUS.	60
5.2	Processo RAMBUS e suas disciplinas.	62
5.3	Cartão de História utilizado para a rede social MySocial.	64
5.4	Mock-ups do Cartão de História da Figura 5.3.	65
5.5	Exemplo de um Diagrama de Tipos.	66
5.6	História do Usuário escrita na LET.	67
5.7	Arquitetura MVC usando C# e ASP.NET MVC 3.	69
5.8	História do Usuário, da Figura 5.6, em C# como teste funcional.	70
5.9	Resultado mostrando que o teste funcional foi inconclusivo, devido aos métodos pendentes.	71

5.10	Resultado mostrando que o teste falhou.	72
5.11	Resultado mostrando que o teste foi aprovado.	72
5.12	Páginas criadas seguindo os testes funcionais das Histórias do Usuário da iteração e baseadas nos Mock-ups da Figura 5.4.	73
5.13	Exemplo de uma ferramenta gerenciando as atividades atuais, em espera e as possíveis atividades futuras de um projeto.	77
6.1	Uma tela da aplicação groupware do experimento.	83
6.2	Nível de experiência dos participantes.	86
6.3	Artefatos da fase de Requisitos entregues aos participantes.	88
6.4	O Product Backlog entregue aos participantes.	88
6.5	O Diagrama de Classes entregue aos participantes.	89
6.6	Satisfação dos participantes com os processos analisados.	91
6.7	Opinião dos alunos sobre os processos em relação ao itens da Tabela 6.6.	92
6.8	Expressão Regular escrita manualmente para utilizar o Cucumber.	98
A.1	Grafo direcionado gerado utilizando DOT.	113

LISTA DE TABELAS

5.1	Diferenças entre modelos de Cartões de História, adaptado de (PATEL; RAMACHANDRAN, 2009).	63
6.1	Diferenças entre Cucumber e Story2CSharp.	80
6.2	Grupos e seus respectivos participantes para o experimento.	86
6.3	Divisão dos processos pelos grupos.	86
6.4	Tempo gasto por cada grupo em cada atividade.	90
6.5	Tempo médio gasto em cada atividade.	90
6.6	Dados comparativos e suas respectivas siglas.	92
6.7	Grupos e seus respectivos participantes para avaliar as ferramentas.	96
6.8	Comparação do tempo de criação das estruturas dos testes e as linhas de código entre o Cucumber e as ferramentas Story2CSharp.	97

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO	16
1.1 Contexto	16
1.2 Motivação e Objetivos	17
1.3 Metodologia de Desenvolvimento do Trabalho	18
1.4 Organização do Trabalho	18
CAPÍTULO 2 – PRINCIPAIS CONCEITOS E TÉCNICAS ANALISADOS	20
2.1 Groupware e a Web 2.0	20
2.2 Conceitos e Técnicas Utilizados nesta Pesquisa	24
2.2.1 Engenharia Web (WebE)	24
2.2.2 História do Usuário	25
2.2.3 Desenvolvimento Orientado a Testes (TDD)	26
2.2.4 Desenvolvimento Orientado a Comportamento (BDD)	27
2.3 Métodos Ágeis	28
2.3.1 Scrum	29
2.3.2 eXtreme Programming (XP)	30
2.3.3 Feature Driven Development (FDD)	31
2.3.4 Agile Unified Process (AUP)	32
2.3.5 Microsoft Solutions Framework (MSF)	33
2.4 Considerações Finais	34

CAPÍTULO 3 – TRABALHOS CORRELATOS	35
3.1 Trabalhos Correlatos às Ferramentas Desenvolvidas Para o Processo	35
3.1.1 Framework for Integrated Tests (FIT)	35
3.1.2 A Process-Complete Automatic Acceptance Testing Framework	37
3.1.3 Call for Testing	37
3.2 Trabalhos Correlatos ao Processo RAMBUS	38
3.2.1 Agile Web Development with Web Framework (AWDWF)	38
3.2.2 A Theoretical Agile Process Framework for Web Applications Deve- lopment in Small Software Firms	39
3.2.3 Web Modeling Language (WebML)	40
3.2.4 Uml-based Web Engineering (UWE)	41
3.3 Considerações Finais	42
CAPÍTULO 4 – FERRAMENTAS DESENVOLVIDAS	44
4.1 Linguagem de Especificação de Testes (LET)	44
4.2 Conversor de LET para a Linguagem C#	49
4.3 Utilização das Ferramentas	51
4.3.1 Especificação de Testes	52
4.3.2 Construção de Testes Funcionais	53
4.3.3 Implementação Dirigida a Testes	55
4.4 Considerações Finais	56
CAPÍTULO 5 – DESENVOLVIMENTO ÁGIL PARA GROUPWARE NA WEB 2.0	59
5.1 Visão Geral do Processo	60
5.2 Disciplinas do Processo	62
5.2.1 Comunicação	63
5.2.2 Modelagem	66
5.2.3 Construção	69

5.3	Fases do Processo	74
5.3.1	Requisitos	74
5.3.2	Planejamento	75
5.3.3	Execução	76
5.3.4	Encerramento	77
5.4	Considerações Finais	78
CAPÍTULO 6 – AVALIAÇÃO		79
6.1	Atividades Preliminares	80
6.2	Experimentação do processo RAMBUS	81
6.2.1	Experimentação em Engenharia de Software	81
6.2.2	Desenvolvimento da Experimentação	82
6.2.2.1	Definição do Experimento	83
6.2.2.2	Planejamento do Experimento	84
6.2.2.3	Execução do Experimento	87
6.2.2.4	Análise e Interpretação dos Resultados	90
6.2.2.5	Avaliação Qualitativa	93
6.2.2.6	Ameaças à validade	93
6.3	Avaliação das ferramentas desenvolvidas	96
6.4	Considerações Finais	98
CAPÍTULO 7 – CONCLUSÃO		100
7.1	Contribuições e Limitações	100
7.2	Trabalhos Futuros	101
REFERÊNCIAS		104
GLOSSÁRIO		109

APÊNDICE A – CÓDIGO DOT DA LET	111
APÊNDICE B – TRECHO DO CÓDIGO DO <i>TEMPLATE</i> T4	114
APÊNDICE C – TRECHO DO CÓDIGO C# GERADO	118

Capítulo 1

INTRODUÇÃO

Este capítulo introduz o contexto e os desafios que deram origem ao desenvolvimento deste trabalho de Mestrado. A Seção 1.1 apresenta o contexto no qual o projeto se insere. A Seção 1.2 apresenta a motivação e a relevância deste trabalho, assim como os principais objetivos almejados. A metodologia empregada encontra-se na Seção 1.3. A Seção 1.4 finaliza o capítulo descrevendo a organização desta dissertação.

1.1 Contexto

Nos últimos anos, mudanças ocorridas nas interações entre usuários e empresas com a Web, fizeram com que o fornecimento de serviços e as interações entre usuários se tornassem mais dinâmicas e acessíveis, levando ao surgimento do termo Web 2.0 (NORRIE, 2008). Essa denominação faz referência a uma nova geração de aplicações Web projetada especialmente para apoiar o compartilhamento do conteúdo gerado pelos usuários. A melhoria do processo de desenvolvimento de software tem motivado inúmeras pesquisas. Por estar contextualizado na Web 2.0, o desenvolvimento de aplicação groupware com melhor qualidade e produtividade faz parte do escopo deste trabalho.

Ultimamente, os métodos ágeis começaram a expandir sua adoção no mercado de desenvolvimento de software. As empresas têm buscado formas menos onerosas de desenvolver suas aplicações, além de melhorar seu ambiente de trabalho. O advento da metodologia ágil tem contribuído para viabilizar essas melhorias. Dessa forma, hoje em dia é comum encontrar empresas que utilizam os principais métodos ágeis: Scrum (SCHWABER, 2004) e eXtreme Programming (XP) (BECK; ANDRES, 2005), ou mesmo uma combinação de ambos.

No contexto da Engenharia de Software, testar a aplicação é considerada uma tarefa prio-

ritária (BERTOLINO, 2007). Isso se deve ao fato de que encontrar um defeito na aplicação quando esta se encontra em produção pode gerar um custo para corrigi-lo muito mais alto do que se o defeito fosse encontrado em etapas anteriores (PERRY, 2006).

Isso se deve ao fato de que a construção de uma aplicação não é uma tarefa trivial. Pelo contrário, pode se tornar muito complexa, dependendo do sistema a ser criado. Dessa forma, diversos problemas podem aparecer durante sua construção. Tais problemas podem acarretar na obtenção de um produto diferente daquele que se esperava (DELAMARO; MALDONADO; JINO, 2007).

Nesta dissertação é apresentado um processo ágil que proporciona facilidades no uso prático do Desenvolvimento Orientado a Testes (TDD) (BECK, 2003) no desenvolvimento de aplicações, tendo como foco aplicações do domínio de groupware. O objetivo final desse processo é construir uma aplicação groupware com maior qualidade em relação às funcionalidades desenvolvidas para atender as necessidades do usuário em comparação aos processos utilizados (Scrum e XP) no mercado atualmente oferecem. Assim, o processo criado neste trabalho visa auxiliar no desenvolvimento de aplicações ao integrar o uso de testes funcionais no ciclo de vida de um projeto.

1.2 Motivação e Objetivos

Motivado pelas ideias de elaborar um processo de desenvolvimento ágil que melhore a produtividade de software sem perder qualidade, este projeto pesquisou um conjunto de tecnologias, integrado com o processo Scrum.

O processo de desenvolvimento de software, tem como base o método Scrum de desenvolvimento ágil, conhecido na academia e no ambiente profissional. O processo proposto foi elaborado considerando o dinamismo do ambiente Web (NORRIE, 2008) e foi testado com aplicações sociais (groupware) da Web 2.0. O processo tem dois objetivos:

- apoiar o desenvolvimento ágil de software groupware; e
- integrar ao processo com a utilização de testes funcionais (no nível de aceitação) e orientar sobre a utilização de testes unitários em conjunto com os funcionais.

Dessa forma, a principal meta do processo é facilitar o desenvolvimento e a manutenção das aplicações Web do domínio groupware.

1.3 Metodologia de Desenvolvimento do Trabalho

Para alcançar tais objetivos, foram analisados conceitos e técnicas diferentes da Engenharia de Software tradicional e do desenvolvimento Web. Dentre eles, destacam-se os conceitos de: Engenharia Web (PRESSMAN; LOWE, 2009), História do Usuário (COHN, 2004), Desenvolvimento Orientado a Testes (Test Driven Development - TDD) (BECK, 2003) e Desenvolvimento Orientado a Comportamento (Behavior Driven Development - BDD) (NORTH, 2006). Além disso, uma pesquisa mais detalhada sobre os principais métodos ágeis – com ênfase em Scrum e XP – foi realizada para obter melhor conhecimento dos mesmos.

O processo proposto utiliza as Histórias do Usuário na forma de cartões em conjunto com os Mock-ups (BROWN, 2011) para especificar os requisitos. Foi pesquisada a utilização do Modelo de Navegação (CERI; FRATERNALI; BONGIO, 2000) para auxiliar na transcrição desses cartões (e dos Mock-ups) para uma versão executável a fim de guiar a implementação, evitando assim, erros de interpretação dos requisitos. O processo foi testado com aplicações Web do domínio groupware.

Ao longo do processo, todas as etapas do desenvolvimento da aplicação Web possuem uma versão das Histórias do Usuário (cartão ou executável) como um dos seus principais artefatos, possibilitando, assim, uma maior integração entre as mesmas. Essas Histórias são utilizadas como base para a criação de testes funcionais no nível de aceitação. Isso ocorre porque tais Histórias, se escritas seguindo o padrão proposto por Dan North (NORTH, 2006), possuem critérios de aceitação que simulam a forma como o usuário utilizaria a aplicação que será desenvolvida. Portanto, se os critérios de aceitação forem cumpridos, é considerado que o usuário aceitará a funcionalidade criada, ou seja, a História do Usuário foi aceita.

1.4 Organização do Trabalho

O Capítulo 2 apresenta os principais conceitos e técnicas utilizados neste trabalho, entre eles encontram-se: Web 2.0, Groupware, Engenharia Web, História do Usuário, Desenvolvimento Orientado a Testes e Desenvolvimento Dirigido por Comportamento, além de uma breve descrição sobre os métodos ágeis estudados. Também são apresentados alguns trabalhos encontrados na literatura que se relacionam ao trabalho deste Mestrado. Dando sequência, os trabalhos correlatos estão no Capítulo 3. O Capítulo 4 apresenta as ferramentas desenvolvidas para dar suporte ao processo. O Capítulo 5 apresenta o processo proposto conforme os objetivos dessa pesquisa. No Capítulo 6 apresenta-se a avaliação do processo com aplicações do domínio

groupware na Web 2.0. Por fim, no Capítulo 7 tem-se a conclusão deste trabalho. Além disso, este trabalho conta com alguns Apêndices (A, B e C), para auxiliar no entendimento de alguns códigos que são apresentados no Capítulo 4.

Capítulo 2

PRINCIPAIS CONCEITOS E TÉCNICAS ANALISADOS

Este capítulo aborda os conceitos previamente citados na Seção 1.3, a fim de proporcionar ao leitor uma melhor compreensão do processo proposto.

Na Seção 2.1 são apresentados os conceitos de groupware e de Web 2.0. Na sequência, a Seção 2.2 apresenta os conceitos e técnicas utilizados nesta pesquisa. Na Seção 2.3 tem-se breves descrições sobre os principais métodos ágeis encontrados na literatura e no mercado profissional. Por fim, na Seção 2.4 apresentam-se as considerações finais.

2.1 Groupware e a Web 2.0

Geralmente, as empresas que desenvolvem aplicações para Web 2.0 usam a Web como uma plataforma para criar sites colaborativos, baseados em comunidades, como os sites de redes sociais, blogs e wikis. Conforme ilustrado na Figura 2.1, o objetivo é fazer com que o ambiente on-line se torne mais dinâmico e que os usuários colaborem para a organização do conteúdo, diferentemente da Web tradicional (Web 1.0) onde o conteúdo é produzido apenas pelo Webmaster e disponibilizado aos usuários.

A Web 2.0 engloba um conjunto de princípios. Na Figura 2.2 são ilustrados tais princípios. Aplicações classificadas como Web 2.0 não necessariamente atendem a todos estes princípios simultaneamente, mas a um subconjunto deles.

De acordo com a Figura 2.2, os sete princípios que giram em torno das aplicações Web 2.0 são:

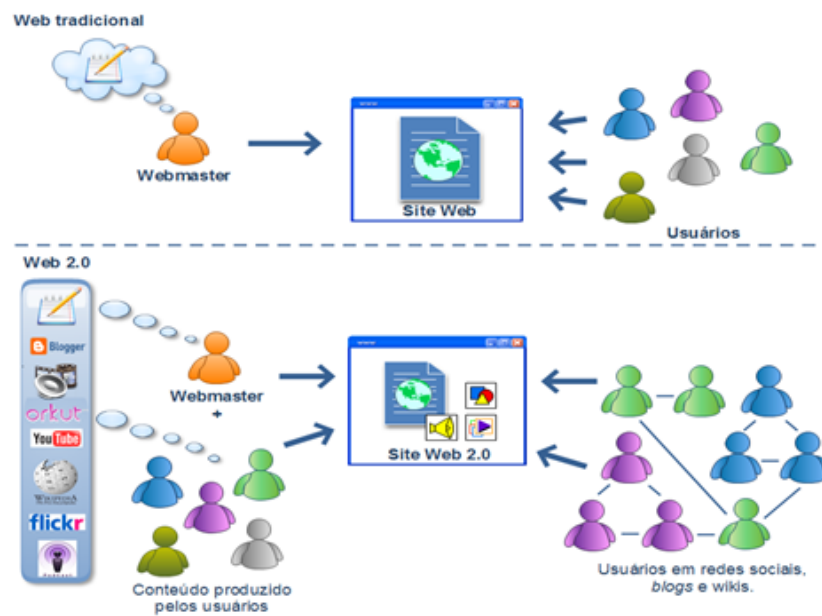


Figura 2.1: Web tradicional X Web 2.0.



Figura 2.2: Princípios da Web 2.0 (adaptado de (GASPAR; YAGUINUMA; PRADO, 2009)).

- A Web como plataforma: a Web tornou-se a plataforma de execução de diversas aplicações tidas tradicionalmente como desktops, tais como jogos, planilhas, editores de texto, agendas, clientes de email, até Sistemas de Planejamento de Recursos Corporativos (ERP - Enterprise Resource Planning);
- Uso da inteligência coletiva: mais do que apenas consumir a informação disponibilizada, o usuário agora passa a fazer parte do processo de elaboração do conteúdo. As informações produzidas pelos usuários são usadas pelas aplicações Web 2.0 para melhorar a qualidade do seu serviço à medida que essas informações são coletadas. Por

exemplo, a Wikipédia¹ fica mais completa a cada edição; redes sociais, como o Facebook², melhoram a cada novo usuário; as ferramentas de idioma do Google³ ficam mais otimizadas a cada sugestão de tradução feita pelos usuários, etc;

- Dado é o mais importante: a informação é hoje um dos ativos mais preciosos no domínio de tecnologia da informação. As informações ficam ainda mais valorizadas se combinadas aos dados coletados de uma massa de usuários;
- Betas perpétuos e desenvolvimento ágil: aplicações Web 2.0 são disponibilizadas como serviços e não em versões. O usuário não instala um software em seu dispositivo, ao invés disso ele acessa um serviço pela Internet. Com isso, as aplicações estão em contínuo desenvolvimento e novas funcionalidades podem ser disponibilizadas com muito mais frequência em questão de meses, semanas e até dias. Se os usuários não responderem às novas funcionalidades, as mesmas são retiradas e outras são adicionadas. Dessa forma, aplicações podem possuir a característica de Beta indefinidamente. Por exemplo, o Gmail⁴ da Google desde seu lançamento possui o rótulo de Beta já que está em constante aprimoramento. Com o Gmail Labs⁵, funcionalidades experimentais podem ser adicionadas e testadas pelos usuários, cujo feedback é usado pelo Google para aprimorar as funcionalidades disponibilizadas;
- Experiência rica do usuário: as interfaces gráficas das aplicações Web 2.0 vão além do Hypertext Markup Language (HTML) básico e contam agora com recursos como arrastar-e-soltar, menus deslizantes, mapas, componentes de captura e exibição de vídeos, etc. Com a evolução da tecnologia, parte do processamento pode ser feita no cliente e, com a comunicação assíncrona com o servidor introduzida com Assynchronous JavaScript and XML (AJAX) (ZAKAS; MCPEAK; FAWCETT, 2007), é possível desenvolver uma interface muito mais rica para o usuário, que oferece a sensibilidade, recursos e funcionalidades que se aproximam das aplicações desktop;
- Modelos leves de programação: o modelo de programação de aplicações Web 2.0 é voltado para a simplicidade, com uso de protocolos de comunicação simples como o Representational State Transfer (REST) (FIELDING; TAYLOR, 2000), linguagens de programação menos complexas e dinâmicas como Ruby⁶, frameworks que favorecem a produtivi-

¹<http://www.wikipedia.org/>

²<http://www.facebook.com/>

³http://www.google.com.br/language_tools/

⁴<http://www.gmail.com/>

⁵<http://www.googlelabs.com/>

⁶<http://www.ruby-lang.org/pt/>

dade como o Rails⁷, metodologias de desenvolvimento ágeis e composição de serviços (mashups). Esses serviços devem ser modulares e fracamente acoplados, de modo que um grande número de aplicações possa reutilizá-los. Por exemplo, o Google Maps⁸ possibilita inserir um mapa com marcadores dentro de qualquer aplicação Web; e

- Software em mais de um dispositivo: a Web 2.0 não se limita à plataforma Personal Computer (PC). As aplicações podem ser executadas em diversos tipos de dispositivos com diferentes sistemas operacionais. Qualquer pessoa que tenha um navegador Web disponível em seu dispositivo poderá acessar a aplicação, seja de um celular ou desktop, seja utilizando Linux ou Windows ou qualquer outra plataforma. A plataforma é a Web e não o sistema do usuário. Além disso, a Web 2.0 vai mais além, fazendo com que as aplicações não se restrinjam à arquitetura cliente-servidor, podendo possuir outras arquiteturas, como a arquitetura Peer-to-Peer (P2P) (GASPAR; YAGUINUMA; PRADO, 2009).

Groupware, também conhecido como Software Colaborativo, é uma aplicação projetada para ajudar as pessoas envolvidas em uma tarefa comum a atingir seus objetivos. É geralmente associada a indivíduos que não estão fisicamente no mesmo local, mas que trabalham juntos por meio da Internet. Pode também incluir sistemas de armazenamento de acesso remoto para arquivar dados de uso comum que podem ser acessados, modificados e recuperados pelos membros do grupo de trabalho distribuído. A Figura 2.3 mostra a interação entre pessoas que estão em locais distintos, além de utilizarem dispositivos diferentes, por meio de um groupware.



Figura 2.3: Groupware auxiliando na interação entre pessoas em locais diferentes.

⁷<http://www.rubyonrails.org/>

⁸<http://maps.google.com/>

Groupware pode ser dividido em três categorias (CARSTENSEN; SCHMIDT, 1999), dependendo do nível de colaboração: (1) ferramentas de comunicação; (2) ferramentas de conferência; e (3) ferramentas de gestão colaborativa.

Ferramentas de comunicação enviam mensagens, arquivos, dados ou documentos entre as pessoas e, conseqüentemente, facilitam a troca de informação. Os exemplos incluem: conferências síncronas e assíncronas, e-mail, wikis, publicações Web e controle de revisões.

Ferramentas de conferência também auxiliam na troca de informação, mas de uma forma mais interativa. Os exemplos incluem: fóruns na Internet, bate-papo online, mensagens instantâneas, vídeo conferências, aplicações compartilhadas e sistemas eletrônicos de reunião.

Ferramentas de gestão colaborativa facilitam e gerenciam as atividades de grupo. Os exemplos incluem: calendários eletrônicos, gerenciamento de projetos, controle de fluxo de trabalho, gerenciamento do conhecimento, aplicações sociais – todo tipo de aplicação que envolve alguma forma de interação entre as pessoas –, planilhas online e previsões do mercado de ações.

A persistência é necessária em alguns exemplos. Bate-papo e comunicações por voz são normalmente não persistentes. Por outro lado, arquivos podem ficar armazenados por anos na rede. O projetista do groupware precisa considerar as necessidades de informação e de duração para implementar corretamente.

2.2 Conceitos e Técnicas Utilizados nesta Pesquisa

São descritos nesta seção os principais conceitos e técnicas relacionados com este trabalho de mestrado. Os conceitos e técnicas têm foco no desenvolvimento ágil de aplicações Web, incluindo duas formas de lidar com a utilização de testes (unitários e funcionais) nas aplicações em desenvolvimento.

2.2.1 Engenharia Web (WebE)

A grande maioria das aplicações Web ainda é desenvolvida com uma abordagem ad hoc (PRESSMAN et al., 1998; PRESSMAN, 2000). Mesmo considerando que o desenvolvimento Web pode se beneficiar de metodologias provenientes de outras áreas, ele possui características específicas que necessitam de uma abordagem especial. Entre essas estão: tráfego de rede, paralelismo, carga imprevisível, desempenho, disponibilidade, foco nos dados, sensibilidade ao contexto, evolução contínua, urgência, segurança e estética (PRESSMAN; LOWE, 2009). A WebE abrange essas necessidades específicas das aplicações Web (GINIGE; MURUGESAN, 2001).

Apesar de utilizar princípios tradicionais da Engenharia de Software, a WebE engloba novas abordagens, metodologias, ferramentas, técnicas e orientações para atender as características citadas.

Os princípios de WebE incluem: (1) um processo diferente e único de desenvolvimento Web (KAPPEL et al., 2006); (2) multidisciplinar. É muito improvável que uma única disciplina possa oferecer uma base teórica completa, com conhecimento e práticas para guiar o desenvolvimento Web (KAPPEL et al., 2006; DESHPANDE; HANSEN, 2001); e (3) a evolução contínua e o gerenciamento do ciclo de vida de um software (ciclos mais curtos possíveis) quando comparados aos métodos tradicionais de desenvolvimento (KAPPEL et al., 2006).

2.2.2 História do Usuário

A História do Usuário descreve funcionalmente o que é solicitado e valioso para o usuário. Em uma História do Usuário há três C's que são: Cartão, Conversação e Confirmação; e segue o princípio INVEST: Independente, Negociável, Valioso para o usuário, Estimável, Pequena (Small) e Testável. O Cartão, mais conhecido como Cartão de História, é a História do Usuário escrita e formalizada. A partir do uso deste a Conversação pode ser restabelecida a fim de obter a Confirmação.

A ideia de a História do Usuário ser escrita em um cartão e não em outra mídia deve-se ao princípio Pequena (ser curta). Porém, se a História do Usuário exceder o limite do cartão, a mesma pode ser segmentada. Um exemplo informal é: “Os estudantes podem comprar passes de estacionamento”. Em seu livro, Mike Cohn sugere uma abordagem mais formal para escrever Histórias do Usuário (COHN, 2004): Como um “papel” eu quero “algo” para ter “um benefício”. Um “papel” é a forma como o usuário irá atuar na história. A funcionalidade que ele deseja é expressa por “algo”. E o ganho que o usuário espera obter com essa funcionalidade é o “benefício”. Essa abordagem facilita a determinação do porque de uma funcionalidade deve ser construída e para quem, sendo assim, essa é a abordagem normalmente utilizada. O mesmo exemplo ficaria dessa forma: “Como um estudante eu quero comprar um passe de estacionamento para que eu possa dirigir até a escola”. A Figura 2.4 mostra, em inglês, a abordagem de Mike Cohn para esse exemplo.

Métodos ágeis favorecem a comunicação interpessoal em relação à ferramentas e processos, e a rápida adaptação à mudança ao invés da preocupação em relatar o problema excessivamente (COHEN; LINDVALL; COSTA, 2004). Utilizando as Histórias do Usuário pode-se obter esse resultado, pois: (1) representam pequenos pedaços do valor de negócio que podem ser implementados em um período de dias ou semanas, (2) necessitam de pouca manutenção, (3) permitem que

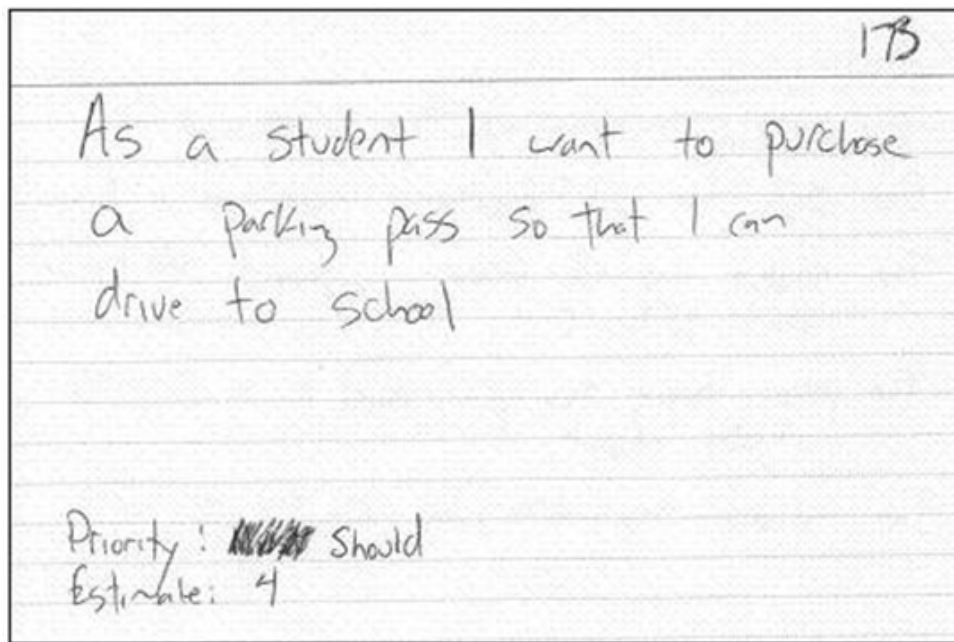


Figura 2.4: Abordagem para utilização das Histórias do Usuário em cartões, retirado de (COHN, 2004).

o desenvolvedor e o usuário discutam os requisitos durante todo o tempo de vida do projeto; (4) possibilitam a segmentação do projeto em pequenos incrementos; (5) são indicadas para projetos nos quais os requisitos são dinâmicos ou de difícil compreensão; e (6) exigem contato próximo com o usuário durante todo o projeto para que as partes de maior valor de negócio da aplicação sejam implementadas corretamente.

Algumas das limitações de Histórias do Usuário em métodos ágeis são: (1) pode ser difícil de escalar em grandes projetos, (2) são consideradas apenas como início da comunicação entre desenvolvedor e usuário, porém não como uma ferramenta para especificação de requisitos.

2.2.3 Desenvolvimento Orientado a Testes (TDD)

Criado por Kent Beck (BECK, 2003), Desenvolvimento Orientado a Testes é uma abordagem que lida com a análise e a especificação do comportamento com base em testes automatizados. Conforme mostra a Figura 2.5, TDD introduz o conceito Red-Green-Refactor: (1) escrever um teste e vê-lo falhar; (2) escrever o código mínimo necessário para fazer o teste passar; e (3) aplicar refatoração com padrões de projeto (FOWLER, 1999) para eliminar redundâncias ou códigos duplicados

Kent Beck considera que TDD incentiva o projeto de código simples e aumenta a confiança no produto final (BECK, 2003). Através de TDD, desenvolvedores podem aplicar o conceito de

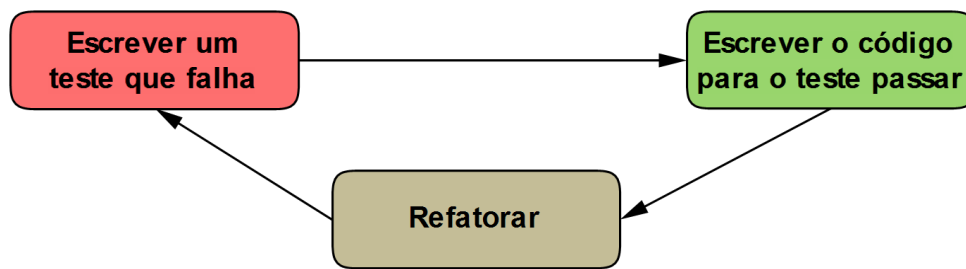


Figura 2.5: Visão geral do TDD, adaptado de (BECK, 2003).

melhorar e depurar código legado desenvolvido a partir de técnicas antigas (FEATHERS, 2004).

No TDD, um teste é um fragmento da aplicação e possui dois objetivos principais: (1) especificação: estabelecer uma regra que a aplicação deve seguir e (2) validação: verificar se a regra foi devidamente implementada na aplicação. Com isso, é possível gerar um código limpo, que de acordo com (MARTIN, 2009) é: “um código que reflete exatamente o que ele foi projetado para fazer, sem artifícios ou obscuridade”.

As principais vantagens do uso de TDD (BECK, 2003) são: (1) o código possui menor acoplamento e maior coesão; (2) o código tem mais qualidade, pois é totalmente testado; (3) a refatoração pode ser executada sem receio de danificar o comportamento atual; (4) é possível saber claramente quando uma tarefa está concluída – quando o(s) teste(s) correspondente(s) não apresenta(m) mais erros; (5) o conjunto de testes serve como base para testes de regressão automatizados; e (6) a grande maioria dos bugs é encontrada logo no início do desenvolvimento, possibilitando resolvê-la com menos esforço e menor custo.

2.2.4 Desenvolvimento Orientado a Comportamento (BDD)

Desenvolvimento Orientado a Comportamento, criado por Dan North (NORTH, 2006), é uma técnica ágil que, durante o processo de desenvolvimento, incentiva a colaboração entre desenvolvedores, equipe de garantia de qualidade e usuários. Resumidamente, no BDD, o usuário define como a aplicação deve se comportar, escrevendo um teste automatizado para a sua verificação. Depois disso, o código necessário para criar esse comportamento é implementado. Assim, sendo considerado como um teste de aceitação e/ou funcional.

