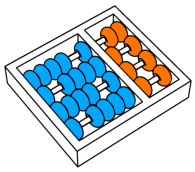
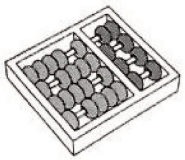


Alan Braz

"Método Ágil aplicado ao desenvolvimento de software confiável baseado em componentes"

CAMPINAS
2013



Universidade Estadual de Campinas
Instituto de Computação

Alan Braz

"Método Ágil aplicado ao desenvolvimento de software confiável baseado em componentes"

Orientador(a): **Profa. Dra. Cecília Mary Fischer Rubira**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

Este exemplar corresponde à versão final da Dissertação defendida por Alan Braz, sob orientação de Profa. Dra. Cecília Mary Fischer Rubira.

Cecília Mary Fischer Rubira

Assinatura do Orientador(a)

CAMPINAS
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

B739m

Braz, Alan, 1980-
Método Ágil aplicado ao desenvolvimento de software confiável baseado em componentes / Alan Braz. – Campinas, SP : [s.n.], 2013.

Orientador: Cecília Mary Fischer Rubira.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Desenvolvimento ágil de software. 2. Tolerância a falha (Computação). 3. Componente de software. 4. Software - Confiabilidade. 5. Tratamento de exceções. I. Rubira, Cecília Mary Fischer,1964-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Reliable component-based software development with Agile Method
Palavras-chave em inglês:
Agile software development
Fault-tolerant computing
Component software
Software - Reliability
Exception handling
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Cecília Mary Fischer Rubira [Orientador]
Regina Lúcia de Oliveira Moraes
Eliane Martins
Data de defesa: 23-05-2013
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 23 de Maio de 2013, pela Banca examinadora composta pelos Professores Doutores:

Profª. Drª. Regina Lúcia de Oliveira Moraes
FT / UNICAMP

Profª. Drª. Eliane Martins
IC / UNICAMP

Profª. Drª. Cecília Mary Fischer Rubira
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Método Ágil aplicado ao desenvolvimento de software confiável baseado em componentes

Alan Braz

23 de maio de 2013

Banca Examinadora:

- Profa. Dra. Cecília Mary Fischer Rubira (Orientadora)
- Profa. Dra. Regina Lúcia de Oliveira Moraes
  FT – UNICAMP
- Profa. Dra. Eliane Martins
  IC – UNICAMP
- Prof. Dr. Ivan Ricarte (Suplente)
  FEEC – UNICAMP
- Profa. Dra. Ariadne Maria Brito Rizzoni Carvalho (Suplente)
  IC – UNICAMP

Resumo

Os Métodos Ágeis, ou Desenvolvimento Ágil de Software (DAS), tem se popularizado, na última década, por meio de métodos como *Extreme Programming* (XP) e Scrum e isso fez com que fossem aplicadas no desenvolvimento de sistemas computacionais de diversos tamanhos, complexidades técnica e de domínio, e de rigor quanto à confiabilidade.

Esse fato evidencia a necessidade de processos de desenvolvimento de software que sejam mais rigorosos e que possuam uma quantidade adequada de modelagem e documentação, em especial no que concerne ao projeto arquitetural, com o objetivo de garantir maior qualidade no seu resultado final. A confiabilidade pode ser alcançada adicionando elementos de tratamento de exceções às fases iniciais do processo de desenvolvimento e à reutilização de componentes.

O tratamento de exceções tem sido uma técnica muito utilizada na verificação e na depuração de erros em sistemas de software. O MDCE+ é um método que auxilia a modelagem do comportamento excepcional de sistemas baseados em componentes que, por ser centrado na arquitetura, melhora a definição e a análise do fluxo de exceções entre os componentes do sistema.

Este trabalho propõe uma solução para guiar o desenvolvimento de sistemas confiáveis baseados em componentes por meio da adição de práticas do MDCE+ ao Scrum, resultando no método Scrum+CE (Scrum com Comportamento Excepcional). Esse processo passa a expor os requisitos excepcionais em nível das Estórias de Usuário, adiciona testes de aceitação mais detalhados, obriga a criação do artefato de Arquitetura Inicial e adiciona um novo papel de Dono da Arquitetura.

Como forma de avaliar esse método proposto, foi realizado um experimento controlado com três equipes, que desenvolveram um sistema com requisitos de confiabilidade, utilizando Scrum e Scrum+CE. Foram coletadas métricas para comparar a eficiência do novo processo e o resultado obtido, com a utilização do Scrum+CE, foi a produção de software com melhor qualidade, porém com menor número de funcionalidades.

Abstract

Agile Software Development (ASD) has been on mainstream through methodologies such as *Extreme Programming* (XP) and Scrum in the last decade enabling them to be applied in the development of computer systems of various size, technical and domain complexity and degress of reliability.

This fact highlights the need for software development processes that are accurate and have an adequate amount of modeling and documentation, especially regarding the architectural design, aiming to increase the quality of the end result.

The reliability can be achieved by adding elements of exception handling at early stages of development and through components reuse. Exception handling has been a widely used technique in detecting and fixing errors in software systems. The MDCE+ is a method that assists exceptional behavior modeling at components based systems, which is architecture-centric what improves the definition and flow analysis of exceptions between system components.

This paper proposes a solution to guide the development of reliable systems based on components by adding MDCE+ practices to Scrum, resulting in the Scrum+CE method (Scrum with Exceptional Behavior). This process exposes the exceptional requirements, at the User Stories level, documents acceptance tests with more details, requires the creation of a high-level architecture artifact and adds a new role of Architecture Owner.

In order to evaluate this proposed method, a controlled experiment was conducted with three teams, who developed a system with reliability requirements using Scrum and Scrum+CE. We collected metrics to compare the efficiency of the new process and the result was the production of software with better quality but with less features using Scrum+CE.

Agradecimentos

Gostaria de dedicar esta dissertação de mestrado à memória de meu pai, Benur Braz, que teve a visão de em 1990, quando eu tinha apenas 9 anos, me dar de presente um PC 286 e me incentivar a aprender à usá-lo e desenvolver meu gosto pela tecnologia.

Agradeço muito às duas mulheres que formam minha base, minha mãe, Maria de Lourdes, por sempre acreditar em mim, torcer, rezar e dizer que "vai dar tudo certo". E minha esposa, Gabriela, por me acompanhar desde antes da graduação até este momento, sempre colocando meus pés no chão e proporcionando longas conversas e divagações, além de não me deixar sequer pensar em desistir!

Muito obrigado ao meu irmão Vitor, ao meu sogro Genaro, à minha sogra Célia e à tia Carmen, por estarem presentes em minha defesa e darem aquele suporte neste momento tão especial e tenso. Foi muito importante tê-los ao vivo me apoiando. Um agradecimento especial também à minha cunhada Juliana, que me deu várias dicas (e algumas broncas) durante todo o processo, mas sempre muito carinhosa, atenciosa e sempre querendo ajudar. E também à amiga Adriana Camargo pela presença e troca de experiências.

Gostaria de salientar o apoio da IBM, que como empresa me deu a flexibilidade necessária para ir à aulas durante o horário comercial, além das reuniões com os orientadores e professores. Ao Marcelo Vessoni, por me inspirar e guiar no início desse processo, me motivando e mostrando a importância da continuação da carreira acadêmica no contexto da maturidade profissional. Ainda obrigado aos colegas, Leandro, Jair, Gabriel, Marco, Wander, Renato, Kleber, Lucas, Guilherme, Fábio, Cristino, e Leonardo, que aceitaram participar do treinamento que viabilizou o experimento deste trabalho. Valeu pessoal.

Obrigado especial para minha orientadora, Profa. Dra. Cecília Rubira, por ter me aceitado em uma situação não usual e embarcado no tema que sugeri. Obrigado ao Prof. Dr. Marco Vieira, da Universidade de Coimbra, por todo apoio, sugestões e ajuda na modelagem do experimento. Agradeço também aos professores da minha banca de qualificação: Prof. Dr. Hans Liesenberg e Profa. Dra. Eliane Martins, que também foi da minha banca de defesa, juntamente com a Profa. Dra. Regina Lúcia de Oliveira Moraes. Obrigado pelos comentários e recomendações. Um agradecimento especial também a Profa. Dra. Ariadne Carvalho pelos convites de aulas introdutórias para as turmas de

graduação e pelo WTD 2011.

Agradeço ao Patrick Brito por ser solícito e ajudar a entender como relacionar o MDCE+ ao meu trabalho.

Obrigado também ao pessoal do LSD (Laboratório de Sistemas Distribuídos), Leonardo Tizzei, Amanda, Marcelo, Rodrigo e Rosana, por todos os papos, dicas, links, impressões, cafés e apoio.

Por fim, gostaria de agradecer ao coordenador do Programa de Pós-Graduação, Prof. Dr. Paulo Lício de Geus, e aos funcionários da secretaria, Daniel Capeleto e Denilson pela paciência e profissionalismo.

Lista de Acrônimos e Siglas

ASD *Agile Software Development*

ATAM *Architecture Trade-off Analysis Method*

CASE *Computer Aided Software Engineering*

CBSE *Component-Based Software Engineering*

CC Complexidade Ciclomática

COTS *Commercial Off-The-Shelf Systems*

CRC *Class Responsibility Collaboration*

CVS *Concurrent Versions System*

DAS Desenvolvimento Ágil de Software

DBC Desenvolvimento Baseado em Componentes

HTTP *Hypertext Transfer Protocol*

IDE *Integrated Development Environment*

JPA *Java Persistence API*

JSON *JavaScript Object Notation*

MDCE Metodologia para Definição do Comportamento Excepcional

MER Modelo Entidade-Relacional

MVC *Model-View Controler*

POJO *Plain Old Java Objects*

REST *REpresentational State Transfer*

RTC *Rational Team Concert*

RUP *Rational Unified Process*

SRE *Software Reliability Engineering*

TDD *Test Driven Development*

UML *Unified Modeling Language*

URL *Uniform Resource Locator*

XP *Extreme Programming*

Sumário

Resumo ix

Abstract xi

Agradecimentos xiii

Lista de Acrônimos e Siglas xv

1 Introdução 1
- 1.1 Trabalhos relacionados . . . 4
- 1.2 Solução proposta . . . 5
- 1.3 Organização deste trabalho . . . 6

2 Fundamentos de Desenvolvimento Ágil de Software e reuso de software 9
- 2.1 Desenvolvimento Ágil de Software . . . 9
  - 2.1.1 Práticas Ágeis . . . 12
- 2.2 Método Scrum . . . 17
- 2.3 Fases do Scrum . . . 18
  - 2.3.1 Fase Pré-jogo . . . 19
  - 2.3.2 Fase Jogo . . . 22
  - 2.3.3 Fase Pós-jogo . . . 22
- 2.4 Pilares do Scrum . . . 24
  - 2.4.1 Transparência . . . 24
  - 2.4.2 Inspeção . . . 24
  - 2.4.3 Adaptação . . . 24
- 2.5 Artefatos do Scrum . . . 25
  - 2.5.1 *Backlog* do Produto . . . 25
  - 2.5.2 Objetivo do *Sprint* . . . 25
  - 2.5.3 *Backlog* do *Sprint* . . . 26
  - 2.5.4 Incremento . . . 26

2.6 Equipe do Scrum . . . 27
2.6.1 Dono do Produto . . . 27
2.6.2 Equipe de Desenvolvimento . . . 28
2.6.3 Scrum Master . . . 29
2.7 Eventos do Scrum . . . 30
2.7.1 *Sprint* . . . 30
2.7.2 Reunião de Planejamento do *Sprint* . . . 31
2.7.3 Scrum Diário . . . 32
2.7.4 Reunião de Revisão do *Sprint* . . . 33
2.7.5 Retrospectiva do *Sprint* . . . 33
2.8 Desenvolvimento Baseado em Componentes . . . 34
2.8.1 Processo *UML Components* . . . 37
2.9 Desenvolvimento Centrado na Arquitetura . . . 39
2.10 Tolerância a falhas e tratamento de exceções . . . 40
2.10.1 Falha, erro e defeito . . . 40
2.10.2 Confiabilidade de sistemas de software . . . 41
2.10.3 Componente tolerante a falhas ideal . . . 41
2.11 Metodologia para Definição do Comportamento Excepcional (MDCE+) . . . 43
2.11.1 FASE 1: Especificação e análise dos requisitos . . . 43
2.11.2 FASE 2: Definição dos aspectos gerenciais . . . 44
2.11.3 FASE 3: Projeto arquitetural . . . 45
2.11.4 FASE 4: Análise do sistema . . . 45
2.11.5 FASE 5: Projeto do sistema . . . 46
2.11.6 FASE 6: Materialização dos componentes . . . 47
2.11.7 FASE 7: Integração dos componentes . . . 48
2.11.8 FASE 8: Implantação (entrega ao cliente) . . . 49
2.12 Resumo . . . 49

3 O Método Scrum+CE **51**
3.1 Visão geral do Método Scrum+CE . . . 51
3.2 Fases do Scrum+CE . . . 53
3.2.1 Fase Pré-jogo . . . 53
3.2.2 Fase Jogo . . . 56
3.2.3 Fase Pós-jogo . . . 57
3.3 Pilares do Scrum+CE . . . 57
3.4 Artefatos do Scrum+CE . . . 57
3.4.1 Arquitetura Inicial . . . 57
3.4.2 *Backlog* do Produto . . . 58
3.4.3 Objetivo do *Sprint* . . . 59

3.4.4 *Backlog* do *Sprint* . . . 59
3.4.5 Incremento . . . 59
3.5 Equipe do Scrum+CE . . . 59
3.5.1 Dono da Arquitetura . . . 60
3.5.2 Dono do Produto . . . 60
3.5.3 Equipe de Desenvolvimento . . . 60
3.5.4 Scrum Master . . . 60
3.6 Eventos do Scrum+CE . . . 61
3.6.1 *Sprint* . . . 61
3.6.2 Reunião de Planejamento do *Sprint* . . . 61
3.6.3 Scrum Diário . . . 64
3.6.4 Reunião de Revisão do *Sprint* . . . 64
3.6.5 Retrospectiva do *Sprint* . . . 65
3.7 Resumo . . . 65

4 Experimento controlado **67**
4.1 Descrição do problema . . . 67
4.2 Definição do experimento . . . 68
4.3 Preparação do experimento . . . 71
4.4 Execução do Pré-jogo . . . 73
4.4.1 Planejamento . . . 73
4.4.2 Arquitetura Inicial . . . 77
4.5 Execução do Jogo . . . 79
4.5.1 *Sprint* 1 . . . 80
4.5.2 *Sprint* 2 . . . 84
4.6 Execução do Pós-jogo . . . 90
4.6.1 Métricas de entrega dos requisitos . . . 90
4.6.2 Métricas de qualidade dos requisitos . . . 90
4.6.3 Métricas de qualidade de código . . . 91
4.7 Análise dos resultados . . . 92
4.7.1 Requisitos entregues . . . 92
4.7.2 Qualidade do sistema . . . 93
4.7.3 Abandono de um participante . . . 95
4.8 Ameaças à validade do experimento . . . 96
4.8.1 Instrumentação . . . 96
4.8.2 Testagem . . . 96
4.8.3 Maturação . . . 97
4.8.4 História . . . 97

4.8.5 Mortalidade seletiva . . . 97
4.8.6 Contaminação . . . 97
4.8.7 Comportamento competitivo . . . 98
4.8.8 Comportamento compensatório . . . 98
4.8.9 Regressão à média . . . 98
4.8.10 Efeito de expectativa do sujeito, efeito placebo ou Hawthorne . . . 98
4.8.11 Efeito de expectativa do experimentador . . . 98
4.8.12 Avaliação subjetiva do pesquisador . . . 99
4.9 Resumo . . . 99

5 Conclusões e trabalhos futuros **101**
5.1 Contribuições . . . 102
5.2 Trabalhos futuros . . . 103
5.3 Publicações . . . 104

Referências Bibliográficas **105**

A Formulário de inscrição no experimento **109**
A.1 Formulário de inscrição . . . 109
A.2 Respostas recebidas . . . 111
A.3 Cálculo do índice técnico . . . 113

B Estórias de Usuário com Testes de Aceitação **115**
B.1 Estória 1 . . . 115
B.2 Estória 2 . . . 117
B.3 Estória 3 . . . 118
B.4 Estória 4 . . . 119
B.5 Estória 5 . . . 120
B.6 Estória 6 . . . 121
B.7 Estória 7 . . . 122
B.8 Estória 8 . . . 123
B.9 Estória 9 . . . 124
B.10 Estória 10 . . . 125
B.11 Estória 11 . . . 125
B.12 Estória 12 . . . 126
B.13 Estória 13 . . . 126

Lista de Tabelas

2.1 Comparação entre metodologias [26]. 18
2.2 Exemplo de *Visão do Produto*. 21
2.3 Exemplos de Objetivo do *Sprint*. 26
2.4 Exemplo de *Backlog* do *Sprint* com estimativas (em horas) por dia. 27
2.5 Características de componente [28]. 35
2.6 Abordagens para a implementação de sistemas confiáveis. 42

3.1 Relação entre as fases do MDCE+ e do Scrum. 54

4.1 Visão do sistema de itinerários de transporte corporativo. 68
4.2 Papéis dos usuários. 68
4.3 *Backlog* do Produto ordenado e estimado em Pontos de Estória. 69
4.4 Objetivo do experimento. 70
4.5 Modelagem do experimento. 71
4.6 Fatores de redução para viabilizar simulação. 71
4.7 Mapeamento dos papéis aos participantes do experimento. 74
4.8 Distribuição dos participantes em grupos. 75
4.9 Atividades de cada dia do *Sprint* 1. 81
4.10 Estórias selecionadas durante o planejamento do primeiro *sprint*. 82
4.11 Estórias demonstradas durante a revisão do primeiro *sprint*. 84
4.12 Atividades de cada dia do *Sprint* 2. 85
4.13 Estórias selecionadas durante o planejamento do segundo *sprint*. 86
4.14 Estórias demonstradas durante a revisão do segundo *sprint*. 89
4.15 Estórias entregues por grupo do experimento. 90
4.16 Testes realizados e defeitos encontrados por grupo experimento. 91
4.17 Métricas de qualidade de código. 92
4.18 Quantidade de horas-homem e produtividade das equipes. 96

A.1 Respostas enviadas pelos candidatos ao experimento. 111
A.2 Respostas a pergunta 2.5. 112
A.3 Valores usados para o cálculo do índice técnico. 113

Lista de Figuras

1.1 Passos realizados para definição do novo método. 3
1.2 Interferências do método MDCE+ nas fases do Scrum. 6

2.1 Modelo de breve descrição de uma Estória de Usuário. 12
2.2 Fases do método Scrum traduzido e adaptado [26]. 19
2.3 Detalhamento da fase Jogo do Scrum. 23
2.4 Modelo de processo genérico de Desenvolvimento Baseado em Componentes (DBC) [28]. 36
2.5 Arquitetura adotada pelo *UML Components* [9]. 38
2.6 Fases do processo *UML Components* [9] em UML. 39
2.7 Ciclo entre a ocorrência de falha, erros e defeito [18]. 40
2.8 Estrutura do componente tolerante a falhas ideal. 42
2.9 Fases do método MDCE+ [8]. 44
2.10 Passos da atividade 5.1 Analisar a interação entre os componentes [8]. . . . 47

3.1 Interferências do método MDCE+ nas fases do Scrum. 52
3.2 Detalhamento da fase Jogo do Scrum+CE. 62

4.1 Diagrama das atividades do experimento. 72
4.2 Índice total dos grupos. 75
4.3 Arquitetura base utilizada pelo grupo de controle (G1). 78
4.4 Arquitetura, utilizada pelos grupos experimentais (G2 e G3), expondo os componentes excepcionais. 79
4.5 Modelo Entidade-Relacional (MER). 80
4.6 Total de Estórias e Pontos entregues. 91
4.7 Comparação das entregas entre G1 e G2, e entre G1 e G3. 93
4.8 Comparação dos testes entre G1 e G2, e entre G1 e G3. 94

Capítulo 1

Introdução

A Engenharia de Software vem sofrendo evoluções nos seus métodos, metodologias e práticas devido à crescente necessidade de se desenvolver sistemas em prazos menores, com custos reduzidos e qualidade satisfatória. Em contrapartida, a complexidade desses sistemas é cada vez maior, bem como a necessidade deles serem confiáveis.

Confiabilidade[1] não é um requisito não funcional inerente apenas a sistemas críticos, como por exemplo, um controlador de voo ou um sistema financeiro, mas sim uma característica que todos os sistemas de software necessitam para não expor informações sigilosas e manter, o máximo possível, um determinado serviço disponível. No contexto deste trabalho, o termo **confiabilidade** será utilizado no sentido de *reliability* que é a capacidade do sistema oferecer seus serviços e funcionalidades conforme suas especificações, ou seja, de forma correta.

Para resolver essas questões, ideias como reutilização de componentes por meio do Desenvolvimento Baseado em Componentes (DBC) [20] e do Desenvolvimento Centrado na Arquitetura [34] têm sido propostas e aplicadas com relativo sucesso. O DBC viabiliza a melhora da confiabilidade utilizando componentes previamente especificados, implementados e testados, para a construção de sistemas de software, que além disso, proporciona um ganho significativo na produtividade durante o ciclo de desenvolvimento. Já o Desenvolvimento Centrado na Arquitetura define, em alto nível, as unidades abstratas do sistema e como essas entidades se relacionam ou se conectam. O fato de ser centrado na arquitetura quer dizer que tais definições devem ser feitas o mais cedo possível dentro do ciclo de desenvolvimento para mitigar os riscos técnicos e, por fim, para aumentar a confiabilidade.

Seguindo essas abordagens, para construir sistemas confiáveis e tolerantes a falhas, o Metodologia para Definição do Comportamento Excepcional (MDCE) [14] foi criado e aplicado ao processo de desenvolvimento *Catalysis* [12]. Em seguida, o MDCE foi evoluído

[1] do inglês *Reliability*

para o MDCE+ [8] enfatizando a modelagem do comportamento excepcional centrado na arquitetura e adaptado, no mesmo trabalho, ao processo *UML Components* [9].

O MDCE+ é um método para o projeto e para a implementação do comportamento excepcional no DBC que apresenta diretrizes para a especificação do comportamento excepcional de um sistema nas fases de identificação de requisitos, análise e projeto, refinando principalmente as fases de projeto arquitetural e implementação. O processo *UML Components* é um processo de desenvolvimento baseado em componentes voltado para o desenvolvimento de sistemas de informação. Este processo adota uma arquitetura predefinida em quatro camadas, sendo que duas delas são destacadas durante o desenvolvimento: camada de sistema e camada de negócio. Por esse motivo, ele é um processo leve, de fácil utilização prática, iterativo e enfatiza que a especificação dos componentes deve utilizar as notações e diagramas da *Unified Modeling Language* (UML) [25].

No entanto, na última década, observou-se uma tendência do uso de métodos mais leves para o desenvolvimento e para o gerenciamento de projetos de software. O termo "leve" se refere à característica de priorizar o sistema de software a ser construído e a colaboração entre as pessoas envolvidas no processo, em detrimento das atividades de modelagem e documentação detalhadas e extensas. Isto ocorre, pois se parte do pressuposto que o escopo do projeto, ou seja, seus requisitos, vão sofrer alterações durante o período de desenvolvimento, portanto, tentar antever todos os requisitos antes de começar o desenvolvimento de código, passou a ser considerado uma forma de desperdício. Esse conjunto de princípios e práticas chamados de leves foi batizado em 2001, de Método Ágil, ou Desenvolvimento Ágil de Software (DAS), quando diversos profissionais se reuniram para formalizar o que já vinham executando há alguns anos, como os métodos XP [3] e Scrum [26], resultando no Manifesto para o Desenvolvimento Ágil de Software [5].

Porém, essa simplicidade é muitas vezes confundida com informalidade ou falta de rigor, principalmente no que diz respeito às atividades de modelagem, à documentação dos requisitos funcionais e não funcionais e ao projeto arquitetural. Essas diferenças entre o desenvolvimento ágil de sistemas de software quando comparado com o desenvolvimento considerado mais tradicional, ou orientado a planos, é a motivação deste trabalho que visa pesquisar aplicações de processos ágeis no desenvolvimento de sistemas baseados em componentes, centrados na arquitetura e, além disso, confiáveis.

Para isso é proposto uma combinação entre o método MDCE+ e o método ágil Scrum, denominada Scrum+CE (Scrum com Comportamento Excepcional). O Scrum é um método ágil que foi concebida como um estilo de gerenciamento de projetos em empresas de fabricação de automóveis e produtos de consumo. Em 1995, Ken Schwaber [26] publicou a definição de Scrum e ajudou a implantá-lo em desenvolvimento de software em todo o mundo. O Scrum integra conceitos de desenvolvimento de produtos [31], *Lean Software Development* [22] e desenvolvimento iterativo e incremental com o objetivo de

otimizar a previsibilidade e o controle de riscos.

O Scrum foi escolhido por ser um processo ágil amplamente utilizado pela indústria de desenvolvimento de software [32].

A definição do Scrum+CE foi executada em duas etapas conforme descrito na Figura 1.1. A primeira fase consistiu no estudo do MDCE e do MDCE+ e suas respectivas aplicações aos processos *Catalysis* e *UML Components*, marcados pelas setas 1 e 2. Na segunda etapa foi feito o mapeamento das atividades das fases do ciclo de desenvolvimento do MDCE+ para o Scrum, suportados pelas práticas de DBC, também provenientes da aplicação do MDCE+ ao *UML Components*, e do Desenvolvimento Centrado na Arquitetura. Esta etapa está sinalizada pelas setas com número 3 e foi suportada indiretamente pelas setas tracejadas.

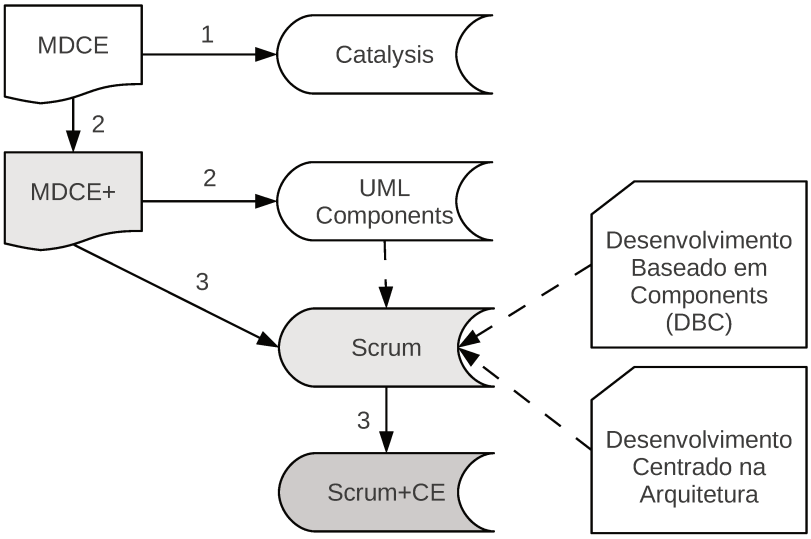


Figura 1.1: Passos realizados para definição do novo método.

Como um estudo de viabilidade do método Scrum+CE, foi realizado um experimento controlado para confrontar o uso da solução proposta ao Scrum tradicional. Esse experimento contou com três grupos de quatro participantes cada, que desenvolveram um sistema de informação com requisitos de confiabilidade. Os três grupos desenvolveram o mesmo sistema paralelamente, porém isolados entre si. Um dos grupos foi selecionado como grupo de controle e utilizou o Scrum e os outros dois grupos foram os experimentais e utilizaram o Scrum+CE.

O experimento teve duração fixa de duas semanas e, ao final desse período, foram analisadas as três soluções desenvolvidas e coletadas, as métricas sobre a quantidade de requisitos entregues, a qualidade dos requisitos e a qualidade do código. Uma vez coletadas as métricas, foi feita a análise dos resultados. Os resultados do experimento

foram satisfatórios ao uso do Scrum+CE que produziu sistemas com menor número de defeitos e, portanto, maior confiabilidade.

As contribuições desta pesquisa se situam na área de Engenharia de Software. Mais especificamente esse trabalho abrange o estudo de modelos, técnicas e processos para apoiar o uso do Método Ágil no desenvolvimento de sistemas de software confiáveis.

1.1 Trabalhos relacionados

Trabalhos anteriores já relataram a possibilidade de se desenvolver software confiável utilizando uma abordagem ou práticas ágeis.

Radinger e Goeschka [23], no artigo "*Agile Software Development for Component Based Software Engineering*", propuseram uma abordagem para integrar o Desenvolvimento Ágil de Software ao Desenvolvimento Baseado em Componentes em um **Desenvolvimento Ágil de Componentes**, considerando projetos de pequena e larga escalas combinando as questões técnicas e gerenciais de ambas as abordagens. Tal abordagem segue todas as fases do DBC, focando em um documento de arquitetura completo e detalhado do sistema, bem como, na análise e projeto igualmente detalhados. Na fase de provisionamento, se for tomada a decisão de que um componente deverá ser desenvolvido, sua construção deverá seguir uma abordagem ágil como se fosse um subprojeto. A pesquisa em questão se diferencia desse, pois considera o processo todo como ágil e não apenas um pedaço ou fase do ciclo de desenvolvimento.

Nord e Tomayko [21], no artigo "*Software Architecture-Centric Methods and Agile Development*", exploraram as relações e sinergias entre a análise e projeto centrados na arquitetura e no método ágil XP, destacando que o segundo enfatiza o desenvolvimento rápido e flexível, enquanto o primeiro prioriza o projeto e a infraestrutura. O trabalho desta pesquisa se diferencia desse, pois eles não criam um novo método, ou estendem o XP, eles apenas listam quais atividades da abordagem centrada na arquitetura se encaixariam nas atividades do XP, enquanto nós usamos o Scrum e propusemos um novo processo.

Behrouz [13] em seu artigo "*Software Reliability Engineering (SRE) for Agile Software Development*" mostra que, apesar de terem enfoques distintos, existe compatibilidade entre o desenvolvimento de software confiável e ágil, principalmente no que cerca os aspectos de medir e avaliar a confiabilidade do software desenvolvido. A discussão gira em torno da suposta incompatibilidade de que SRE executa testes no software criado com o objetivo de validar os parâmetros de confiabilidade previamente definidos e modelados. Por outro lado, o método ágil se apoia em uma abordagem dirigida a testes, *Test Driven Development* (TDD), em que os casos de testes são escritos antes do código e executados constantemente, preferencialmente de forma automática, durante o processo de desenvolvimento. Ele propõe um novo processo, chamado *Agile-SRE*, que mostra que é possível

desenvolver software confiável com uma abordagem ágil e que tal confiabilidade é atingida por meio do uso da prática TDD. No nosso caso do trabalho desta pesquisa, buscou-se aumentar a confiabilidade do sistema de software graças à antecipação, à documentação e à exposição na arquitetura do comportamento excepcional.

Patrick Brito [8], em sua dissertação de mestrado, definiu o MDCE+ e citou como uma das sugestões de melhoria na Seção de Trabalhos Futuros: "Uma possibilidade para agilizar o desenvolvimento dos sistemas seria estudar alguns processos ágeis e tentar introduzir algumas de suas características no MDCE+. Porém, deve-se ter sempre em mente que não se pode comprometer a confiabilidade do produto final produzido, uma vez que esse é o objetivo principal do método MDCE+." (BRITO, 2005, p. 171). Essa sugestão foi a principal motivação para este trabalho. Porém, ao invés de tentar adicionar práticas ágeis ao MDCE+, iremos adicionar práticas do MDCE+ ao método ágil Scrum.

1.2 Solução proposta

O objetivo principal deste trabalho é desenvolver a combinação do método que auxilia a construção de sistemas de software com requisitos de confiabilidade ligados à tolerância a falhas, chamado de MDCE+ [8], com um processo ágil de desenvolvimento de sistemas de software, o Scrum [26]. Isso foi feito por meio da utilização das tarefas de identificação de exceções e da especificação de seus tratadores nas fases de desenvolvimento do Scrum. O método resultante é chamado Scrum+CE (Scrum com Comportamento Excepcional) e possibilita o desenvolvimento de software confiável utilizando um método ágil.

O Scrum+CE adiciona novas práticas nas fases iniciais do Scrum no que diz respeito a elicitar e a formalizar os requisitos excepcionais no formato de Estórias Excepcionais e Testes de Aceitação mais criteriosos, e a refinar a arquitetura com elementos excepcionais explícitos. Também foram adicionados ao Scrum, a obrigatoriedade do artefato de "Arquitetura Inicial" e o novo papel de "Dono da Arquitetura".

A Figura 1.2 mostra as fases do Scrum com as atividades do MDCE+ (em cinza) que devem ser desenvolvidas nas respectivas fases. Note que as maiores modificações são na fase Pré-jogo, na qual o comportamento excepcional é documentado na forma de Estórias Excepcionais e de Testes de Aceitação nas Estórias de Usuário [10] que validem especificamente os fluxos excepcionais. Além disso, a Arquitetura Inicial passará a expor os componentes excepcionais.