A principal diferença entre TDD e BDD é que o primeiro é mais indicado para testes com baixo nível de abstração - como os testes unitários - enquanto o segundo possui foco nas regras de negócio que devem respeitadas pela aplicação. Um exemplo dessa diferença seria: “a tela inicial deve listar todos os clientes” e “o método HomeController.index necessita criar uma variável chamada clientes”. A primeira abordagem é compreensível por um usuário, enquanto

a segunda é voltada para um programador. O primeiro exemplifica o comportamento de uma determinada tela, uma possível solicitação do usuário. O segundo exemplifica como um programador interpretaria o código-fonte que realiza essa funcionalidade.

BDD oferece algumas vantagens (NORTH, 2006): (1) aumenta a integração entre usuários finais, desenvolvedores e testadores, já que todos utilizam o mesmo canal de comunicação; (2) mesmo quando os testadores e desenvolvedores estão em equipes diferentes, há possibilidade de trabalharem em conjunto para descrever “o que” a aplicação deve resolver. Utilizar Histórias do Usuário é uma ferramenta indicada para conter essas descrições; e (3) as descrições comportamentais criadas fornecem casos de teste, que podem ser automatizados ou não, além funcionarem como parte das especificações do projeto. Com isso, pode-se dizer que o usuário final é capaz de “escrever o código” para os testes de aceitação.

Finalmente, além de incentivar a escrita de um código de melhor qualidade, BDD reduz a carga total de trabalho das equipes e aprimora a comunicação de todos os envolvidos, algo essencial para um processo de desenvolvimento mais ágil.

2.3 Métodos Ágeis

Alguns métodos ágeis, incluindo os mais conhecidos tanto no mercado de trabalho quanto no meio acadêmico, são discutidos nesta seção. Os primeiros a serem abordados são aqueles mais comumente encontrados quando o termo “ágil” é utilizado no meio computacional.

O termo “ágil” apareceu pela primeira vez em 2001, quando um grupo de dezessete desenvolvedores de software se reuniu para discutir sobre métodos de desenvolvimento mais leves. O resultado dessa reunião foi o Manifesto Ágil⁹, que possui os seguintes valores: (1) indivíduos e suas interações sobre processos e ferramentas; (2) aplicação em funcionamento sobre documentação abrangente; (3) colaboração com o cliente sobre negociação de contratos; e (4) responder a mudanças sobre seguir um plano. Dessa forma, mesmo havendo valor nas partes não sublinhadas, são mais valorizadas as sublinhadas.

Os métodos ágeis são diferentes das propostas tradicionais que são fundamentadas no processo cascata. A diferença essencial é que o modelo tradicional, baseado nos ensinamentos do PMI (Project Management Institute) propõe que Escopo é uma constante, Recursos podem ser tratados como uma variável e Tempo é uma variável. Por sua vez, em métodos ágeis isso se inverte: Tempo passa a ser uma constante, Recursos continuam podendo ser tratados como variável e Escopo é uma variável. Ou seja, essa inversão de importância é o que torna os

⁹<http://agilemanifesto.org/>

métodos ágeis adequados para lidar com um ambiente onde o usuário muda constantemente de opinião.

2.3.1 Scrum

Scrum é um arcabouço para desenvolvimento ágil de software que é interativo e incremental. Inicialmente foi concebido como um estilo de gerenciamento de projetos para o setor automobilístico e indústrias de consumo. Esses setores notaram que ao utilizar pequenas equipes multidisciplinares (*cross funcional*) em seus projetos, o resultado final foi consideravelmente melhor. Ken Schwaber formalizou a definição de Scrum e ajudou a introduzir o conceito no desenvolvimento de aplicações em todo o mundo (SCHWABER; BEEDLE, 2002).

Atualmente a principal função do Scrum é ser utilizado para gerenciar projetos que lidam com o desenvolvimento de aplicações. Pode ser utilizado também em qualquer outro contexto no qual um grupo de pessoas necessite trabalhar em conjunto para alcançar um objetivo comum. A Figura 2.6 apresenta a visão geral do método Scrum.

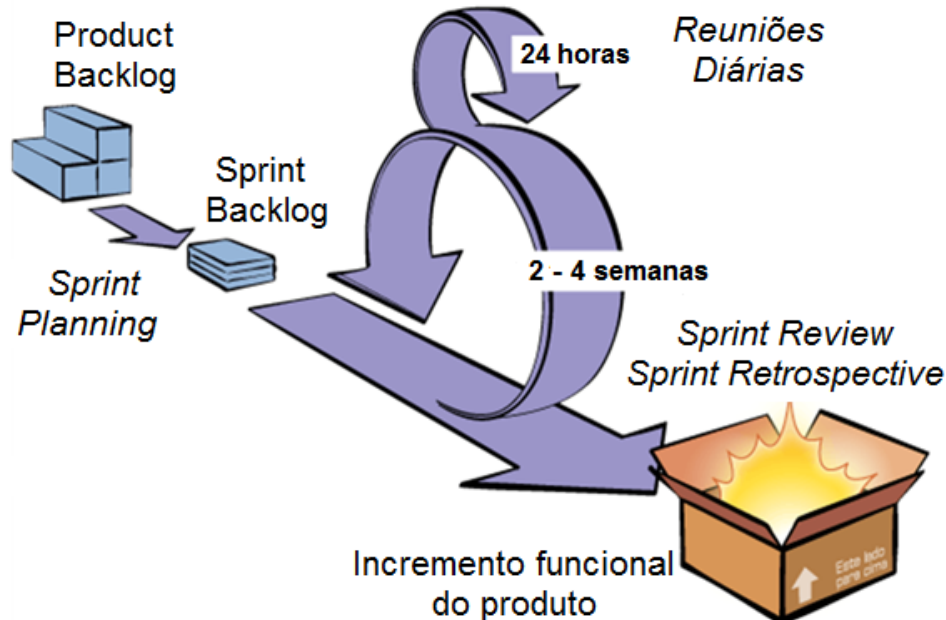


Figura 2.6: O método Scrum, adaptado de (SCHWABER, 2004).

Scrum possui três papéis principais: (1) o Scrum Master, que mantém os processos; (2) o Product Owner, que representa as partes interessadas e de negócio; (3) o Scrum Team, um grupo multifuncional composto por cerca de sete pessoas que fazem análise, projeto, implementação, testes, entre outras tarefas do desenvolvimento da aplicação.

Durante cada Sprint, tipicamente um período de duas a quatro semanas, o Scrum Team cria um incremento de produto que pode ser colocado em produção. Os recursos que entram em um Sprint vêm do Product Backlog. Este, criado pelo Product Owner, é um conjunto de requisitos priorizados pelo usuário. Os itens do Product Backlog que irão para o Sprint são determinados durante o Sprint Planning Meeting. Durante esta reunião, o Product Owner informa ao Scrum Team sobre os itens do Product Backlog com maior prioridade para o usuário. O Scrum Team então determina com quantos itens eles podem se comprometer a concluir durante o próximo Sprint.

Uma vez que o Sprint teve início, ninguém é autorizado a mudar o conteúdo do mesmo, isso significa que os requisitos daquele Sprint são inalteráveis. O desenvolvimento tem o prazo fixo de tal forma que o Sprint deve terminar no tempo previsto. Diariamente, o Scrum Team se reúne no chamado Daily Scrum Meeting, que é uma reunião rápida em pé, para expor o que está sendo realizado e quais as dificuldades encontradas. Fica a cargo do Scrum Master resolver ou minimizar essas dificuldades. Se algum requisito não é concluído por algum motivo nesse período, ele é deixado de lado e volta para o Product Backlog.

Ao término de um Sprint, o Scrum Team se reúne para analisar os pontos positivos e negativos (Sprint Retrospective). Também demonstra a aplicação resultante para o usuário (Sprint Review).

2.3.2 eXtreme Programming (XP)

O eXtreme Programming, é um conjunto de métodos para equipes pequenas e médias que irão desenvolver aplicações com requisitos vagos e dinâmicos, visando melhorar a qualidade da aplicação e a capacidade de resposta às exigências dos usuários. O XP adota a estratégia de constante acompanhamento e execução de vários pequenos ajustes durante o desenvolvimento da aplicação. Os quatro valores fundamentais da metodologia XP são: comunicação, simplicidade, feedback e coragem. A partir desses valores, tem como princípios básicos: feedback rápido, mudanças incrementais, comprometimento com as mesmas e trabalho de qualidade. Na Figura 2.7 são apresentadas as principais práticas do XP.

Entre as variáveis de controle em projetos (custo, qualidade, tempo e escopo), há um foco explícito em escopo. O XP recomenda a priorização de recursos os quais representam o maior valor possível para o negócio. Assim, quanto mais o escopo do projeto for reduzido, mais funcionalidades com menor valor de negócio serão adiadas ou canceladas.

O XP também incentiva o controle da qualidade, pois o pequeno ganho em curto prazo

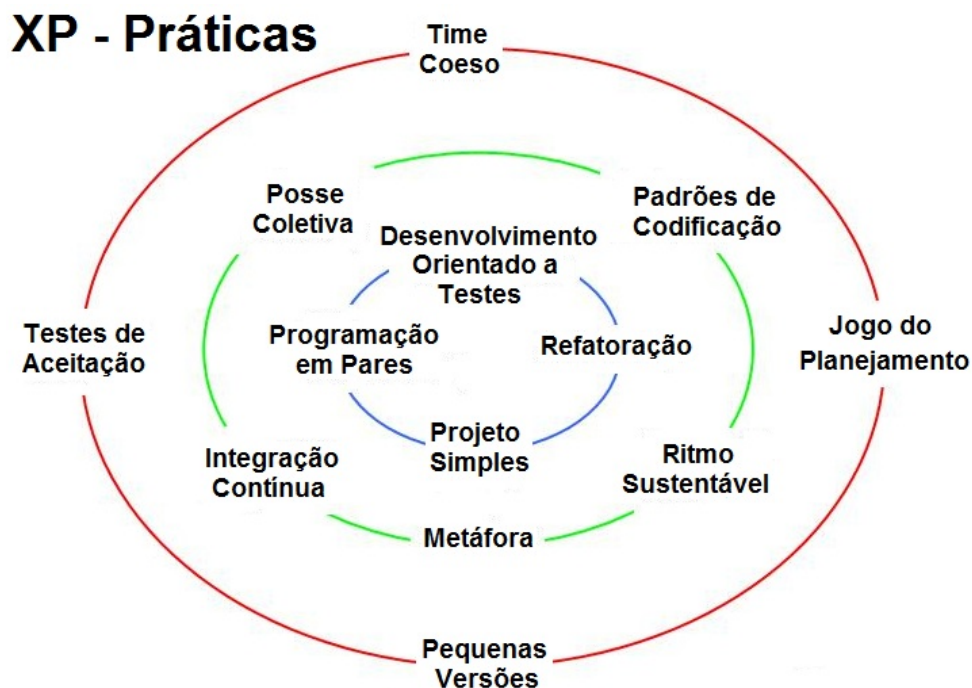


Figura 2.7: As principais práticas do XP, adaptado de (JEFFRIES; ANDERSON; HENDRICKSON, 2000).

na produtividade gerado pela baixa qualidade, não é compensado pelas possíveis perdas (ou impedimento) em médio e longo prazo.

2.3.3 Feature Driven Development (FDD)

Desenvolvimento Guiado por Funcionalidade teve origem a partir de um projeto bancário em Cingapura, entre 1997 e 1999. Seu lema é “Resultados frequentes, tangíveis e funcionais” (PALMER; FELSING, 2002).

O FDD é, classicamente, descrito por cinco processos, os quais estão mostrados na Figura 2.8:

Desenvolver um Modelo Abrangente: pode envolver desenvolvimento de requisitos, análise orientada por objetos, modelagem lógica de dados e outras técnicas para entendimento do domínio de negócio em questão. O resultado é um modelo de objetos (e/ou de dados) de alto nível, que guiará a equipe durante os ciclos de construção.

Construir uma Lista de Funcionalidades: decomposição funcional do modelo do domínio, em três camadas típicas: áreas de negócio, atividades de negócio e passos automatizados da atividade (funcionalidades). O resultado é uma hierarquia de funcionalidades que representa o produto a ser construído (também chamado de Product Backlog, ou lista de espera do produto).

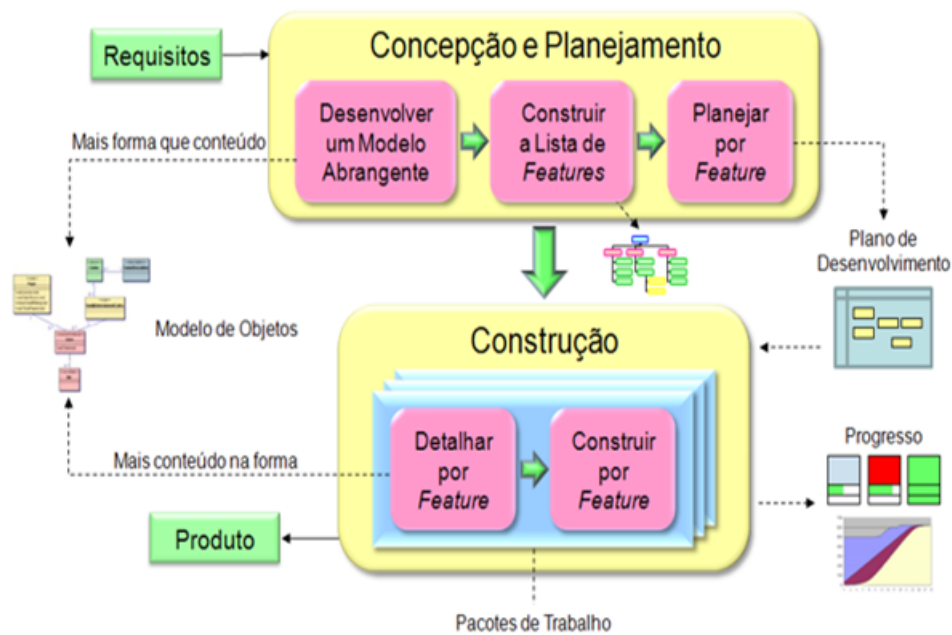


Figura 2.8: Os cinco processos do FDD.

Planejar por Funcionalidade: abrange a estimativa de complexidade e dependência das funcionalidades, também levando em consideração a prioridade e valor para o negócio/cliente. O resultado é um plano de desenvolvimento, com os pacotes de trabalho na sequência apropriada para a construção.

Detalhar por Funcionalidade: já dentro de uma iteração de construção, a equipe detalha os requisitos e outros artefatos para a codificação de cada funcionalidade, incluindo os testes. O projeto para as funcionalidades é inspecionado. O resultado é o modelo de domínio mais detalhado e os esqueletos de código prontos para serem preenchidos.

E **Construir por Funcionalidade:** cada esqueleto de código é preenchido, testado e inspecionado. O resultado é um incremento do produto integrado ao repositório principal de código, com qualidade e potencial para ser usado pelo cliente/usuário.

2.3.4 Agile Unified Process (AUP)

AUP é uma simplificação do RUP (Rational Unified Process) desenvolvido por Scott Ambler. Ele descreve um método para o desenvolvimento de aplicações de negócios usando técnicas e conceitos ágeis, sem deixar de permanecer fiel aos conceitos do RUP (LEFFINGWELL, 2011).

Faz uso de técnicas ágeis incluindo TDD, gerenciamento de mudanças e refatoração, visando a melhoria da produtividade. Todo o processo é baseado nos seguintes princípios: (1)

Sua equipe sabe o que está fazendo. As pessoas não irão ler a documentação do processo detalhado, mas eles querem alguma orientação de alto nível e/ou treinamento de vez em quando; (2) Simplicidade. Tudo é descrito de forma concisa usando um punhado de páginas; (3) Agilidade. O AUP está em conformidade com os valores e princípios do Manifesto Ágil; (4) Concentre-se em atividades de alto valor. O foco é sobre as atividades que realmente importam; (5) Independente de ferramentas. Pode-se usar qualquer conjunto de ferramentas. A recomendação é que se utilizem as mais adequadas para o trabalho, que muitas vezes são as mais simples; e (6) Pode-se adaptar o AUP para atender às necessidades.

Diferente do RUP, o AUP contém apenas sete disciplinas, sendo elas: Modelagem, Implementação, Testes, Implantação, Gerenciamento de Configuração, Gerenciamento de Projeto e Ambiente. A Figura 2.9 apresenta essas disciplinas ao longo do ciclo de desenvolvimento do projeto.

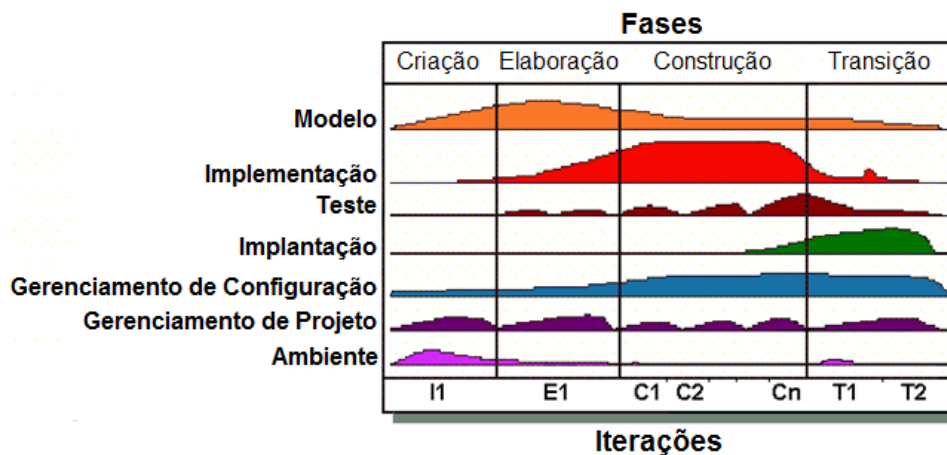


Figura 2.9: Ciclo de vida do AUP, retirado de (SLIGER; BRODERICK, 2008).

Além disso, AUP distingue dois tipos de iterações. Uma Iteração de Desenvolvimento resulta em uma versão do produto para o Controle de Qualidade. Por sua vez, a Iteração de Produção resulta em uma versão final. Este é um aperfeiçoamento significativo em relação ao RUP.

2.3.5 Microsoft Solutions Framework (MSF)

Atualmente na versão 5.0, o MSF é um conjunto de princípios, modelos, disciplinas, conceitos e diretrizes para fornecer soluções em Tecnologia da Informação criada pela Microsoft (POWERS, 2006). Não se limita apenas ao desenvolvimento de aplicações, sendo também aplicável a outros projetos de TI, como implantação de projetos de redes ou infraestrutura.

A espinha dorsal do MSF é formada por oito princípios fundamentais: (1) promover uma comunicação aberta; (2) trabalhar no sentido de uma visão compartilhada; (3) capacitar os

membros da equipe; (4) estabelecer uma responsabilização clara e compartilhada; (5) foco em agregar valor de negócios; (6) manter-se ágil, esperar por mudanças; (7) investir na qualidade; e (8) aprender com todas as experiências.

Além disso, o MSF possui dois modelos. O Modelo de Equipe descreve papéis para os membros da equipe. Entre esses papéis estão: gerente do produto, gerente do projeto, arquiteto, desenvolvedor e testador. O Modelo de Governança descreve as cinco etapas do processo. Cada uma possui uma meta de qualidade definida. Essas etapas são: (1) visualizar: pensar sobre o que precisa ser realizado e identificar as restrições; (2) planejar: planejar e projetar uma solução para atender as necessidades e expectativas dentro das limitações; (3) construir: construir a solução; (4) estabilizar: validar se a solução atende às necessidades e às expectativas; e (5) implantar: implantar a solução.

Por fim, o MSF atualmente é subdividido em duas abordagens: uma para os padrões do CMMi e outra para o desenvolvimento ágil. Este, conhecido como MSF para Agile Software Development (MSF4ASD) se destina a ser um processo leve, iterativo e adaptável, fornecendo uma orientação de processo que incide sobre as pessoas e mudanças.

2.4 Considerações Finais

O capítulo abordou conceitos úteis para o trabalho, como História do Usuário, que é o principal artefato utilizado neste projeto, juntamente com os conceitos de TDD e BDD, que proporcionam uma base de guia para execução do processo, e Scrum, que possui suas principais atividades incorporadas no processo proposto.

No próximo capítulo são apresentados os trabalhos correlatos encontrados na literatura, tanto as ferramentas desenvolvidas quanto ao processo ágil proposto.

Capítulo 3

TRABALHOS CORRELATOS

Trabalhos relacionados ao desenvolvimento ágil de aplicações Web e ao uso de testes funcionais têm sido pesquisados e propostos pela comunidade acadêmica, demonstrando a relevância do assunto abordado neste trabalho. Este capítulo apresenta alguns desses trabalhos que possuem certa relação com o trabalho desenvolvido nesta pesquisa.

A organização do capítulo está conforme segue: a Seção 3.1 apresenta os principais trabalhos encontrados em relação as ferramentas desenvolvidas para dar suporte ao processo RAMBUS; a Seção 3.2 apresenta os principais trabalhos encontrados na literatura em relação ao processo RAMBUS; por fim, a Seção 3.3 conclui o capítulo tecendo algumas considerações finais.

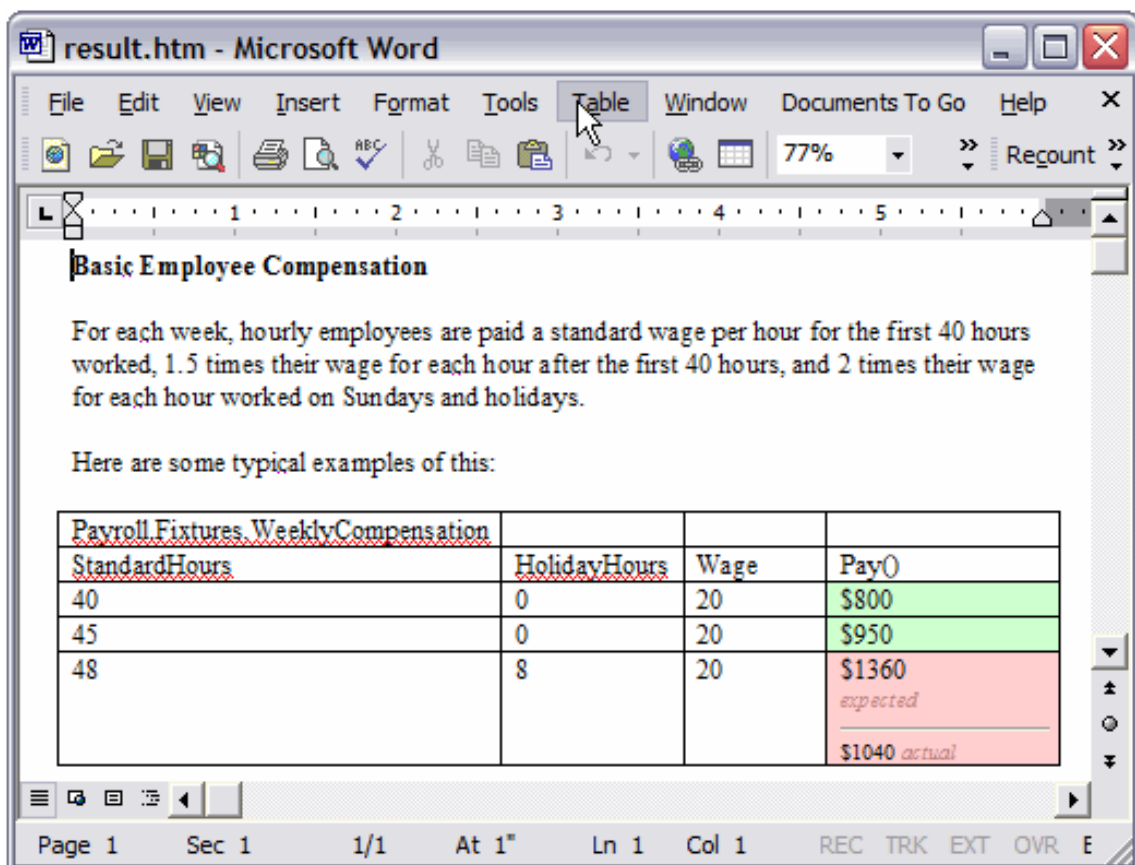
3.1 Trabalhos Correlatos às Ferramentas Desenvolvidas Para o Processo

Nessa seção são apresentados trabalhos correlatos ao das ferramentas desenvolvidas para dar suporte ao processo RAMBUS, na integração de testes funcionais durante o desenvolvimento das aplicações.

3.1.1 Framework for Integrated Tests (FIT)

O FIT (Framework for Integrated Tests) (MUGRIDGE; CUNNINGHAM, 2005), descreve os testes na forma de tabelas escritas em HTML ou utilizando um editor de textos como o Microsoft Word. Essas tabelas têm nome para ser chamado no código do software e um atributo em cada coluna, seguido de seus métodos, com os exemplos para testar descritos por linha. Possuem, também, uma descrição do problema que elas representam para melhorar a compre-

ensão da funcionalidade do software e auxiliar na realização dos testes. O código dos testes escritos na linguagem alvo utilizando o FIT examinam os resultados obtidos e os comparam com o resultado esperado presente nas tabelas. Caso o teste tenha sido aprovado, a célula da tabela correspondente a esse teste é preenchida com a cor verde. Caso contrário, é preenchida em vermelho e o framework mostra qual o resultado esperado e qual foi obtido, para análise e correção por parte dos desenvolvedores. A Figura 3.1 mostra um exemplo do funcionamento do FIT.



<u>Payroll.Fixtures.WeeklyCompensation</u>			
<u>StandardHours</u>	<u>HolidayHours</u>	<u>Wage</u>	<u>Pay()</u>
40	0	20	\$800
45	0	20	\$950
48	8	20	\$1360 <i>expected</i> \$1040 <i>actual</i>

Figura 3.1: Documento FIT com sucessos e falhas, mostrando o que era esperado e qual o valor obtido, retirado de (MUGRIDGE; CUNNINGHAM, 2005).

Na abordagem proposta as ferramentas desenvolvidas diferem do FIT (MUGRIDGE; CUNNINGHAM, 2005) por não utilizar tabelas para criar os testes funcionais e automatizá-los. Essa decisão foi porque não é muito prático utilizar tabelas para descrever as atividades relativas a uma página Web e suas possíveis navegações entre outras páginas. Além disso, as ferramentas utilizadas oferecem suporte para desenvolvimento com maior orientação ao comportamento esperado pelo usuário, pois utiliza descrições textuais para geração dos testes funcionais.

3.1.2 A Process-Complete Automatic Acceptance Testing Framework

Outra ferramenta apresentada por (TALBY et al., 2005) propõe uma abordagem para a definição de testes funcionais em detrimento do uso de descrições textuais.

Essa abordagem é justificada, segundo os autores, para facilitar a automatização dos testes. A abordagem definida utiliza tabelas com uma coluna para cada item (*step*) da aplicação em desenvolvimento que está sendo testado (tais como editar campos, realizar uma ação e emitir mensagens) e seus respectivos parâmetros. A Figura 3.2 mostra essa abordagem, utilizando como exemplo o equivalente a descrição textual “Write 999 in the amount field and hit 'Save’”.

<i>Step Type</i>	<i>Parameter 1</i>	<i>Parameter 2</i>
Edit Field	Amount	999
Do Action	Save	

Figura 3.2: A abordagem proposta por (TALBY et al., 2005).

Os resultados obtidos em cada passo são armazenados na última coluna de cada tabela. Um outro exemplo (mais completo), mostrando os resultados dos testes pode ser visto na Figura 3.3.

<i>Step Type</i>	<i>Param 1</i>	<i>Param 2</i>	<i>Result</i>
Open Form	Customer		Passed
Edit Field	Full Name	John Doe	Passed
Edit Field	Amount	100.0	Passed
Check Field	Amount	100.0	Passed
Do Action	Save		Failed
Check Action	Save	Enabled	Not Run

Figura 3.3: Resultado do teste apresentado dentro da tabela, na última coluna. Retirado de (TALBY et al., 2005).

Diferente deste trabalho, a abordagem proposta consegue gerar testes funcionais a partir de Histórias do Usuário, que são descrições textuais, sem a necessidade de adotar uma abordagem com um nível de abstração mais baixo.

3.1.3 Call for Testing

Outra ferramenta, chamada Call For Testing (CFT) (YU et al., 2009), emprega a comunidade *opensource* para criar os testes para o software em desenvolvimento. Um servidor CFT gerencia o acesso a aplicação e armazena os testes criados pelos usuários da comunidade *opensource* que

se candidatam para testar o software. Esses testes são criados utilizando uma notação própria que separa as ações em quatro possibilidades: link, evento, entrada de dados e relacionamento. Essas ações são agrupadas segundo um diagrama de estados que representa a navegação pelas páginas do software. O servidor CFT executa esses testes e encaminha os resultados para os desenvolvedores analisarem e realizarem as correções necessárias. A Figura 3.4 mostra como é a interação entre as partes do processo CFT e sua plataforma.

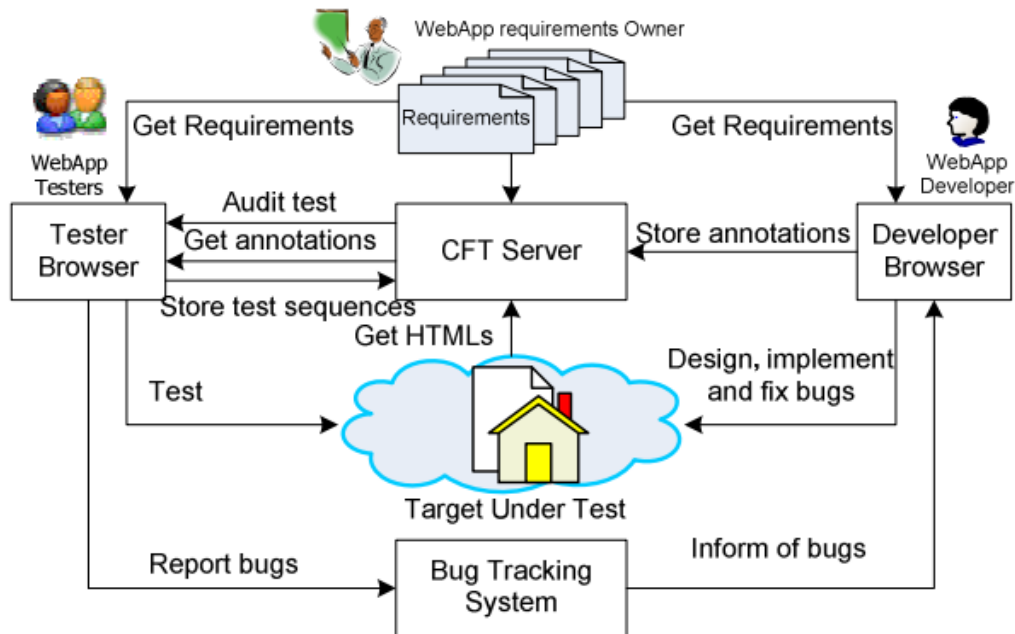


Figura 3.4: Processo colaborativo e a plataforma CFT, retirado de (YU et al., 2009).

Diferente deste trabalho, a abordagem proposta utiliza ferramentas para gerar testes funcionais baseados nos requisitos especificados pelo usuário através das Histórias do Usuário. Dessa forma, substitui a tarefa de atribuir a pessoas que não fazem parte do projeto (a comunidade *opensource*) a função de definir testes, o que pode não ser muito interessante ou seguro para determinados projetos.

3.2 Trabalhos Correlatos ao Processo RAMBUS

Nessa seção são apresentados os trabalhos correlatos ao processo RAMBUS encontrados na literatura.

3.2.1 Agile Web Development with Web Framework (AWDWF)

O processo Agile Web Development with Web Framework (RAN et al., 2008) é a união do conceito de desenvolvimento ágil com a maturidade de *frameworks* Web. AWDWF de-

fende que as pessoas são fatores determinantes no processo de desenvolvimento, sendo assim a comunicação entre os membros das equipes e entre estes e os usuários é vital para o sucesso da aplicação.

Conforme mostra a Figura 3.5, AWDWF consiste na utilização de *frameworks* para orientar a disciplina de Projeto da Engenharia de Software tradicional. A ideia é usar os arcabouços para a parte do desenvolvimento que não está ligada à lógica de negócios, aumentando assim sua estabilidade e eficiência, além de garantir a qualidade desta parte da aplicação Web. Desta forma, o foco da equipe de desenvolvimento fica orientado para a lógica de negócio.