Dentro da fase Jogo, teremos como entradas as Estórias de Usuário e Estórias Excepcionais igualmente priorizadas no *Backlog* do Produto. No Scrum, o Dono do Produto é o responsável por priorizar as estórias no *Backlog* do Produto por ordem de valor de negócio. Como as Estórias Excepcionais têm um aspecto não funcional e técnico, o novo papel de Dono da Arquitetura fica responsável por ajudar o Dono do Produto a priorizá-las de

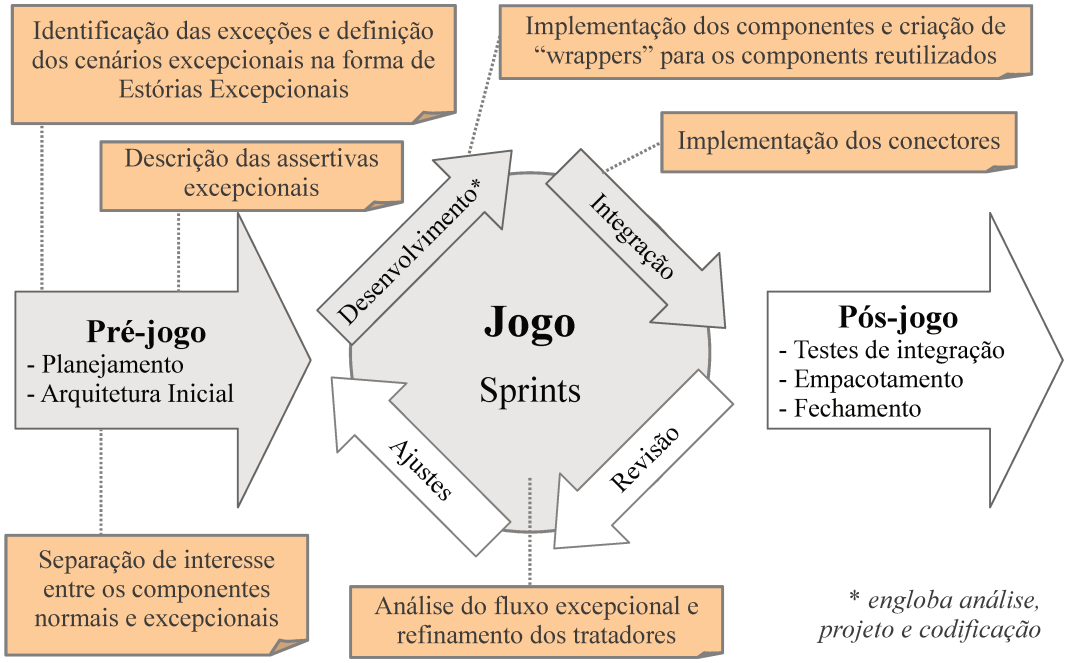


Figura 1.2: Interferências do método MDCE+ nas fases do Scrum.

forma coerente.

A combinação com o MDCE+ também afetará o artefato *Backlog* do *Sprint* com adição das tarefas específicas de implementação das exceções, oriundas das Estórias Excepcionais. Uma vez dentro dos *backlogs*, as estórias e tarefas excepcionais são tratadas e implementadas normalmente, como um requisito funcional comum.

1.3 Organização deste trabalho

Este documento foi dividido em 5 capítulos e dois apêndices, organizados da seguinte forma:

- **Capítulo 2 – Fundamentos de Desenvolvimento Ágil de Software e reuso de software**: O segundo capítulo apresenta os fundamentos teóricos necessários para embasar este trabalho. Esses conceitos estão separados em: Desenvolvimento Ágil de Software apresentando suas características gerais, seus valores e princípios, além de quatro práticas ágeis; o método Scrum, elicitando sua história, características e componentes principais (fases, pilares, artefatos, equipe, e eventos); o Desenvolvimento Baseado em Componentes (DBC), assim como o processo de desenvolvimento *UML Components*; os conceitos de Desenvolvimento Centrado na

Arquitetura; as definições de tolerância a falhas e tratamento de exceções; e, finalmente, é descrito modelo MDCE+.

- **Capítulo 3 – O Método Scrum+CE**: O terceiro capítulo apresenta o método proposto por este trabalho seguindo a mesma estrutura apresentada na **Seção 2.2 Método Scrum**, salientando as diferenças e as adições feitas para lidar com o comportamento excepcional e explicitá-lo.

- **Capítulo 4 – Experimento controlado**: O quarto capítulo apresenta, em detalhes, a preparação, a execução, a coleta e a análise dos resultados do experimento realizadas para validar a aplicação do Scrum+CE, e a comparação com o Scrum.

- **Capítulo 5 – Conclusões e trabalhos futuros**: Nesse capítulo são apresentadas as conclusões deste trabalho. Além disso, são apontadas as principais contribuições e algumas sugestões para trabalhos futuros.

- **Apêndice A – Formulário de inscrição no experimento**: esse apêndice mostra o formulário de inscrição utilizado pelos participantes do experimento, as respostas recebidas às perguntas desse formulário, bem como a fórmula utilizada para quantificar a experiência técnica dos participantes e para auxiliar na separação dos grupos.

- **Apêndice B – Estórias de Usuário com Testes de Aceitação**: por fim, esse apêndice lista as treze Estórias de Usuário que formaram o *Backlog* do Produto do projeto implementado pelas equipes do experimento do Capítulo 4.

Capítulo 2

Fundamentos de Desenvolvimento Ágil de Software e reuso de software

Este capítulo apresenta os fundamentos teóricos necessários para embasar este trabalho. Esses conceitos estão agrupados em cinco seções.

A Seção 2.1 introduz os conceitos do Desenvolvimento Ágil de Software apresentando suas características gerais, os seus valores e seus princípios, além de quatro práticas ágeis para documentação no formato de Estória de Usuário), estimativas em Pontos de Estória, desenvolvimento guiado a testes e integração contínua. A Seção 2.2 apresenta o método ágil Scrum, escolhida como base para a solução proposta neste trabalho, elicitando sua história, características e componentes principais. Todas as fases, pilares, artefatos, papéis e eventos do Scrum são detalhados entre as Seções 2.3 e 2.7.

A Seção 2.8 trata do conceito de Desenvolvimento Baseado em Componentes (DBC), assim como introduz o processo de desenvolvimento *UML Components*. A Seção 2.9 apresenta os conceitos de Desenvolvimento Centrado na Arquitetura e, finalmente, as Seções 2.10 e 2.11 mostram os conceitos de tolerância a falhas, tratamento de exceções, e a arquitetura adotada pelo método MDCE+ para a estruturação do comportamento excepcional do software.

2.1 Desenvolvimento Ágil de Software

Até o início da década de 90, havia um consenso de que um planejamento cuidadoso, a formalização da garantia da qualidade, o uso de ferramentas *Computer Aided Software Engineering* (CASE) na análise e projeto e o controle rigoroso do processo de desenvolvimento eram maneiras mais eficazes de produzir software com a qualidade adequada. O contexto da época era de sistemas de software grandes, de longa vida, desenvolvidos por equipes com grande número de pessoas, as quais ocasionalmente trabalhavam em

empresas ou até em geografias diferentes e que levavam anos para serem concluídos. Um contexto com tamanha complexidade justificava o uso de uma abordagem baseada em planos e *pesada*, em termos de quantidade e alto nível de detalhes de documentação.

Entretanto, em ambientes de equipes e projetos menores, tal abordagem passou a ser um fardo a ser carregado, prejudicando assim o desenvolvimento do software ao invés de guiá-lo, uma vez que o tempo gasto para determinar o que o sistema deveria fazer, e como fazer, acabava levando mais tempo do que o próprio desenvolvimento em si [28]. Além disso, o próprio dinamismo dos negócios resultava na constante mudança nos requisitos, o que gerava uma imensa quantidade de retrabalho e alterações muito onerosas nos planos do sistema, especificações e projeto. O impacto de tais alterações é notado nos cronogramas e custos dos projetos.

Com isso, começaram a ser propostas e testadas abordagens mais *leves*, no sentido de permitir que a equipe focasse no desenvolvimento do software ao invés do seu projeto e documentação. Nessas abordagens, a especificação, a análise, o desenvolvimento e a entrega do software seguem uma abordagem iterativa e incremental. Desta forma, o impacto das mudanças dos requisitos durante o ciclo de desenvolvimento seria minimizado, uma vez que os clientes podem alterar ou solicitar novas funcionalidades a serem atendidas nas próximas iterações do sistema.

Motivados por essas novas abordagens, 17 especialistas em desenvolvimento de software se reuniram em fevereiro de 2001, resultando na criação da *Agile Alliance*[1] e na declaração do **Manifesto para Desenvolvimento Ágil de Software** [5], ou simplesmente **Manifesto Ágil**. Nesse manifesto foram consolidados os valores e os princípios que os autores já vinham utilizando na prática e com isso ocorreu a popularização de vários métodos que respeitam tais valores e princípios, entre eles destacam-se o Scrum [26] [27] e XP [3] (ou Programação Extrema).

Outros métodos ágeis também ganharam sua importância e foram descritos no livro de Highsmith [16], são eles: *Crystal* [16], *Dynamic Systems Development Method* (DSDM) [16], *Feature-Driven Development* (FDD) [16], *Lean Development* (LD) [16] e *Adaptive Software Development* (ASD) [16].

Os quatro valores do Manisfesto Ágil passam a valorizar:

1. **Indivíduos e interações** mais que processos e ferramentas;
2. **Software em funcionamento** mais que documentação abrangente;
3. **Colaboração com o cliente** mais que negociação de contratos; e
4. **Responder à mudança** mais que seguir um plano.

[1] http://www.agilealliance.com/. Acessado em: 12 de out. 2012.

Os doze princípios transcritos a seguir expandem os valores de forma mais detalhada, dando mais ênfase aos valores dos itens à esquerda do que aos da direita, não lhes descartando a importância, mas apenas não os priorizando. Eles ditam que:

1. A maior prioridade deve ser satisfazer o cliente por meio da entrega contínua e adiantada de software com valor agregado;

2. As mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis ganham com as mudanças, ao visarem à vantagem competitiva para o cliente;

3. A entrega frequente software funcionando (instalado em um ambiente de testes de forma que o cliente possa validá-lo), de poucas semanas a poucos meses, com preferência a menor escala de tempo é fundamental;

4. As pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto;

5. Os projetos devem ser construídos em torno de indivíduos motivados. É fundamental proporcionar ambiente adequado, oferecer suporte necessário e confiar neles para fazer o trabalho;

6. A conversa face a face é o método mais eficiente e eficaz de transmitir informações, para e entre uma Equipe de desenvolvimento;

7. A medida primária de progresso é software funcional (demonstrável);

8. A promoção de um ambiente sustentável no qual clientes, desenvolvedores, usuários e outros envolvidos[2] sejam capazes de manter um ritmo de trabalho constante indefinidamente[3] é imprescindível;

9. A contínua atenção, a excelência técnica e o bom projeto[4] aumenta a agilidade;

10. A simplicidade – a arte de maximizar a quantidade de trabalho que não precisou ser feito – é essencial;

11. As melhores arquiteturas, requisitos e projetos emergem de times autoorganizáveis;

12. A equipe deve refletir, em intervalos regulares, sobre como se tornar mais eficaz e então refinar e ajustar seu comportamento.

[2]do inglês *stakeholders*
[3]do inglês *sustainable pace*
[4]do inglês *design*

Pela simplicidade de seus valores e enfoque na organização das equipes e aspectos gerenciais, tem-se a impressão que documentar os requisitos, definir um projeto arquitetural, fazer análise e projeto não são obrigatórios ou não precisam de nenhum tipo de rigor. Porém, um dos princípios dita que é necessário ter **"atenção contínua à excelência técnica e a um bom projeto"** salientando a importância de se ter um projeto satisfatório, o que só é possível com certo rigor e formalismo no projeto arquitetural, que por sua vez é baseado nos requisitos elicitados.

2.1.1 Práticas Ágeis

Nesta seção são descritas quatro práticas ágeis usadas em diversos processos, incluindo o próprio Scrum, apesar de elas não fazerem parte formal do processo.

2.1.1.1 Estórias de Usuário

Uma das técnicas ágeis mais usadas para elicitar e documentar os requisitos que compõem o *Backlog* do Produto é a de Estórias de Usuário[5], ou simplesmente estórias, que são uma forma simples de capturar os requisitos dos usuários durante um projeto de software como uma alternativa às especificações longas e extensas dos Casos de Uso.

Uma Estória de Usuário [10] é uma simples declaração daquilo que o usuário necessita, de uma funcionalidade de software, portanto ela é escrita sob a perspectiva do usuário. Logo, a linguagem usada deve ser relacionada ao negócio, não contendo jargões técnicos.

Essa técnica é derivada do XP, entretanto, pode e vem sendo usada como principal forma de elicitar requisitos em outros métodos ágeis, incluindo o Scrum.

Inicialmente, elas eram escritas em cartões de papel, os *Class Responsibility Collaboration* (CRC) *cards* [4], com dimensões de cinco por três polegadas, o que limitava fisicamente a quantidade de informação que uma estória comporta. Tais cartões, fazem parte dos 3 Cs que caracterizam as Estórias de Usuário: (i) *cartão*; (ii) *confirmação* e (iii) *conversação*.

O *cartão* contém a breve descrição ou declaração da estória em um de seus lados. Essa descrição foca no "quem", "o que" e "por que" de uma funcionalidade e não no "como", e sendo assim, ela segue um modelo consistente e padronizado, mostrado na Figura 2.1.

```
Como um [papel de usuário], eu quero [objetivo],
  para que eu possa [justificativa de negócio].
```

Figura 2.1: Modelo de breve descrição de uma Estória de Usuário.

[5]do inglês *User Stories*. A tradução para o português foi escolhida como **Estória** pelo fato do termo em inglês ser *Story* e não *History*.

Dois exemplos de estórias seriam:

1. Como um candidato, eu quero buscar uma vaga de emprego, para que eu evolua em minha carreira.

2. Como um recrutador, eu quero publicar uma vaga de emprego, para que eu possa preencher uma posição disponível.

Conforme o projeto e a equipe amadurecem, a parte da descrição "`para que eu possa [justificativa de negócio]`" passa a ser opcional, porém ela é muito importante, tanto para a parte técnica quanto para a de negócios lembrarem da razão pela qual tal estória foi documentada e sua utilização para os usuários.

A *confirmação* é definida para cada Estória de Usuário e, se usado um cartão físico, é formalizada no verso dele. É composta por sentenças de Testes de Aceitação, ou seja, frases cujas respostas possíveis sejam apenas Sim ou Não. Exemplos de Testes de Aceitação da estória número 1, do exemplo anterior seriam:

- Consigo buscar uma vaga por área de atuação?

- Consigo buscar uma vaga por palavra-chave?

- Consigo buscar uma vaga por cidade?

- As vagas encontradas estão ordenadas por data de abertura?

- A lista das vagas encontradas contém, no máximo, 15 vagas por página?

Os Testes de Aceitação são fundamentais na Reunião de Revisão do *Sprint* para guiar a validação do Dono do Produto sobre a completude e o comportamento correto da entrega da Estória de Usuário.

Por fim, a *conversação* acontece mais intensamente durante a Reunião de Planejamento dos *sprints*, quando efetivamente os requisitos são detalhados, as dúvidas são tiradas e as possíveis soluções de implementação são discutidas. É uma conversa real e verbal entre o Dono do Produto e toda a Equipe de Desenvolvimento. Esse momento é crucial para eliminar os riscos de comunicação e também para forçar a colaboração entre as partes.

Uma vez elicitadas as estórias que formam o *Backlog* do Produto, elas devem ser estimadas. A seguir será introduzida a técnica de Pontos de Estória para estimar o esforço de se implementar uma Estória de Usuário.

2.1.1.2 Pontos de Estória

Nenhum projeto de desenvolvimento de software sobrevive muito tempo sem começar a ser questionado: *quando o sistema ficará pronto?*. A melhor abordagem para se estimar uma estória seria a que:

- possibilite mudança dos requisitos assim que forem descobertos novos detalhes sobre uma estória;
- funcione para estórias grandes e pequenas;
- não seja onerosa, levando muito tempo para se estimar;
- proveja informações úteis sobre o progresso e o trabalho faltante;
- seja tolerante à imprecisão das estimativas; e
- possa ser utilizada no planejamento de *releases*.

A técnica que satisfaz cada um desses objetivos é a estimativa em Pontos de Estória[6]. Os Pontos de Estória [11], ou simplesmente pontos, são uma maneira rápida, simples e relativa de estimar o tamanho, a complexidade e a grandeza de uma determinada estória, isto é, cada equipe determina o que significa um ponto. Com isso, um ponto de estória não tem unidade e não pode ser comparada a uma hora, um dia, uma semana ou um mês.

Na fase Pré-jogo, durante a subfase de Planejamento, todas as estórias que compõem o *Backlog* do Produto devem ser estimadas em Pontos de Estória pela Equipe de Desenvolvimento. Isso dará uma noção de tamanho do projeto, e aliado com a Velocidade da Equipe, pode-se estimar a quantidade de *sprints* necessários na fase Jogo para implementar o *backlog*. A Velocidade é uma métrica simples e direta que representa os Pontos de Estória implementados por *sprint* de uma determinada Equipe de Desenvolvimento. Assim, como os pontos são relativos a uma determinada equipe, a velocidade, que é diretamente ligada aos pontos, também será relativa àquela equipe.

Durante as Reuniões de Planejamento dos *sprints*, as estimativas podem ser discutidas e modificadas, uma vez que, nesse momento, os detalhes das estórias são discutidos com o Dono do Produto e, com mais informações e detalhes, é possível ajustar a estimativa, tanto para mais quanto para menos. Como a própria Equipe de Desenvolvimento fez a estimativa inicial, ela tem total liberdade para ajustá-la a qualquer momento. Esse comportamento é totalmente natural em um processo no qual os requisitos são dinâmicos.

[6] do inglês *Story Points*

2.1.1.3 *Test Driven Development* (TDD)

Desenvolvimento dirigido por testes ou *Test Driven Development* é uma prática que utiliza testes para ajudar os desenvolvedores a tomar as decisões certas no momento certo. TDD não é sobre o teste e sim sobre o uso de testes para criar software de uma forma simples e incremental, melhorando assim, não só a qualidade e o projeto do software, mas também simplificando o processo de desenvolvimento.

A efetividade do TDD é que, com uma pequena mudança em sua dinâmica de desenvolvimento e algumas ferramentas adicionais, pode-se produzir software melhor e mais rápido [15].

TDD representa uma evolução simples na forma de como um software pode ser desenvolvido. É a percepção de que sistemas de software complexos podem ser desenvolvidos em pequenos incrementos que usam testes para conduzir o projeto e implementação do software.

O maior desafio em utilizar TDD é a mudança na maneira de pensar sobre a escrita de código. Ao invés de criar um projeto para lhe dizer como estruturar seu código, cria-se um teste que define como uma pequena parte do sistema deve funcionar. Esse teste impulsiona o projeto do código necessário para implementar a funcionalidade, liberando assim tomadas de decisões de projeto antecipadas e gerando um certo grau de liberdade para descobrir a melhor forma de estruturar o código. TDD também se concentra na criação do software de forma incremental. Não é importante obter o código certo da primeira vez, pois continuamente o código é refatorado assim como os testes, até que o software atenda aos requisitos.

De forma geral, o TDD é uma prática muito simples, que se baseia em dois conceitos principais: testes de unidade e de refatoração. TDD é basicamente composto pelas seguintes etapas:

1. Escreva um teste de unidade que defina como uma pequena parte do software deve se comportar;

2. Execute todos os testes de unidade existentes para validar que o novo teste esteja falhando. Apesar de parecer óbvio, este passo é muito importante para validar o conjunto de testes como um todo e para garantir que a funcionalidade desejada realmente não foi implementada ainda;

3. Escreva o código mais simples possível que implemente a funcionalidade nova ou melhoria;

4. Execute os testes novamente até que todos finalizem com sucesso. Caso algum falhe, volte ao passo anterior;

5. Limpe o código (refatore), pois agora que o código está funcionando corretamente, pode-se melhorar o projeto e remover qualquer duplicação ou outros problemas que foram introduzidos para que o teste adicionado seja executado com sucesso;

6. Repita esses passos até implementar todas as funcionalidades.

Um dos resultados dessa abordagem metódica e iterativa é a criação de um código enxuto, uma vez que se adiciona o código apenas o suficiente para implementar o comportamento desejado. Isso implica um aumento na possibilidade de satisfazer as exigências em tempo hábil, pois não se perde tempo desenvolvendo código que nunca será usado.

Outra vantagem evidente é que à medida que mais códigos são desenvolvidos, o número de testes cresce rapidamente compondo um conjunto de testes que pode ser executado a qualquer momento. Esse conjunto de testes é crucial para o processo de TDD, pois pode ser usado para monitorar constantemente o software e identificar imediatamente quando ocorrem problemas. Uma vez que se executa esse conjunto de testes continuamente durante o desenvolvimento, pode-se encontrar e corrigir defeitos muito mais rapidamente. Além disso, seguindo essa abordagem, tem-se um sistema com uma grande cobertura de testes, ou seja, praticamente todo o código estará relacionado a um teste de unidade, e isto acrescenta confiança para o processo de desenvolvimento.

2.1.1.4 Integração Contínua

Essa técnica é fundamental e muitos autores a consideram essencial para o desenvolvimento ágil. Ela consiste na integração de um novo trecho de código ao sistema, assim que ele esteja pronto. Desta forma, a qualquer momento do ciclo de desenvolvimento se tem uma versão integrada do sistema.

No caso de usar testes de unidade, pode-se automatizá-los e rodar todo o conjunto de testes como validação antes da integração efetivamente ocorrer. Dessa forma, a integração em si só ocorrerá se todos os testes passarem com sucesso. Caso contrário, um relatório com os testes falhos pode ser enviado para a equipe. Essa é uma prática muito poderosa, pois além de fazer testes de regressão continuamente, diminui o ciclo de detecção de defeitos.

A integração pode ser configurada para ocorrer no intervalo desejado e estar ligada ao repositório de versionamento de código-fonte. Essa frequência pode ser a cada atualização de código, diária ou semanal. A decisão vai depender da criticidade do projeto. Um modelo bem utilizado é o de *Nightly Builds* (ou "Construção Noturna") que ocorre de madrugada, quando teoricamente nenhum desenvolvedor está alterando o código, e provê, diariamente, uma versão testada e integrada do sistema, ou um relatório com as falhas inseridas no dia anterior.

Na seção, a seguir, são apresentados os conceitos básicos do método Scrum, que serão explorados mais detalhadamente no Capítulo 3.

2.2 Método Scrum

O Scrum assume que o processo de desenvolvimento de sistemas é um processo imprevisível e complicado que só pode ser descrito de maneira geral. Desta forma, define o processo de desenvolvimento de sistemas como um conjunto disperso de atividades, combinando conhecimento, ferramentas e técnicas com o melhor que uma equipe de desenvolvimento pode conceber na construção de sistemas. Uma vez que essas atividades são desacopladas, usam-se controles para gerenciar o processo e seus riscos.

O Scrum trata as partes do desenvolvimento de sistemas como "caixas-pretas controladas"[7]. Esse termo, que vem dos conceitos de controles de processos industriais aplicados ao desenvolvimento de sistemas, define um processo como "teórico" (ou completamente definido) ou "empírico" (ou caixa-preta). Quando um processo do tipo caixa-preta é tratado como se fosse complemente definido, resultados inesperados ocorrem [26]. A maioria dos processos de desenvolvimento de sistemas conhecidos não é complemente definido mas é tratado como tal, gerando resultados imprevisíveis.

A nova abordagem para o desenvolvimento de sistemas proposta pelo Scrum é baseada em processos de gerenciamento teóricos e empíricos, visando ao aprimoramento da abordagem iterativa e incremental e propondo uma nova forma de organizar e gerenciar a equipe de desenvolvimento.

O nome Scrum [31] é uma referência a uma formação do jogo de rúgbi, utilizada para reinício da partida, quando os atacantes de cada time se abraçam e têm que atuar em conjunto para empurrar o time adversário para conquistar a posse da bola. É uma metáfora que ilustra a forma como os membros de uma equipe de desenvolvimento devem se comportar e colaborar entre si para conquistar seus objetivos com qualidade. Tal inspiração faz sentido, já que foi criado observando-se que equipes pequenas e multidisciplinares produziam melhores resultados. Com isso, oferece uma abordagem empírica permitindo que os membros da equipe trabalhem independente e coordenadamente em um ambiente criativo, reconhecendo assim, o aspecto social e colaborativo da engenharia de software.

O processo é rápido, adaptável e auto-organizado[8] representando uma grande mudança no processo sequencial de desenvolvimento. Acredita-se que o software não deve ser desenvolvido como um típico processo industrial, que é repetitivo e baseado em parâmetros de entrada e saída previsíveis e descritivos.

[7] do inglês *controlled black boxes*
[8] do inglês *self-organizing*

A principal diferença entre o Scrum e as metodologias tradicionais como Cascata ou *Waterfall* [24], Espiral [6] e Iterativa, como o *Rational Unified Process* (RUP) [17]), é que a abordagem do Scrum assume que análise, projeto, desenvolvimento, testes e integração, que ocorrem em *sprints*, são imprevisíveis e, por conta disso, não devem ser descritos como uma sequência de atividades ordenadas e padronizadas. Porém, um mecanismo de controle é usado para gerenciar a imprevisibilidade e controlar o risco, resultando em uma maior flexibilidade e agilidade. A Tabela 2.1 compara as características primárias do Scrum com as das metodologias tradicionais.

Tabela 2.1: Comparação entre metodologias [26].

Característica	**Cascata**	**Espiral**	**Iterativo**	**Scrum**
Processo definido	Requerido	Requerido	Requerido	Planejamento e Fechamento
Definição do produto final	Durante o planejamento	Durante o planejamento	Durante o processo	Durante o processo
Custo do projeto	Durante o planejamento	Parcialmente variável	Durante o processo	Durante o processo
Data de entrega	Durante o planejamento	Parcialmente variável	Durante o processo	Durante o processo
Adaptação ao ambiente	Apenas no Planejamento	Principalmente no Planejamento	Ao final de cada iteração	**Sempre**
Flexibilidade e criatividade da equipe	Limitada (guia de regras)	Limitada (guia de regras)	Limitada (guia de regras)	**Livre durante as iterações**
Transferência de conhecimento	Treinamento prévio	Treinamento prévio	Treinamento prévio	**Colaborativa durante o projeto**
Probabilidade de sucesso	Baixa	Média/baixa	Média	**Alta**

A seguir estão listadas as fases, pilares, artefatos, papéis e eventos que constituem o processo Scrum.

2.3 Fases do Scrum

A Figura 2.2 ilustra as três fases que compõem o Scrum, são elas: Pré-jogo, Jogo e Pós-jogo. As fases Pré e Pós-jogo consistem em processos bem definidos, ou seja, têm entradas e saídas predefinidas e o conhecimento é explicito seguindo um fluxo linear, ainda que a etapa de Planejamento (parte do Pré-jogo) possa ser composta por algumas iterações.

A nomenclatura de fases do Scrum, no formato de Pré-jogo, Jogo e Pós-jogo, descrito em [26], foi escolhida por questões didáticas e de organização do texto. Na prática,

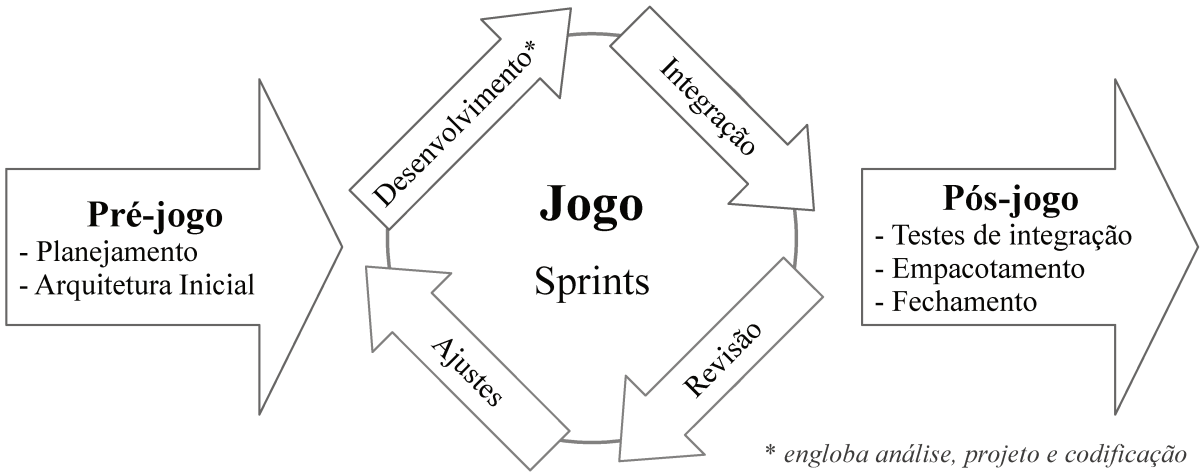


Figura 2.2: Fases do método Scrum traduzido e adaptado [26].

nas organizações como a *Agile Alliance*, e em outros processos ágeis, como o *Disciplined Agile Delivery (DAD)*[9], pode-se encontrar denominações da fase Pré-jogo como *Sprint 0* (zero), *Sprint Inicial* ou *Concepção*. A fase Jogo também é referenciada como *Sprints de Desenvolvimento*, ou *Contrução*, ou simplesmente, *Sprints*. E a fase Pós-jogo como *Sprint Final*, ou *Transição*.

A fase Jogo, composta por *sprints*, segue um processo empírico, ou seja, suas atividades são indefinidas e desordenadas, sendo tratada como uma "caixa-preta" submetida a controles externos, incluindo gerenciamento de riscos, que visa evitar o caos e maximizar a flexibilidade. Os *sprints* são não lineares e flexíveis, porém, podem fazer uso do conhecimento de processos explícitos quando disponíveis. Caso contrário, o conhecimento tácito e métodos de tentativa e erro são utilizados para evoluir o sistema até sua entrega final.

O ciclo de vida de desenvolvimento do projeto é visto como um ambiente aberto a mudanças de requisitos, a prazos, a custos, a qualidade ou a qualquer outra natureza até o Pós-jogo, desde que seja mantida a complexidade do projeto.

Uma *release* é definida pela disponibilização de uma versão do sistema de software, instalada em uma infraestrutura para utilização dos usuários finais. Isto ocorre ao final da fase Pós-jogo.

A seguir, as três fases são descritas em maiores detalhes.

2.3.1 Fase Pré-jogo

A fase Pré-jogo é dividida em duas atividades: (i) planejamento; e (ii) arquitetura e projeto iniciais em alto nível.

[9]Disponível em: http://disciplinedagiledelivery.wordpress.com/. Acessado em: 24 jul. 2013.

2.3.1.1 Planejamento

No planejamento, será definida uma nova *release* baseada nos requisitos do *Backlog* do Produto atual, ou nos requisitos levantados para compor um *backlog*, bem como nas suas estimativas de custo e prazo. Essa etapa é composta das seguintes atividades:

- Criação da *Visão do Produto* (Tabela 2.2);
- Elicitação dos requisitos que irão compor o *Backlog* do Produto;
- Definição da data de entrega e das funcionalidades-chave de uma ou mais *releases*;
- Seleção da *release* mais apropriada para o desenvolvimento imediato;
- Mapeamento dos itens do *backlog* necessários para implementar as funcionalidades da *release* selecionada;
- Definição dos membros da Equipe de Desenvolvimento, do Dono do Produto e do Scrum Master;
- Avaliação dos riscos e definição dos controles apropriados;
- Revisão e ajustes dos itens do *backlog* selecionados;
- Validação e preparação das ferramentas de desenvolvimento e infraestrutura necessária;
- Estimativas do custo da *release*, incluindo desenvolvimento, materiais adicionais, marketing, treinamento dos usuários, e lançamento;
- Definição da duração e da quantidade de *sprints* necessários para implementar os itens *backlog*;
- Estabelecimento da Definição de "*Pronto*"; e
- Aprovação gerencial e financeira.

2.3.1.2 Definição de "*Pronto*"

A definição de "*Pronto*"[10] tem um papel fundamental no Scrum, visto que ela é um consenso sobre quais atributos devem ser contemplados para que se declare um item do *Backlog* do Produto ou um Incremento do sistema como *Pronto*. O significado de *Pronto*

[10] do inglês *Done*

Tabela 2.2: Exemplo de *Visão do Produto*.

Para	Professores e alunos.
Que	precisam aplicar e responder testes de múltipla escolha.
O	*TesteJá* é um sistema Web.
Que	permite o controle de questões, provas e notas, além da realização do próprio teste por parte do aluno.
Ao contrário	da forma manual feita com papel impresso.
Nosso produto	fornecerá correção automática e disponibilização dos resultados imediatamente após a conclusão da prova, além de permitir ao professor o reuso de questões em diversos testes.

pode variar de equipe para equipe, portanto deve estar muito bem definido e entendido por todos os envolvidos no processo, desde os membros da Equipe de Desenvolvimento ao Dono do Produto. O compartilhamento e o entendimento coletivo desta definição possibilitam maior transparência.

A partir deste ponto, toda citação a *Pronto* (em itálico com P maiúsculo) será refente à Definição de "*Pronto*" estabelecida nesta fase.

2.3.1.3 Arquitetura e projeto

Na arquitetura é definido como os itens selecionados do *backlog* serão implementados. Se essa *release* for de um projeto já em andamento, é feita a análise de como as novas funcionalidades irão impactar a arquitetura, bem como as modificações necessárias para acomodá-las. Nesta etapa, também é realizada uma análise e projeto em alto nível, além das atividades a seguir:

- Revisão dos requisitos selecionados do *Backlog* do Produto;
- Identificação das mudanças necessárias para implementar tais requisitos;
- Análise de domínio do problema a ser construído ou incrementado, atualizando os modelos de dados para refletir o contexto do novo sistema e seus requisitos;
- Refinar a arquitetura do sistema para acomodar o novo contexto;
- Identificar possíveis problemas que ocorrerão durante a implementação de tais mudanças; e
- Preparação para a primeira Reunião de Planejamento do *Sprint*, na qual as sugestões de arquitetura e projeto serão apresentadas, discutidas e escolhidas.

Uma vez finalizado o planejamento e definida a arquitetura, ou as possíveis candidatas, a fase Jogo está habilitada a ser iniciada.

2.3.2 Fase Jogo

A fase Jogo é composta por *sprints*, ou iterações, de período de tempo fixo e predefinido que podem variar de uma a quatro semanas. Esses ciclos iterativos se repetem até que as restrições de tempo, competição, qualidade e funcionalidades sejam alcançadas de forma que o Dono do Produto esteja satisfeito. Quando isso ocorrer, inicia-se a fase Pós-jogo.