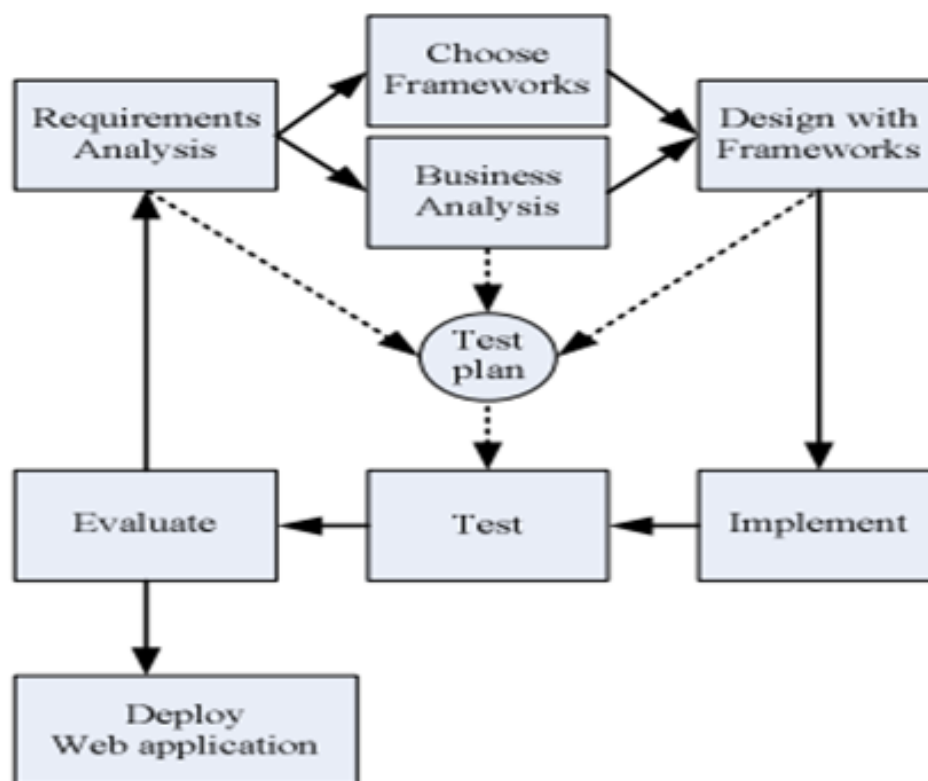


Figura 3.5: Processo AWDWF, retirado de (RAN et al., 2008).

Em comparação com a proposta AWDWF (RAN et al., 2008), o processo apresentado nesta dissertação, além do uso de *frameworks*, mostra como interagir com o cliente e como usar os artefatos desta interação para guiar a implementação de aplicações *groupware*, utilizando alguns destes artefatos para criar testes funcionais.

3.2.2 A Theoretical Agile Process Framework for Web Applications Development in Small Software Firms

Nesse método ágil (ALTARAWNEH; SHIEKH, 2008) são propostas melhorias no ciclo de vida do XP e acrescenta-se uma atividade de gestão de projeto. Esta adição se justifica porque o

gerenciamento de projetos no XP pode ser considerado limitado. O processo é dividido em seis etapas: (1) começar com pequenos projetos Web; (2) adotar o processo XP modificado; (3) aplicar XPMM (eXtreme Programming Maturity Model) (NAWROCKI; WALTER; WOJCIECHOWSKI, 2001); (4) educação e formação; (5) avaliação interna e revisões formais mais leves; e (6) avaliação externa. Todas estas etapas devem ser executadas utilizando as melhores práticas na Web Engenharia.

O processo RAMBUS não é baseado apenas no XP, como o proposto por (ALTARAWNEH; SHIEKH, 2008), mas em TDD e com o Scrum como base. Outra diferença é que o processo RAMBUS utiliza o conceito de desenvolvimento guiado por comportamento, para que os testes funcionais sejam o mais próximo possível do que o usuário espera.

3.2.3 Web Modeling Language (WebML)

WebML é uma linguagem de alto nível para projetar aplicações Web com fluxo de dados intenso. Foi criada em 1998 e apresentada formalmente em 2000 (CERI; FRATERNALI; BONGIO, 2000). É adotada em várias universidades ao redor do mundo, segundo seus autores e possui uma aplicação comercial, denominada WebRatio¹.

Conforme demonstra a Figura 3.6, o processo é dividido nas fases de Estrutura, Hipertexto e Apresentação. Dentro de cada fase existem modelos para guiar o desenvolvimento da aplicação Web, sendo que as maiorias desses modelos se encontram na fase de Hipertexto. Os modelos são: Modelo de Estrutura, Modelo de Navegação, Modelo de Composição, Modelo de Usuário e Modelo de Apresentação.

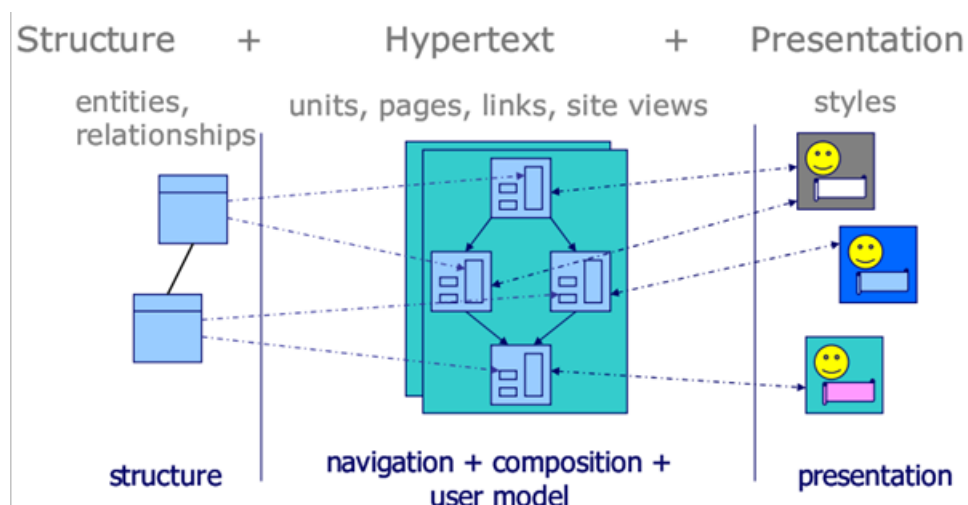


Figura 3.6: Processo WebML, retirado de (CERI; FRATERNALI; BONGIO, 2000).

¹<http://www.webratio.com/>

O Modelo de Estrutura expressa o conteúdo dos dados do site, em termos de entidades relevantes e os relacionamentos. Não é proposta outra linguagem de modelagem de dados, sendo a WebML compatível com as notações clássicas, como o modelo ER e os diagramas de classe da UML (Unified Modeling Language).

O Modelo de Navegação expressa como páginas e unidades de conteúdo estão ligadas à forma de hipertexto. Links são tanto não contextuais, quando eles conectam páginas semanticamente independentes (por exemplo, a página de um artista para a página inicial do site), quanto contextual, quando o conteúdo de destino depende do conteúdo da fonte.

O Modelo de Composição especifica quais as páginas que compõem o hipertexto e quais unidades de conteúdo compõem uma página.

No Modelo de Usuário o administrador da Web define as características de perfis de usuários, com base em requisitos de personalização. Os potenciais utilizadores e grupos de usuários são mapeados para a notação da WebML e, possivelmente, uma visão diferente do site é criada para cada grupo.

Por fim, o Modelo de Apresentação expressa a aparência e layout gráfico das páginas, independentemente do dispositivo de saída e da linguagem de programação escolhida, por meio de uma sintaxe XML abstrata.

Diferente do processo WebML, o processo RAMBUS não utiliza uma linguagem de alto nível própria, mas uma adaptação do texto descritivo utilizado nas Histórias do Usuário. Dessa forma, os desenvolvedores utilizam uma forma textual de fácil entendimento para especificar os requisitos da aplicação, facilitando, também, o diálogo com o Usuário para resolver possíveis dúvidas. Por fim, o RAMBUS utiliza essas histórias como testes funcionais para guiar o desenvolvimento da aplicação.

3.2.4 Uml-based Web Engineering (UWE)

UWE é uma abordagem de engenharia de software para o desenvolvimento de aplicações Web. Fornece um perfil UML (extensão da UML), um meta modelo, um processo de desenvolvimento orientado por modelo e ferramenta de suporte - ArgoUWE - para o projeto de aplicações Web. UWE segue a separação de interesses, possuindo diagramas separados para a construção de modelos distintos para os requisitos, para descrição do conteúdo, hipertexto, apresentação, processo de adaptabilidade e arquitetura (KOCH et al., 2008). A Figura 3.7 mostra um exemplo da extensão da UML criada pelo UWE.

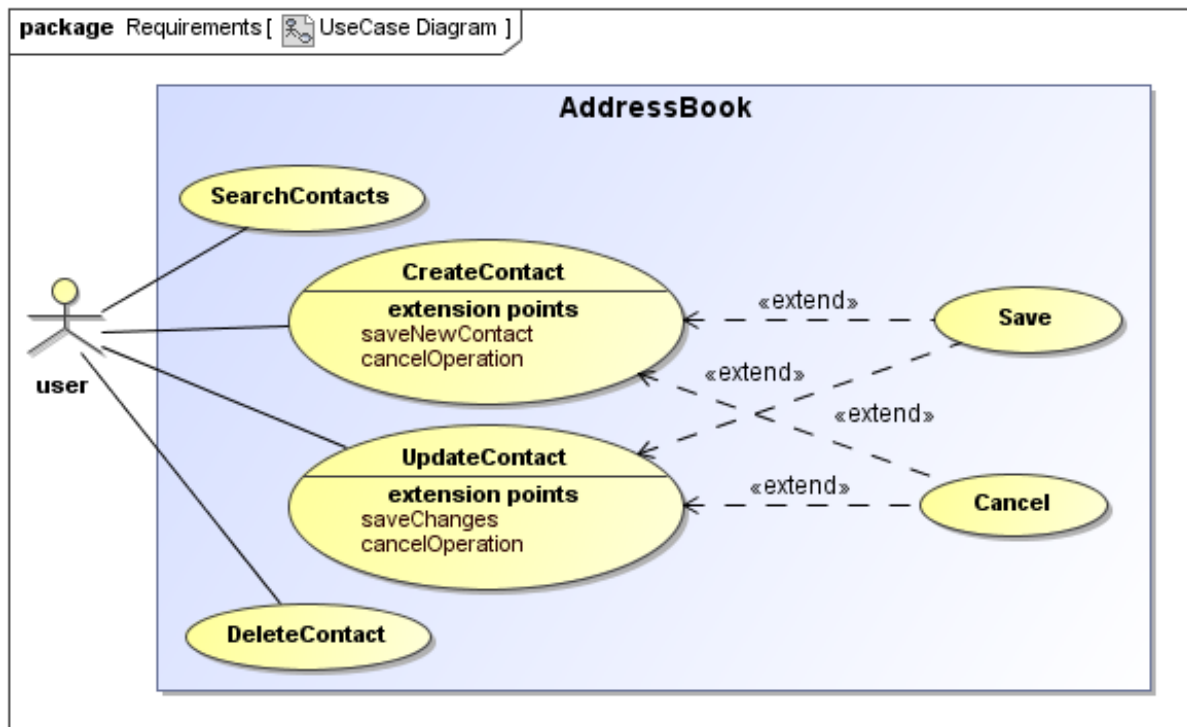


Figura 3.7: Exemplo da notação do UWE, retirado de (KOCH et al., 2008).

Diferente deste processo, o RAMBUS não utiliza uma extensão da UML. O RAMBUS utiliza outros artefatos (como a História do Usuário e os Mocku-ps) em conjunto com alguns diagramas UML para, dessa forma, oferecer aos desenvolvedores um processo ágil com o uso de testes funcionais integrado ao desenvolvimento.

3.3 Considerações Finais

Este capítulo apresentou os trabalhos com foco no desenvolvimento de aplicações Web e a utilização de testes funcionais em processos de desenvolvimento que possuem correlação ao trabalho desenvolvido nesta pesquisa.

O processo RAMBUS é baseado em algumas características desses trabalhos correlatos, porém têm suas próprias contribuições por meio da adequação e evolução dessas características. Mais precisamente, as contribuições deste trabalho estão voltadas para o suporte ao desenvolvimento de aplicações groupware na Web 2.0 com o apoio de testes funcionais integrados ao processo.

Assim, no processo proposto são combinadas as concepções de TDD, BDD, testes funcionais e métodos ágeis para a formulação de um processo apropriado que define atividades e artefatos para apoio ao desenvolvimento Web.

Para auxiliar no uso do processo ágil – que será apresentado no Capítulo 5 –, duas ferramentas foram criadas. Essas ferramentas são contribuições deste trabalho e seu desenvolvimento é apresentado em detalhes no Capítulo 4.

Capítulo 4

FERRAMENTAS DESENVOLVIDAS

Foram desenvolvidas duas ferramentas para automação parcial de testes funcionais (PEREIRA; PRADO, 2012). Essas ferramentas funcionam especificamente com a plataforma Visual Studio, da Microsoft, e com a linguagem C#. A primeira ferramenta a ser apresentada é a Linguagem de Especificação de Testes (LET), seguida do Conversor de LET para C#.

Este capítulo apresenta, na Seção 4.1, a linguagem criada para especificar a História do Usuário de forma a melhor convertê-la para testes funcionais. Na sequência, na Seção 4.2, é mostrada a ferramenta de conversão desenvolvida para dar suporte ao processo. Por fim, na Seção 4.4 estão as Considerações Finais.

4.1 Linguagem de Especificação de Testes (LET)

A Linguagem de Especificação de Testes foi criada para ser uma forma descritiva da História do Usuário. Essa forma possui elementos que facilitam a conversão do texto descritivo para testes funcionais escritos na linguagem C#. Sendo assim, as Histórias do Usuário, uma vez especificadas em cartões, são posteriormente descritas na LET, para possibilitar a integração dos testes funcionais no processo.

A gramática dessa linguagem foi desenvolvida com base no conceito de Interface Fluente (FOWLER, 2011). Isto significa que a implementação da API (*Application Programming Interface* - Interface de Programação de Aplicativos) orientada a objetos é projetada visando fornecer um código mais legível. A Figura 4.1 apresenta um trecho de código escrito sem utilizar o conceito de Interface Fluente (a) e outro utilizando o conceito (b). Visando utilizar esses dois trechos, a Figura 4.2 apresenta como chamar os métodos criados na Figura 4.1.

```
public interface IConfiguration
{
    void SetColor(string newColor);
    void SetDepth(int newDepth);
    void SetHeight(int newHeight);
    void SetLength(int newLength);
}

public sealed class Configuration : IConfiguration
{
    private string color;
    private int depth;
    private int height;
    private int length;

    public void SetColor(string newColor)
    {
        this.color = newColor;
    }

    public void SetDepth(int newDepth)
    {
        this.depth = newDepth;
    }
}
```

(a) Trecho do código não utilizando Interface Fluente.

```
public interface IConfigurationFluente
{
    IConfigurationFluente SetColor(string newColor);
    IConfigurationFluente SetDepth(int newDepth);
    IConfigurationFluente SetHeight(int newHeight);
    IConfigurationFluente SetLength(int newLength);
}

public sealed class ConfigurationFluente : IConfigurationFluente
{
    private string color;
    private int depth;
    private int height;
    private int length;

    private ConfigurationFluente()
    {
    }

    public static IConfigurationFluente New()
    {
        return new ConfigurationFluente();
    }

    public IConfigurationFluente SetColor(string newColor)
    {
        this.color = newColor;
        return this;
    }
}
```

(b) Trecho do código utilizando Interface Fluente.

Figura 4.1: Diferença entre não usar (a) e usar (b) o conceito de Interface Fluente.

```

IConfiguration config = new Configuration();
config.SetColor("blue");
config.SetHeight(1);
config.SetLength(2);
config.SetDepth(3);

```

(a) Utilizando o código da Figura 4.1 (a).

```

IConfigurationFluent fluentConfig =
    ConfigurationFluent
        .New()
        .SetColor("blue")
        .SetHeight(1)
        .SetLength(2)
        .SetDepth(3);

```

(b) Utilizando o código da Figura 4.1 (b).

Figura 4.2: Diferença entre utilizar os códigos da Figura 4.1.

Além do conceito de Interface Fluente, a LET utiliza as Histórias do Usuário seguindo o padrão formalizado por Dan North (NORTH, 2006). Esse padrão expande o formato proposto por Mike Cohn (COHN, 2004). Dentro desse contexto, uma História do Usuário inicia com a palavra *Feature*: ou *Story is*. Na sequência, uma história pode ter N benefícios, seguidos de N papéis assumidos pelos atores que interagem na história e N objetivos que esses atores desejam realizar. Até então, essa sequência é basicamente a mesma proposta por Mike Cohn (COHN, 2004). Após isso, a história pode ter vários cenários. Eles representam as interações do usuário com a aplicação. Por exemplo, uma história que contempla o “Gerenciamento de Usuários” pode ter três cenários: Criar, editar e apagar usuário. Cada cenário tem seus respectivos critérios de aceitação, que são divididos em três tipos: condições iniciais, operações realizadas e resultados esperados. A palavra-chave “Given” indica as condições iniciais. As operações são indicadas pela palavra-chave “When”. Por fim, os resultados esperados são indicados pela palavra-chave “Then”.

Conforme mostra a Figura 4.3, a LET foi dividida em duas partes para a sua implementação. Os vértices (os estados que a LET pode assumir, como condições, operações e resultados) e as suas respectivas arestas (palavras-chaves, como *Given*, *Then* e *When*) são descritas como um *template* escrito em DOT (GANSNER; NORTH, 2000). DOT é uma notação textual para representação de grafos. A Figura 4.4 mostra um trecho do código escrito em DOT.

O restante da sua implementação, como a sintaxe e a semântica para construir sua classe em C#, é descrito como um *template* do *framework* T4 (Text Template Transformation Toolkit)¹, do Visual Studio². Esse *template* T4 utiliza o *template* escrito em DOT para analisar a estrutura do grafo e, assim, entender a sequência em que as palavras-chave devem aparecer.

¹<http://blogs.msdn.com/b/t4/>

²<http://www.microsoft.com/visualstudio/en-us>

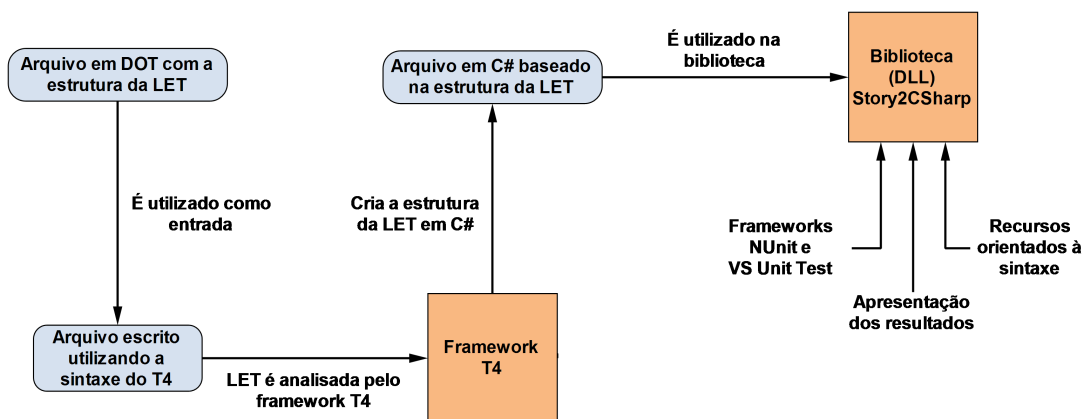


Figura 4.3: O processo de geração do código C#.

```

digraph Story2CSharp{
    //states
    Story
    Benefit
    Role
    Feature
    Scenario
    Condition
    Operation
    Outcome

    //transitions
    Story->Benefit      [label="InOrderTo"]
    Benefit->Benefit    [label="And"]

    Benefit->Role       [label="AsA"]
    Role->Role          [label="OrAsA"]

    Role->Feature       [label="IWant"]
    Feature->Feature    [label="And"]

    Feature -> Scenario [label="Scenario"]
    Scenario -> Condition [label="Given"]
    Condition -> Condition [label="And"]

    Condition -> Operation [label="when"]
    Operation -> Operation [label="And"]

    Operation -> Outcome [label="Then"]
    Outcome -> Outcome [label="And"]

    Outcome -> Scenario [label="Scenario"]
}
  
```

Figura 4.4: Trecho do código escrito em DOT.

A Figura 4.5 mostra trechos dos códigos do *template* T4 (a) e do arquivo gerado na linguagem C# (b). Em (a) tem-se um trecho do *template* do *framework* T4. Neste *template* estão comentados (1 a 4) os locais onde são incluídos as regras do *template* DOT. O código entre os símbolos <# e #> são próprios do *template* T4 e os demais códigos são trechos a serem incluídos no código C# a ser gerado. Portanto, conforme é apresentado em (b), o *framework* T4 é responsável por analisar os *templates* e gerar o correspondente código C#.

```

...
// (1) graph.States representa o grafo gerado pelo
// template DOT
<# foreach(var state in graph.States) { #>
...
// (2) state.Label.Name = Benefit
public class <#=state.Label.Name#> : FragmentBase {
...
  <# foreach(var exit in state.OutgoingTransitions) {
    var target = exit.ToState;
    string prefix = UnCamel(exit.Label.Name);
    ...
  #>
    ...
    // (3) Loop 1 in Benefit:
    // target.Label.Name = Benefit
    // exit.Label.Name = And

    // (4) Loop 2 in Benefit:
    // target.Label.Name = Role
    // exit.Label.Name = AsA
    public <#=target.Label.Name#>
      <#=exit.Label.Name#>(string text) {
        Step s = new Step("<#=prefix#>", text);
        return new <#=target.Label.Name#>(s, this);
      }
  }
}
}
}
#>
...

```

(A)

```

public class Benefit : FragmentBase
{
  public Benefit And(string text)
  {
    Step s = new Step("and", text);
    return new Benefit(s, this);
  }

  public Role AsA(string text)
  {
    Step s = new Step("as a", text);
    return new Role(s, this);
  }
}

```

(B)

(a) Trecho do código do *template* T4 da Figura 4.3.

(b) Trecho do código C# gerado da Figura 4.3.

Figura 4.5: Trecho do *template* T4 (a) e o código C# gerado por ele (b).

Pequenas modificações foram introduzidas na estrutura da História do Usuário prevista por Mike Cohn (COHN, 2004), visando a implementação em C#. Ao invés de utilizar aspas duplas (“ ”) para designar todos os elementos “editáveis” de uma História do Usuário, o símbolo “\$” é usado para referenciar identificadores (campos, variáveis e botões) e chaves ({ }) são usadas para indicar strings.

Caso necessário, a LET pode ser alterada tanto mudando o comportamento dos seus estados (e suas respectivas transições ou palavras-chave) quanto adicionando novos estados ou palavras-chave. Um exemplo disso seria adicionar as palavras-chave *Or* para os estados de Condição, Operação e/ou Resultado. Para isso, basta alterar o arquivo DOT e gerar uma nova versão da linguagem (compilar no Visual Studio), tornando-a mais adequada para uma determinada situação. A Figura 4.6 mostra um exemplo de uma História do Usuário definida utilizando a LET.

```
Feature: Change email
In order to keep my profile updated
As a register user
I want to be able to change my email

Scenario: Change my email
Given I am signed in
And I click on my name in the header
And I follow $settings
When I am on my account settings page
And I fill in $user_email with (new_email@example.com)
And I press $change_email
Then I should see (Email changed)
```

Figura 4.6: Exemplo de uma História do Usuário escrita na LET.

Para torná-la operacional, a LET é implementada como uma DLL (Dynamic-Link Library), no ambiente Visual Studio. Essa DLL, denominada Story2CSharp, oferece suporte para a escrita dos testes funcionais no Visual Studio e tem recursos orientados à sintaxe, como auto-completar e sugestões para a sequência da escrita dos testes funcionais. Na execução dos testes funcionais, esses recursos auxiliam o time de desenvolvimento apresentando os resultados como pendentes, falhos e aceitos.

A execução dos testes construídos, por sua vez, pode ser realizada utilizando tanto o *framework* NUnit³ quanto o VS Unit Test⁴, padrão do Visual Studio. A utilização desses frameworks possibilita que o Story2CSharp realize testes funcionais integrados com ferramentas que realizam testes unitários.

Para mais detalhes, o Apêndice A apresenta o código completo escrito em DOT e o grafo direcionado gerado a partir do código DOT. Na sequência, o Apêndice B mostra o código completo do *template* T4. Por fim, o Apêndice C mostra o código na linguagem C# gerado.

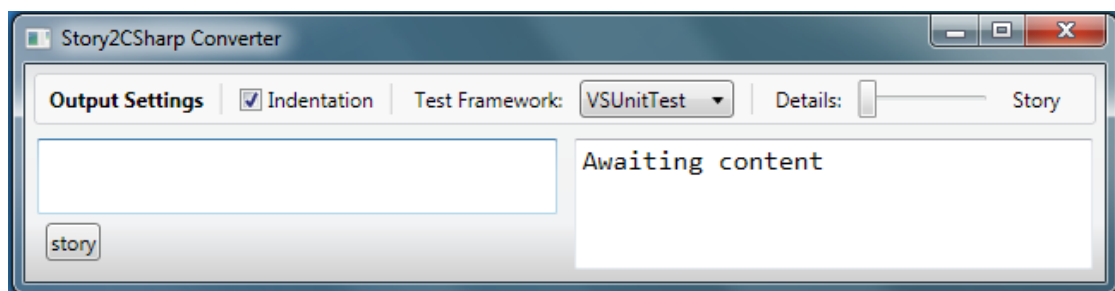
4.2 Conversor de LET para a Linguagem C#

Outro recurso desenvolvido para apoiar o processo é o conversor de Histórias do Usuário, descritas em LET, para a linguagem C#, chamado Story2CSharp Converter. Operacionalmente, o conversor tem uma interface com duas telas conforme mostra a Figura 4.7.

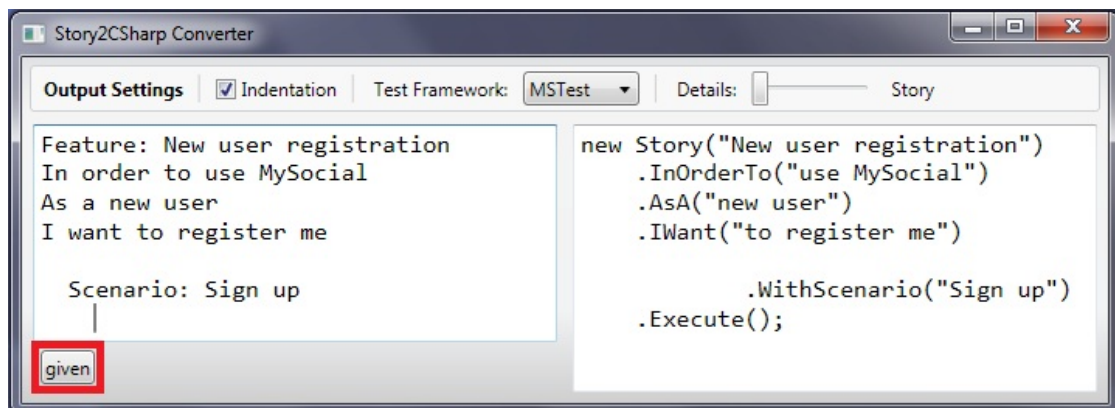
À esquerda tem-se a História do Usuário escrita na LET e à direita tem-se a história convertida para um teste funcional escrito em C#. Na parte inferior da interface (à esquerda) têm-se botões de interações que dão sequência na escrita da História do Usuário. A tela à direita serve

³<http://www.nunit.org/>

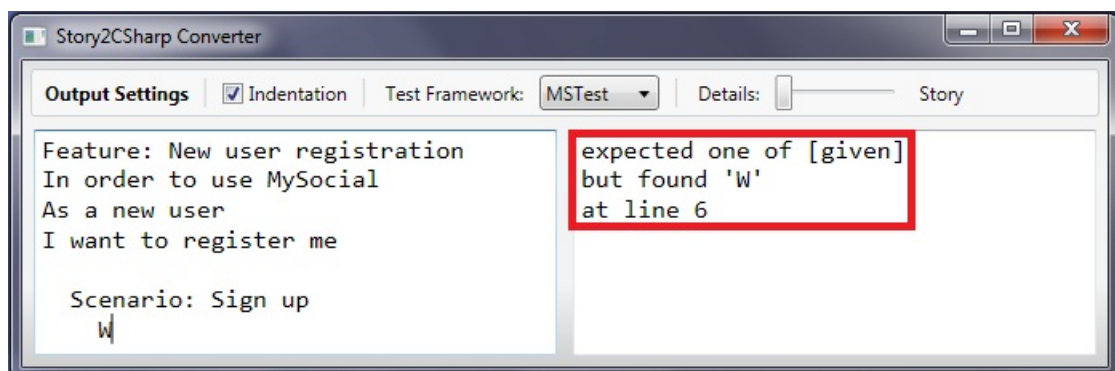
⁴<http://msdn.microsoft.com/en-us/library/ms243147.aspx>



(a) Tela inicial do conversor.



(b) Botões de interação de acordo com a necessidade.



(c) Mensagem de erro em tempo de execução.

Figura 4.7: Descrições na LET convertidas para Teste Funcional no C#.

também para mostrar erros na conversão, orientando a sintaxe da escrita da história. O desenvolvedor pode copiar e colar uma História do Usuário escrita na LET, criada anteriormente, na tela da esquerda do conversor ou escrevê-la diretamente no conversor. A medida que a história é escrita, a ferramenta realiza a conversão em tempo de execução.

A ferramenta de conversão possui algumas opções, destacadas na Figura 4.8. As opções são: (1) *Indentation*. Se estiver marcada, o teste gerado ficará indentado; (2) *Test Framework*. O desenvolvedor pode escolher entre dois *frameworks* para executar os testes, sendo eles o NUnit e o Visual Studio (VS) Unit Testing; e (3) *Details*. O desenvolvedor escolhe dentre quatro níveis de detalhamento para a conversão da história para o teste funcional.

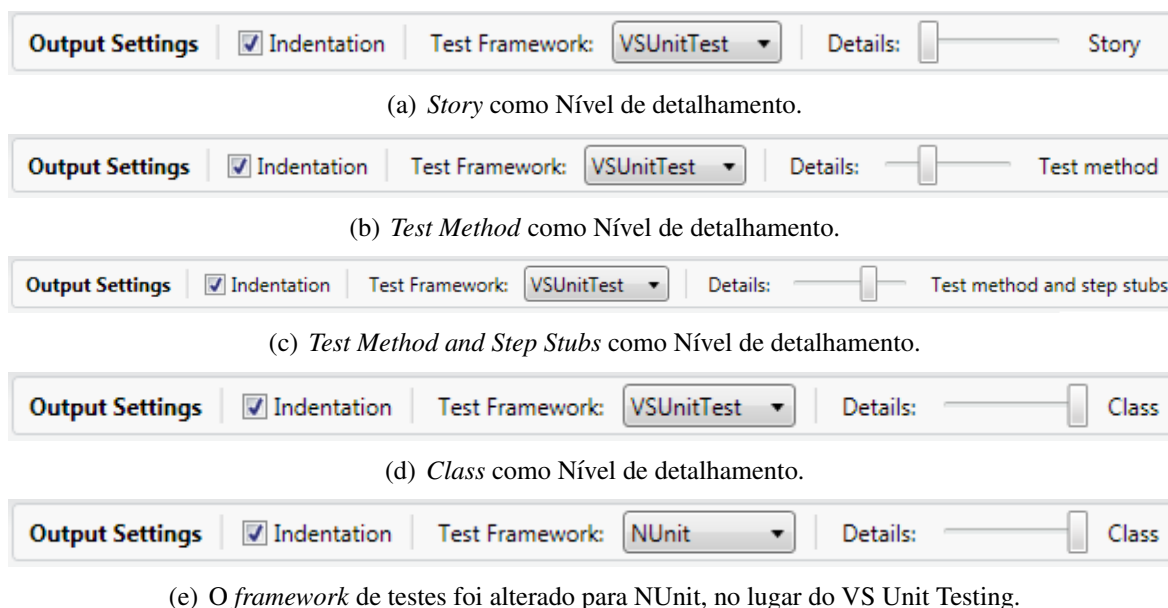


Figura 4.8: Opções do Story2CSharp Converter.

No conversor, a diferença entre escolher o *framework* NUnit ou VSUnitTest está no nome do classe de teste (*TestFixture* para o NUnit e *TestClass* para o VSUnitTest), o nome do método testável (*Test* para o NUnit e *TestMethod* para o VSUnitTest) e o nome da biblioteca antes da declaração da classe (*NUnit.Framework* ou *Microsoft.VisualStudio.TestTools.UnitTesting*), além de diferenças técnicas (como compatibilidade, desempenho, escolhas de projeto, etc.) que não são analisadas neste trabalho.

Os quatro níveis de detalhamento do código de testes gerado tem as seguintes diferenças: (1) *Story*, uma conversão simples da LET para o seu equivalente em código C#; (2) *Test method*, o código gerado faz parte de um método de teste; *Test methods and step stubs*, os passos (*Given*, *When* e *Then*) têm seus métodos criados de forma a lançar uma exceção para avisar que ainda eles não foram implementados e estão pendentes; e *Class*, onde o código parcialmente completo é gerado, com nome da classe e as bibliotecas requeridas para sua execução.

Uma vez que a história tenha sido convertida, pode-se dar continuidade na implementação da lógica da regra de negócio, copiando os testes criados pela conversão e executando-os dentro do Visual Studio.

4.3 Utilização das Ferramentas

A utilização das ferramentas segue o que foi proposto em (PEREIRA; PRADO, 2012). Ela é dividido em duas etapas obrigatórias, com uma terceira etapa opcional no começo (antes das

duas obrigatórias). Essas três etapas são: Especificação dos Testes (opcional), Construção de Testes Funcionais e Implementação Dirigida a Testes.

A forma de utilizar as ferramentas baseia-se nos conceitos de TDD de “criar um teste, verificar sua falha, escrever o código para que o teste tenha sucesso” e os conceitos de desenvolver orientado pelo comportamento esperado do software. Para auxiliar no entendimento e na inter-relação dessas três etapas, a Figura 4.9 mostra, em um diagrama SADT (Structured Analysis and Design Technique) (ROSS, 1977), as entradas e as saídas de cada etapa, assim como seus respectivos mecanismos e controles. Nas subseções seguintes, tais etapas são apresentadas em detalhes.

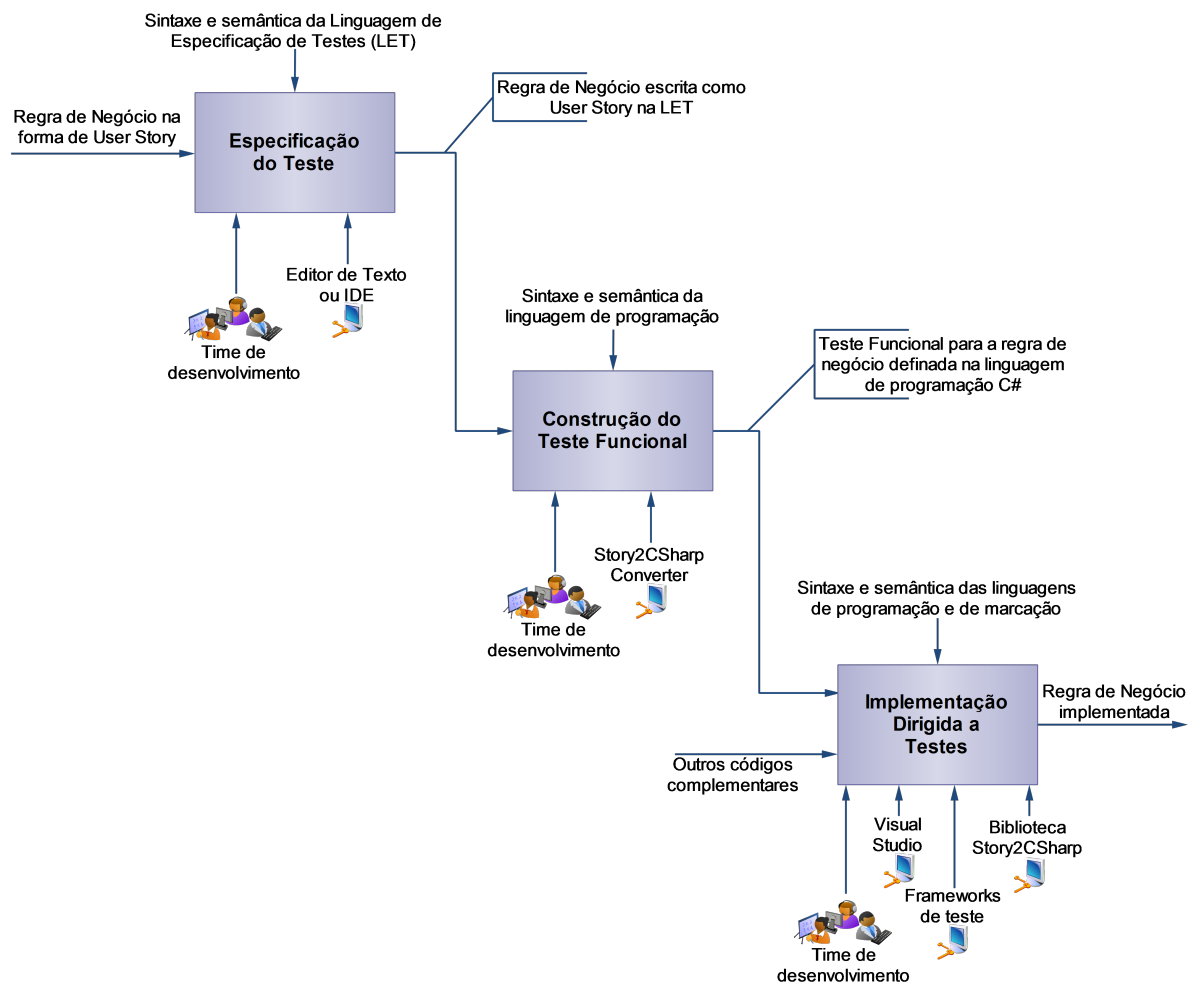


Figura 4.9: SADT das etapas para utilização das ferramentas.

4.3.1 Especificação de Testes

Nesta etapa opcional, uma regra de negócio, inicialmente descrita na forma de uma História do Usuário seguindo os padrões propostos por (NORTH, 2006), é adaptada para a LET. A Figura 4.10 mostra as diferenças entre as descrições da História do Usuário escrita de acordo com

(NORTH, 2006) e da História do Usuário escrita na LET. Essa adaptação é necessária para que a “nova” História do Usuário seja compreensível para a DLL e o conversor (as ferramentas Story2CSharp desenvolvidas). Caso o time de desenvolvimento escreva as Histórias do Usuário utilizando a LET desde o início, essa etapa foi realizada implicitamente.

```

Feature: Change email
In order to keep my profile updated
As a register user
I want to be able to change my email

Scenario: Change my email
Given I am signed in
And I click on my name in the header
And I follow "settings"
When I am on my account settings page
And I fill in "user_email" with "new_email@example.com"
And I press "change_email"
Then I should see "Email changed!!"

```

(a) Padrão proposto por (NORTH, 2006).

```

Feature: Change email
In order to keep my profile updated
As a register user
I want to be able to change my email

Scenario: Change my email
Given I am signed in
And I click on my name in the header
And I follow $settings
When I am on my account settings page
And I fill in $user_email with {new_email@example.com}
And I press $change_email
Then I should see {Email changed!!}

```

(b) Escrita na LET.

Figura 4.10: Diferenças entre as descrições das Histórias do Usuário.

4.3.2 Construção de Testes Funcionais

Nesta etapa, o time de desenvolvimento utiliza a História do Usuário escrita na LET para criar a regra de negócio instrumentada como teste funcional. Essa etapa é auxiliada pelo Conversor Story2CSharp, que converte da LET para a linguagem C#. Operacionalmente o desenvolvedor copia o conteúdo da História do Usuário descrita na LET para a tela da esquerda do Conversor Story2CSharp.

O desenvolvedor pode, também, escrever a História do Usuário na LET direto no conversor (na tela da esquerda). O conversor gera o código C#, mostrando-o na tela à direita. A Fi-

gura 4.11 mostra como o conversor é utilizado. E a Figura 4.12 mostra o resultado dessa etapa de conversão, com o maior nível de detalhamento possível (Class).

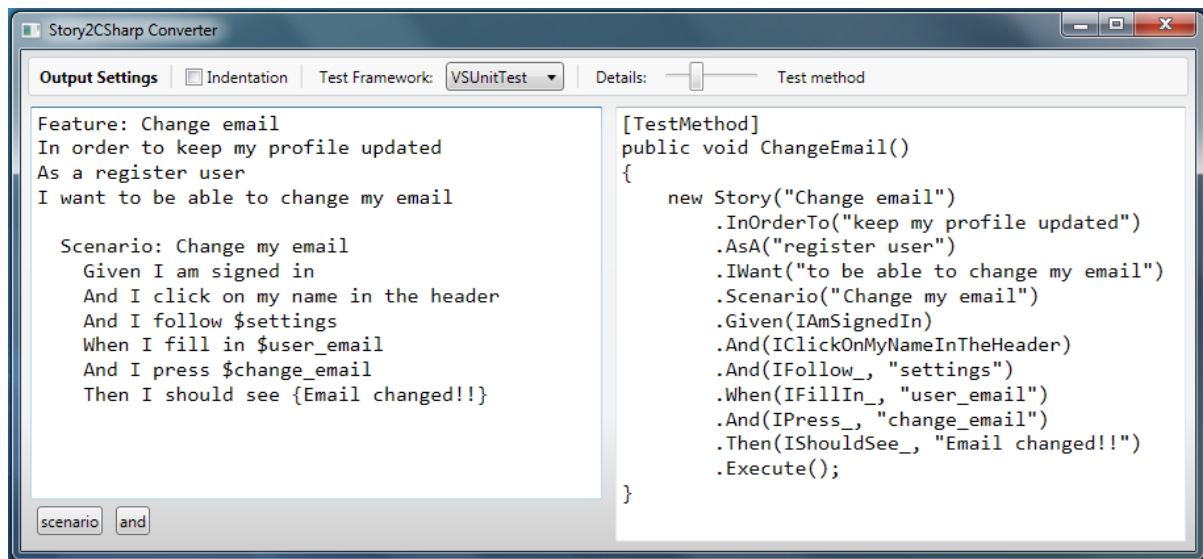


Figura 4.11: História do Usuário, da Figura 4.10 (b), sendo convertida.

```
using System;
using Story2CSharp;
using Microsoft.VisualStudio.TestTools.UnitTesting;

[TestClass]
public class Story2CSharpTestClass
{
    [TestMethod]
    public void ChangeEmail()
    {
        new Story("Change email")
            .InOrderTo("keep my profile updated")
            .AsA("register user")
            .IWant("to be able to change my email")
            .Scenario(": Change my email")
            .Given(IAMSignedIn)
            .And(IClickOnMyNameInTheHeader)
            .And(IFollow_, "settings")
            .When(IFillIn_, "user_email")
            .And(IPress_, "change_email")
            .Then(IShouldSee_, "Email changed!!")
            .Execute();
    }

    private void IAMSignedIn()
    {
        throw new NotImplementedException();
    }
    ...
}
```

Figura 4.12: História convertida com o maior nível de detalhamento possível.

Uma observação a respeito da Figura 4.12: o método na parte inferior da figura é criado para cada critério de aceitação da História do Usuário. Sendo assim, o método citado faz referência a condição inicial indicada pela palavra-chave “Given”. Os demais métodos (para os outros critérios) foram retirados da figura, para que a mesma não ficasse grande demais.

Dessa forma, o time de desenvolvimento constrói o teste funcional integrado com a História do Usuário. Note que a lógica de negócio não é gerada automaticamente. Apenas a estrutura do teste é gerada, ficando a escrita da lógica para a próxima etapa.

4.3.3 Implementação Dirigida a Testes

Nesta última etapa, o teste funcional gerado na etapa anterior deve orientar a implementação da lógica de negócio, seguindo os princípios do BDD (NORTH, 2006). Outros códigos podem ter sido criados, automaticamente ou não, para dar suporte a essa etapa, como por exemplo a criação do CRUD (Create, Read, Update, Delete) (MARTIN, 1983) e do banco de dados. No ambiente Visual Studio, a utilização é apoiada pelos *frameworks* de testes, NUnit ou VS Unit Test, e pela biblioteca Story2CSharp. O time de desenvolvimento inclui o código gerado pelo conversor no projeto criado no Visual Studio.

Uma vez construídos os testes funcionais, eles podem ser executados e ter seus resultados analisados pelo time de desenvolvimento. Se os testes forem executados da forma como foram convertidos (sem acréscimo de código), o resultado será inconclusivo, como mostra a Figura 4.13. Isso ocorre porque para cada critério de aceitação é criado um método com a seguinte linha de comando: “*throw new NotImplementedException();*”.

```
ChangeEmail : Inconclusive

Story is Change email
  In order to keep my profile updated
  As a register user
  I want to be able to change my email

  With scenario Change my email
    Given I am signed in => Pending !!
      And I click on my name in the header => Pending !!
      And I follow settings => Pending !!
    When I fill in user_email => Pending !!
      And I press change_email => Pending !!
    Then I should see Email changed!! => Pending !!
```

Figura 4.13: Resultado inconclusivo ao executar a História, da Figura 4.12, recém convertida.

Na sequência, o time de desenvolvimento deve escrever o código para testar a lógica de negócio da História do Usuário na estrutura gerada pela conversão. Assim, o time faz com que

os passos, de cada teste funcional, acusem falhas. A Figura 4.14 mostra um exemplo disso. Para corrigir as falhas, o time deve implementar a lógica de negócio, para que os respectivos passos sejam aprovados nos seus testes. A Figura 4.15 mostra uma História sendo aprovada. O resultado final é a lógica de negócio implementada, funcionalmente testada, conforme os requisitos especificados nas Histórias do Usuário.

```
ChangeEmail : Failed

Story is Change email
  In order to keep my profile updated
  As a register user
  I want to be able to change my email

  With scenario Change my email
    Given I am signed in                => Passed
      And I click on my name in the header => Passed
      And I follow settings              => Passed
    When I fill in user_email           => Failed:
      "Assert.Fail failed. Field does not exist [1]"
      And I press change_email          => Pending !!
    Then I should see Email changed!!   => Pending !!
```

Figura 4.14: Resultado apresentando uma falha.

```
ChangeEmail : Passed

Story is Change email
  In order to keep my profile updated
  As a register user
  I want to be able to change my email

  With scenario Change my email
    Given I am signed in                => Passed
      And I click on my name in the header => Passed
      And I follow settings              => Passed
    When I fill in user_email           => Passed
      And I press change_email          => Passed
    Then I should see Email changed!!   => Passed
```

Figura 4.15: Resultado mostrando que a História da Figura 4.12 foi aprovada.

4.4 Considerações Finais

As ferramentas desenvolvidas encapsulam as tarefas de transformar as Histórias do Usuário para testes funcionais, automatizando esse processo. Fornecem, também, funcionalidades para a execução das Histórias do Usuário (na forma de testes funcionais), apresentado os resultados dentro do ambiente Visual Studio. Para isso, é criada uma adaptação da descrição textual das Histórias do Usuário, denominada LET. Essa adaptação é estruturada com outras funções dentro

de uma biblioteca (DLL). A DLL é utilizada dentro do Visual Studio, para que o mesmo entenda a sintaxe das histórias escritas na LET, além de ser utilizada como base para a ferramenta de conversão de História do Usuário para testes funcionais na linguagem C#.

Inicialmente desenvolvidas para dar suporte computacional ao processo RAMBUS, é viável supor que a utilização das ferramentas – que visa integrar os testes funcionais com o processo de desenvolvimento – pode ser independente do processo RAMBUS. Sendo assim, essa utilização pode ser estendida para qualquer processo de desenvolvimento, independente do processo ser ágil ou não. É necessário apenas que o time de desenvolvimento faça uso dos conceitos e técnicas do Desenvolvimento Dirigido a Testes (TDD) (BECK, 2003).

Para ilustrar isso, a Figura 4.16 mostra como seria o processo Scrum utilizando as três etapas da utilização das ferramentas desenvolvidas, descritas neste Capítulo. Nesse caso, as etapas podem ser tratadas como uma “abordagem para integração de testes funcionais” à um determinado processo. Como pode ser visto, as três etapas são acopladas ao Sprint do Scrum. Essa “abordagem” será utilizada para testar cada História do Usuário presente em cada Sprint. O mesmo princípio pode ser aplicado a outros processos de desenvolvimento, pois uma vez que os desenvolvedores utilizem TDD, fica mais prático criar e/ou transcrever uma História do Usuário nos moldes da LET.

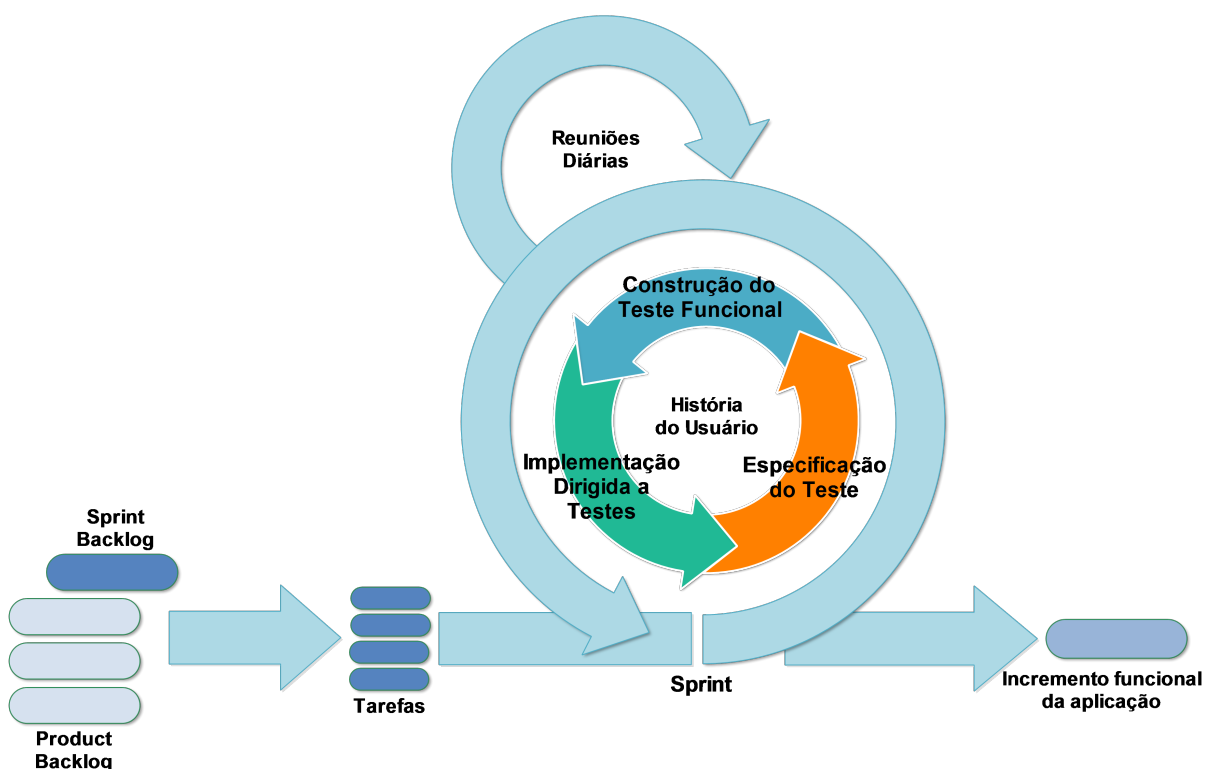


Figura 4.16: Utilizando a “Abordagem para Integração de Testes Funcionais” com o processo Scrum.

No próximo capítulo é apresentado o processo ágil RAMBUS que, de acordo com o tema proposto, engloba os principais conceitos citados no Capítulo 2 e pode utilizar as ferramentas apresentadas neste Capítulo, ou qualquer outra ferramenta que tenha funcionalidades parecidas, como por exemplo, a ferramenta Cucumber⁵ para Ruby, uma das mais conhecidas no mercado. Entretanto, no caso de utilizar o Cucumber, a transcrição das Histórias do Usuário para testes funcionais é feita de forma puramente manual. Essa transcrição é feita escrevendo Expressões Regulares (ER) para cada critério de aceitação. A Figura 4.17 mostra um exemplo de uma ER escrita ao se utilizar o Cucumber.

```
Then /the result should be (.*) on the screen/ do
  ...
end
```

Figura 4.17: Uma Expressão Regular no Cucumber.

⁵<http://cukes.info/>

Capítulo 5

DESENVOLVIMENTO ÁGIL PARA GROUPWARE NA WEB 2.0

Um processo é definido como um roteiro com passos previsíveis e diretrizes relacionadas ao desenvolvimento de sistemas computacionais. Compreende uma abordagem que é utilizada quando a aplicação é elaborada, assim como as tecnologias que o constituem, tais como métodos, técnicas e ferramentas automatizadas. Seu objetivo é apoiar a criação de produtos com alta qualidade dentro dos prazos e orçamento previstos (PRESSMAN, 2010; PFLEEGER; ATLEE, 2010).

Neste capítulo é apresentado o processo para *Desenvolvimento Ágil para Groupware na Web 2.0*, desenvolvido com este projeto de pesquisa. Tendo em vista o domínio das aplicações sociais na Web e o dinamismo inerente ao mesmo, no processo proposto são utilizados os conceitos do Scrum integrado com o uso de Desenvolvimento Orientado a Testes (TDD), com foco em testes funcionais sendo utilizados a nível de aceitação (BDD). Dessa forma, ao utilizar o processo proposto, os desenvolvedores têm em mãos um processo gerencial para lidar com o projeto e artefatos que os induzem a criar testes e utilizá-los para orientar a implementação da aplicação Web.

Este capítulo está organizado da seguinte forma: a Seção 5.1 apresenta uma visão geral da proposta de processo ágil desenvolvido no Mestrado. O processo está dividido em Disciplinas do Processo (Seção 5.2) e Fases do Processo (Seção 5.3). Por fim, a Seção 5.4 encerra o capítulo apresentando algumas considerações finais.

5.1 Visão Geral do Processo

RAMBUS (Rambus Agile Methodology Based on User Stories) (PEREIRA; PRADO, 2011) é um processo iterativo e incremental para o desenvolvimento de aplicações Web composto por quatro fases - Requisitos, Planejamento, Execução e Encerramento - e três disciplinas - Comunicação, Modelagem e Construção. Com o auxílio de Mockups (BROWN, 2011), o processo utiliza Histórias do Usuário para obter os requisitos e orientar o processo de desenvolvimento. Ao final de cada iteração (Sprint), são desenvolvidas versões funcionais da aplicação, baseadas nas Histórias do Usuário utilizadas, que no decorrer do tempo incluirão todas as funcionalidades solicitadas. Para construir a aplicação Web, o processo é executado em ciclos, onde as disciplinas (Comunicação, Modelagem e Construção) se repetem para gerar versões cada vez mais refinadas da aplicação. Na Figura 5.1 é apresentado o ciclo de vida do processo RAMBUS, no formato conhecido como *hump chart*. Na vertical encontram-se as disciplinas do processo proposto neste trabalho e na horizontal, representados por elementos do Scrum, estão às fases de Requisitos, Planejamento, Execução e Encerramento.

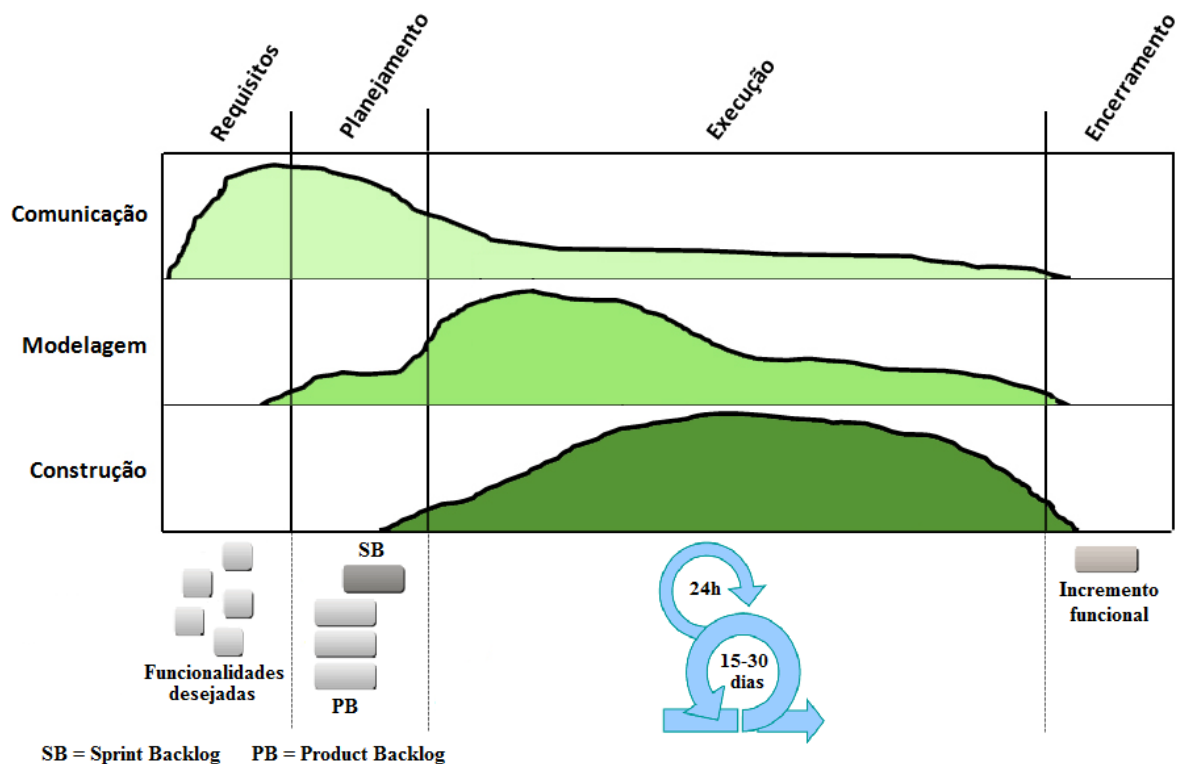


Figura 5.1: Ciclo de vida do processo RAMBUS.

Observa-se na figura que as disciplinas têm seus picos de uso em determinadas fases do processo. Isso significa, por exemplo, que enquanto a especificação dos requisitos é realizada, o time está na fase de Requisitos, com foco maior na disciplina Comunicação. Quando é cri-

ado o Product Backlog e é selecionado o que será feito na iteração, o time está na fase de Planejamento, ainda focado em Comunicação. A fase de Execução representa um ciclo de implementação e testes, como o Sprint do Scrum. Assim, primeiramente o foco é maior na disciplina Modelagem, para melhor entender como serão realizadas as tarefas presentes em cada ciclo. À medida que a fase de Execução avança, o foco do time muda da disciplina Modelagem para a disciplina Construção, onde os testes são criados e o código escrito. Para finalizar, a fase de Encerramento engloba a entrega do incremento funcional criado na iteração e a análise da iteração atual para melhoras nas iterações futuras.

Conforme mostra a Figura 5.1, uma disciplina não termina após o seu pico de uso e mantém-se presente até o fim da iteração. Por exemplo, apesar de a disciplina de Comunicação ser a de maior foco nas fases de Requisitos e Planejamento, ela continua presente na fase de Execução, para ilustrar que a interação entre as pessoas envolvidas no projeto ainda é (muito) necessária.

O processo também utiliza as teorias de Desenvolvimento Dirigido por Comportamento (BDD), que por meio das histórias testam o comportamento do usuário (NORTH, 2006), conceitos consagrados do Scrum, entre eles Product Backlog e Sprint (SCHWABER, 2004) e as ideias do TDD de criar testes antes de escrever o código da aplicação e usar isso para orientar o desenvolvimento, além de sugerir o uso de algumas boas práticas, como programação pareada e integração contínua.

Para melhorar a compreensão, na Figura 5.2 é mostrado, em um diagrama SADT, as disciplinas do processo RAMBUS. A figura destaca as principais entradas, saídas, controles e mecanismos de execução de cada disciplina.

Ao longo do desenvolvimento novas funcionalidades podem ser solicitadas, por exemplo quando o usuário interagir com a aplicação. Essas solicitações gerarão mudanças no projeto e o time deve manter o sistema para evoluir com o mínimo de dificuldades possível. Além disso, no decorrer do processo é válido testar ideias e obter o feedback do usuário, pois a descoberta mais cedo de problemas pode poupar tempo e reduzir os gastos para repará-los (PERRY, 2006). Portanto, o processo deve tratar essas mudanças ao longo do projeto.

No decorrer da próxima seção, é utilizada a LET (apresentada no Capítulo 4) como padrão para a escrita da descrição textual das Histórias do Usuário. Isso faz com que as ferramentas Story2CSharp (a biblioteca e o conversor) se integrem com o processo RAMBUS. Tal integração não é obrigatória. Por exemplo, o formalismo proposto por (NORTH, 2006) pode ser utilizado nas Histórias e outros *frameworks* podem ser utilizados para executá-las – neste caso, as necessidades de cada *framework* devem ser cumpridas –, como o Cucumber.

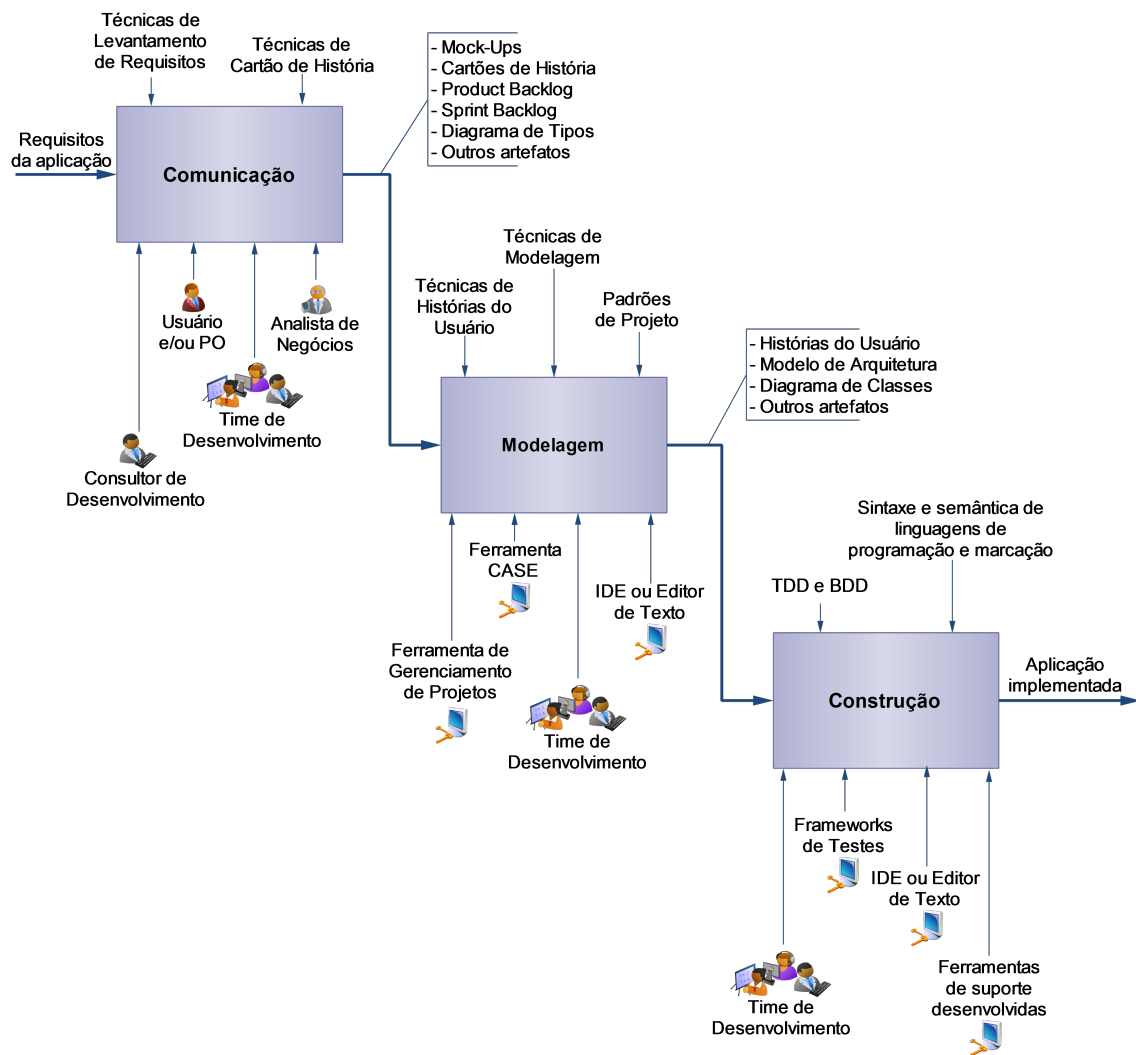


Figura 5.2: Processo RAMBUS e suas disciplinas.