Essa abordagem segue o conceito de processo "caixa-preta", dando total flexibilidade para a Equipe de Desenvolvimento executar suas atividades de implementação de forma concorrente e colaborativa. Por "caixa-preta" entende-se que, a entrada, neste caso as estórias selecionadas no início do *sprint*, e a saída, a implementação e testes de tais estórias, são bem definidas, enquanto que, os detalhes sobre quais atividades devem ser realizadas para transformar tal entrada na saída esperada, não são relevantes do ponto de vista das pessoas fora da Equipe de Desenvolvimento.

As atividades que compõem esta fase são:

- **Desenvolvimento**: microprocesso de descoberta, invenção, experimentação e implementação, composto por análise de domínio, projeto, codificação, testes e documentação das mudanças;
- **Empacotamento**[11] **ou Integração**: criação de uma versão executável ou instalável das mudanças implementadas;
- **Revisão**: a Equipe de Desenvolvimento se reúne e apresenta o trabalho executado para o Dono do Produto (evento de Reunião de Revisão do *Sprint*) e também revisa o próprio processo (evento de Retrospectiva do *Sprint*). Nesse momento, os defeitos podem ser documentados bem como a adição de novos itens ao *Backlog* do Produto. Os riscos são revistos e as ações corretivas são definidas;
- **Ajustes**: consolidação das informações levantadas na revisão e preparação para o próximo *sprint*.

Cada *Sprint* segue uma série de eventos, mostrados na Figura 2.3, que manipula os artefatos de acordo com a necessidade, avaliando os riscos e respondendo a mudanças constantemente. Veja mais detalhes na Seção 2.7.1.

2.3.3 Fase Pós-jogo

Na fase Pós-jogo, ocorre o fechamento[12] que é a preparação para o lançamento de uma *release* em produção, ou seja, no ambiente em que os usuários finais utilizarão o sistema. Essa

[11] do inglês *Wrap*
[12] do inglês *Closure*

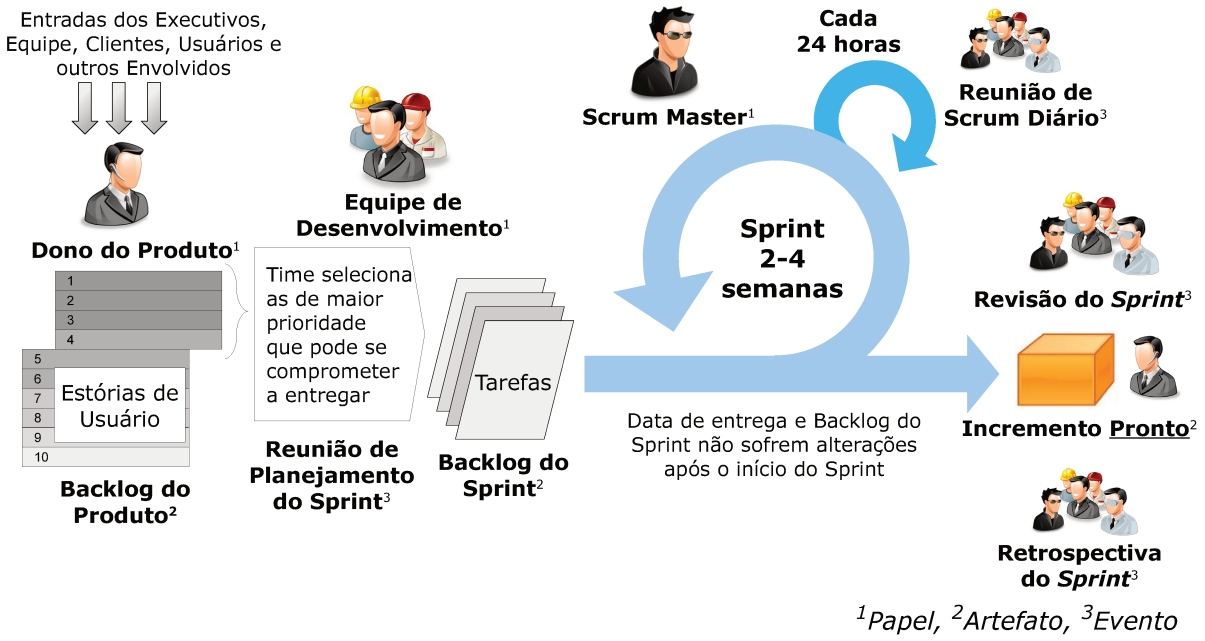


Figura 2.3: Detalhamento da fase Jogo do Scrum.

fase inicia-se quando o número de *sprints* planejados é executado, ou a implementação dos itens do *Backlog* do Produto é concluída. Pode também ter seu inicio antecipado se o Dono do Produto se declarar satisfeito com os atributos de qualidade e funcionalidades implementadas até determinado *sprint*, uma vez que, uma versão utilizável do sistema é demonstrada ao término de cada *sprint* na Reunião de Revisão do *Sprint*.

Nessa fase, as seguintes atividades poderão ser executadas:

- Testes de integração e de sistema;
- Documentação dos usuários;
- Preparação de material de treinamento, bem como a aplicação do treinamento aos usuários finais;
- Material de marketing;
- Aceitação final e formal do Dono do Produto, declarando o encerramento dessa *release*;
- Migração de dados existentes; e
- Instalação efetiva no ambiente de produção.

2.4 Pilares do Scrum

O Scrum aplica uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos, além de ser baseado nas teorias empíricas de controle de processo. O empirismo dita que as decisões tomadas são sempre baseadas no que se conhece e que, por sua vez, o conhecimento vem por meio da experiência.

Os três pilares que sustentam a implementação de um controle de processo empírico são: (i) transparência, (ii) inspeção e (iii) adaptação.

2.4.1 Transparência

Toda e qualquer tipo de informação pertinente a negócio, tecnologia, ou andamento do projeto deve estar visível para os responsáveis pelos resultados. Para tal, é necessário que os aspectos sejam definidos por um padrão comum para que os observadores compartilhem do mesmo entendimento do que está sendo visto.

Por exemplo, os próprios conceitos e regras do processo Scrum devem estar disponíveis e ser compartilhados entre todos os participantes. Outro exemplo é a visibilidade total e constante do *Backlog* do Produto e suas prioridades.

As métricas colhidas e analisadas, a lista de defeitos em aberto, o andamento do projeto, os requisitos a serem implementados e os demais artefatos gerados devem estar sempre acessíveis.

2.4.2 Inspeção

Todos os envolvidos no Scrum devem, frequentemente, inspecionar seus artefatos e monitorar o seu progresso para detectar possíveis variações. A frequência das inspeções não deve ser alta a ponto de burocratizar o fluxo de trabalho.

Os eventos do Scrum são ótimas oportunidades para se executar inspeções em diversas esferas. Por exemplo, no Scrum Diário pode-se ter uma ideia do quão bem, ou mal, a implementação das tarefas que compõem o *sprint* está caminhando. Na Reunião de Revisão do *Sprint*, pode-se concluir se o ritmo está adequado para o objetivo da *release*.

2.4.3 Adaptação

Uma vez detectado desvios inaceitáveis que gerarão riscos ao projeto durante as inspeções, ajustes devem ser feitos para se retomar o fluxo planejado. Tais ajustes devem ser feitos tão logo quanto possível para minimizar os desvios futuros.

O Scrum prescreve quatro oportunidades formais para inspeção e adaptação, como descritas na Seção 2.7, são elas os eventos de: (i) Reunião de Planejamento do *Sprint*, (ii) Scrum Diário, (iii) Reunião de Revisão do *Sprint*, e (iv) Retrospectiva do *Sprint*.

2.5 Artefatos do Scrum

Os artefatos do Scrum representam o trabalho ou valor que são úteis no sentido de oferecer máxima transparência e oportunidades para inspeção e adaptação. Eles são manipulados pela Equipe Scrum com objetivo de entregar um Incremento *Pronto*.

2.5.1 *Backlog* do Produto

É uma lista única e ordenada de tudo que possa ser necessário para o desenvolvimento e entrega de um sistema de software. Pode ser visto como o repositório central de todos os requisitos (funcionais e não funcionais), incluindo funcionalidades, melhorias e defeitos. Cada item do *Backlog* do Produto possui no mínimo uma descrição, um número de ordem e uma estimativa.

O *Backlog* do Produto é dinâmico e nunca estará completo, evoluindo junto com o produto, mudando constantemente para melhor atender às necessidades de negócio, mantendo assim a competitividade do sistema. Ele deve existir enquanto o produto existir, ou seja, conforme o sistema vai sendo usado, o seu verdadeiro valor vai sendo criado a partir da colaboração dos usuários reais provocando assim, o surgimento de melhorias e as novas funcionalidades. Outras mudanças de negócios, de mercado, de governo ou até mesmo de tecnologia também são responsáveis pelas alterações em seus itens.

O Dono do Produto é o responsável pelo conteúdo, disponibilidade e ordenação do *Backlog* do Produto. Mesmo que qualquer membro da Equipe possa adicionar itens, apenas o Dono do Produto deve ordená-los com base nas decisões de negócio condizentes, como valor para o negócio, riscos, prioridade e necessidade. Os itens de menor ordem se tornam candidatos a serem implementados nos próximos *sprints*, portanto eles devem estar menos maduros e menos detalhados que os de maior ordem. O tamanho de um item deve ser adequado para poder ser desenvolvido em apenas um *sprint*. Caso isso não seja possível, o item deve ser decomposto em itens menores.

A Equipe de Desenvolvimento é responsável pelas estimativas dos itens do *Backlog* do Produto, mesmo que auxiliada pelo Dono do Produto que ajuda a equipe a entender mais detalhadamente, bem como a fazer as negociações. Porém, quem estabelece a estimativa final é sempre quem executará o trabalho, ou seja, a Equipe de Desenvolvimento.

2.5.2 Objetivo do *Sprint*

O Objetivo do *Sprint* é uma sentença declarada pela Equipe de Desenvolvimento juntamente com a ajuda do Dono do Produto durante a Reunião de Planejamento do *Sprint*. Ela guia a equipe durante o *sprint* em direção às funcionalidades a serem implementadas, salientando, normalmente, a funcionalidade principal que se deseja ter ao final do ciclo.

Também pode ser usada como marco, para se declarar, durante a Reunião de Revisão do *Sprint*, que o *sprint* foi concluído com sucesso.

Alguns exemplos de Objetivo do *Sprint* são listados na Tabela 2.3.

Tabela 2.3: Exemplos de Objetivo do *Sprint*.

Preparar infraestrutura de banco de dados.
Implementar mecanismo de persistência.
Implementar cadastro de usuários.
Implementar pagamento via boleto bancário.
Implementar busca de produtos.
Integrar componentes.
Finalizar manual do usuário.

2.5.3 *Backlog* do *Sprint*

O *Backlog* do *Sprint* é um conjunto de itens do *Backlog* do Produto selecionados para o *sprint* durante a Reunião de Planejamento do *Sprint*, juntamente com um plano para obter o incremento do produto e atingir o Objetivo do *Sprint*. Este conjunto é uma previsão da Equipe de Desenvolvimento sobre quais funcionalidades estarão no próximo incremento e quais tarefas são necessárias para entregá-las.

O *Backlog* do *Sprint* define o trabalho que a Equipe de Desenvolvimento irá desempenhar para transformar os itens do *Backlog* do Produto em Incrementos *Prontos*. Ele torna visível todas as tarefas que a Equipe de Desenvolvimento identifica como necessárias para atingir o Objetivo do *Sprint*. A Equipe de Desenvolvimento modifica o *Backlog* do *Sprint* ao longo do *sprint*, adicionando, se necessário, novas tarefas, ou removendo as desnecessárias, visando atingir o objetivo do *sprint*. Na medida em que o trabalho é desempenhado e completado, a estimativa do trabalho restante é atualizada.

Monitoramento. Cada tarefa que compõe o *Backlog* do *Sprint* deve estar estimada em horas. Durante o Scrum Diário, a equipe ajusta a estimativa de quantidade de horas restantes para finalizar determinada tarefa, conforme exemplo da Tabela 2.4. Somando-se o tempo restante de todas as tarefas, consegue-se monitorar e gerenciar os riscos da previsão de entrega do *sprint*.

2.5.4 Incremento

O Incremento é o produto final da implementação de todas as tarefas do *Backlog* do *Sprint*, testadas, integradas e documentadas. Ele deve estar em uma condição utilizável e demonstrável, pois será apresentado pela Equipe de Desenvolvimento ao Dono do Produto

Tabela 2.4: Exemplo de *Backlog* do *Sprint* com estimativas (em horas) por dia.

Tarefa	**Segunda**	**Terça**	**Quarta**	**Quinta**	**Sexta**
Codificar interface	8	4	8		
Codificar classe de negócio	16	12	10	4	
Testar classe de negócio	8	16	16	11	8
Escrever texto de ajuda	12				
Criar tabela no banco de dados	4	4	4	4	4
Adicionar *log* de erros			8	3	

na Reunião de Revisão do *Sprint*. Para isso, ele deve respeitar a Definição de *Pronto* estabelecida na fase de planejamento.

Esse será o artefato principal para a validação do Dono do Produto e para o monitoramento do progresso do projeto como um todo, uma vez que segue um processo incremental. Ele evoluirá com o passar dos *sprints* até conter a implementação de todos os itens estimados para uma *release*.

2.6 Equipe do Scrum

A Equipe do Scrum é formada pelo Dono do Produto, pela Equipe de Desenvolvimento e pelo Scrum Master. Essa equipe é auto-organizada e multifuncional, isto significa que seus membros são independentes e autossuficientes para escolher a melhor forma de realizar seu trabalho, ao invés de serem dirigidas ou gerenciadas por outros de fora da equipe. Ser multifuncional significa que não existe função ou cargo exclusivo, como arquiteto, analista de requisitos, desenvolvedor, testador. Todos os membros da equipe trabalham colaborativamente na execução de suas tarefas, criando uma atmosfera de propriedade coletiva. Com esse modelo de equipe, consegue-se alcançar melhor a flexibilidade, a criatividade e a produtividade.

2.6.1 Dono do Produto

O Dono do Produto[13] é responsável por maximizar o valor de negócio do produto e a qualidade do trabalho entregue pela Equipe de Desenvolvimento, por meio do gerenciamento do *Backlog* do Produto. É representado por uma única pessoa, dedicada ao projeto, que deve entender o processo Scrum e as responsabilidades intrínsecas a ele. Deve ser desempenhado por uma pessoa de negócio, normalmente o próprio cliente, mas também por uma pessoa de marketing, um usuário-chave, ou até mesmo de vendas que represente o cliente internamente.

[13]do inglês *Product Owner*

Gerenciar o *Backlog* do Produto consiste nas seguintes atividades:

- Expressar com clareza os itens do *backlog*;
- Ordenar os itens para melhor alcançar os objetivos e missões;
- Garantir o valor do trabalho desempenhado pela Equipe de Desenvolvimento;
- Garantir visibilidade, transparência e clareza dos itens que equipe irá trabalhar em seguida; e
- Garantir que a Equipe de Desenvolvimento entenda os itens no nível de detalhe necessário.

2.6.2 Equipe de Desenvolvimento

A Equipe de Desenvolvimento consiste nos profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto "*Pronto*" ao final de cada *sprint*. Ela tem as seguintes características:

- É auto-organizada, ou seja, nem mesmo o Scrum Master diz para a equipe como transformar o *Backlog* do Produto em incrementos de funcionalidades potencialmente utilizáveis. Essas atividades devem ser totalmente empíricas, possibilitando assim maior flexibilidade, criatividade e inovação;
- É multifuncional, com todas as habilidades necessárias para criar o incremento do produto;
- Não reconhece perfis dos membros além de desenvolvedor, independente do trabalho que está sendo realizado pela pessoa. Não há exceções a essa regra;
- Faz com que seus membros, mesmo que tenham habilidades especializadas e áreas de especialização, respondam à Equipe de Desenvolvimento como parte de um todo; e
- Não possua subequipes dedicadas a domínios particulares do conhecimento, como testes ou análise de negócio.

O tamanho ótimo de uma Equipe de Desenvolvimento é suficientemente pequeno para se manter ágil, e grande o suficiente para completar uma parcela representativa do trabalho. Equipes muito pequenas podem encontrar limitações nas habilidades necessárias para um *sprint*, e como consequência não conseguem entregar um incremento potencialmente utilizável. Ter muitos membros acaba gerando muita complexidade para gerenciar

um processo empírico. Portanto o tamanho recomendado que ela tenha entre 5 e 9 membros. Esse número não inclui o Dono do Produto nem o Scrum Master, a não ser que eles também executem o trabalho do *Backlog* do *Sprint*.

2.6.3 Scrum Master

O Scrum Master é o responsável por garantir que o Scrum seja entendido e aplicado, obedecendo aos pilares, fases, práticas, eventos e regras do Scrum. O Scrum Master não tem um papel de Gerente de Projetos mas sim atua como um líder facilitador para a equipe do Scrum, blindando a equipe do Scrum de interrupções ou práticas que não fazem parte do processo. Com isso, acaba ajudando nas interações dos envolvidos para maximizar o valor criado pela equipe de Scrum.

Por ser um facilitador, o Scrum Master serve aos outros papéis da Equipe Scrum e à organização como um todo, como as listadas a seguir:

- Ao **Dono do Produto:**
  - Encontrar técnicas para o gerenciamento efetivo do *Backlog* do Produto;
  - Comunicar claramente a visão, os objetivos e os itens do *Backlog* do Produto para a Equipe de Desenvolvimento;
  - Ensinar a Equipe de Desenvolvimento como criar itens do *Backlog* do Produto claros e concisos;
  - Entender o planejamento de longo prazo do produto em um ambiente empírico;
  - Entender e praticar a agilidade; e
  - Facilitar os eventos do Scrum.
- À **Equipe de Desenvolvimento:**
  - Orientar sua auto-organização e multifuncionalidade;
  - Liderar a criação de um produto de alto valor;
  - Remover os impedimentos para o progresso da implementação durante o *sprint*;
  - Facilitar os eventos do Scrum; e
  - Orientar a equipe no ambiente organizacional no qual o Scrum ainda não é amplamente adotado e compreendido.
- À **Organização:**
  - Liderar e orientar a organização na adoção do Scrum;

- Planejar as implementações do Scrum dentro da organização;
- Ajudar empregados e envolvidos a entender e implantar o Scrum e o desenvolvimento empírico de produtos;
- Provocar mudanças que aumentem a produtividade na Equipe de Scrum; e
- Trabalhar com outros Scrum Masters para aumentar a efetividade da aplicação do Scrum na organização.

2.7 Eventos do Scrum

Os eventos prescritos pelo Scrum são usados para criar regularidade e minimizar a necessidade de outras reuniões. Todos os eventos têm duração máxima predefinida[14]. Isso garante que uma quantidade adequada de tempo seja gasta, não permitindo que ocorra desperdício no processo de planejamento.

Além do próprio *Sprint*, que é um contêiner para todos os outros eventos, cada evento do Scrum é uma oportunidade formal para inspecionar e adaptar o software ou o processo. Deixar de incluir qualquer um dos eventos resulta em perda de transparência e distorce o processo.

A seguir são detalhados os cinco eventos.

2.7.1 *Sprint*

O *Sprint* é a iteração do Scrum, que tem duração fixa e predefinida de um mês ou menos e que se repete seguidamente na implementação dos requisitos do *Backlog* do Produto. Durante um *sprint* ocorre a criação de uma versão potencialmente utilizável do produto, ou seja, um Incremento. A duração dos *sprints* bem como os membros da Equipe de Desenvolvimento e o Objetivo do *Sprint* devem se manter constantes durante todo o ciclo de desenvolvimento.

O objetivo de um *sprint* é ter uma duração fixa e relativamente curta (entre duas e quatro semanas) permitindo assim uma maior previsibilidade e, por fim, garantindo a inspeção e adaptação de um conjunto limitado de requisitos. Além disso, em um período pequeno de tempo há um risco menor de ocorrerem mudanças. Em casos extremamente excepcionais pode ocorrer o cancelamento de um *sprint* que deve apenas ser declarado pelo Dono do Produto quando o Objetivo do *Sprint* ficar obsoleto pelo contexto do negócio. Se isso acontecer, deve ocorrer uma revisão mais profunda do *Backlog* do Produto e uma nova Reunião de Planejamento do *Sprint* deve ser marcada. Devido à duração curta de um *sprint*, tais cancelamentos são raros.

[14] do inglês *time-box*

A fase Jogo é composta por *sprints* que consistem em uma Reunião de Planejamento do *Sprint*, Scrum Diários, o trabalho de desenvolvimento em si (atividades de desenvolvimento, empacotamento, revisão, e ajustes), Revisão e Retrospectiva do *Sprint*. Estes eventos são detalhados a seguir.

2.7.2 Reunião de Planejamento do *Sprint*

A Reunião de Planejamento do *Sprint* é o primeiro evento de um *sprint* e visa criar um plano sobre o trabalho que será executado. Tal plano é feito de forma colaborativa e com a participação de toda Equipe de Scrum. Essa reunião tem duração fixa de oito horas para *sprints* de quatro semanas, para *sprints* mais curtos, o evento é proporcionalmente menor, ou seja, *sprints* de duas semanas têm Reunião de Planejamento do *Sprint* de quatro horas.

Esse evento tem como artefatos de entrada o *Backlog* do Produto e o Incremento atual do sistema, e de saída, o Objetivo do *Sprint* e o *Backlog* do *Sprint*.

A Reunião de Planejamento é dividida em duas partes de mesma duração. Cada metade visa responder às seguintes questões, respectivamente: (i) **O que** vai ser entregue como resultado do Incremento do próximo *sprint*? (ii) **Como** será realizado o trabalho necessário para entregar o Incremento?

Cada uma das partes será descrita a seguir:

Parte 1: O que será *Pronto* neste *sprint*? Nesta parte, o Dono do Produto apresenta os itens ordenados do *Backlog* do Produto para a Equipe de Desenvolvimento. Toda a Equipe de Scrum colabora no entendimento do trabalho do *sprint*, levando em consideração o último Incremento do produto, a capacidade projetada e o desempenho da Equipe de Desenvolvimento durante os *sprints* anteriores. Com esses dados em mãos, a Equipe de Desenvolvimento avalia o quanto consegue prever para ser implementado no próximo *sprint*. Uma vez feita essa previsão, a Equipe de Scrum produz o Objetivo do *Sprint*.

Parte 2: Como o trabalho escolhido será *Pronto*? Tendo selecionado o trabalho do *sprint*, a Equipe de Desenvolvimento decide como irá transformar essas funcionalidades em um Incremento *Pronto* durante o *sprint*. Isto é feito decompondo os requisitos dos *Backlog* do Produto em tarefas executáveis, com duração estimada de, no máximo, 2 dias de trabalho. Tais tarefas formam o *Backlog* do *Sprint*. Durante esse processo ocorrem a análise e projeto dos requisitos, ao mesmo tempo que o planejamento de como implementá-los.

A participação do Dono do Produto é obrigatória na **Parte 1** e altamente recomendada na **Parte 2**, visto que é nesse evento que serão capturados os detalhes sobre os requisitos que, devido à sua ordenação, já estão maduros o suficiente para serem implementados. Para esse evento também podem ser convidadas outras pessoas, como especialistas no domínio do sistema ou técnicos, para colaborar no entendimento dos requisitos e desenho da solução.

Ao final da Reunião de Planejamento do *Sprint*, a Equipe de Desenvolvimento deve estar apta a explicar ao Dono do Produto e ao Scrum Master como pretende trabalhar como uma equipe auto-organizada para conseguir criar o Incremento desejado e atingir o Objetivo do *Sprint*.

2.7.3 Scrum Diário

A Reunião do Scrum Diário é um evento com 15 minutos fixos, coordenado pelo Scrum Master, na qual a Equipe de Desenvolvimento sincroniza as atividades e cria um plano para as próximas 24 horas. Isso se faz, inspecionando o trabalho desde o último Scrum Diário e prevendo o trabalho que pode ser *Pronto* até o próximo.

Para reduzir a complexidade, o Scrum Diário deve ocorrer no mesmo horário e local e seguir um roteiro simples, composto por três perguntas que devem ser respondidas por cada membro da Equipe de Desenvolvimento:

1. O que conseguiu realizar desde o último encontro?
2. O que vai ser realizado até o próximo encontro?
3. Quais são os obstáculos que estão no seu caminho?

O roteiro deve ser respeitado à risca, não fugindo do escopo de cada pergunta. Apenas um membro deve falar por vez, para que seja respeitado o limite de tempo de 15 minutos. Todas as discussões sobre possíveis soluções ou desvios devem ser feitas fora do encontro e apenas com as pessoas interessadas, evitando assim o desperdício de tempo dos outros membros da equipe. Portanto se alguém comunicar um obstáculo, o Scrum Master deve endereçar a solução fora do encontro e apenas com o membro da equipe afetado.

A Equipe de Desenvolvimento usa o Scrum Diário para avaliar o progresso na direção do Objetivo do *Sprint*, aumentando assim a probabilidade de atingi-lo no prazo planejado.

Os Scrum Diários melhoram a comunicação, eliminam outras reuniões, identificam e removem os impedimentos para o desenvolvimento, destacam e promovem uma tomada de decisão rápida, e melhoram o nível do conhecimento do projeto da Equipe de Desenvolvimento. Essa é a reunião-chave para a inspeção e a adaptação de todo o processo.

2.7.4 Reunião de Revisão do *Sprint*

A Reunião de Revisão do *Sprint* é executada ao final do *sprint* para inspecionar o Incremento e adaptar ao *Backlog* do Produto, caso necessário. Assim como os outros eventos, a revisão tem tempo fixo de 4 horas para *sprints* de um mês, sendo proporcionalmente menor para *sprints* de menor duração.

As principais atividades realizadas durante este evento são:

- A Equipe de Desenvolvimento discute o que ocorreu de forma satisfatória durante o *sprint*, quais problemas ocorreram e como foram resolvidos;
- A Equipe de Desenvolvimento demonstra o trabalho que foi *Pronto* e responde às perguntas sobre o Incremento;
- O Dono do Produto valida e identifica o que foi *Pronto* e o que não foi *Pronto*;
- O Dono do Produto discute o *Backlog* do Produto e projeta a data mais provável de término, baseando-se no progresso até o momento; e
- Todos colaboram no que deve ser *Pronto* em seguida, fornecendo assim, uma entrada valiosa para as próximas reuniões de planejamento do *sprint*.

É neste momento que ocorre a validação, pelo Dono do Produto, do Incremento produzido durante o *sprint* em relação ao Objetivo do *Sprint* declarado na Reunião de Planejamento. Caso algum item não tenha sido reconhecido como *Pronto*, ele deve voltar ao *Backlog* do Produto para ser reordenado, ou podem ser criados novos itens para corrigir eventuais defeitos ou ajustes.

Ao final dessa reunião, o *Backlog* do Produto está revisado, atualizado e reordenado, levando-se em consideração o que foi entregue neste *sprint* e as novas demandas que possam ter surgido nesse período de tempo, ficando assim pronto para a próxima Reunião de Planejamento de *sprint*.

2.7.5 Retrospectiva do *Sprint*

A Retrospectiva do *Sprint* é uma oportunidade para a equipe do Scrum se auto inspecionar e criar um plano de melhorias que deve ser aplicado durante o próximo *sprint*. Ela ocorre após a Reunião de Revisão do *Sprint* e antes da próxima Reunião de Planejamento do *Sprint*. Tem também tempo fixo: 3 horas para *sprints* de um mês (ou menos proporcionalmente).

O propósito da Retrospectiva do *Sprint* é de inspecionar como foi o último *sprint* no que diz respeito a pessoas, a relações, a processos e a ferramentas, visando aumentar a efetividade no uso do Scrum e na qualidade do produto entregue, podendo-se até adaptar a definição de *Pronto* quando apropriado.

No final da Retrospectiva do *Sprint*, a Equipe de Scrum deve identificar melhorias que serão aplicadas no próximo *sprint*. Essas melhorias devem ser implementadas no próximo *sprint* e a inspeção deve ser adaptada pela própria Equipe do Scrum. Apesar de melhorias poderem ser adotadas a qualquer momento, esse evento oferece um momento exclusivo, dedicado e focado na inspeção e adaptação do processo Scrum.

2.8 Desenvolvimento Baseado em Componentes

Apesar de ser um conceito muito difundindo atualmente na produção de software, o Desenvolvimento Baseado em Componentes (DBC) foi sugerido pela primeira vez, em 1968, como uma alternativa para a padronização de rotinas e métodos comumente utilizados, fazendo-se uma analogia com peças de estruturas mecânicas. Tal padronização visava criar repositórios de componentes robustos, genéricos e de alta qualidade, de forma que pudessem ser utilizados na construção de sistemas de software com maior qualidade e velocidade. Além disso, McIlroy [20] já salientava que rotinas similares deveriam estar disponíveis nesse repositório de modo que o uso de uma delas poderia ser intercambiável com as demais.

Mesmo com essas ideias já de longa data, o uso do DBC ainda vem amadurecendo, tendo sido impulsionado quase 30 anos depois, após a realização do Primeiro Workshop Internacional em Programação Orientada a Componentes, o WCOP'96[15]. Porém, a definição de componente ainda é um tanto abstrata, mesmo havendo um consenso na comunidade de que um componente é uma unidade de software independente que pode ser composta por outros componentes para criar um sistema de software [28].

Uma definição muito utilizada é a de Szyperski [30], que dita que um componente de software é uma unidade de composição com interfaces especificadas por meio de contratos e dependências de contexto explícitas. Além de poder ser implantado independentemente e estar sujeito à composição por terceiros. Podem-se interpretar as definições de "interfaces especificadas por meio de contratos" como sendo as **interfaces providas** e as de "dependências de contexto explícitas" como sendo as **interfaces requeridas**, sendo que toda e qualquer interação fora da fronteira de um componente deve ser obrigatoriamente feita por intermédio dessas interfaces.

[15] *International Workshop on Component-Oriented Programming*, http://research.microsoft.com/en-us/um/people/cszypers/events/WCOP1996. Acessado em: 22 ago. 2011

Uma **interface provida** define os serviços ou funções fornecidas pelo componente, ou seja, provê acesso aos métodos que podem ser chamados pelos seus usuários. Uma **interface requerida** especifica quais serviços devem ser providos pelos outros componentes do sistema para o componente em questão funcionar. Isso não compromete a independência ou a facilidade de implantação do componente uma vez que não há necessidade de que um componente específico seja usado como fornecedor, mas qualquer componente que satisfaça a sua interface requerida.

Independente dos autores e de suas definições de componente, existe um consenso sobre suas características. A Tabela 2.5 mostra essas características essenciais baseadas no DBC.

Tabela 2.5: Características de componente [28].

Plano	**Descrição**
Padronizado	Um componente precisa estar de acordo com algum modelo padronizado que defina as interfaces, metadados, documentação, composição e implantação.
Independente	Um componente eeve ser composto e implantado sem a necessidade de outros componentes específicos. Se houver alguma dependência, ela deve ser explicitamente definida por uma interface requerida.
Passível de Composição	Todas as interações entre componentes devem ocorrer por meio de interfaces publicamente definidas que forneçam acesso externo às informações sobre si próprio, como seus métodos e atributos.
Implantável	Um componente precisa ser autocontido e deve ser capaz de operar como uma entidade independente sobre uma determinada plataforma, de forma que não precise ser compilado antes de ser implantado.
Documentado	Um componente deve ser completamente documentado, incluindo as especificações de todas as suas interfaces, de forma que os usuários possam decidir se atendem ou não suas necessidades.

A utilização de componentes previamente especificados, implementados e testados para a construção de sistemas de software tem proporcionado um ganho significativo na produtividade durante o ciclo de desenvolvimento e na qualidade final do software produzido. O ganho de produtividade é decorrente da reutilização desses componentes já existentes e a melhoria na qualidade vem como consequência de esses componentes já terem sido testados e empregados em outros contextos.

Embora o reuso, principalmente de código, já ocorra de forma intuitiva e informal independentemente do processo de desenvolvimento usado, a inclusão do reuso sistemático de componentes já está consolidada e vem sendo cada vez mais utilizada. A essa abordagem dá-se o nome de DBC ou Engenharia de Software Baseada em Componentes[16]. A Figura 2.4 mostra um modelo genérico de DBC com as seis fases do processo.

[16]do inglês *Component-Based Software Engineering* (CBSE) [28]

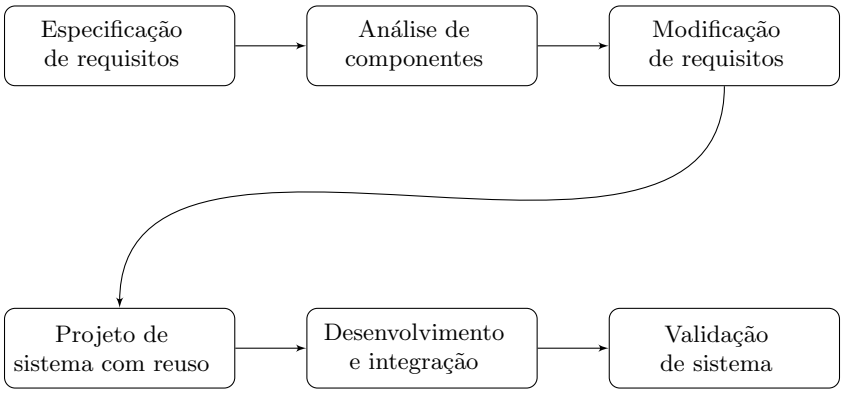


Figura 2.4: Modelo de processo genérico de DBC [28].

Neste processo, os estágios de especificação inicial dos requisitos e de validação são compatíveis a outros processos, já os estágios intermediários são diferenciados no que se refere à reutilização de componentes. Esses estágios são:

1. **Análise de componentes.** Feita a especificação de requisitos, busca-se por componentes que satisfaçam total ou parcialmente as funcionalidades elicitadas. Essa busca é feita tanto em repositórios internos da própria organização ou no mercado, adquirindo sistemas comerciais independentes ou *Commercial Off-The-Shelf Systems* (COTS).