Nas próximas seções são apresentadas as disciplinas do processo, com seus respectivos artefatos, e posteriormente as fases do processo.

5.2 Disciplinas do Processo

Nesta seção são descritas as disciplinas do processo, mostrando em detalhes seus respectivos artefatos, além de como criá-los e utilizá-los. Para facilitar o entendimento, utiliza-se como exemplo, o desenvolvimento de uma aplicação do domínio de rede social, denominada MySocial. Essa aplicação integra recursos do Facebook¹ e do Google+², e tem como objetivo ser um site de relacionamentos com foco em construir e refletir redes sociais e relações sociais entre pessoas. Essas pessoas compartilham interesses e/ou atividades de relacionamentos. Neste site,

¹<https://www.facebook.com/>

²<https://plus.google.com/>

as pessoas, após efetuarem seus cadastros, podem publicar posts e fotos, procurar por amigos e comentar suas publicações e as de seus amigos. Portanto, todas as figuras mostram exemplos da especificação de um rede social para ilustrar a descrição dos artefatos.

5.2.1 Comunicação

Nesta disciplina, o foco está na interação entre as pessoas envolvidas no processo, especialmente para a especificação de requisitos e a elaboração do que deverá ser feito em cada iteração (Sprint).

Para auxiliar na especificação dos requisitos, o processo faz uso dos Cartões de História e de Mock-ups. Não existe uma regra sobre qual dos dois artefatos deve ser criado primeiro, porque o processo pressupõe que eles se complementam. Vale ressaltar que no decorrer do processo, o termo Cartão de História expressa uma História do Usuário no formato de cartão (seja em papel ou eletrônico). Ao final de cada interação com o usuário, os Cartões de História e os Mock-ups devem ter sido criados, para que juntos possam exibir um quadro geral das funcionalidades da aplicação que foram especificadas pelo usuário.

O modelo de Cartão de História utilizado no processo RAMBUS é baseado no modelo INSERT, proposto por (PATEL; RAMACHANDRAN, 2009). Esse modelo possui vantagens em relação aos modelos propostos por (COHN, 2004) e por (BECK; ANDRES, 2005) para o XP, como a definição de uma estrutura padrão para o cartão e a maior facilidade para alterar os cartões. A Tabela 5.1 mostra algumas dessas vantagens.

Tabela 5.1: Diferenças entre modelos de Cartões de História, adaptado de (PATEL; RAMACHANDRAN, 2009).

Funções e Orientações	Mike Cohn	Kent Beck	INSERT
Cartões devem ser pequenos e completos	X	X	X
Define um padrão de estrutura para os cartões	-	-	X
Define cartões com simplicidade	X	X	X
Define cartões consistentes	-	-	X
Os cartões devem apresentar requisitos de negócio	X	X	X
Faz os cartões serem fáceis de alterar	-	-	X
Valioso para o cliente	X	X	X
Os cartões devem ser estimáveis	X	X	X
Cartões suportam requisitos não-funcionais	-	-	X

Para fins de apresentação, o Cartão de História é o primeiro artefato com explicações sobre a sua utilização. A Figura 5.3 mostra um exemplo de Cartão de História solicitando a funcionalidade de cadastrar novos usuários para a rede social do exemplo, denominada MySocial.

<i>Story Card No.: 2</i>		<i>Prioridade: Alta</i>	
<i>Nome do projeto: MySocial</i>		<i>Data: 20/09/2011</i>	
<i>Funcionalidade: Cadastro de novo usuário</i>		<i>Estimativa: 2</i>	
<i>Descrição:</i> <i>Como um novo usuário</i> <i>Eu quero me cadastrar</i> <i>Para utilizar o MySocial</i>		<i>Testes para se considerar:</i> <i>1 – Nenhum campo no cadastro</i> <i>pode ser nulo</i> <i>2 – O usuário passa pelo assistente</i> <i>3 – O usuário não passa pelo</i> <i>assistente</i>	
<i>Anotações:</i> <i>Necessita de username, password e email</i> <i>Depois de criar a conta, o usuário pode</i> <i>utilizar o assistente de configuração ou não</i> <i>O assistente de configuração tem os campos</i> <i>Primeiro Nome, Último Nome e Tags</i>			
<i>Requisitos não funcionais:</i> <i>Não há requisitos não funcionais até o momento</i>			

Figura 5.3: Cartão de História utilizado para a rede social MySocial.

Como pode ser visto na figura, o Cartão de História é simples e objetivo. Ele permite que o usuário tenha uma maior compreensão do processo e assim colabore da melhor forma para identificar e elicitare os requisitos. No cartão consta, por exemplo, a sua prioridade para o usuário e uma estimativa inicial sobre a dificuldade (relativa a cada empresa ou time de desenvolvimento) para a realização da funcionalidade solicitada. Porém, a coleta de dados apenas dessa forma não é o suficiente. Isso se deve ao fato de que essas informações estão em um nível mais alto de abstração.

O usuário informa o que espera que a aplicação Web faça e como quer que a mesma funcione. Pode até ser o suficiente para algumas aplicações, mas visando melhorar o entendimento dos requisitos, é viável fazer uso de Mock-ups para ter esboços das páginas da aplicação Web que fazem referência aos Cartões de História elaborados. Os Mock-ups são uma forma de projetar as interfaces do usuário em papel ou através de imagens. Um Mock-up define o *layout* de elementos fundamentais da interface, incluindo sua navegação, além de auxiliar no entendimento dos requisitos da aplicação. A Figura 5.4 mostra como exemplo dois Mockups para a rede social MySocial, relacionados com o Cartão de História apresentado na Figura 5.3.

Nada impede que outros artefatos sejam criados para expressar os requisitos de forma mais clara. Quanto mais prático o artefato for, sem perder conteúdo, maior é a probabilidade do usuário interagir. Um exemplo é o Diagrama de Tipos (D'SOUZA; WILLS, 1999), que é extremamente simples, fazendo uso apenas das classes (com os possíveis nomes dos atributos) e seus

(a) Mockup para a página Sign Up.

(b) Mockup para a página Getting Started.

Figura 5.4: Mock-ups do Cartão de História da Figura 5.3.

relacionamentos, como mostra a Figura 5.5. O usuário não precisa saber o que é uma classe, basta entender o que o desenho representa e interagir. O Diagrama de Tipos pode ser utilizado também para especificar as informações em um banco de dados legado que será usado pela aplicação.

Problemas técnicos, como por exemplo a arquitetura da aplicação, não são discutidos nos Cartões de História, mas podem ser anotados para uma análise posterior pelo time de desenvolvimento. Se esses problemas possuírem um relacionamento direto com uma ou mais funcionalidade da aplicação, é importante anotar com quais cartões eles estão relacionados.

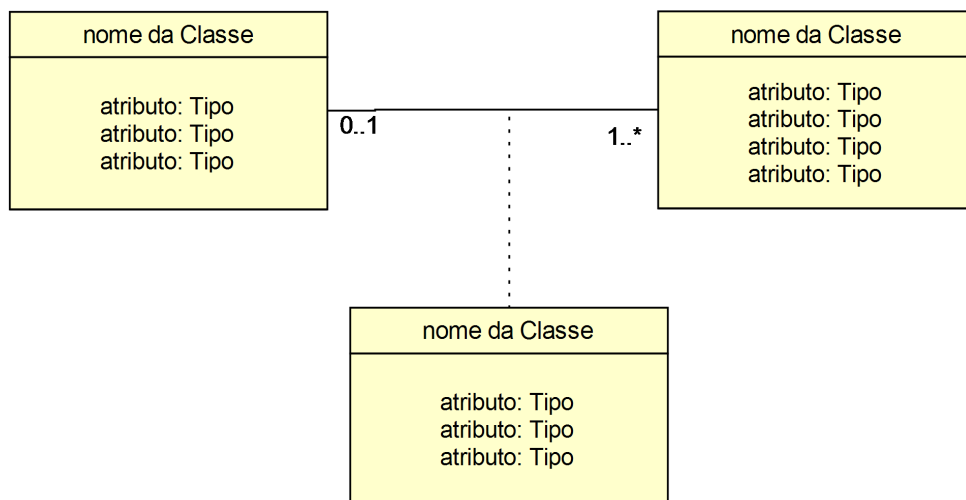


Figura 5.5: Exemplo de um Diagrama de Tipos.

Usando os Cartões de História e os Mock-ups, um desenvolvedor do time, no papel de Product Owner (PO), deve criar uma lista de funcionalidades desejadas e priorizá-las de acordo com as necessidades exigidas pelo usuário. Esta lista é conhecida como Product Backlog no Scrum. Assim, durante a reunião de planejamento da iteração, o PO apresenta ao time de desenvolvimento as funcionalidades priorizadas e suas descrições. Nessa reunião, são especificados quais itens da lista serão feitos na iteração atual, criando uma lista de tarefas para a iteração. Essa lista é similar ao Sprint Backlog, que é feito na reunião antes do Sprint no Scrum.

Para melhor gerenciar os Backlogs, uma boa alternativa é usar uma ferramenta ágil de gerenciamento de projetos como Pivotal Tracker³, IceScrum⁴ e TargetProcess⁵. Outra recomendação é imprimir os itens da iteração atual e colocá-los em um lugar de fácil visualização para todos os envolvidos, no estilo sugerido pelo KanBan (ANDERSON, 2010).

A cada iteração da disciplina de Comunicação têm-se diferentes artefatos, destacando-se: os Cartões de História e os Mock-ups (ambos são essenciais para orientar a construção dos artefatos da próxima disciplina); o Product Backlog e o Sprint Backlog para controlar as funcionalidades desejadas e definir quais destas funcionalidades são realizadas em cada iteração.

5.2.2 Modelagem

Nesta disciplina, o time de desenvolvimento utiliza os Cartões de História, os Mock-ups e outros artefatos criados na Comunicação (para auxiliar a especificação de requisitos), para analisar e projetar a aplicação Web. Nessa disciplina são especificados novos artefatos e ainda

³<http://www.pivotaltracker.com/>

⁴<http://www.icescrum.org/>

⁵<http://www.targetprocess.com/product/scrum/>

refinados os produzidos na Comunicação.

Por exemplo, o time de desenvolvimento deve preparar um Diagrama de Classes, ou refinar um Diagrama de Tipos previamente criado, que irá fornecer assistência para análise dos dados da aplicação. O Diagrama de Classes, possivelmente simples no começo, pode ser refinado a cada iteração. O time de desenvolvimento deve verificar, por exemplo, a necessidade de criar uma classe abstrata ou uma interface para manipular os dados. É de responsabilidade do time analisar e definir se vai utilizar o Diagrama de Classes para criar o banco de dados (se ele não for legado) ou se vai criar um Modelo de Banco de Dados. O objetivo é criar o banco de dados de acordo com as necessidades da aplicação, podendo o mesmo ser estendido gradualmente.

O próximo passo é especificar as Histórias do Usuário com base nos artefatos que têm sido utilizados até agora. Essas Histórias devem representar exatamente o que foi expresso pelo usuário nos Cartões de História e nos Mock-ups. Na Figura 5.6 é mostrada uma transcrição de uma História do Usuário escrita na LET, baseada nos Mock-ups da Figura 5.4 e no Cartão de História da Figura 5.3, usando a LET. Utilizar uma IDE (Integrated Development Environment), ou um editor de textos, facilita a especificação de uma História do Usuário.

```
Feature: New user registration  
In order to use MySocial  
As a new user  
I want to register me  
  
Scenario: Sign up  
Given I go to the new user registration page  
When I fill in $username with {testuser}  
And I fill in $email with {test@test.test}  
And I fill in $user_password with {secretpwd}  
And I fill in $password_confirmation with {secretpwd}  
And I press $create_my_account  
Then I should be on the getting started page  
And I should see {Welcome}  
And I should see {Fill out your profile}  
And I should see {Connect with cool people}  
And I should see $Finished
```

Figura 5.6: História do Usuário escrita na LET.

No exemplo da Figura 5.6, a condição inicial é navegar para a página de cadastro de usuário. As operações são preencher os campos e pressionar o botão para criar a conta. Os resultados esperados incluem o redirecionamento para uma nova página com uma mensagem de boas-vindas e os campos para completar o processo de cadastro. A História do Usuário é considerada

finalizada quando todos os critérios de aceitação de cada um dos seus cenários são atendidos.

A História do Usuário transcrita neste formato representa uma descrição executável das especificações da aplicação, em um nível de abstração que facilita o entendimento do usuário. O usuário pode verificar as Histórias do Usuário escritas dessa forma e aprová-las ou não.

É importante destacar que uma forma de melhorar o uso das Histórias do Usuário é criá-las apenas para as regras de negócio e não para funcionalidades simples, como um CRUD (MARTIN, 1983). Isso se deve ao fato de que uma vez que o time de desenvolvimento tenha definido um Diagrama de Classes, o CRUD pode ser gerado automaticamente a partir de várias ferramentas CASE (Computer Aided Software Engineering). Portanto, há a possibilidade de poupar tempo de desenvolvimento, não só na implementação, mas também na manutenção deste código automático.

Analisando as Histórias do Usuário e os outros artefatos, o time deve procurar o que pode ser reutilizado de projetos anteriores e o que do projeto atual pode gerar um artefato reutilizável para futuras aplicações deste domínio. Além disso, é válido identificar onde um padrão de projeto pode ser incluído para facilitar o entendimento, a criação e a futura manutenção da aplicação. Um exemplo de padrão é o uso do MVC (Model-View-Controller) (LEFF; RAYFIELD, 2001), para estruturar as camadas do modelo, controle e visão.

Nesta disciplina, o exemplo utiliza a DLL que possibilita utilizar a LET dentro do ambiente Visual Studio, para gerar mensagens de erro e avisos auto explicativos. Caso o projeto não utilize a linguagem C# (como o exemplo que ilustra este capítulo), outros *frameworks* podem ser utilizados para auxiliar na especificação das Histórias do Usuário que orientam a implementação da aplicação. Nesse caso, o time de desenvolvimento deve escolher qual *framework* usar para executar as Histórias do Usuário criadas nesta disciplina. Algumas opções são: easyb⁶ para Groovy, Jasmine⁷ para JavaScript, Cucumber⁸ e RSpec⁹ para Ruby, e JBehave¹⁰ para Java.

Também deve ser definida a plataforma de *hardware* e *software* adotada para o projeto. Por exemplo a adoção de C# como a linguagem de programação, que possibilita também a integração do padrão MVC na plataforma ASP.NET MVC 3¹¹. A Figura 5.7 mostra detalhes dessa arquitetura. Outra decisão importante é a do SGBD (Sistema de Gerenciamento de Banco de Dados) que será utilizado, no caso de aplicação com persistência.

⁶<http://www.easyb.org/>

⁷<http://pivotal.github.com/jasmine/>

⁸<http://cukes.info/>

⁹<http://relishapp.com/rspec>

¹⁰<http://jbehave.org/>

¹¹<http://www.asp.net/mvc/mvc3>

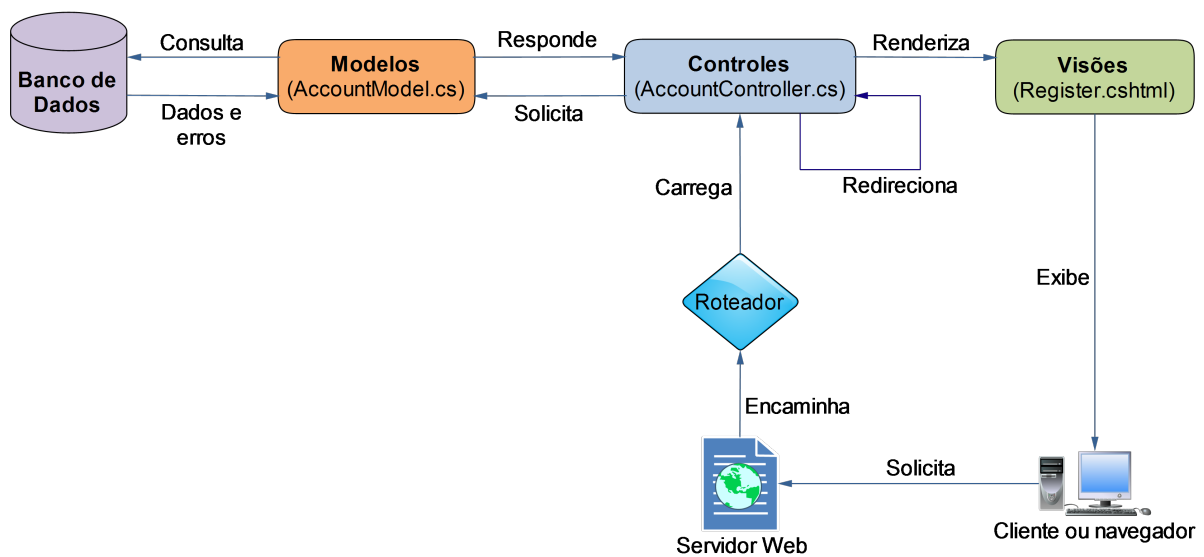


Figura 5.7: Arquitetura MVC usando C# e ASP.NET MVC 3.

Conforme a Figura 5.2, os principais artefatos criados são: as Histórias do Usuário em uma forma executável, o Diagrama de Classes e um Modelo de Arquitetura. Entretanto, nada impede que outros artefatos sejam criados para que o time de desenvolvimento tenha um melhor entendimento do problema.

5.2.3 Construção

Esta disciplina compreende a implementação e os testes da aplicação. Práticas como Reuniões Diárias, Programação Pareada, Integração Contínua e Padrões de Código são úteis nesta disciplina, assim como qualquer outra prática que ajude o time de desenvolvimento a implementar a aplicação da melhor forma possível.

Tendo como base os artefatos da disciplina anterior, o time de desenvolvimento deve criar a infraestrutura necessária para dar suporte a iteração em cada Sprint. Provavelmente, a primeira iteração terá mais atividade de suporte, como criar um banco de dados baseado no Diagrama de Classe ou no Modelo de Banco de Dados. É recomendado utilizar esse momento para gerar todos os possíveis códigos automáticos (como o CRUD). Só então escrever e testar o código referente as regras de negócio.

O time de desenvolvimento deve então converter as Histórias do Usuário da lista de atividades da iteração do Sprint (Sprint Backlog) para obter a estrutura dos testes funcionais na linguagem alvo da implementação, no caso o C#. Essa conversão é realizada pelo conversor Story2CSharp. A Figura 5.8 mostra o resultado dessa conversão.

```
using System;
using Story2CSharp;
using Microsoft.VisualStudio.TestTools.UnitTesting;

[TestClass]
public class Story2CSharpTestClass {
    [TestMethod]
    public void NewUserRegistration() {
        new Story("New user registration")
            .InOrderTo("use MySocial")
            .AsA("new user")
            .IWant("to register me")
            .WithScenario("Sign up")
            .Given(IGoToTheNewUserRegistrationPage)
            .When(IFillIn_With_,"username","testuser")
            .And(IFillIn_With_,"email","test@test.test")
            .And(IFillIn_With_,"user_password","secret")
            .And(IFillIn_With_,"password_confirmation","secret")
            .And(IPress_,"Create_my_account")
            .Then(IShouldBeOnTheGettingStartedPage)
            .And(IShouldSee_,"Welcome")
            .And(IShouldSee_,"Fill out your profile")
            .And(IShouldSee_,"Connect with cool people")
            .And(IShouldSee_,"Finished")
            .Execute();
    }

    private void IGoToTheNewUserRegistrationPage()
    { throw new NotImplementedException(); }

    private void IFillIn_With_(string arg1, string arg2)
    { throw new NotImplementedException(); }

    private void IPress_(string arg1)
    { throw new NotImplementedException(); }

    private void IShouldBeOnTheGettingStartedPage()
    { throw new NotImplementedException(); }

    private void IShouldSee_(string arg1)
    { throw new NotImplementedException(); }
}
```

Figura 5.8: História do Usuário, da Figura 5.6, em C# como teste funcional.

Conforme mostra a Figura 5.8, foi gerada uma classe de testes com o nome indicando o uso da DLL Story2CSharp e cada história se transforma em um método testável. É importante notar que apesar de convertida para C#, a história é legível, visando a manter um nível de abstração próximo ao oferecido pela LET.

Na sequência é criada uma variável *Story*, que conterá as informações da História do Usuário. O conteúdo de cada cenário é dividido em métodos que são declarados externamente ao método testável. O nome de cada método é precedido de um underscore (_) onde os seus respectivos atributos se encaixam no nome, para facilitar sua leitura. Esses métodos são criados com o objetivo de lançar uma exceção para indicar que ainda não foram implementados. No final do método testável, uma função interna é chamada para executar a história.

Uma vez que os testes funcionais foram gerados, o time de desenvolvimento pode executá-los e analisar seus resultados. Como o código acabou de ser gerado, o resultado mais provável é que o teste funcional seja “inconclusivo”, uma vez que os métodos de cada cenário são apenas estruturas, que lançam exceções avisando que cada método está pendente, e as regras de negócio ainda não foram implementadas. A Figura 5.9 mostra esse resultado.

NewUserRegistration : Inconclusive

```
Story is New user registration
  In order to use MySocial
  As a new user
  I want to register me
Scenario: Sign up
  Given I go to the new user registration page      => Pending !!
  When I fill in Username with testuser            => Pending !!
    And I fill in Email with test@test.test        => Pending !!
    And I fill in user_password with secret        => Pending !!
    And I fill in Password confirmation with secret => Pending !!
    And I press Create my account                  => Pending !!
  Then I should be on the getting started page     => Pending !!
    And I should see Welcome                       => Pending !!
    And I should see Fill out your profile         => Pending !!
    And I should see Connect with cool people     => Pending !!
    And I should see Finished                     => Pending !!
```

Figura 5.9: Resultado mostrando que o teste funcional foi inconclusivo, devido aos métodos pendentes.

Prosseguindo, o time de desenvolvimento escreve o código para testar a lógica de negócio da História do Usuário, para que seus passos acusem falhas. Para corrigir as falhas, implementa-se a lógica de negócio, para que os passos sejam aprovados nos testes. A Figura 5.10 mostra o caso de falha do teste funcional e a Figura 5.11 mostra o teste funcional sendo aprovado.

```

NewUserRegistration : Failed

Story is New user registration
  In order to use MySocial
  As a new user
  I want to register me
  Scenario: Sign up
    Given I go to the new user registration page      => Passed
    When I fill in Username with testuser            => Passed
      And I fill in Email with test@test.test        => Passed
      And I fill in user_password with secret        => Passed
      And I fill in Password confirmation with secret => Passed
      And I press Create my account                  => Passed
    Then I should be on the getting started page     => Failed:
                                                    "Page not found [1]"
      And I should see Welcome                       => Pending !!
      And I should see Fill out your profile         => Pending !!
      And I should see Connect with cool people     => Pending !!
      And I should see Finished                     => Pending !!

```

Figura 5.10: Resultado mostrando que o teste falhou.

```

NewUserRegistration : Passed

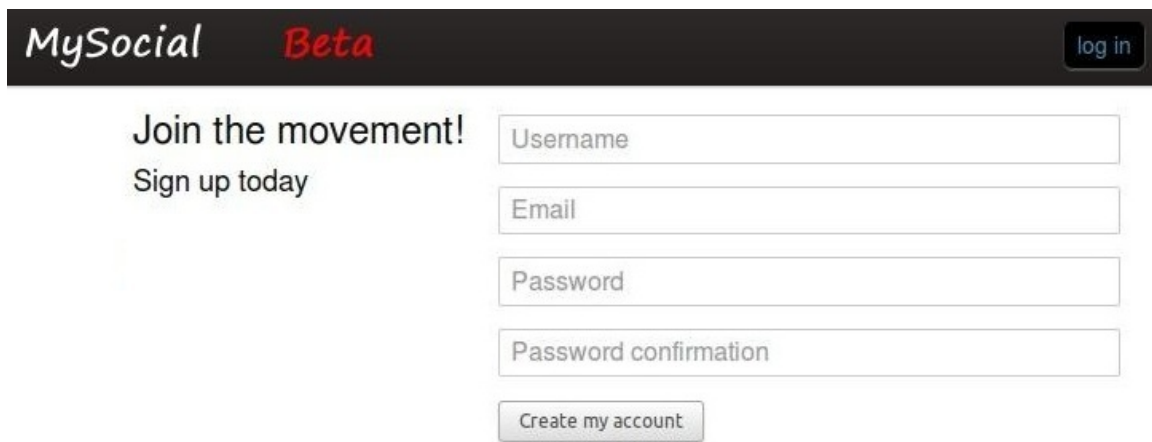
Story is New user registration
  In order to use MySocial
  As a new user
  I want to register me
  With scenario Sign up
    Given I go to the new user registration page      => Passed
    When I fill in Username with testuser            => Passed
      And I fill in Email with test@test.test        => Passed
      And I fill in user_password with secret        => Passed
      And I fill in Password confirmation with secret => Passed
      And I press Create my account                  => Passed
    Then I should be on the getting started page     => Passed
      And I should see Welcome                       => Passed
      And I should see Fill out your profile         => Passed
      And I should see Connect with cool people     => Passed
      And I should see Finished                     => Passed

```

Figura 5.11: Resultado mostrando que o teste foi aprovado.

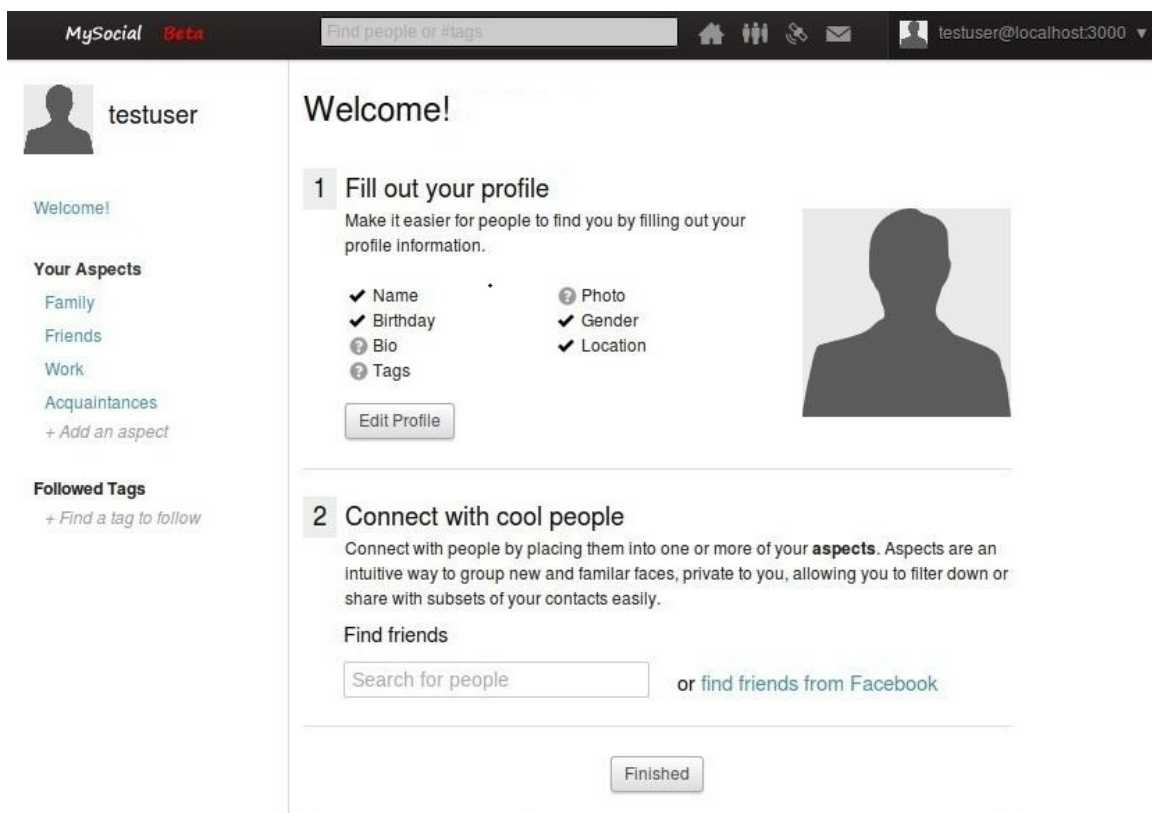
O uso desta técnica não impede a utilização de testes unitários. Ao contrário, a utilização destes testes ajuda a complementar a avaliação da História do Usuário. Formas de integrar os testes unitários com os testes funcionais pode ser encontrada nos livros *The RSpec Book* (CHELIMSKY et al., 2010) e *The Cucumber Book* (WYNNE; HELLESØY, 2012).

Uma vez que os Mock-ups são artefatos do processo, o projeto final das páginas da aplicação correspondentes às regras de negócio das Histórias do Usuário seguem o que esses Mock-ups definiram com o usuário. A Figura 5.12 mostra a versão entregue ao usuário dos Mock-ups Sign Up e Getting Started, ambos utilizados como exemplo da aplicação MySocial.



The screenshot shows the sign-up page for 'MySocial Beta'. At the top left is the logo 'MySocial Beta' and at the top right is a 'log in' button. The main heading is 'Join the movement! Sign up today'. Below this are four input fields: 'Username', 'Email', 'Password', and 'Password confirmation'. At the bottom is a 'Create my account' button.

(a) Página Sign Up.



The screenshot shows the 'Getting Started' page for 'MySocial Beta'. The top navigation bar includes the logo, a search bar 'Find people or #tags', and user information 'testuser@localhost:3000'. The main content area is titled 'Welcome!' and features two main sections: '1 Fill out your profile' and '2 Connect with cool people'. The first section includes a list of profile fields: Name, Birthday, Bio, Tags, Photo, Gender, and Location, with checkboxes indicating completion status. An 'Edit Profile' button is present. The second section includes a 'Find friends' search bar and a link 'or find friends from Facebook'. A 'Finished' button is at the bottom.

(b) Página Getting Started.

Figura 5.12: Páginas criadas seguindo os testes funcionais das Histórias do Usuário da iteração e baseadas nos Mock-ups da Figura 5.4.

O resultado final é a lógica de negócio implementada, funcionalmente testada, conforme os requisitos especificados nas Histórias do Usuário. Essas histórias são tratadas como testes de aceitação que simulam o comportamento esperado pelo usuário. É importante perceber que corrigindo os erros que aparecem durante esses testes, o time de desenvolvimento pode criar uma aplicação mais próxima das necessidades do usuário, particularmente no que se refere ao processamento das regras de negócio.

Esse processo é repetido iterativamente, através de todos os critérios de aceitação de cada cenário presente nas Histórias do Usuário. O código necessário é implementado até que todas as Histórias do Usuário da iteração não apresentem mais erros. No final desta última disciplina, a iteração do Sprint termina e um protótipo totalmente funcional está pronto.

5.3 Fases do Processo

O processo de desenvolvimento é realizado em ciclos, ao longo de quatro fases consecutivas. Cada fase é composta por seus objetivos, suas atividades e seus artefatos gerados e refinados. A principal disciplina de cada fase também é destacada. Para cada Sprint essas fases se repetem produzindo versões cada vez mais completas e refinadas da aplicação.

5.3.1 Requisitos

Esta fase visa a definir os objetivos do projeto e se o mesmo é viável. Por englobar a especificação de requisitos e muitas interações interpessoais, nesta fase a principal disciplina é a **Comunicação**. As pessoas envolvidas nesta fase são: o Consultor de Desenvolvimento, responsável pela especificação dos requisitos e pela entrevista com o usuário; o Usuário, que descreve a necessidade que a aplicação Web deve suprir e suas funcionalidades; e o Analista de Negócios, uma pessoa com conhecimento na área de negócios do usuário que auxilia o Consultor de Desenvolvimento na busca pelos requisitos e na entrevista. Nada impede que o Consultor tenha conhecimento na área de negócios e, assim, acumule as funções do Analista de Negócios.

Para o Consultor de Desenvolvimento é sugerido que mantenha um contato com o usuário (ou alguém indicado pelo usuário) para solucionar problemas ou especificar informações sobre a aplicação Web. Não deixar para ter essas conversas apenas em reuniões formais. Essas informações são vitais para o time de desenvolvimento analisar e projetar o que será feito em cada iteração, estimando esforço e tempo. Os requisitos especificados nessas conversas são utilizados na próxima fase.

As atividades necessárias nesta fase envolvem a identificação dos diferentes tipos de usuário que irão interagir com a aplicação e como essas interações irão ocorrer. Isto será realizado nas conversas entre as partes interessadas (stakeholders) através da utilização de Cartões de História e de Mock-ups. A disciplina de **Modelagem** pode ocorrer nesta fase, porque uma possível atividade aqui é a criação de elementos de modelagem, como um Diagrama de Tipos, para

auxiliar na compreensão dos requisitos e na comunicação com o usuário, ou mesmo a escrita de algumas Histórias do Usuário.

5.3.2 Planejamento

Os objetivos desta fase são analisar o domínio do problema, desenvolver o plano do projeto, estabelecer a fundamentação arquitetural e eliminar os elementos de risco. Estes elementos são os riscos de requisito, tecnológico (referentes à capacidade das ferramentas disponíveis) e habilidade (referentes aos membros do projeto). As pessoas envolvidas nesta fase são o time de desenvolvimento e o Product Owner (PO). A disciplina **Comunicação** continua como a principal nesta, devido as interações entre o PO e o time de desenvolvimento e os artefatos que serão gerados.

Deve ser preparado um Product Backlog com as funcionalidades desejadas para a aplicação Web. O PO tem a responsabilidade de criar o Product Backlog e gerenciá-lo. O Product Backlog deve ser apresentado para o time de desenvolvimento em uma reunião para definir o que será feito no ciclo de implementação e testes (ou iteração), como o Sprint do Scrum. O time de desenvolvimento deve analisar este Backlog, selecionar o que será feito na iteração e dividir as funcionalidades em tarefas que serão executadas dentro da iteração. Essa lista de tarefas a serem realizadas é semelhante ao Sprint Backlog.

É recomendado que a iteração não dure mais do que algumas semanas (PRESSMAN; LOWE, 2009). Considerando as orientações do Scrum em relação ao Sprint, essas “algumas semanas” podem ser refinadas para um período de duas a quatro semanas. Todos os itens selecionados para participar da iteração devem ser completados durante esse período. Isso é um compromisso que o time de desenvolvimento deve assumir e o seu descumprimento levar à perda de confiança por parte do usuário.

O usuário pode requisitar mudanças antes do início da iteração ou de itens que não fazem parte da mesma. Os Backlogs devem ser constantemente atualizados e, o mais importante, o usuário deve ter informações sobre o progresso do projeto durante todo o tempo.

Para documentação, é indicado o uso de uma ferramenta gráfica ou CASE e o armazenamento das diferentes versões dos artefatos – em relação aos ciclos de iterações. Entretanto, devido a grande dinamicidade das reuniões do time de desenvolvimento e das conversas com usuário, é indicado utilizar esboços devido a comodidade e rapidez que eles proporcionam.

Além disso, a disciplina **Modelagem** está nesta fase, porque o time de desenvolvimento pode definir a arquitetura adotada para suportar a aplicação, como por exemplo o uso do padrão

MVC, criando modelos para melhor visualizar como adotar o padrão. Outros elementos de modelagem (como diagramas UML (ERIKSSON; PENKER; LYONS, 2004)) também podem ser elaborados para auxiliar o time de desenvolvimento a solucionar possíveis dúvidas sobre a aplicação. O mesmo é válido para a disciplina **Construção**, pois é possível realizar a implementação (e teste) de protótipos, visando a encontrar problemas no entendimento da aplicação e escolher entre diferentes formas de abordar a construção da mesma.

5.3.3 Execução

Nesta fase são desenvolvidos os artefatos necessários para criar uma versão (ou refinar uma previamente criada) da aplicação Web baseada nas funcionalidades da iteração atual (incrementando o que foi criado em iterações anteriores) e respeitando os artefatos desenvolvidos nas outras fases. Por esse motivo, essa fase tem duas disciplinas principais. Primeiramente, a principal disciplina desta fase é a **Modelagem**, quando o time de desenvolvimento se reúne para dividir as tarefas da iteração, modelar o problema e verificar como abordá-lo. Na sequência, a disciplina **Construção** ganha maior foco, devido ao início da implementação e dos testes.

Ao representar a implementação da aplicação Web, esta fase pode ser considerada um equivalente ao Sprint do Scrum. No entanto, este processo entra em alguns detalhes sobre como a aplicação pode ser feita em busca de um resultado mais próximo das solicitações do usuário. Para alcançar esses objetivos, as atividades envolvem o total entendimento dos requisitos identificados nos artefatos e baseados neles, criar novos artefatos para auxiliar no desenvolvimento do projeto, como um Diagrama de Classes ou o refinamento do Diagrama de Tipos previamente criado.

Outro artefato que deve ser gerado é a História do Usuário de acordo com o que foi visto na disciplina Modelagem. Ela é uma transcrição o mais próxima possível de uma (ou mais) funcionalidade(s) visualizada nos Mock-ups e seus respectivos Cartões de História. Assim, todas as funcionalidades da lista de atividades da iteração devem ser transcritas para o formato das Histórias do Usuário, as quais serão úteis durante essa fase. É aconselhável o uso de uma ferramenta para gerenciamento do projeto, preferencialmente com a função de lidar com as Histórias do Usuário, como mostra a Figura 5.13. Assim, as histórias podem ser separadas entre aceitas e não aceitas, presentes na iteração atual e aguardando por futuras iterações.

É importante perceber que utilizar TDD (BECK, 2003) é muito importante neste processo, não “apenas” para criar testes antes do código da aplicação, visando diminuir a probabilidade de ter um produto final de pior qualidade. Mas, para que os testes ajudem o desenvolvedor a entender melhor o “problema” e resolvê-lo de uma forma mais eficiente.

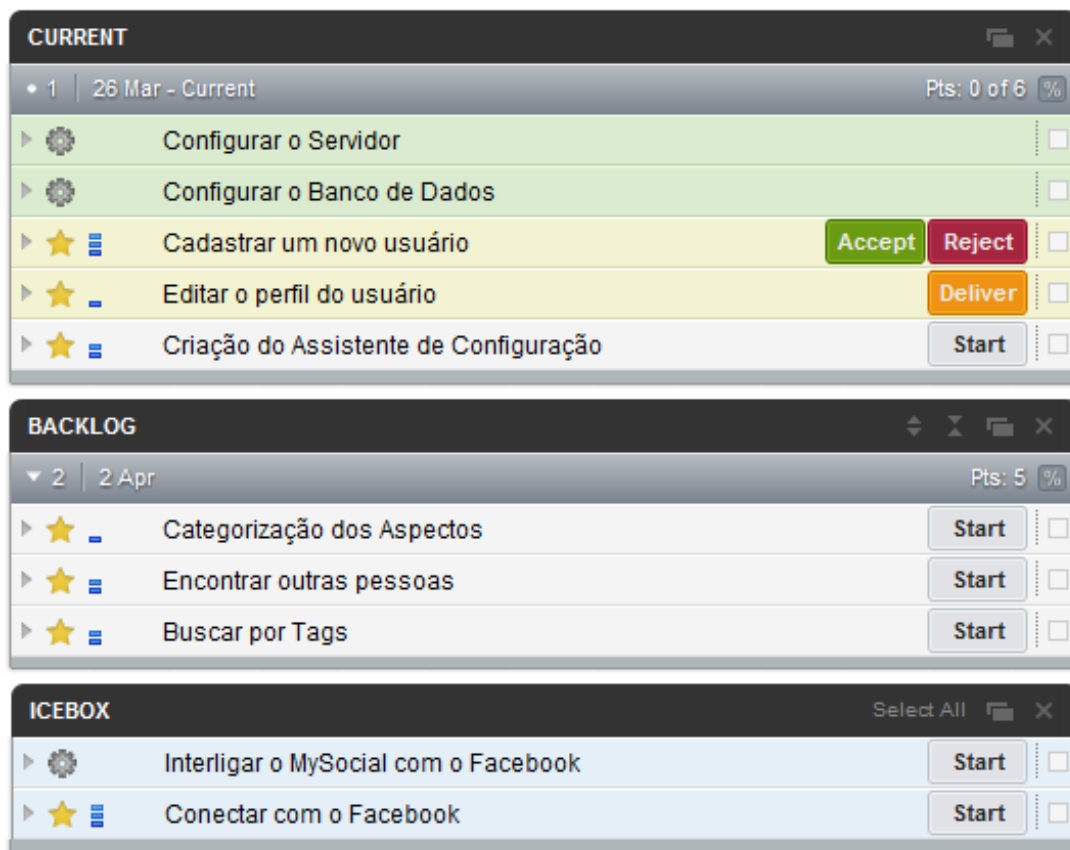


Figura 5.13: Exemplo de uma ferramenta gerenciando as atividades atuais, em espera e as possíveis atividades futuras de um projeto.

Assim, o processo faz uso de testes orientados ao comportamento. Isso se deve ao fato de que as Histórias do Usuário criadas nesta fase são tratadas como testes de aceitação e orientam a implementação. Vale ressaltar que essa lógica é o núcleo deste processo. Todos os artefatos gerados até o momento têm como objetivo orientar a implementação com base no comportamento esperado que a aplicação Web deve ter.

Durante toda essa fase, a disciplina **Comunicação** também está presente, porém em menor intensidade. Isso acontece porque o usuário pode acompanhar todo o ciclo de desenvolvimento e solicitar ao PO novas funcionalidades (através dos Cartões de História e dos Mock-ups), que são adicionadas ao Product Backlog para futuras iterações. O usuário também pode ser consultado para resolver possíveis dúvidas sobre as regras de negócio que estão sendo implementadas na iteração atual.

5.3.4 Encerramento

Finalmente, o objetivo desta fase é apresentar o produto criado na iteração para o usuário e colocá-lo em produção, se essa for a vontade do usuário.

O usuário pode ou não solicitar mudanças. Caso solicite, o código escrito como foi mostrado neste trabalho pode ser mais fácil de alterar, devido aos testes de aceitação que irão estar em execução para mostrar onde a alteração solicitada (ou uma nova funcionalidade) gerou erro na aplicação. Tanto as mudanças quanto novas funcionalidades são tratadas em um novo ciclo do projeto, retornando para a fase de Requisitos. Se nenhuma alteração ou nova funcionalidade for solicitada, o ciclo do processo retorna para a fase de Planejamento, onde o time de desenvolvimento fará uma nova reunião com o Product Owner para selecionar as Histórias do Usuário para a nova iteração.

Outra atividade a ser realizada nesta fase é uma reunião entre o time de desenvolvimento para analisar como foi a fase de Execução e resolver potenciais problemas para a próxima iteração.

5.4 Considerações Finais

Este capítulo apresentou o processo RAMBUS desenvolvido neste trabalho para apoio à construção de aplicações groupware na Web 2.0. O processo baseia-se nos conceitos de TDD e Scrum, focando-se no uso de testes funcionais, a partir de Cartões de História até a criação de Histórias do Usuário (escritas na LET, por exemplo), para geração parcial de código dos casos de testes (geração da estrutura), e na orientação que esses testes oferecem para a implementação das aplicações. Além disso, as experiências iniciais com o processo RAMBUS serviram para o seu refinamento. O principal refinamento consistiu em utilizar os Mockp-Ups para complementar os requisitos especificados pelos Cartões de História.

Apesar de utilizar as ferramentas apresentadas no Capítulo 4 no decorrer da explicação do processo, o mesmo é independente das ferramentas. Isso significa que qualquer outra ferramenta que utilize as Histórias do Usuário como testes funcionais (a nível de aceitação) pode ser integrado ao processo. Alguns exemplos de outras ferramentas que podem ser utilizadas são: Cucumber¹² e RSpec¹³ para a linguagem de programação Ruby, e o JBehave¹⁴ para Java.

No próximo capítulo é abordada a avaliação do processo RAMBUS e das ferramentas desenvolvidas.

¹²<http://cukes.info/>

¹³<http://relishapp.com/rspec>

¹⁴<http://jbehave.org/>

Capítulo 6

AVALIAÇÃO

Avaliação é um processo para a melhoria da qualidade do processo de produção de software (ARES et al., 1998). Através da avaliação é que se pode verificar a efetividade de novas ferramentas, métodos, processos e estratégias de desenvolvimento de software antes de colocá-los em prática. Se nenhuma avaliação é realizada, a introdução de uma nova tecnologia no desenvolvimento de software pode ser ineficiente e até mesmo ter impactos negativos na qualidade do software desenvolvido. A avaliação na Engenharia de Software, fornece as bases para validar as teorias a respeito de uma tecnologia e confrontá-las com a realidade, com o intuito de descobrir se irão, de fato, surtir o efeito esperado tanto no produto quanto no processo (TRAVASSOS; GUROV; AMARAL, 2002; WOHLIN et al., 2000).

O desenvolvimento de software é, na maioria das vezes, uma tarefa complexa. Projetos de software podem se estender por longos períodos de tempo, envolver diferentes pessoas com habilidades distintas, e consistir de várias atividades que produzem diversos documentos intermediários para especificação do software a ser entregue. Essa complexidade acarreta em dificuldades para a otimização dos processos de desenvolvimento (WOHLIN et al., 2000). Neste sentido, a avaliação e o aperfeiçoamento dos processos de software têm sido estratégias aplicadas ao longo dos anos em busca da melhoria da qualidade do software desenvolvido (e.g. Quality Improvement Paradigm - QIP (BASILI; GREEN, 1994), Capability Maturity Model - CMM (PAULK et al., 1995)). Em geral, os métodos de avaliação baseiam-se na premissa de que a qualidade do produto de software é determinada pela qualidade de seu processo de desenvolvimento, considerando a máxima de que um bom produto é resultado direto de um bom processo (ARES et al., 1998).

De fato, a literatura sugere que novas abordagens de desenvolvimento de software devem ser avaliadas no sentido de verificar sua efetividade antes de colocá-las em prática (TRAVASSOS; GUROV; AMARAL, 2002; WOHLIN et al., 2000). Nesse sentido, neste capítulo é apresentada a

validação do processo RAMBUS, elaborado neste trabalho. Para avaliação do processo foram conduzidas atividades de treinamento nas técnicas e tecnologias utilizadas no experimento. Na época do experimento, as ferramentas de suporte ao processo RAMBUS não se encontravam em condições de testes, por isso foi definido o uso de uma outra ferramenta, com funcionalidades equivalentes, para suprir a necessidade de um *framework* para Histórias do Usuário. Para esse fim, foi escolhido o Cucumber¹, por ser o mais próximo das funcionalidades existentes na biblioteca Story2CSharp. A Tabela 6.1 mostra as principais diferenças entre esses *frameworks* e a biblioteca Story2CSharp.

Tabela 6.1: Diferenças entre Cucumber e Story2CSharp.

Características	Cucumber	Story2CSharp
Plataforma	Ruby e JRuby	.Net
Uso de ER para mapear os CA	Sim, parcialmente gerado	Não utiliza ER
Geração automática da ET	Não	Sim
Apresentação dos resultados	Recurso próprio	NUnit ou VS UnitTesting

ER - Expressão Regular; CA - Critérios de Aceitação; ET - Estrutura dos Testes.

Sobre a segunda e a terceira características apresentadas na tabela, o Cucumber utiliza expressões regulares para fazer a ligação entre a descrição textual das Histórias do Usuário e o correspondente código a ser testado. Por sua vez, como foi apresentado no Capítulo 4, a biblioteca Story2CSharp converte a descrição textual automaticamente para uma estrutura de testes funcionais.

Após as atividades preliminares de treinamento, uma avaliação controlada, com a aplicação da metodologia experimental (WOHLIN et al., 2000), foi conduzida com o intuito de avaliar o impacto do processo proposto na eficiência de equipes que desenvolvem aplicações groupware.

Este capítulo está organizado da seguinte forma: a Seção 6.1 apresenta as atividades preliminares realizadas; a Seção 6.2 descreve o desenvolvimento da experimentação do processo, incluindo a análise e interpretação dos resultados obtidos; a Seção 6.3 descreve uma avaliação realizada sobre as ferramentas de suporte desenvolvidas; e na Seção 6.4 estão as considerações finais que encerram este capítulo.

6.1 Atividades Preliminares

Como o experimento utilizou técnicas, conceitos e tecnologias que não são comuns de serem apresentadas em cursos de graduação, no decorrer do segundo semestre de 2011 aulas

¹<http://cukes.info/>

foram ministradas pelo experimentador para suprir essas carências.

Sendo assim, os estudantes da disciplina de Desenvolvimento de Software Orientado a Objetos, do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de São Carlos (UFSCar), tiveram aulas sobre a linguagem de programação Ruby e o seu *framework* Rails, visando o uso da ferramenta Cucumber para executar as Histórias do Usuário. Também foram ministradas aulas sobre Test Driven Development (TDD) e Behavior Driven Development (BDD), para apresentar os seus respectivos conceitos. Todas as aulas teóricas foram seguidas de atividades práticas, visando o aprendizado dos estudantes para ter um experimento sem grandes problemas devido a falta de experiência dos mesmos.

6.2 Experimentação do processo RAMBUS

Na Engenharia de Software a experimentação é importante para testar hipóteses sobre uma nova tecnologia, observando se os efeitos da sua adoção estão alinhados com o que foi declarado nas hipóteses (WOHLIN et al., 2000). A experimentação, portanto, verifica a previsão teórica de encontro à realidade, de forma a prover evidências de que o objeto avaliado (e.g. processo, método, ferramenta, abordagem de desenvolvimento, recurso, modelo, teoria) está ou não indo na direção esperada (TRAVASSOS; GUROV; AMARAL, 2002).

Na próxima seção faz-se uma breve introdução sobre os princípios experimentais aplicados à Engenharia de Software, nos quais se baseou esta parte da avaliação. Em seguida, apresenta-se o experimento conduzido que avaliou o impacto na eficiência de equipes que desenvolveram aplicações groupware utilizando o processo RAMBUS (PEREIRA; PRADO, 2011) em comparação ao processo Scrum (SCHWABER, 2004).

6.2.1 Experimentação em Engenharia de Software

A Engenharia de Software é descrita formalmente pelo IEEE como “a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software” (IEEE, 1990). Tratando-se de uma abordagem de engenharia, dentre outros aspectos, a Engenharia de Software implica num processo que seja previsível e disciplinado para o desenvolvimento de produtos de software confiáveis de forma controlada e eficiente. Esse mesmo rigor com que o software é produzido certamente deverá ser aplicado quando da avaliação e aperfeiçoamento da construção de software (WOHLIN et al., 2000).

Neste sentido, a experimentação oferece um modo sistemático, disciplinado, quantificável e controlado para avaliação da atividade humana envolvida num processo de criação de software. Em Engenharia de Software os experimentos normalmente são realizados em laboratórios, sob condições controladas. O objetivo é manipular uma ou mais variáveis relacionadas com o objeto sendo estudado enquanto mantêm-se outras variáveis em um nível fixo. O efeito dessa manipulação observado nos resultados do experimento é medido, e, com base nisso, análises são desempenhadas para validar ou refutar as hipóteses formuladas (TRAVASSOS; GUROV; AMARAL, 2002; WOHLIN et al., 2000).

6.2.2 Desenvolvimento da Experimentação

O experimento, conduzido no segundo semestre de 2011, consistiu de um estudo comparativo entre o uso do RAMBUS para a construção de uma aplicação groupware e o uso do processo Scrum, sem o emprego de TDD, para o mesmo fim. Pelo fato do RAMBUS ser baseado no processo Scrum, o foco da análise foi verificar se a forma como a utilização de testes funcionais foi integrada no processo era ou não eficaz.

A experimentação foi realizada com estudantes do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de São Carlos (UFSCar). Alguns deles eram recém-formados, com e sem experiência profissional, e alguns com curso de especialização em desenvolvimento Web. Eles foram divididos em quatro grupos (três grupos com três pessoas e outro com quatro pessoas) para desenvolver uma aplicação groupware.

Para a realização do experimento foi utilizada uma aplicação groupware para dicas e indicações sobre restaurantes e hotéis, elaborada especialmente para a execução do experimento. Nessa aplicação, os usuários registram restaurantes e hotéis (com seus atributos), indicam o quanto foi gasto e que nota dada ao restaurante/hotel. Além disso, os usuários podem deixar comentários sobre os restaurantes e/ou os hotéis. A aplicação é gerenciada pelos próprios usuários. Uma vez feito o login, o usuário pode adicionar, excluir e editar as informações sobre os restaurantes e hotéis, bem como suas qualificações. Visitando o site <http://www.tripadvisor.com/> pode-se encontrar um sistema semelhante e muito mais completo. Este sistema inspirou o exemplo para o experimento. A Figura 6.1 mostra uma tela da aplicação a ser desenvolvida no experimento.

Para essa tarefa, os grupos realizaram duas atividades: (1) desenvolver uma parte do aplicativo groupware utilizando Scrum; e (2) desenvolver outra parte do aplicativo groupware utilizando RAMBUS. Dessa forma, dois grupos criaram toda a parte referente ao restaurante utilizando primeiro o Scrum, enquanto os outros dois grupos realizaram a mesma atividade

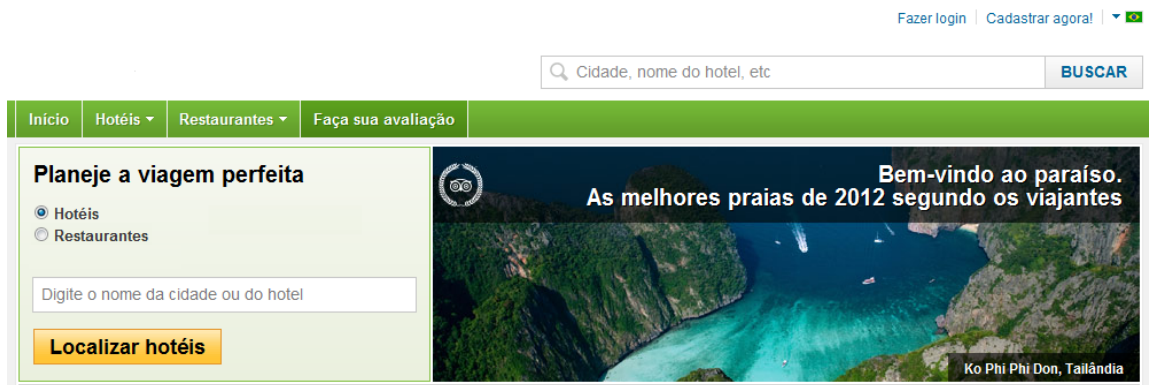


Figura 6.1: Uma tela da aplicação groupware do experimento.

utilizando RAMBUS. Depois os grupos trocaram de processo e desenvolveram o restante da aplicação, ou seja, o módulo referente ao hotel.

Devido ao fator tempo, os dois processos foram fixados em duas iterações para cada atividade. A primeira foi desenvolver as funções de gerenciamento do estabelecimento (restaurante e, posteriormente, hotel), clientes e qualificações. Na segunda iteração, os grupos deveriam qualificar um estabelecimento, calcular a média das notas e dos preços de cada estabelecimento e seu respectivo número de qualificações.

Para auxiliar na utilização do processo RAMBUS, como material de apoio foram disponibilizados os exemplos utilizados nas Atividades Preliminares, como os códigos dos exercícios com Ruby e Rails, e as matérias de TDD e BDD, incluindo os exemplos práticos de como utilizar o *framework* Cucumber. Para o Scrum, os grupos poderiam desenvolver a aplicação seguindo qualquer técnica que considerassem importante, tendo como limite apenas construir os artefatos que compõem o processo Scrum.

Após cada atividade, os participantes responderam a um questionário sobre qual foi a experiência no desenvolvimento do aplicativo requerido, utilizando o processo da atividade proposta e como eles avaliam esse processo. Finalmente, no final das duas atividades, um questionário foi utilizado para possibilitar aos alunos a comparação dos processos e a citação dos pontos fortes e fracos do processo de RAMBUS.

As fases experimentais realizadas no estudo são descritas a seguir.

6.2.2.1 Definição do Experimento

O objetivo do estudo foi:

- **Analisar** o uso do processo RAMBUS na construção de aplicações groupware;

- **Com o propósito** de avaliação;
- **Com respeito à** eficiência dos artefatos em integrar os testes funcionais ao processo;
- **Do ponto de vista de** desenvolvedores de software;
- **No contexto de** estudantes de pós-graduação em Ciência e da Computação.

O contexto de um experimento pode ser caracterizado em termos do número de participantes e objetos envolvidos no estudo experimental. No caso do experimento realizado, considerando o envolvimento de diversos participantes e de um objeto (aplicação groupware), o contexto do experimento classificou-se como **estudo de objeto com vários testes** (WOHLIN et al., 2000). Isto significa que o estudo consistiu de diversos testes experimentais, onde em cada teste houve um grupo de participantes aplicando um determinado processo para construir aplicação proposta pelo experimentador.

6.2.2.2 Planejamento do Experimento

O experimento ocorreu em ambiente universitário, sendo realizado com estudantes de pós-graduação no Laboratório de Ensino do Departamento de Computação da Universidade Federal de São Carlos (UFSCar), no âmbito da disciplina Desenvolvimento de Software Orientado a Objetos.

Foram elaboradas três hipóteses para o experimento com relação ao efeito do processo de desenvolvimento no resultado. As hipóteses são:

- **Hipótese nula (H0):** Em geral, não há diferença entre as equipes de desenvolvimento que utilizam o processo RAMBUS e equipes que utilizam o processo Scrum para a construção de aplicações groupware;
- **Hipótese alternativa (H1):** As equipes que utilizam o processo RAMBUS para a construção de aplicações groupware ficam, em geral, mais satisfeitas do que as equipes que utilizam o processo Scrum;
- **Hipótese alternativa (H2):** As equipes que utilizam o processo Scrum para a construção de aplicações groupware ficam, em geral, mais satisfeitas do que as equipes que utilizam o processo RAMBUS.

A satisfação das equipes é analisada tendo como base o tempo necessário para desenvolver a aplicação e a opinião dos participantes sobre cada um dos processos. As variáveis consideradas

foram: o processo de desenvolvimento, a aplicação desenvolvida, o ambiente de desenvolvimento e as tecnologias de desenvolvimento. Uma vez que o objetivo era investigar a satisfação dos participantes com o processo de desenvolvimento, foi estabelecido que o mesmo receberia tratamento distinto e teria seu efeito observado. As demais variáveis foram mantidas constantes durante o estudo, conforme segue:

- **Aplicação:** Aplicação groupware similar ao TripAdvisor;
- **Ambiente de desenvolvimento:** IDE Netbeans;
- **Tecnologias de desenvolvimento:** Ruby, Rails, HTML, CSS, jQuery e Cucumber.

A seleção dos participantes foi feita através de amostragem não probabilística por conveniência (WOHLIN et al., 2000), selecionando os indivíduos disponíveis mais próximos para participarem do experimento. Participaram do estudo 13 alunos de pós-graduação do Programa de Pós-Graduação em Ciência da Computação da UFSCar, matriculados na disciplina Desenvolvimento de Software Orientado a Objetos.

Dessa forma, os alunos foram divididos em 4 grupos (blocos), de modo que cada grupo possuísse, tanto quanto possível, médias semelhantes de nível de experiência. A experiência dos participantes foi avaliada pelo formulário de caracterização dos participantes— documento utilizado para capturar a experiência profissional e experiência nos assuntos relacionados ao estudo. Esse formulário foi entregue a cada participante semanas antes da execução do experimento, de forma que fosse possível planejar o estudo antecipadamente. O gráfico da Figura 6.2 mostra os níveis de experiência individuais de cada participante, conforme as informações apresentadas pelos alunos em seus respectivos formulários de caracterização. Esses níveis foram obtidos através da média das escalas do grau de experiência nas tecnologias e nos conceitos/técnicas empregados no experimento, reportados pelos participantes nos formulários. São eles: (1) **Tecnologias:** linguagem Ruby, *framework* Rails, linguagem de marcação HTML, folhas de estilo (CSS), banco de dados MySQL, IDE NetBeans, UML e ferramentas de testes; e (2) **Conceitos e Técnicas:** Desenvolvimento Orientado a Testes (TDD), Desenvolvimento Orientado a Comportamento (BDD) e Scrum.

A alocação dos participantes nos grupos foi realizada de maneira mais balanceada possível, uma vez que o número de participantes é ímpar (13 alunos). Sendo assim, foram criados três grupos com 3 alunos e um grupo com 4 alunos. A Tabela 6.2 apresenta a composição dos grupos.

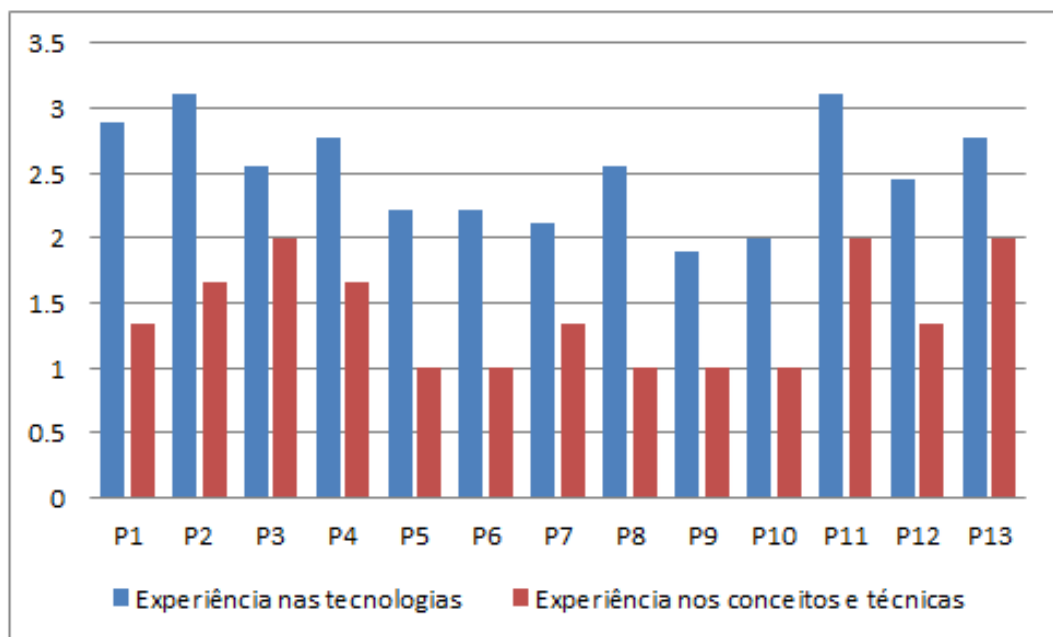


Figura 6.2: Nível de experiência dos participantes.

Tabela 6.2: Grupos e seus respectivos participantes para o experimento.

Grupo 1	Grupo 2	Grupo 3	Grupo 4
P2, P9 e P11	P1, P7 e P8	P3, P4 e P10	P5, P6, P12 e P13

O tipo do projeto do experimento foi de um fator (processo de desenvolvimento) com dois tratamentos (RAMBUS e Scrum) completamente diferentes (WOHLIN et al., 2000). Nesse tipo de projeto um mesmo objeto a ser manipulado durante o estudo é utilizado em todos os tratamentos e os participantes (ou grupos) são aleatoriamente atribuídos a cada tratamento. No caso do experimento, a aplicação groupware foi desenvolvida utilizando ambos processos, sendo metade através do processo RAMBUS e a outra metade pelo processo Scrum. Visando minimizar o efeito dos grupos terem maiores dificuldades no começo das atividades (independente do processo utilizado), foi definido que dois grupos iriam começar as atividades com o processo Scrum e os outros dois grupos com o processo RAMBUS. Essa definição foi aleatória. A Tabela 6.3 apresenta o resultado dessa divisão.