2. **Modificação de requisitos.** Os requisitos são analisados levando-se em conta os componentes encontrados e modificados para acomodá-los. Se tais modificações forem impossíveis, a atividade de análise de componentes pode ser realizada novamente.

3. **Projeto de sistema com reuso.** O esqueleto do sistema é projetado organizando e conectando os componentes que serão reusados. Se funcionalidades não forem satisfeitas com componentes já existentes, será necessário projetá-las neste estágio.

4. **Desenvolvimento e integração.** O software externo que não pode ser reusado é desenvolvido. Os componentes e os sistemas COTS são integrados. Tal integração, nesse modelo, faz parte do processo de desenvolvimento, em vez de ser uma fase separada, já que as adaptações e as conversões podem ser necessárias e, por conta disso, precisam ser testadas.

A vantagem evidente do uso do DBC é que, com o reuso, reduz-se a quantidade de software a ser desenvolvido, por sua vez diminuem os custos, os riscos e, por fim, o tempo de

entrega. No entanto, deve-se considerar que as funcionalidades de um componente preexistente dificilmente serão idênticas às necessidades do sistema a ser desenvolvido, levando à negociação dos requisitos. Além disso, a evolução do sistema deve sofrer um controle mais rigoroso quanto às versões dos componentes reusados, principalmente, quando eles não estiverem sob controle da organização que os utiliza.

O aumento da qualidade do sistema é uma consequência do fato de os componentes utilizados já terem sido empregados e testados em outros contextos. Porém, apesar de os testes de cada componente serem benéficos, a reutilização de componentes não dispensa a execução dos testes no novo contexto onde o componente está sendo reutilizado, integrando-se a outros componentes já existentes e recém-desenvolvidos. Esses testes de integração são fundamentais para garantir a integridade do comportamento dos componentes em tal contexto, principalmente pelo fato de que nem todas as funcionalidades de um componente serão utilizadas, e mesmo não utilizadas, uma vez integradas elas podem produzir defeitos inesperados.

A seguir, será introduzido um processo de DBC, chamado *UML Components*.

2.8.1 Processo *UML Components*

O processo *UML Components* [9] é um processo baseado em componentes voltado para o desenvolvimento de sistemas de informação. Por esse motivo, ele é um processo simples e de fácil utilização prática. Devido a sua simplicidade, uma arquitetura predefinida em quatro camadas é adotada sendo que duas delas são destacadas durante o desenvolvimento: camada de sistema e camada de negócio. A camada de sistema é responsável por agrupar os componentes que implementam as funcionalidades especificadas para o sistema.

Porém, para que essas funcionalidades possam ser implementadas, esses componentes podem necessitar de algumas funcionalidades comuns ao domínio. Os componentes que implementam essas funcionalidades gerais são posicionados na camada de negócio. As quatro camadas da arquitetura e o papel de cada uma podem ser vistos na Figura 2.5.

O desenvolvimento é dividido em seis fases, ilustradas na Figura 2.6: (i) especificação de requisitos; (ii) especificação dos componentes; (iii) provisionamento dos componentes; (iv) montagem do sistema; (v) testes; e (vi) implantação. Esse processo é iterativo e enfatiza, basicamente, a fase de especificação dos componentes. Por essa razão, essa fase é detalhada em três subfases: (i) identificação dos componentes; (ii) interação entre os componentes; e (iii) especificação final.

Na **Fase 1 Especificação dos requisitos** ocorre a criação de casos de uso para representar as funcionalidades do sistema e é especificado o modelo conceitual do negócio, a ser implementado na Fase 6, que representa as entidades básicas da sua lógica. Esses artefatos são considerados as principais saídas desta fase.

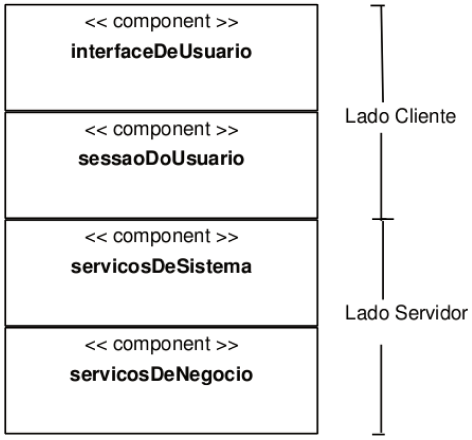


Figura 2.5: Arquitetura adotada pelo *UML Components* [9].

A **Fase 2 Especificação dos componentes** é a fase mais detalhada e visa especificar os componentes do sistema, tomando como entradas os artefatos da fase anterior e produzindo três artefatos de saída: (i) especificação das interfaces (refinamento das assinaturas); (ii) especificação dos componentes (interfaces providas e requeridas); e (iii) a arquitetura do sistema, como consequência da definição das dependências entre os componentes.

A **Fase 3 Provisionamento** deve garantir a materialização dos componentes especificados por meio da reutilização de componentes já existentes ou da implementação de componentes novos. Normalmente, os componentes da camada de negócio são candidatos à reutilização, uma vez que representam características básicas para sistemas de um mesmo domínio, enquanto os componentes da camada de sistema oferecem implementações específicas, peculiares às necessidades de um sistema em particular, sendo assim é necessário especificar as suas estruturas internas.

A **Fase 4 Montagem** é responsável pela materialização da configuração da arquitetura do sistema, ou seja, a integração dos componentes para construção da aplicação como um todo. Basicamente, existem duas atividades desempenhadas durante esta fase: (i) a construção dos conectores, que implementam um código de mapeamento entre os componentes do sistema; e (ii) a construção do programa principal, que deve conter o código de instanciação dos componentes e dos conectores.

A **Fase 5 Testes** tem a missão de realizar verificações entre o sistema desenvolvido e os requisitos especificados, garantindo que os requisitos estejam corretos e coerentes, ou seja, não contraditórios entre si, e que o sistema atenda minimamente a todos eles. Essa etapa é fundamental para garantir a qualidade do sistema entregue.

A **Fase 6 Implantação** coloca o sistema desenvolvido disponível para utilização não significando que ele tenha sido finalizado, pois muito provavelmente serão necessários alguns ciclos de manutenção corretiva para que o sistema seja considerado estável.

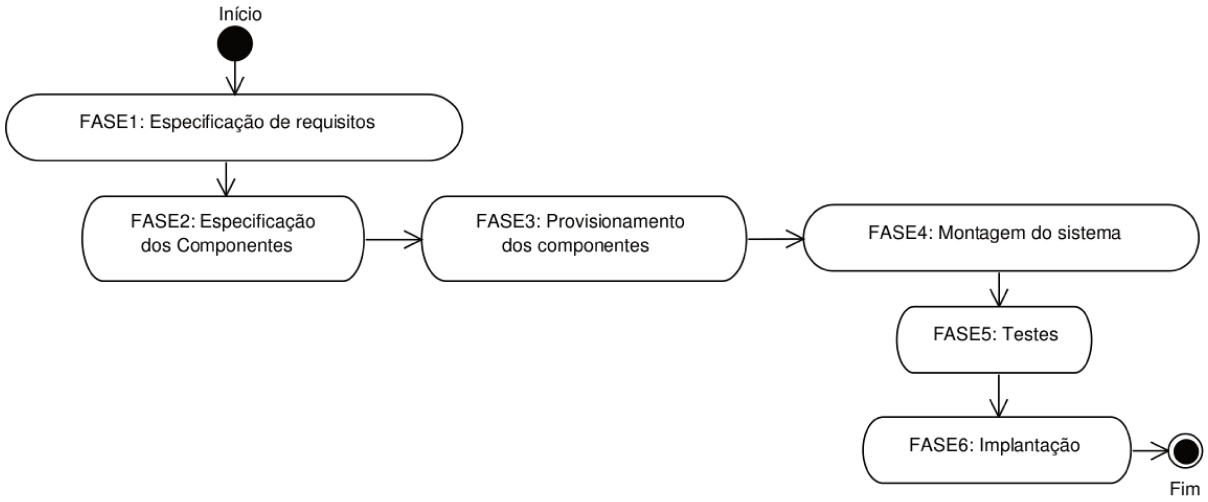


Figura 2.6: Fases do processo *UML Components* [9] em UML.

2.9 Desenvolvimento Centrado na Arquitetura

A arquitetura de software define o sistema, em um alto nível de abstração, em termos de: seus componentes arquiteturais, que representam unidades abstratas do sistema; a interação entre essas entidades, que são materializadas explicitamente por meio dos conectores; os atributos e funcionalidades de cada um, conforme definido por Sommerville [28]. Por conhecerem o fluxo interativo entre os componentes do sistema, os conectores possibilitam a criação de protocolos de comunicação e a coordenação da execução dos serviços que envolvam mais de um componente do sistema.

As propriedades arquiteturais são derivadas dos requisitos do sistema e influenciam, direcionam e restringem todas as fases do seu ciclo de vida. Sendo assim, a arquitetura de software é um artefato essencial no processo de desenvolvimento de sistemas de software modernos, sendo útil em todas as suas fases. A importância da arquitetura fica ainda mais clara no contexto do desenvolvimento baseados em componentes. Isso acontece, uma vez que na composição de sistemas, os componentes precisam interagir entre si para oferecer as funcionalidades desejadas. Além disso, devido à diferença de abstração entre a arquitetura e a implementação de um sistema, um processo de desenvolvimento baseado em componentes deve apresentar uma distinção clara entre os conceitos de componente arquitetural, que é abstrato e não é necessariamente instanciável; e componente de implementação, que representa a materialização de uma especificação em alguma tecnologia específia e deve, necessariamente, ser instanciável.

2.10 Tolerância a falhas e tratamento de exceções

Esta seção descreve os conceitos de falha, de erro, de defeito, da característica de confiabilidade e do componente tolerante a falhas ideal.

2.10.1 Falha, erro e defeito

Falha[17] é um evento que causa erros[18]. Caso o estado errôneo do sistema não seja tratado em um tempo determinado, ocorrerá um defeito[19], que se manifestará pela não execução ou mudança indesejada no serviço especificado. Como pode ser visto na Figura 2.7, além de indicar um estado errôneo irreversível, a ocorrência de um defeito realimenta o ciclo e pode gerar novos erros. Esse fenômeno é conhecido como propagação de erro e deve ser contida por meio do seu confinamento.

Inicialmente, o conceito de falhas se relacionava apenas às falhas de origem física, por exemplo, a mudança de um bit na memória por interferência eletromagnética. Entretanto, dado o aumento da complexidade dos sistemas de software, falhas de projetos se tornaram frequentes e quase inevitáveis.

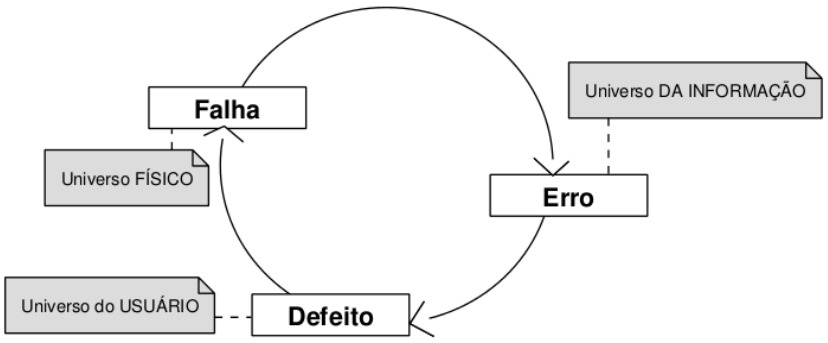


Figura 2.7: Ciclo entre a ocorrência de falha, erros e defeito [18].

As falhas também podem ser classificadas segundo a sua duração, podendo ser distribuídas em três grupos: (i) transientes, quando existe a chance das falhas ocorrem mais de uma vez, mas não obrigatoriamente, por exemplo, a invasão do sistema por um usuário mal-intencionado; (ii) intermitente, quando a ocorrência da falha se repete, mas não em períodos definidos, por exemplo, a manifestação de um vírus; e (iii) permanentes, que são caracterizadas pela sua presença constante no sistema, por exemplo, falhas de especificação e implementação.

[17] do inglês *Fault*
[18] do inglês *Error*
[19] do inglês *Failure*

Um fato importante de se observar é que uma falha pode resultar em um, mais de um ou nenhum erro. Esse fato, aliado à impossibilidade de se conhecer todas as causas de uma falha dificultam seu processo de detecção. Outro fato é que várias falhas distintas podem acarretar um mesmo erro, por esse motivo, a identificação do erro não resulta naturalmente na identificação da falha responsável por ele. Dessa forma, faz-se necessário uma análise rigorosa do contexto de manifestação dos erros para que seja possível inferir suas causas e, em seguida, tratá-las.

2.10.2 Confiabilidade de sistemas de software

O grau de confiança no funcionamento de um sistema de software depende, principalmente, do número de falhas contidas no sistema e da maneira como ele se comporta na presença dessas falhas.

Um sistema confiável[20] é um sistema que oferece um grau de confiabilidade de funcionamento mensurável aos seus usuários [28]. As principais dimensões dessa confiança são: (i) disponibilidade[21], que é a capacidade do sistema oferecer seus serviços quando requerido; (ii) confiabilidade[22], que é a capacidade do sistema oferecer seus serviços conforme a especificação; (iii) segurança operacional no funcionamento[23], que é a capacidade do sistema operar sem apresentar defeitos catastróficos; e (iv) segurança da informação[24], que é a capacidade do sistema evitar falhas maliciosas, isto é, falhas provenientes do meio externo e que podem ser intencionais.

As principais abordagens utilizadas para o desenvolvimento de sistemas de software com requisitos críticos de confiabilidade no funcionamento são [28]: (i) prevenção de falhas[25]; (ii) remoção de falhas[26]; (iii) tolerância a falhas[27]; e (iv) avaliação de falhas[28]. Essas técnicas são complementares e podem ser combinadas para o desenvolvimento de sistemas de software robustos. A Tabela 2.6 descreve sucintamente cada uma delas.

2.10.3 Componente tolerante a falhas ideal

O conceito de componente tolerante a falhas ideal, ou simplesmente **componente ideal**, foi introduzido por Anderson e Lee [1].

[20] do inglês *Dependable*
[21] do inglês *Availability*
[22] do inglês *Reliability*
[23] do inglês *Safety*
[24] do inglês *Security*
[25] do inglês *Fault prevention*
[26] do inglês *Fault removal*
[27] do inglês *Fault tolerance*
[28] do inglês *Fault forecasting*

Tabela 2.6: Abordagens para a implementação de sistemas confiáveis.

Abordagem	**Descrição**
Prevenção de Falhas	O sistema é desenvolvido de modo a evitar a inserção de falhas humanas. O processo de desenvolvimento é organizado para detectar e corrigir falhas, antes da entrega do sistema ao usuário final.
Remoção de Falhas	O sistema é projetado para que sejam utilizadas técnicas de verificação e validação (formais ou não) para detectar e corrigir falhas. Essas técnicas também são executadas antes da entrega do sistema ao usuário final.
Tolerância a Falhas	O sistema é projetado de modo que a presença de falhas durante a sua execução não resulte em defeitos visíveis ao usuário.
Avaliação de Falhas	O sistema é simulado, a fim de identificar os estados impossíveis de serem alcançados. Essa identificação é utilizada para restringir o modelo de falhas do sistema, reduzindo os custos e aumentando a eficiência das técnicas de tolerância a falhas.

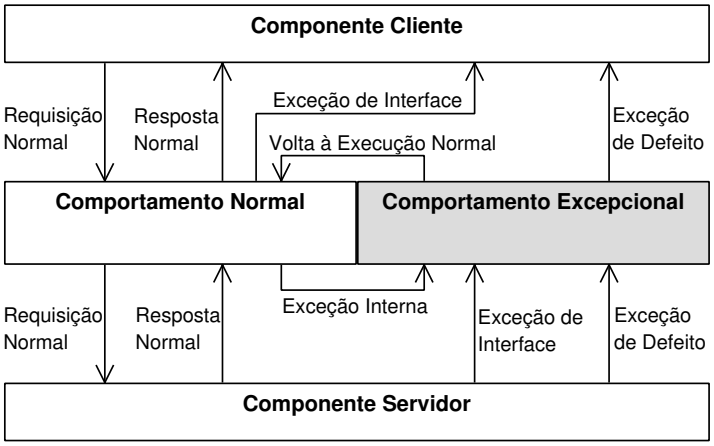


Figura 2.8: Estrutura do componente tolerante a falhas ideal.

A Figura 2.8 mostra a estrutura do componente ideal, cujo comportamento deve ser categorizado em dois tipos: normal ou excepcional (ou anormal). O comportamento normal é aquele que executa adequadamente os serviços, produzindo respostas corretas. Entretanto, na ocorrência de alguma falha, o componente excepcional assume a responsabilidade, tentando tratá-la de maneira adequada.

As exceções ocorridas em um componente ideal podem ser classificadas como internas ou externas. As exceções externas, por sua vez, podem ser exceções de defeito, quando indicam a incapacidade de fornecer um serviço, ou de interface, decorrentes de requisições incorretas por parte dos clientes. Na ocorrência de qualquer um dos tipos, a execução normal do componente é interrompida e é ativado o tratador específico para a exceção ocorrida. Caso não seja possível a recuperação do estado normal do sistema, é retornada uma exceção de defeito à sua camada superior, que pode tratá-la ou continuar propagando.