Tabela 6.3: Divisão dos processos pelos grupos.

	Scrum	RAMBUS
Atividade 1	Grupo 1 e 4	Grupo 2 e 3
Atividade 2	Grupo 2 e 3	Grupo 1 e 4

Por fim, os materiais necessários para apoiar os participantes no experimento foram previamente planejados, compreendendo a definição dos objetos que seriam manipulados, as diretrizes

para orientar os grupos na execução dos processos, bem como os instrumentos para a coleta de dados e medição.

6.2.2.3 Execução do Experimento

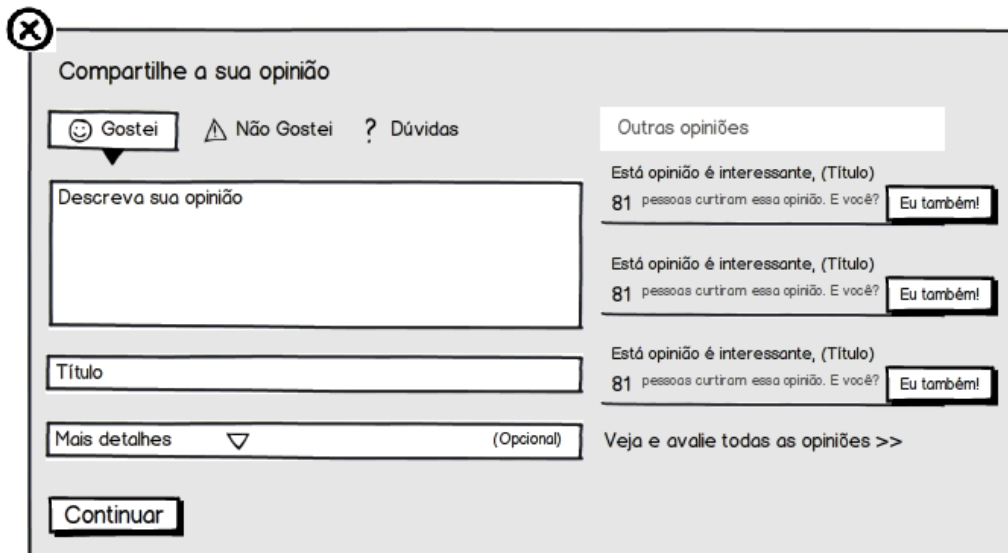
A execução do experimento foi realizada em duas aulas (uma aula, com quatro horas de duração, por semana), sendo a primeira para a atividade 1 e a segunda aula para a atividade 2, tendo como foco a fase de Execução do processo RAMBUS e o Sprint do processo Scrum. Assim, os participantes tiveram que construir as Histórias do Usuário e outros elementos de modelagem que considerassem necessários (tendo como base os artefatos das fases anteriores entregues pelo experimentador), e implementar e testar a aplicação Web. Por fim, a fase de Encerramento envolveu a apresentação da aplicação Web criada e a análise dos grupos sobre como foi desenvolver seguindo o processo utilizado.

Nesta fase os materiais definidos na instrumentação do experimento foram efetivamente elaborados, a saber:

- **Objetos.** As imagens que deveriam ser utilizadas nas páginas, os arquivos CSS e JavaScript referentes à biblioteca jQuery empregada na construção da aplicação Web foram entregue aos grupos e as alterações necessárias, de acordo com o projeto de cada grupo, contou com o auxílio do experimentador. Em relação ao processo RAMBUS, a fase de Requisitos foi entregue aos participantes, com os artefatos Cartões de História e os Mock-ups criados (Figura 6.3). A Fase de Planejamento foi entregue com os artefatos Product Backlog (Figura 6.4) e Sprint Backlog definidos, além de uma versão simplificada do Diagrama de Classes (Figura 6.5), apenas para orientar os participantes para a próxima fase. O mesmo se aplica ao processo Scrum, uma vez que os Backlogs contêm uma breve descrição das funcionalidades desejadas e o Diagrama de Classes também pode ser utilizado como auxílio.
- **Diretrizes.** Os seguintes documentos foram produzidos para serem utilizados no estudo: 1) *Formulário de caracterização dos participantes*, para obtenção da experiência profissional e nos assuntos relacionados diretamente ao estudo; 2) *Formulário de consentimento*, para aprovação e ciência dos participantes dos objetivos do estudo e das condições de participação; 3) *Descrição da tarefa*, com as instruções da sua execução; e 4) *Descrição da aplicação e material de apoio*, contendo o detalhamento da aplicação groupware, bem como tutoriais baseados nas aulas das Atividades Preliminares.
- **Instrumentos para Coleta de Dados.** Medições em experimentos são conduzidas através

Cartão de História No.: 2	Prioridade: Alta
Nome do projeto: Dicas de Lugares	Data: 22/09/2011
Funcionalidade: Gerenciar Restaurantes	Estimativa: 2
Descrição: Para que eu possa usar o sistema Enquanto um usuário Eu quero gerenciar restaurantes	Testes para se considerar: 1 – Nenhum campo no cadastro pode ser nulo
Anotações: Necessita de username, senha e e-mail.	
Requisitos não funcionais: Não há requisitos não funcionais até o momento	

(a) Cartão de História.



Compartilhe a sua opinião

Outras opiniões

Está opinião é interessante, (Título)
81 pessoas curtiram essa opinião. E você?

Está opinião é interessante, (Título)
81 pessoas curtiram essa opinião. E você?

Está opinião é interessante, (Título)
81 pessoas curtiram essa opinião. E você?

Descreva sua opinião

Título

Mais detalhes (Opcional)

Veja e avalie todas as opiniões >>

(b) Mock-up para a opinião do usuário.

Figura 6.3: Artefatos da fase de Requisitos entregues aos participantes.

Descrição	Valor de Negócio	Frequência de Uso
Gerenciar Restaurantes	ALTA	ALTA
Gerenciar Clientes	ALTA	ALTA
Gerenciar Qualificação	ALTA	ALTA
Médias (Preço e Nota)	MÉDIA	-
Número de Qualificações	MÉDIA	-
Qualificar por Restaurante	BAIXA	MÉDIA
Qualificar por Cliente	BAIXA	MÉDIA

Figura 6.4: O Product Backlog entregue aos participantes.

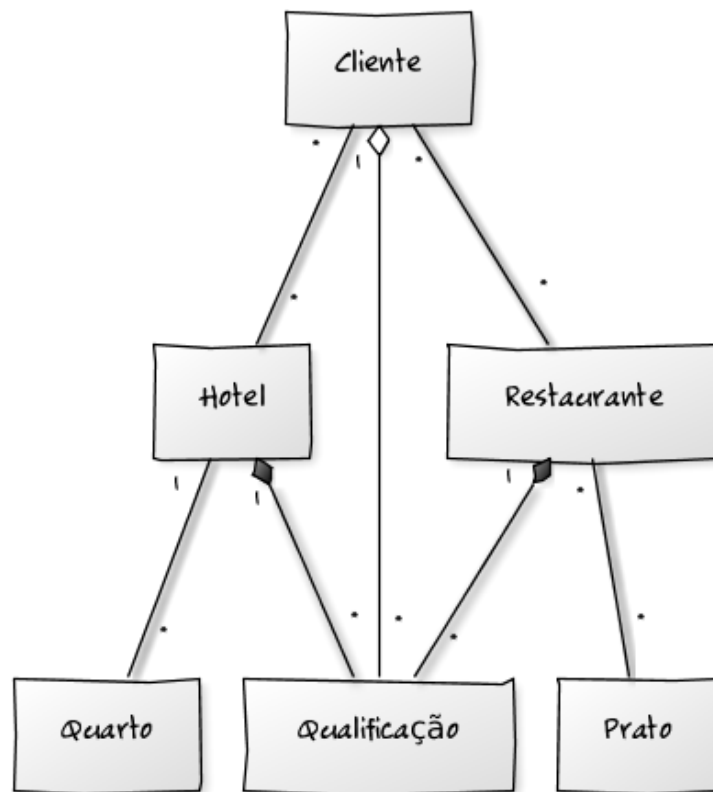


Figura 6.5: O Diagrama de Classes entregue aos participantes.

dos dados que são coletados. Em experimentos envolvendo atividades executadas por seres-humanos, os dados geralmente são coletados manualmente por meio de formulários ou entrevistas. Assim, para a coleta de dados do experimento, foram elaborados também dois tipos de formulário de avaliação qualitativa para cada um dos processos, nos quais cada participante individualmente, após a execução do experimento, deveria reportar sua percepção sobre a facilidade de utilização dos processos para desenvolver a aplicação. Um terceiro formulário foi elaborado para que, no final do experimento, os participantes dessem sua opinião sobre os pontos positivos e negativos do processo RAMBUS.

Durante a execução do experimento, alguns problemas técnicos ocorreram, mas, felizmente, foram solucionados rapidamente, sem comprometer o andamento do experimento. Dúvidas conceituais tanto sobre os processos quanto ao uso das ferramentas também foram levantadas e sanadas, de forma que tais dúvidas não inviabilizassem o estudo.

Após a execução do experimento, verificou-se se os participantes preencheram os formulários de maneira correta. De forma geral, constatou-se que os grupos desempenharam bem a tarefa experimental que lhes foi proposta, executando as atividades com seriedade. Os treinamentos realizados no decorrer de algumas aulas pelas Atividades Preliminares colaboraram para que os participantes se habituassem a linguagem Ruby e seu framework Rails, assim como

os conceitos de TDD e BDD, e o uso da ferramenta Cucumber. Dessa forma, os problemas para aplicá-los em ambos os processos foi minimizado e durante o experimento os grupos não tiveram maiores problemas para executarem as atividades. Os tratamentos, portanto, foram aplicados corretamente conforme planejado no projeto experimental, sendo os dados coletados considerados válidos para o experimento.

6.2.2.4 Análise e Interpretação dos Resultados

A partir de uma análise preliminar dos dados presentes nos formulários, alguns aspectos interessantes foram observados entre os grupos que utilizaram o processo RAMBUS e o processo Scrum. A Tabela 6.4 apresenta o tempo gasto por cada grupo para realizar as duas atividades do experimento. A segunda atividade foi mais rápida do que a primeira em ambos os processos, devido ao fato dos grupos estarem mais preparados e da codificação dos hotéis reutilizar componentes já criados para os restaurantes. Por sua vez, a Tabela 6.5 mostra o tempo médio gasto em cada processo, tendo como base os tempos da Tabela 6.4. Como pode ser visto, o processo RAMBUS é aproximadamente 11% (21 minutos) mais lento que o processo Scrum na Atividade 1 e aproximadamente 12% (17 minutos) mais lento na Atividade 2. Na média total do tempo, o RAMBUS é por volta de 11.5% (19 minutos) mais lento que o Scrum.

Tabela 6.4: Tempo gasto por cada grupo em cada atividade.

		Tempo gasto por atividade			
		Grupo 1	Grupo 2	Grupo 3	Grupo 4
Scrum	Atividade 1	3h05min	-	-	3h17min
	Atividade 2	-	2h13min	2h26min	-
RAMBUS	Atividade 1	-	3h34min	3h29min	-
	Atividade 2	2h41min	-	-	2h33min

Tabela 6.5: Tempo médio gasto em cada atividade.

	Scrum	RAMBUS
Tempo Médio - Atividade 1	3h11min	3h32min
Tempo Médio - Atividade 2	2h20min	2h37min
Tempo Médio Total	2h46min	3h05min

Entretanto, a Figura 6.6 mostra que a satisfação (opinião) dos participantes, em relação a cada processo ao desenvolver a aplicação do experimento, é maior com o processo RAMBUS do que com o processo Scrum.

Como pode ser visto, a satisfação dos participantes com o processo Scrum varia entre FRACO e ÓTIMO. Com isso, pode ser inferido que por volta de 76.9% (10) dos participantes

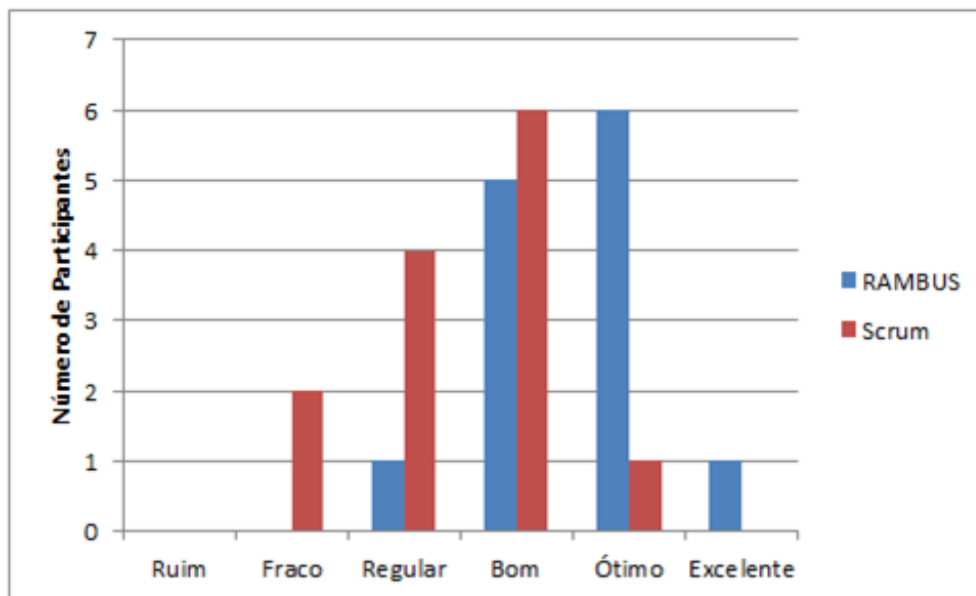


Figura 6.6: Satisfação dos participantes com os processos analisados.

consideram a satisfação com o Scrum entre **REGULAR** e **BOM**. Por outro lado, na opinião dos participantes, a satisfação com o processo RAMBUS varia entre **REGULAR** e **EXCELENTE**. Dessa forma, é possível verificar que cerca de 84.6% (11) dos participantes classificam o RAMBUS entre **BOM** e **ÓTIMO**. Tendo isso como base, apesar de o RAMBUS ser mais lento em comparação ao Scrum (sem testes), a opinião dos participantes mostra que é válido o processo, pois eles se mostraram mais satisfeitos com o resultado e a forma como ele foi alcançado.

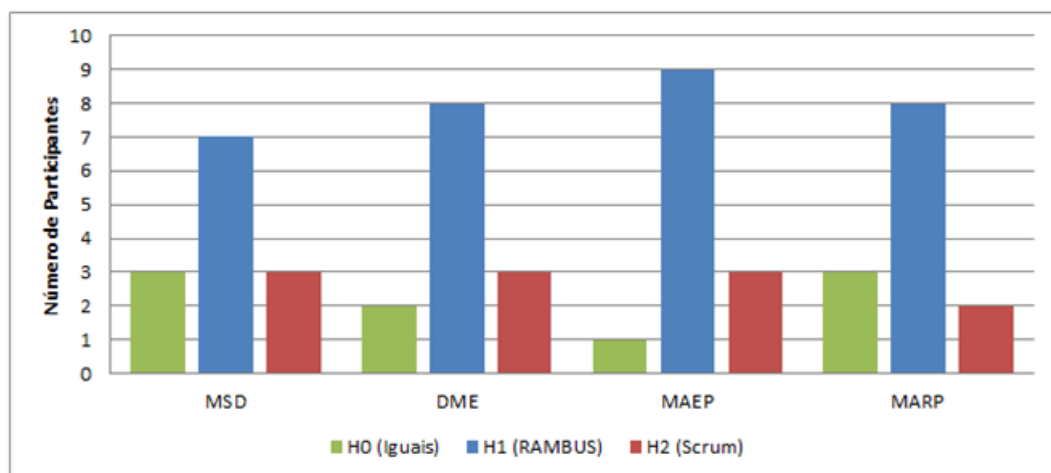
Para concluir a apresentação dos dados coletados, a comparação dos processos envolveu também a opinião dos participantes em relação a quatro tipos de dados: (1) maior segurança no desenvolvimento – o quão seguro o estudante ficou no decorrer do processo sobre “o que fazer” e “como fazer”; (2) documentação mais eficiente – se os documentos gerados foram suficientes para auxiliar no processo; (3) maior auxílio na compreensão do problema – o quanto o processo ajudou no entendimento do problema; e (4) maior auxílio na resolução do problema – o quanto o processo ajudou de alguma forma na busca pelo produto final. A Tabela 6.6 apresenta as respectivas siglas criadas para cada um dos quatro tipos de dados comparativos. Cada participante escolheu uma das três opções disponíveis para cada um dos dados: Iguais (H0), RAMBUS (H1) ou Scrum (H2).

Com base nas respostas de cada participante envolvido no estudo de caso sobre estes quatro dados, a Figura 6.7 mostra que pelo menos neste estudo os resultados obtidos com o processo RAMBUS foram melhores do que o esperado. Por utilizar o Scrum como base, acrescentando artefatos para integrar o uso de testes funcionais ao processo, era esperado que o processo RAMBUS tivesse um desempenho superior em relação a esses dados comparativos. Porém,

Tabela 6.6: Dados comparativos e suas respectivas siglas.

Dados comparativos	Siglas
Maior Segurança no Desenvolvimento	MSD
Documentação Mais Eficiente	DME
Maior Auxílio no Entendimento do Problema	MAEP
Maior Auxílio na Resolução do Problema	MARP

o fato de que em todos os dados comparativos o RAMBUS ser superior a soma das outras hipóteses foi uma agradável surpresa. A título de exemplificação, no dado onde o desempenho do RAMBUS foi “pior” (MSD) obteve 7 (sete) opiniões favoráveis, contra 6 (seis) opiniões contrárias, em relação a soma do Scrum (H2) com a opção Iguais (H0). Isso representa cerca de 53.85% do total das opiniões dos participantes. E no dado onde foi obtido o “melhor” desempenho (MAEP), o resultado de opiniões favoráveis ao RAMBUS foi de 9 (nove) contra 4 (quatro), por volta de 69.23% do total das opiniões.

**Figura 6.7: Opinião dos alunos sobre os processos em relação ao itens da Tabela 6.6.**

Finalmente, considerando que o experimento foi realizado *in-vitro* sob condições controladas, é importante manter em mente que as conclusões a respeito dos resultados observados neste trabalho se restringem ao escopo de desenvolvedores de software em ambiente universitário no qual o estudo foi conduzido. Por questões de validade, para expandir a generalização da avaliação para um contexto mais amplo, é necessário que novos estudos com um número maior de equipes sejam realizados em ambientes *in-vivo*, comparando-se também o uso do RAMBUS em relação a outras abordagens de desenvolvimento, tais como XP, FDD, Scrum com algumas variações, com outras ferramentas de TDD e BDD, e com outras linguagens de programação, por exemplo. Isto permitirá obter uma validação mais abrangente das hipóteses de pesquisa. A replicação deste experimento em ambiente industrial, portanto, torna-se importante já que outros fatores ausentes no ambiente acadêmico poderão ser controlados e estudados.

6.2.2.5 Avaliação Qualitativa

Entre os pontos positivos do processo RAMBUS está o fato que as Histórias do Usuário terem sido muito úteis para os participantes, que conseguiram utilizá-las de maneira eficiente para entender as funcionalidades a serem criadas e como fonte de testes para orientar a implementação. Os participantes gostaram do fato de serem orientados para implementar de acordo com a “vontade” do usuário, ou seja, criar as funcionalidades que o usuário mais valoriza e de visualizar essa criação de uma maneira mais próxima da visão do usuário.

Gostaram também do desenvolvimento ser “de fora para dentro”, ou seja, primeiro escreveram o código para as interfaces com o usuário – de maneira simples, sem CSS, AJAX e com HTML básico –, para depois escrever os códigos dos controladores e alguns acréscimos e/ou correções nos modelos que lidam com os dados no banco de dados. Vale salientar que o banco de dados foi criado a partir do Diagrama de Classes criado pelo experimentador e entregue aos participantes.

O maior problema relatado, em relação ao RAMBUS, foi a falta de uma forma mais simples de transpor dos requisitos (Cartões de História e Mock-ups) para a Modelagem (Histórias do Usuário executáveis). Apesar de essa etapa ter sido feita pelo experimentador, os participantes tiveram algumas oportunidades (após o experimento) de tentar transcrever alguns Cartões e Mock-ups para Histórias. Até o momento, isso é um processo puramente manual e dependente da compreensão dos desenvolvedores sobre a melhor forma de transcrever os dados.

É importante ressaltar que, na época do estudo de caso, as ferramentas de suporte desenvolvidas para o processo RAMBUS ainda se encontravam em fase de concepção. Por isso, as mesmas foram tratadas em uma outra avaliação.

6.2.2.6 Ameaças à validade

Uma das questões fundamentais envolvendo os resultados de um experimento é determinar quão válidos eles são. Por isso, é importante considerar a questão da validade da avaliação desde a fase de planejamento de forma a guiar a execução do experimento no sentido de obter uma validade adequada dos resultados.

Neste caso, validade adequada significa que os resultados são válidos para a população de interesse, tanto para o subconjunto populacional que executou o estudo, quanto para uma população mais ampla para a qual os resultados podem ser generalizados (WOHLIN et al., 2000). As principais ameaças que podem colocar em risco a validade de um experimento se classificam em: interna, de construção e externa. Essas ameaças foram tratadas conforme segue:

1) Validade Interna: tem relação com as questões que afetam a habilidade de assegurar que os resultados foram obtidos em decorrência dos tratamentos e não por uma coincidência. Exemplos disso são: o modo como os participantes são selecionados, recompensados e tratados.

No caso do estudo, na tentativa de minimizar esta ameaça, buscou-se realizar o experimento com estudantes da área de Computação no nível de pós-graduação (Mestrado e Doutorado). É viável acreditar que eles possuem um certo nível de experiência com desenvolvimento de software, conforme foi reportado em seus formulários de caracterização. Dessa forma, é possível assumir que eles são representativos para a população dos desenvolvedores de software. Além disso, os estudantes foram agrupados adequadamente conforme seus níveis de experiência para que os grupos ficassem homogêneos, minimizando diferenças na velocidade com que cada grupo executava os processos à medida que interagiam com o experimento. Nenhum tipo de recompensa ou favorecimentos quanto à nota da disciplina foi oferecida aos estudantes de modo a não criar expectativas e evitar que se comportassem com empenho diferente do normal durante o estudo. Por fim, o estudo foi realizado de forma que todos os participantes estivessem sob as mesmas circunstâncias durante a execução do mesmo.

Tudo isso ajuda a minimizar um pouco essa ameaça, mas não é o suficiente para anulá-la. Um experimento com pessoas mais experientes pode gerar resultados diferentes. Além disso, os participantes são alunos da pós-graduação e o experimento foi realizado em uma matéria na qual eles se matricularam. Isso pode gerar uma “vontade de agradar” por parte deles, pensando que se discordarem ou se não elogiarem o processo analisado pelo experimento eles podem ficar com notas menores. Isso pode mascarar o resultado encontrado ao analisar os dados.

2) Validade de Construção: tem relação com as questões que afetam a habilidade de generalizar o resultado do experimento ao conceito ou teoria envolvidos no estudo. Por exemplo, a definição adequada das medidas e tratamentos.

Visando minimizar essa ameaça, no estudo realizado o objetivo era comparar dois processos de desenvolvimento com respeito a seu impacto na segurança da equipe e na satisfação dos participantes com o processo. Para os propósitos do estudo, uma equipe segura é aquela que produz código e finaliza a implementação encontrando menos dificuldade (e maior facilidade para resolvê-los) no projeto e se sentindo mais confiante em relação ao trabalho realizado. Um processo é mais eficiente que outro, se a relação entre tempo de desenvolvimento e satisfação dos desenvolvedores do processo A for melhor que a do processo B.

Portanto, os dados para coleta e os tratamentos foram definidos de modo que fosse possível realizar adequadamente a análise do efeito na segurança das equipes e da satisfação dos participantes com os processos em conformidade com os objetivos do estudo. Além disso, a fim de

evitar interações dos participantes voltadas para maximização ou minimização dos dados de interesse (confiança, facilidade em resolver problemas, dificuldades, tempo, satisfação), evitou-se divulgar exatamente quais dados seriam analisados quando os mesmos foram informados dos objetivos.

Porém, isso não impede que essa ameaça esteja presente na avaliação. Outras medidas poderiam ser utilizadas no experimento. Neste caso, talvez os resultados poderiam ser diferentes. Deve ser considerado também que uma das medidas é a opinião de cada participante. Isso é uma medida subjetiva, podendo mudar de pessoa para pessoa e em um experimento com um número maior de participantes isso pode gerar dados totalmente diferentes dos apresentados no experimento.

3) Validade Externa: tem relação com as questões que afetam a habilidade de generalizar os resultados do experimento para um contexto mais amplo do que o que foi selecionado para o estudo. Neste caso, existem três riscos principais: (1) a escolha errada dos participantes; (2) conduzir o experimento em um ambiente inapropriado; e (3) desempenhar o estudo em um período de tempo que afete os resultados.

No caso dos estudos, para minimizar a ameaça, os participantes do estudo podem ser considerados representativos para a população dos desenvolvedores de software (risco 1). O experimento foi conduzido em um laboratório informatizado com equipamentos adequados, bem como com ferramentas e tecnologias de desenvolvimento atualizadas comumente empregadas em ambientes do mercado de trabalho, tais como a IDE NetBeans, a linguagem de programação Ruby e o *framework* Cucumber (risco 2). Com relação às questões temporais, o experimento foi planejado de forma que os participantes pudessem desempenhá-lo dentro de dois períodos (duas aulas, uma por semana) de 4 horas, evitando que os resultados fossem afetados devido ao desinteresse (no decorrer do tempo) ou ao desgaste excessivo dos participantes (risco 3).

Entretanto, os participantes poderiam ser outros, assim como a aplicação criada poderia ser uma aplicação real. Sendo assim, essa ameaça não é anulada. Por exemplo, o experimento poderia ser realizado em uma empresa, em um projeto real. Porém, realizar um experimento desse tipo é muito difícil, tanto por ser complicado analisá-lo quanto pela dificuldade de encontrar uma empresa disposta a ser “cobaia” de um processo de desenvolvimento novo. Deve ser considerado também que um projeto real pode sofrer inúmeras alterações, ser cancelado ou a empresa perder o contrato de desenvolvimento e o cliente levar o que foi criado até então para outra empresa.

6.3 Avaliação das ferramentas desenvolvidas

Para analisar as ferramentas criadas para dar suporte ao processo RAMBUS, alguns participantes do experimento anterior foram convidados a fazer uma avaliação, no primeiro semestre de 2012, sobre as ferramentas em comparação ao Cucumber. Devido ao tempo disponível, essa avaliação foi mais simples do que o experimento anterior. Esses participantes foram escolhidos por terem um bom conhecimento em desenvolvimento Web com a linguagem C#. São eles os participantes P1, P4, P6, P10, P11 e P13 da Tabela 6.2.

Divididos em três duplas (conforme mostra a Tabela 6.7), eles refizeram parte da aplicação groupware baseada no site TripAdvisor utilizando o processo RAMBUS (a Atividade 1 do experimento anterior). Para evitar confusões sobre o funcionamento do processo, o mesmo foi (re)explicado parte-a-parte. Primeiramente, eles reutilizaram a ferramenta Cucumber visando lembrar como era o funcionamento da mesma, para o caso de não a terem utilizado após o experimento relatado anteriormente. Em seguida, as duplas criaram a aplicação utilizando a linguagem C#, o ambiente Visual Studio e as ferramentas criadas (a DLL para o Visual Studio e o Story2CSharp Converter).

Tabela 6.7: Grupos e seus respectivos participantes para avaliar as ferramentas.

Grupo 1	Grupo 2	Grupo 3
P1 e P11	P4 e P10	P6 e P13

Foram elaboradas três hipóteses para a avaliação em relação a utilidade das ferramentas desenvolvidas para dar suporte ao processo RAMBUS. As hipóteses são:

- **Hipótese nula (H0):** Em geral, não há diferença de uso entre o *framework* Cucumber e as ferramentas Story2CSharp dentro do processo RAMBUS;
- **Hipótese alternativa (H1):** As ferramentas Story2CSharp são, em geral, mais úteis para as equipes no processo RAMBUS do que o *framework* Cucumber;
- **Hipótese alternativa (H2):** O *framework* Cucumber é, em geral, mais útil para as equipes no processo RAMBUS do que as ferramentas Story2CSharp.

Como o objetivo da avaliação foi verificar se as ferramentas Story2CSharp possuem no mínimo utilidade equivalente a ferramenta utilizada no experimento (Cucumber), os participantes puderam reutilizar os artefatos criados para a primeira parte da avaliação. Tais artefatos eram os Cartões de História, os Mock-ups, o Diagrama de Classes e as Histórias do Usuário.

No caso das Histórias do Usuário, elas foram adaptadas de acordo com a etapa Especificação de Testes, como apresentado na Seção 4.3. Na sequência, os participantes converteram as histórias para os testes funcionais utilizando o Story2CSharp Converter e copiaram os testes para dentro do projeto no Visual Studio. Por fim, eles utilizaram esses testes para guiar o desenvolvimento da aplicação, seguindo as disciplinas e as fases do RAMBUS.

A Tabela 6.8 mostra os resultados em relação ao tempo de criação das estruturas dos testes (sem considerar o código que deve testar a aplicação) e as suas respectivas linhas de códigos (LOC). Para o Cucumber, os participantes criaram todas as expressões regulares (ER) necessárias para os critérios de aceitação de cada cenário presente nas histórias. No caso das ferramentas Story2CSharp, os participantes adaptaram as Histórias do Usuário para descrevê-las conforme a LET e depois convertê-las para os testes funcionais. O tempo dedicado ao aprendizado das ferramentas não foi incluído.

Tabela 6.8: Comparação do tempo de criação das estruturas dos testes e as linhas de código entre o Cucumber e as ferramentas Story2CSharp.

		Tempo de criação das estruturas dos testes	LOC
Grupo 1	Cucumber	36min	31
	Story2CSharp	6min	83
Grupo 2	Cucumber	31min	37
	Story2CSharp	5min	87
Grupo 3	Cucumber	39min	34
	Story2CSharp	8min	85
Média	Cucumber	35min	34
	Story2CSharp	6min	85

Como pode ser visto na tabela, o tempo médio gasto pelos grupos para criar as estruturas dos testes funcionais utilizando o Cucumber foi de 31 minutos, enquanto que a mesma atividade durou em média 6 minutos utilizando as ferramentas Story2CSharp. Isso se deve a dois fatores: (1) os testes serem convertidos automaticamente para testes funcionais uma vez escritos na LET; e (2) o tempo que os grupos levam para analisar uma história e verificar como criar uma expressão regular que atenda o maior número de critérios de aceitação possível (considerando que esses critérios também devem possuir o código a ser testado parecido).

Com relação as linhas de código, parte do aumento se deve ao fato da conversão passar pelo nome da história, pela parte referente a documentação da mesma e pelo nome de cada cenário. O Cucumber “ignora” esses dados, sendo que a criação de expressões regulares é feita apenas para os critérios de aceitação. A Figura 6.8 mostra um exemplo de uma expressão regular no Cucumber, para o critério de aceitação “*Given I have entered 50 into the calculator*”.

A expressão regular da Figura 6.8 é válida apenas para o critério de aceitação na condição

```
Given /I have entered (.*) into the calculator/ do  
...  
end
```

Figura 6.8: Expressão Regular escrita manualmente para utilizar o Cucumber.

inicial *Given*. Isso significa essa “frase” é global para as condições iniciais, mas não para as operações ou para os resultados. Portanto, é válido generalizar que quanto mais Histórias de Usuário existirem no projeto, mais ERs devem ser escritas (e essa escrita é manual) e gerenciadas, pois uma alteração em uma História pode causar alterações em N expressões regulares.

Em compensação, utilizando as ferramentas desenvolvidas para suporte ao processo RAMBUS, o tempo “gasto” para escrever e gerenciar essas Histórias é minimizado consideravelmente, devido a transformação automática das histórias nas estruturas dos testes funcionais.

Por fim, todos os participantes deram opiniões favoráveis em relação as ferramentas, considerando-as eficazes no que elas se propõem: utilizar as Histórias do Usuário criadas no processo RAMBUS para orientar a implementação das funcionalidades desejadas na aplicação groupware, através da transformação automatizada das Histórias do Usuário em testes funcionais. Dessa forma, a Hipótese Alternativa (H1), de que as ferramentas Story2CSharp são mais úteis para os desenvolvedores no processo RAMBUS é válida. Vale ressaltar que o tempo de criação do código interno dos testes (o código que testa a aplicação) não foi considerado por ser “igual” independente das ferramentas utilizadas, uma vez que testa a mesma lógica de negócio. O que difere é a sintaxe e a semântica das linguagens C# e Ruby, que não foi considerado nos tempos coletados e nas linhas de código.

6.4 Considerações Finais

Este capítulo apresentou a avaliação do trabalho desenvolvido. A avaliação compreendeu um estudo comparativo, seguindo a metodologia experimental, onde avaliou-se o impacto do uso do processo RAMBUS e do processo Scrum na satisfação da equipe de desenvolvimento. Apresentou, também, uma avaliação para investigar o comportamento das ferramentas desenvolvidas em um projeto de desenvolvimento.

Os resultados favoráveis obtidos com a avaliação do trabalho superam as expectativas iniciais do processo desenvolvido, que incluem redução dos esforços, aumento da produtividade e melhoria da qualidade das aplicações groupware desenvolvidas através do uso do TDD. Assim como os resultados com as ferramentas vão de encontro as expectativas iniciais, que incluem

criar uma forma de tratar testes funcionais em um ambiente mais amplo que o Ruby com uma eficiência parecida ao da ferramenta Cucumber.

Capítulo 7

CONCLUSÃO

Esse trabalho teve como objetivo contribuir com as pesquisas relacionadas ao desenvolvimento de aplicações Web, com foco no domínio de groupware da Web 2.0. Dessa forma, dá continuidade aos trabalhos desenvolvidos pelo Grupo de Engenharia de Software (GES)¹ do Departamento de Computação da UFSCar. A pesquisa contou com o apoio financeiro do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico)².

As contribuições e as limitações identificadas em relação ao trabalho desenvolvido, estão presentes na Seção 7.1. Na Seção 7.2, são discutidos possíveis trabalhos futuros, sinalizando o desfecho desta dissertação.

7.1 Contribuições e Limitações

O processo ágil para desenvolvimento Web proposto neste trabalho é uma forma de associar as disciplinas de Requisitos; Análise e Projeto; e Implementação e Testes; por meio dos artefatos que são altamente relacionados. Outra contribuição importante é que o trabalho mostra eficiência em garantir que o requisito solicitado pelo usuário foi de fato implementado, por causa do aprimoramento da comunicação possibilitado pela melhor utilização dos artefatos. Esse aprimoramento é possível principalmente devido ao uso das Histórias do Usuário. Elas simulam o comportamento do usuário, ou seja, como ele utilizaria a aplicação. Essa simulação é realizada através dos critérios de aceitação presentes em cada cenário da História do Usuário. Se esses critérios são satisfeitos, o usuário “aceita” a funcionalidade. Portanto ele valida que a funcionalidade foi criada corretamente.

¹<http://ges.dc.ufscar.br/>

²<http://www.cnpq.br/>

Quanto maior for o sistema a ser desenvolvido, maiores serão as dificuldades para lidar com o alto número de Cartões de História e criar os Mockups, e posteriormente utilizá-los para transcrever as Histórias do Usuário, visto que este procedimento demanda um bom apoio visual. Nesse sentido, o processo proposto é adequado para aplicações Web de pequeno e médio porte. Atributos não funcionais são não-triviais de serem colocados em uma História do Usuário, limitando, também, o uso do processo proposto.

Em relação as ferramentas desenvolvidas, a abordagem proposta auxilia na criação de software, orientando o desenvolvimento com base no comportamento esperado pelo usuário. Isto é possível através da implementação das Histórias do Usuário escritas na Linguagem de Especificação de Testes (LET), que possui um alto nível de abstração. Essas histórias são a base da abordagem, descrevendo cada regra de negócio como um cartão, que evolui até os testes funcionais. Estes testes estão no nível de teste de aceitação e são executados para orientar a implementação. Para isso, duas ferramentas foram desenvolvidas para apoiar a abordagem: a biblioteca Story2CSharp, uma DLL para ser usada com o Visual Studio e o Story2CSharp Converter, que utiliza a DLL e auxilia na construção do teste funcional da História do Usuário escrita na LET.

Ao longo do desenvolvimento deste trabalho, foi observado que apesar do processo RAMBUS ter sido utilizado para desenvolver aplicações no domínio de *groupwares*, o mesmo pode ser generalizado para atender a outros domínios de aplicações Web. Para tal, os desenvolvedores devem utilizar os Cartões de História e os Mock-ups para especificar os requisitos e depois construir as Histórias do Usuário, utilizando as ferramentas de apoio à abordagem.

Em resumo, as principais contribuições deste trabalho são:

- Um processo de desenvolvimento ágil de aplicações Web que combina as práticas dos métodos ágeis, como Scrum, com atividades de testes;
- Suporte computacional constituído das ferramentas Story2CSharp: biblioteca (DLL) e concorsor; que apoiam o desenvolvedor na utilização do processo proposto; e
- A verificação de que é possível utilizar testes integrados com métodos ágeis, sem prejudicar a produtividade do time.

7.2 Trabalhos Futuros

No decorrer do desenvolvimento deste trabalho, algumas oportunidades de melhoria, tanto do processo quanto das ferramentas de suporte computacional, foram destacadas.

Os possíveis trabalhos futuros identificados incluem:

- Estudos sobre como aplicar o uso de testes funcionais do processo proposto nos níveis de testes de sistema e de integração. Uma possível forma de realizar isso seria reduzir o nível de abstração das Histórias do Usuário escritas na LET;
- Melhorias das ferramentas construídas para apoiar o processo. Um exemplo seria importar diretamente o código C# gerado no conversor para o Visual Studio;
- Uma análise mais detalhada para comprovar a viabilidade de utilizar a proposta de utilizar a “abordagem de testes funcionais” com outros processos, como citado no final do Capítulo 4; e
- A aplicação do processo RAMBUS em ambiente industrial para a construção de aplicações de maior porte, onde outros fatores potencialmente ausentes no ambiente acadêmico poderão ser controlados e estudados. Além disso, outros efeitos também poderão ser estudados, tais como qualidade e usabilidade dos artefatos desenvolvidos, por exemplo.

PUBLICAÇÕES

A partir do trabalho apresentado nesta dissertação, foram publicados os seguintes artigos:

- Pereira, V.; Prado, A. F. . Introducing a New Agile Development for Web Applications Using a Groupware as Example. In: 1st International Conference on Integrated Computing Technology, 2011, São Carlos, Brasil. Communications in Computer and Information Science. Springer-Verlag. v. 165. p. 144-160.
- Pereira, V.; Prado, A. F. . A New Agile Process for Web Development. In: 6th International Conference on Evaluation of Novel Software Approaches to Software Engineering, 2011, Pequim, China. ENASE 2011 - 6th International Conference on Evaluation of Novel Software Approaches to Software Engineering. SciTePress. p. 177-187.
- Pereira, V.; Prado, A. F. . RAMBUS: An Agile Process for Developing Web Applications. Journal of Intelligent Computing, v. 2, p. 42-53, 2011. Digital Information Research Foundation.
- Pereira, V.; Prado, A. F. . Providing Facilities for the Use of TDD in Practice. In: 8th International Conference on Web Information Systems and Technologies, 2012, Porto, Portugal. WEBIST 2012 - 8th International Conference on Web Information Systems and Technologies. SciTePress. p. 242-245.

REFERÊNCIAS

- ALTARAWNEH, H.; SHIEKH, A. E. A theoretical agile process framework for web applications development in small software firms. In: *6th International Conference on Software Engineering Research, Management and Applications*. Prague, Czech Republic: IEEE Computer Society, 2008. p. 125 – 132. ISBN 978-0-76-953302-5.
- ANDERSON, D. J. *Kanban: Successful Evolutionary Change for Your Technology Business*. First edition. Sequim, WA, United States of America: Blue Hole Press, 2010. ISBN 978-0-98-452140-1.
- ARES, J. et al. Formalising the software evaluation process. In: *XVIII International Conference of the Chilean Computer Science Society*. Chile: IEEE Computer Society, 1998. p. 15–24. ISBN 0-81-868616-2.
- BASILI, V.; GREEN, S. Software process evolution at the sel. *IEEE Software*, IEEE Computer Society Press, v. 11, n. 4, p. 58 – 66, 1994. ISSN 0740-7459.
- BECK, K. *Test-Driven Development by Example*. First edition. Boston, MA, United States of America: Addison-Wesley, 2003. ISBN 978-0-32-114653-3.
- BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change*. Second edition. Boston, MA, United States of America: Addison-Wesley, 2005. ISBN 978-0-32-127865-4.
- BERTOLINO, A. Software testing research: Achievements, challenges, dreams. In: *Future of Software Engineering*. Minneapolis, MN, United States of America: IEEE Computer Society, 2007. p. 85 – 103. ISBN 0-76-952829-5.
- BROWN, D. M. *Communicating Design: Developing Web Site Documentation for Design and Planning*. Second edition. Berkeley, CA, United States of America: New Riders, 2011. ISBN 978-0-13-138541-2.
- CARSTENSEN, P. H.; SCHMIDT, K. Computer supported cooperative work: New challenges to systems design. In: *Handbook of Human Factors*. [S.l.]: Asakura Publishing, 1999. p. 619 – 636.
- CERI, S.; FRATERNALI, P.; BONGIO, A. Web Modeling Language (WebML): A modeling language for designing web sites. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Elsevier North-Holland, Inc., Amsterdam, Netherlands, v. 3, n. 1 - 6, p. 137 – 157, 2000. ISSN 1389-1286.

- CHELIMSKY, D. et al. *The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends*. First edition. Lewisville, TX, United States of America: Pragmatic Bookshelf, 2010. ISBN 978-1-93-435637-1.
- COHEN, D.; LINDVALL, M.; COSTA, P. An introduction to agile methods. In: *Advances in Computers*. New York, NY, United States of America: Elsevier Science, 2004. v. 62, p. 1–66. ISBN 0-12-012162-X.
- COHN, M. *User Stories Applied: For Agile Software Development*. First edition. Boston, MA, United States of America: Addison-Wesley, 2004. ISBN 978-0-32-120568-1.
- DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao teste de software*. First edition. Rio de Janeiro, RJ, Brasil: Elsevier, 2007. ISBN 978-8-53-522634-8.
- DESHPANDE, Y.; HANSEN, S. Web engineering: Creating discipline among disciplines. *IEEE Multimedia*, IEEE Computer Society Press, v. 8, n. 2, p. 81–86, 2001. ISSN 1070-986X.
- D'SOUZA, D. F.; WILLS, A. C. *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach*. First edition. Reading, MA, United States of America: Addison-Wesley, 1999. ISBN 978-0-20-131012-2.
- ERIKSSON, H. E.; PENKER, M.; LYONS, B. *UML 2 toolkit*. Second edition. [S.l.]: Wiley Pub., 2004. ISBN 978-0-47-146361-0.
- FEATHERS, M. *Working Effectively with Legacy Code*. First edition. Upper Saddle River, NJ, United States of America: Prentice Hall PTR, 2004. ISBN 978-0-13-117705-5.
- FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. In: *22nd International Conference on Software Engineering*. Limerick, Ireland: ACM, 2000. p. 407–416. ISBN 1-58-113206-9.
- FOWLER, M. *Refactoring: Improving the design of existing code*. First edition. Reading, MA, United States of America: Addison-Wesley, 1999. ISBN 978-0-20-148567-7.
- FOWLER, M. *Domain-Specific Languages*. First edition. Upper Saddle River, NJ, United States of America: Addison-Wesley, 2011. ISBN 978-0-32-171294-3.
- GANSNER, E. R.; NORTH, S. C. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, v. 30, n. 11, p. 1203 – 1233, 2000.
- GASPAR, T. C.; YAGUINUMA, C. A.; PRADO, A. F. do. Desenvolvimento de aplicações colaborativas síncronas na web 2.0. In: *XV Simpósio Brasileiro de Sistemas Multimídia e Web*. Minicurso: Livro Texto. Fortaleza, Brasil: ACM, 2009. p. 168–207. ISBN 978-1-60-558880-3.
- GINIGE, A.; MURUGESAN, S. Web engineering: An introduction. *IEEE Multimedia*, IEEE Computer Society Press, v. 8, n. 1, p. 14–18, 2001. ISSN 1070-986X.
- IEEE. *IEEE standard glossary of software engineering terminology/IEEE Std 610.12-1990*. New York, NY, United States of America: Institute of Electrical and Electronics Engineers, 1990. Reaffirmed in 2002. ISBN 978-1-55-937067-7.
- JEFFRIES, R.; ANDERSON, A.; HENDRICKSON, C. *Extreme Programming Installed*. First edition. [S.l.]: Addison-Wesley Professional, 2000. ISBN 978-0-20-170842-4.

- KAPPEL, G. et al. *Web Engineering: The Discipline of Systematic Development of Web Applications*. First edition. Hoboken, NJ, United States of America: John Wiley and Sons, 2006. ISBN 978-0-47-001554-4.
- KOCH, N. et al. UML-based web engineering: An approach based on standards. In: *Web Engineering: Modelling and Implementing Web Applications*. First edition. London, England: Springer, 2008, (Human-Computer Interaction Series). cap. 7, p. 157 – 191. ISBN 978-1-84-628923-1.
- LEFF, A.; RAYFIELD, J. T. Web-application development using the model/view/controller design pattern. In: *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing (EDOC'01)*. [S.l.]: IEEE Computer Society, 2001. p. 118 – 127. ISBN 0-7695-1345-X.
- LEFFINGWELL, D. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. First edition. Upper Saddle River, NJ, United States of America: Addison-Wesley, 2011. ISBN 978-0-32-163584-6.
- MARTIN, J. *Managing the data-base environment*. Englewood Cliffs, NJ, EUA: Prentice-Hall, 1983. ISBN 0-13-550582-8.
- MARTIN, R. *Clean Code: A Handbook of Agile Software Craftsmanship*. First edition. Upper Saddle River, NJ, United States of America: Prentice Hall PTR, 2009. ISBN 978-0-13-235088-4.
- MUGRIDGE, R.; CUNNINGHAM, W. *Fit for Developing Software: Framework for Integrated Tests*. First edition. Upper Saddle River, NJ, United States of America: Prentice Hall PTR, 2005. (Robert C. Martin Series). ISBN 978-0-32-126934-8.
- NAWROCKI, J. R.; WALTER, B.; WOJCIECHOWSKI, A. Toward maturity model for extreme programming. In: *27th Euromicro Conference*. Warsaw, Poland: IEEE Computer Society, 2001. p. 233 – 239. ISBN 0-76-951236-4.
- NORRIE, M. C. PIM meets web 2.0. In: *27th International Conference on Conceptual Modeling*. Barcelona, Spain: Springer-Verlag, 2008. p. 15–25. ISBN 978-3-54-087876-6.
- NORTH, D. Behavior modification - the evolution of behavior-driven development. *Better Software*, March, 2006.
- PALMER, S.; FELSING, J. M. *A Practical Guide to Feature-Driven Development*. First edition. Upper Saddle River, NJ, United States of America: Prentice Hall PTR, 2002. ISBN 978-0-13-067615-3.
- PATEL, C.; RAMACHANDRAN, M. Story card based agile software development. *International Journal of Hybrid Information Technology*, SERSC, Tasmania, Australia, v. 2, n. 2, p. 125 – 140, 2009. ISSN 1738-9968.
- PAULK, M. C. et al. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA, United States of America: Addison-Wesley, 1995. ISBN 978-0-20-154664-4.

- PEREIRA, V.; PRADO, A. F. do. RAMBUS: An agile process for developing web applications. *Journal of Intelligent Computing*, Digital Information Research Foundation, Springfield, MO, United States of America, v. 2, n. 1, p. 42–53, 2011. ISSN 0976-9005.
- PEREIRA, V.; PRADO, A. F. do. Providing facilities for the use of TDD in practice. In: *8th International Conference on Web Information Systems and Technologies*. Porto, Portugal: SciTePress, 2012. p. 242 – 245. ISBN 978-9-89-856508-2.
- PERRY, W. *Effective methods for software testing*. Third edition. Indianapolis, IN, United States of America: Wiley, 2006. ISBN 978-0-76-459837-1.
- PFLEEGER, S. L.; ATLEE, J. M. *Software Engineering: Theory and Practice*. Forth edition. Upper Saddle River, NJ, United States of America: Prentice Hall, 2010. ISBN 978-0-13-606169-4.
- POWERS, M. K. *Microsoft Solutions Framework (MSF): A Pocket Guide*. First edition. Norwich, England: Van Haren Pub, 2006. ISBN 978-9-07-721216-5.
- PRESSMAN, R. S. What a tangled web we weave. *IEEE Software*, IEEE Computer Society Press, v. 17, n. 1, p. 18–21, 2000. ISSN 0740-7459.
- PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. Seventh edition. New York, NY, United States of America: McGraw-Hill Higher Education, 2010. ISBN 978-0-07-337597-7.
- PRESSMAN, R. S. et al. Can internet-based applications be engineered? *IEEE Software*, IEEE Computer Society Press, v. 15, n. 5, p. 104–110, 1998. ISSN 0740-7459.
- PRESSMAN, R. S.; LOWE, D. *Web Engineering: A Practitioner's Approach*. First edition. Boston, MA, United States of America: McGraw-Hill Higher Education, 2009. ISBN 978-0-07-352329-1.
- RAN, H. et al. Agile web development with web framework. In: *4th International Conference on Wireless Communications, Networking and Mobile Computing*. Dailan, China: IEEE, 2008. p. 1 – 4. ISBN 978-1-42-442107-7.
- ROSS, D. T. Structured analysis: A language for communicating ideas. *IEEE Transactions on Software Engineering*, IEEE Press, v. 3, n. 1, p. 16 – 34, 1977. ISSN 0098-5589.
- SCHWABER, K. *Agile Project Management with Scrum*. First edition. Redmond, WA, United States of America: Microsoft Press, 2004. ISBN 978-0-73-561993-7.
- SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. First edition. Upper Saddle River, NJ, United States of America: Prentice Hall, 2002. ISBN 978-0-13-067634-4.
- SLIGER, M.; BRODERICK, S. *The Software Project Manager's Bridge to Agility*. First edition. [S.l.]: Addison-Wesley Professional, 2008. ISBN 978-0-32-150275-9.
- TALBY, D. et al. A process-complete automatic acceptance testing framework. In: *IEEE International Conference on Software - Science, Technology and Engineering*. Herzelia, Israel: IEEE Computer Society, 2005. p. 129 – 138. ISBN 0-76-952335-8.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G. *Introdução à Engenharia de Software Experimental*. Rio de Janeiro, RJ, Brasil, 2002. Relatório Técnico RT-ES-590/02. Programa de Engenharia de Sistemas e Computação, UFRJ.

WOHLIN, C. et al. *Experimentation in Software Engineering: an introduction*. First edition. Boston, MA, United States of America: Kluwer Academic, 2000. ISBN 978-0-79-238682-7.

WYNNE, M.; HELLESØY, A. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. First edition. Dallas, TX, United States of America: Pragmatic Bookshelf, 2012. ISBN 978-1-93-435680-7.

YU, L. et al. Towards call for testing: An application to user acceptance testing of web applications. In: *33rd Annual IEEE International Computer Software and Applications Conference*. Seattle, WA, United States of America: IEEE Computer Society, 2009. v. 1, p. 166–171. ISBN 978-0-76-953726-9.

ZAKAS, N. C.; MCPEAK, J.; FAWCETT, J. *Professional AJAX*. First edition. Indianapolis, IN, United States of America: Wiley, 2007. ISBN 978-0-47-010949-6.

GLOSSÁRIO

AJAX – *Assynchronous JavaScript and XML*

API – *Application Programming Interface*

AUP – *Agile Unified Process*

AWDWF – *Agile Web Development with Web Framework*

BDD – *Behavior Driven Development*

CASE – *Computer Aided Software Engineering*

CFT – *Call for Testing*

CNPq – *Conselho Nacional de Desenvolvimento Científico e Tecnológico*

CRUD – *Create, Read, Update, Delete*

DBMS – *Database Management System*

DLL – *Dynamic-Link Library*

DSL – *Domain-Specific Language*

EBNF – *Extended Backus-Naur Form*

ERP – *Enterprise Resource Planning*

ER – *Expressoes Regulares*

FDD – *Feature Driven Development*

FIT – *Framework for Integrated Tests*

HTML – *HyperText Markup Language*

IDE – *Integrated Development Environment*

LET – *Linguagem de Especificacao de Testes*

- MSF4ASD** – *MSF for Agile Software Development*
- MSF** – *Microsoft Solutions Framework*
- MVC** – *Model-View-Controller*
- PC** – *Personal Computer*
- PMI** – *Project Management Institute*
- PPGCC** – *Programa de Pos-Graduacao em Ciencia da Computacao*
- RAMBUS** – *Rambus Agile Methodology Based on User Stories*
- SADT** – *Structured Analysis and Design Technique*
- SGBD** – *Sistema de Gerenciamento de Banco de Dados*
- T4** – *Text Template Transformation Toolkit*
- TDD** – *Test Driven Development*
- UFSCar** – *Universidade Federal de Sao Carlos*
- UML** – *Unified Modeling Language*
- UWE** – *Uml-based Web Engineering*
- VS** – *Visual Studio*
- WebE** – *Web Engineering*
- WebML** – *Web Modeling Language*
- XML** – *Extensible Markup Language*
- XPMM** – *eXtreme Programing Maturity Model*
- XP** – *eXtreme Programing*

Apendice A

CÓDIGO DOT DA LET

No código a seguir, é definido que o vértice *Story* será apresentado no grafo com duas linhas na elipse, para simbolizar a raiz. Os *aliases* indicam como o vértice pode ser referenciado no futuro código C#. Por sua vez, os vértices *Condition*, *Operation* e *Outcome* serão apresentados no grafo como elipses preenchidas.

Nas arestas (transições), é definido o nome de cada aresta e o nível de indentação. A aresta *Scenario* (*Feature ->Scenario* e *Outcome ->Scenario*) possui um *alias* para ter uma outra opção de nome, caso gere confusão diferenciar o vértice *Scenario* da aresta *Scenario*.

```
digraph Story2CSharp{

    //states
    Story [shape=doublecircle, aliases="Story is, Feature:"]
    Benefit
    Role
    Feature
    Scenario
    Condition [fillcolor=skyblue, style=filled]
    Operation [fillcolor=skyblue, style=filled]
    Outcome [fillcolor=skyblue, style=filled]

    //transitions - name and documentation
    Story -> Benefit [label="InOrderTo", indentlevel="1"]
    Benefit -> Benefit [label="And", indentlevel="2"]
```

```
Benefit -> Role [label="AsA", indentlevel="1"]
Role -> Role [label="OrAsA", indentlevel="2"]

//transitions - feature and scenario name
Role -> Feature [label="IWant", indentlevel="1"]
Feature -> Feature [label="And", indentlevel="2"]

Feature -> Scenario [label="Scenario", indentlevel="3",
                    aliases="WithScenario:"]

//transitions - acceptance criteria
Scenario -> Condition [label="Given", indentlevel="4"]
Condition -> Condition [label="And", indentlevel="5"]

Condition -> Operation [label="When", indentlevel="4"]
Operation -> Operation [label="And", indentlevel="5"]

Operation -> Outcome [label="Then", indentlevel="4"]
Outcome -> Outcome [label="And", indentlevel="5"]

//transitions - new scenario
Outcome -> Scenario [label="Scenario", indentlevel="3",
                    aliases="WithScenario:"]
}
```

Na sequência encontra-se o grafo direcionado, Figura A.1, que pode ser gerado a partir da descrição em DOT.

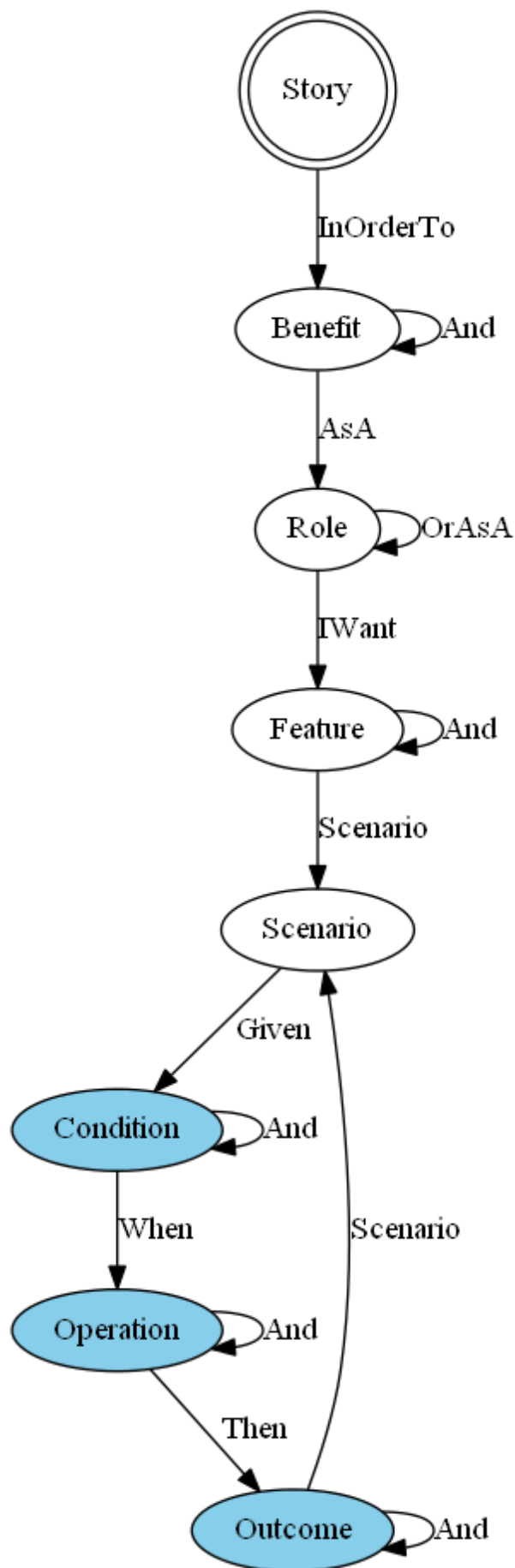


Figura A.1: Grafo direcionado gerado utilizando DOT.

Apendice B

TRECHO DO CÓDIGO DO *template* T4

Segue-se um trecho do código do *template* T4 escrito para gerar o código C#.

```
<#@ import namespace="Irony.Parsing"#>
<#@ import namespace="System.Text.RegularExpressions"#>
<#@ import namespace="System.Linq"#>
<#@ import namespace="System.Collections.Generic"#>
<#@ template language="C#" hostspecific="True" debug="True" #>
<#@ assembly name="$(SolutionDir)..\\lib\\development\\Flit.dll"#>
<#@ assembly name="$(SolutionDir)..\\lib\\development\\Irony.dll"#>
<#@ assembly name="System.Core.dll"#>
<#@ output extension=".g.cs" #>
<#
    var inputFile = GetParameter("inputFile",
        "FluentInterface\\Story2CSharp.txt");
    var nameSpace = GetParameter("nameSpace",
        "Story2CSharp").Replace('-', '_');

    var compiler = new Irony.Parsing.Parser(new FlitGrammar());
    ParseTree tree = compiler.Parse(System.IO.File
        .ReadAllText(Host.ResolvePath(inputFile)));
    ParseTreeNode node = tree.Root;
    Graph graph = node.AstNode as Graph;

#>
```

```

using System;
using System.ComponentModel;
using Story2CSharp.Infrastructure;

// tells the parser what our entry points are:
[assembly:ParserEntryPointAttribute(
    typeof(<#=nameSpace#>.Infrastructure.Story2CSharpEntryPoints))]

namespace <#=nameSpace#>
{
<#
    foreach(var state in graph.States)
    {
        string type = state.Label.Name;
#>

    /// <summary>
    /// The [<#=state.Label.Name#>] story fragment.
    /// <#=Tooltip(state)#>
    /// <h1>Transitions:</h1><ul>
<#
        foreach(var exit in state.OutgoingTransitions)
        {
            var target = exit.ToState;
            string prefix = UnCamel(exit.Label.Name);
#>

            /// <li><#=prefix#> [<see cref="<#=target.Label.Name#>"/>]: <see cref =
                "<#=exit.Label.Name#" /> </li> <# }
#>

        /// </ul>
        /// </summary>
        public class <#=state.Label.Name#> : FragmentBase
        {
<#
            if (IsEntryPoint(state))

```

```

    {
        var text = GetAliases(state.GenericAttributes)
            .FirstOrDefault()??state.Label.Name; #>
    }
    /// <summary>
    /// Starts a new Story2CSharp <#=state.Label.Name#>.
    /// </summary>
    /// <param name="text"> The name of the new <#=state.Label.Name#>
    </param>

    public <#=state.Label.Name#>(string text) : base(new Step("<#=
        UnCamel(text)#>", 0, text, Step.DoNothing), null){} <#
    }
    else
    {
#>
...
<#
}
foreach(var exit in state.OutgoingTransitions)
    {
        var target = exit.ToState;
        string prefix = UnCamel(exit.Label.Name);
        ...
#>

    /// <summary>
    /// <#=prefix#> [<#=target.Label.Name#>].
    /// </summary>
    /// <param name="text">
    /// A textual description. This story fragment is not executable.
    /// </param>
    /// <returns>The next fragment of your story, a <see cref =
    "<#=target.Label.Name#>" /> </returns>

<#
        foreach(var alias in GetAliases(exit.GenericAttributes))
            {

```

```
#>
    [Alias("<#=#alias#>")]
<#
    }
#>
<#=#IsExecutable(target)?"protected":"public"#> <#=#target.Label.Name#>
    <#=#exit.Label.Name#>(string text)
{
    Step s = new Step("<#=#prefix#>", <#=#Indent(exit)#>, text,
        <#=#IsExecutable(target)?"null":"Step.DoNothing"#>);
    return new <#=#target.Label.Name#>(s, this);
}

<#
}
#>

    /// <summary>
    /// Adds a tag to this step. Tags can be used make disparate steps
    ///     searchable.
    /// </summary>
    /// <param name="tag"></param>
    /// <returns></returns>
    public <#=#state.Label.Name#> Tag(string tag)
    {
        ((IStepContainer)this).Step.Tags.Add(tag.Trim().Trim('#'));
        return this;
    }

}

<#
}
#>

...
}
...
```

Apendice C

TRECHO DO CÓDIGO C# GERADO

Segue-se um trecho do código C# gerado de acordo com o código mostrado no Apêndice B.

```
using System;
using System.ComponentModel;
using Story2CSharp.Infrastructure;

// tells the parser what our entry points are:
[assembly:ParserEntryPointAttribute(typeof(
    Story2CSharp.Infrastructure.Story2CSharpEntryPoints))]

namespace Story2CSharp
{
    /// <summary>
    /// The [Story] story fragment.
    /// This is the root item of any story
    /// <h1>Transitions:</h1><ul>
    /// <li>in order to [<see cref="Benefit"/>]: <see
        cref="InOrderTo(string)"/></li>
    /// </ul>
    /// </summary>
    public class Story : FragmentBase
    {
        /// <summary>
        /// Starts a new Story2CSharp Story.
```



```
    /// </summary>
    /// <param name="text"> The name of the new Story </param>

    public Story(string text) : base(new Step("story is", 0, text,
        Step.DoNothing), null) { }

    /// <summary>
    /// in order to [Benefit].
    /// </summary>
    /// <param name="text">
    /// A textual description. This story fragment is not executable.
    /// </param>
    /// <returns> The next fragment of your story, a <see cref="Benefit" />
    </returns>

    public Benefit InOrderTo(string text)
    {
        Step s = new Step("in order to", 1, text, Step.DoNothing);
        return new Benefit(s, this);
    }

    /// <summary>
    /// Adds a tag to this step. Tags can be used make disparate steps
    /// searchable.
    /// </summary>
    /// <param name="tag"></param>
    /// <returns></returns>
    public Story Tag(string tag)
    {
        ((IStepContainer)this).Step.Tags.Add(tag.Trim().Trim('#?'));
        return this;
    }
}
...
}
```