Porém, devido ao contexto das informações, o tratamento será mais eficiente quanto mais próximo estiver da origem do erro [1].

2.11 Metodologia para Definição do Comportamento Excepcional (MDCE+)

O MDCE+ [8] tem como objetivo sistematizar a especificação, modelagem e implementação do comportamento excepcional no desenvolvimento de sistemas confiáveis baseados em componentes, fazendo uso dos mecanismos de tratamento de exceções já existentes nas linguagens de programação. No caso da especificação dos requisitos, baseado no método MDCE [14], é feita a antecipação das atividades excepcionais relacionadas ao domínio do problema.

Já na modelagem, ou análise e projeto, são especificadas as exceções relacionadas às interações entre os componentes no nível da arquitetura do sistema, fazendo uma separação explícita entre os comportamentos normal e excepcional antes da fase de implementação. Tal ênfase na arquitetura possibilita uma melhor análise dos fluxos de exceções que ocorrem entre os componentes arquiteturais, permitindo assim a construção de tratadores mais eficientes, além de antecipar a correção de possíveis falhas de especificação.

O método é iterativo e prescritivo, ou seja, todas as atividades que devem ser executadas e a ordem delas são totalmente descritas e distribuídas em sete fases que constituem o ciclo de desenvolvimento, atuando desde a especificação dos requisitos até a construção final do software,como ilustrado na Figura 2.9.

A seguir são detalhadas cada fase do método MDCE+, descrevendo suas características gerais, relativas tanto à especificação do comportamento normal, quanto à especificação do comportamento excepcional dos sistemas de software baseados em componentes.

2.11.1 FASE 1: Especificação e análise dos requisitos

Esta fase inicial tem por objetivo o entendimento do problema a ser resolvido pelo sistema. Isto é feito por meio da elicitação das funcionalidades, ou requisitos, a serem implementados. Um requisito pode ser classificado como **funcional** ou **não funcional** [28]. Os requisitos funcionais podem ser mapeados para os serviços, tarefas ou funções que o sistema deve oferecer. Já os não funcionais, ou de qualidade, não representam funcionalidades diretamente mas interferem na forma em que o sistema deve executá-las, afetando portanto diretamente a arquitetura do software.

As seis atividades que descrevem esta fase são: (i) compreender o problema por meio de descrições; (ii) analisar e representar o domínio do problema; (iii) definir o escopo do sistema;

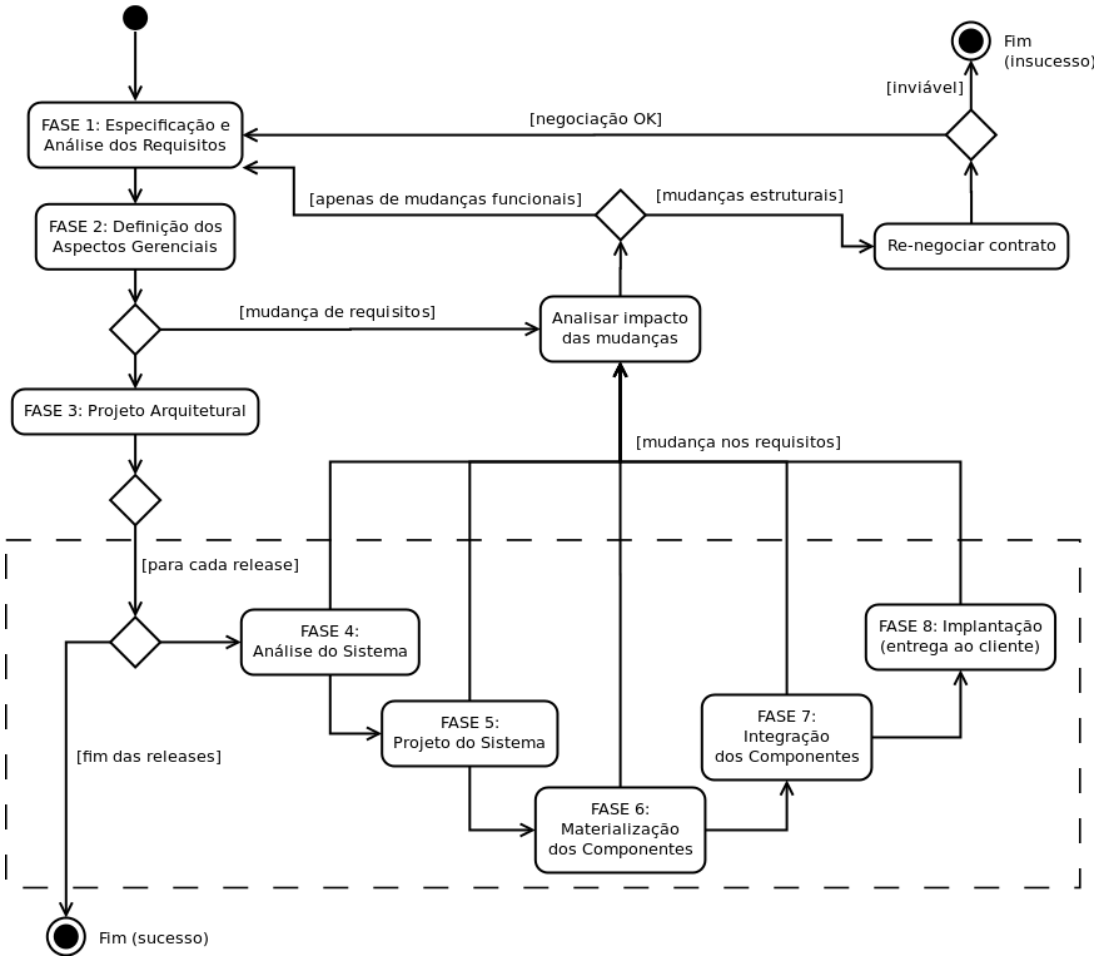


Figura 2.9: Fases do método MDCE+ [8].

(iv) especificar os requisitos funcionais; (v) especificar o comportamento excepcional; e (vi) construir/atualizar o diagrama de casos de uso.

A atividade de **especificar o comportamento excepcional** foi adicionada pelo MCDE+ e visa salientar os desvios do curso normal decorrentes da necessidade de se tratar situações excepcionais ocorridas durante a execução de um dos passos de seus cenários. Para isso, devem ser descritas todas as situações que podem ocorrer para impedir o sucesso, além da funcionalidade do caso de uso. Esta atividade é ainda dividida em duas atividades: (i) identificar exceções; e (ii) especificar os cenários excepcionais.

2.11.2 FASE 2: Definição dos aspectos gerenciais

Introduzida pelo MDCE+, essa fase é responsável por estabelecer as *releases* que definem um desenvolvimento iterativo, além de elucidar as principais restrições para o desenvolvi-

mento e reutilização de componentes. Quanto ao comportamento excepcional, foi definida uma atividade para identificar as funcionalidades críticas para direcionar melhor o custo com a tolerância a falhas. Também é definido, juntamente com o cliente, o cronograma de entrega dos módulos do sistema na forma de entregas sucessivas, o que caracteriza o desenvolvimento iterativo.

É composta por três atividades: (i) definir as restrições para o desenvolvimento; (ii) identificar as funcionalidades críticas; (iii) definir as *releases* para iterações.

A atividade de **identificar as funcionalidades críticas** foi adicionada para melhorar a aplicação dos recursos disponíveis para a implementação efetiva de técnicas de tolerância a falhas, sugerindo a classificação das funcionalidades especificadas de acordo com a sua criticidade: (i) essenciais; (ii) importantes; e (iii) desejáveis.

2.11.3 FASE 3: Projeto arquitetural

Essa fase também foi adicionada pelo método MDCE+ em relação ao MDCE [14]. Diferentemente dos processos clássicos de desenvolvimento, essa fase é executada antes da fase de análise, possibilitando assim uma maior contextualização dos componentes identificados com a estrutura geral do sistema, reforçando assim o fato do MDCE+ ser um método centrado na arquitetura.

Ao enfatizar a arquitetura do sistema consegue-se as seguintes vantagens: (i) uma estruturação abstrata que facilita o entendimento da solução proposta e que propicia uma comunicação mais eficiente entre as partes interessadas; (ii) uma alta granularidade de reutilização de componentes, que juntamente com os princípios do DBC, resultam na diminuição do tempo de desenvolvimento; e (iii) uma maior facilidade para adaptar, manter, evoluir e portar o sistema para outras plataformas pelo fato do uso de conectores arquiteturais que diminuem o acoplamento e explicitam a comunicação entre os componentes.

Para orientar o arquiteto na escolha de uma arquitetura adequada, propõe-se uma simplificação da abordagem adotada pelo método *Architecture Trade-off Analysis Method* (ATAM) [2], a qual é baseada no entendimento de como os atributos de qualidade podem ser satisfeitos pela arquitetura. Tal abordagem é constituída por cinco atividades principais: (i) avaliar restrições arquiteturais; (ii) listar arquiteturas compatíveis; (iii) priorizar os atributos de qualidade; (iv) selecionar arquiteturas candidatas; e (v) selecionar a arquitetura final.

2.11.4 FASE 4: Análise do sistema

Com base nos casos de uso e no modelo de representação do domínio de negócio, ambos criados na fase de Análise de Requisitos, e da arquitetura definida na fase anterior, é feita a identificação das entidades envolvidas na resolução do problema em questão. No

caso do DBC, tais entidades são os componentes em si, portanto a modelagem deve levar em conta suas interfaces (providas e requeridas). Durante esse processo de identificação dos componentes, deve-se sempre ter em mente o posicionamento deles na arquitetura definida.

Os componentes derivados dos casos de uso são geralmente específicos ao domínio do problema, o que dificulta sua reutilização e os torna bons candidatos a uma nova implementação. Já os componentes relativos à lógica de negócio, derivados do modelo de representação do domínio, constituem a infraestrutura relacionada ao domínio, sendo considerados candidatos a reutilização.

Esta fase é composta pelas seguintes atividades: (i) especificar as interfaces da aplicação; (ii) processar as exceções agendadas; (iii) identificar as interfaces de infraestrutura; e (iv) criar os componentes e posicioná-los na arquitetura.

2.11.5 FASE 5: Projeto do sistema

A fase de projeto foi uma das fases que apresentou as principais alterações, devido ao maior foco na arquitetura do sistema.

O objetivo principal da fase de projeto do software é refinar os modelos provenientes da análise, com o objetivo de aproximá-los da implementação, ou seja, a solução do problema elaborada na fase de análise é interpretada e aprofundada do ponto de vista específico da implementação. Por se tratar de desenvolvimento baseado em componentes, o projeto será conduzido utilizando os componentes como unidade básica de abstração, mas a forma como eles serão materializados será decidida na próxima fase.

Para estruturar melhor as suas atividades, o projeto do sistema foi dividido em dois passos principais: (i) análise do aspecto interativo entre os componentes; e (ii) formalização da especificação dos componentes.

2.11.5.1 Analisar a interação entre os componentes

Analisar o aspecto interativo entre as entidades que compõem um sistema é essencial durante o seu desenvolvimento, principalmente do ponto de vista de tolerância a falhas. Um componente isolado não é capaz de oferecer os recursos necessários para identificação e para tratamento dos erros de forma efetiva, característica que fica ainda mais evidente quando a arquitetura deixa explícito o comportamento excepcional.

Como pode ser visto na Figura 2.10, esse passo do projeto é constituído das atividades relacionadas diretamente com o aspecto interativo entre os componentes. Portanto, além da identificação das operações requeridas pelos componentes do sistema e um consequente refinamento da arquitetura, são necessárias oito perspectivas complementares: (i) detalhar os tratadores; (ii) estruturar os componentes arquiteturais; (iii) refinar as entidades

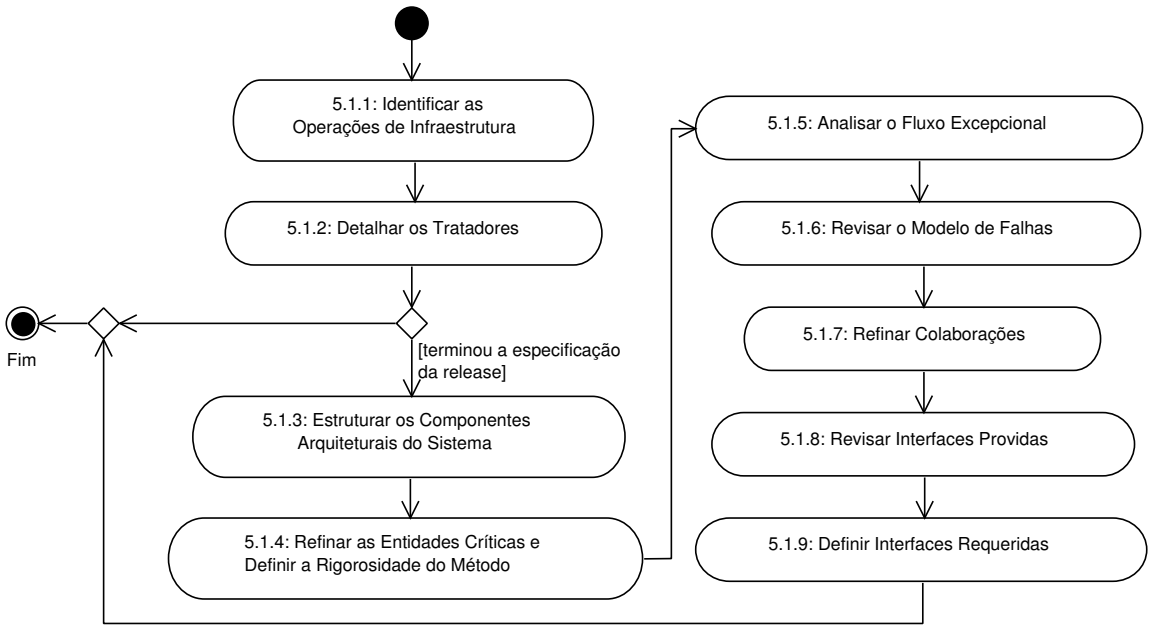


Figura 2.10: Passos da atividade 5.1 Analisar a interação entre os componentes [8].

críticas e definir a rigorosidade do método; (iv) analisar o fluxo excepcional; (v) revisar o modelo de falhas; (vi) refinar as colaborações; (vii) revisar as interfaces providas dos componentes; e (v) definir as interfaces requeridas.

2.11.5.2 Formalização da especificação dos componentes

Com os componentes do sistema já especificados, essa fase consiste no refinamento das suas especificações por meio da formalização das assertivas do componente. Por se tratar de uma etapa de custo elevado do ponto de vista do tempo de desenvolvimento, essa atividade é considerada opcional.

2.11.6 FASE 6: Materialização dos componentes

Por adotar um modelo de componentes, o MDCE+ apresenta refinamentos importantes para essa fase. As principais características apresentadas são: (i) possibilitar a reutilização e a implementação de componentes; (ii) apresentar diretrizes para a reutilização de componentes COTS; (iii) adotar uma hierarquia de exceções; e (iv) adotar o modelo COSMOS para especificação interna dos componentes.

Neste ponto do desenvolvimento, os componentes que implementam a *release* atual estão especificados. O próximo passo é materializar cada um desses componentes, sejam eles normais ou excepcionais.

O método MDCE+ define três formas de se materializar um componente: (i) reutilização de um componente já utilizado pela empresa; (ii) aquisição do componente a partir de um catálogo de terceiros; e (iii) implementação do componente. Mas antes de escolher a forma mais adequada para a materialização de cada componente do sistema, é necessário executar algumas atividades gerais, que são comuns às três abordagens. Estas atividades são:

- Criar os *data types*, que são as especificações dos tipos de dados dos parametros de entrada e do valor de retorno, presentes na assinatura especificada durante a análise interativa entre os componentes;
- Criar as classes de exceções;
- Posicionar as exceções na hierarquia proposta.

Após a materialização dos componentes individuais (normais e excepcionais isolados), dependendo do método de testes adotado, podem ser procedidas atividades de teste de unidade, que visam à verificação da conformidade das funcionalidades de cada componente isoladamente. Adiante, no incio da fase de integração dos componentes, será possível realizar essa mesma verificação para os componentes arquiteturais do sistema, já estruturados segundo o modelo do componente tolerante a falhas ideal.

2.11.7 FASE 7: Integração dos componentes

Apesar de nesse ponto do desenvolvimento os componentes da *release* atual já estarem disponíveis para serem utilizados, esses componentes, isoladamente, não são capazes de oferecer os seus serviços especificados. Para que isso seja possível, é necessário indicar, para cada um dos componentes, quem supre as suas dependências. Essa ligação entre as interfaces requeridas de um componente e as respectivas interfaces providas de outro é feita por meio dos conectores que, consequentemente, são dependentes dos componentes envolvidos.

O método MDCE+ divide essa fase em duas etapas conceitualmente distintas: (i) montar os componentes arquiteturais, onde é realizada a ligação entre os componentes normais e excepcionais; e (ii) materializar a configuração do sistema, onde é realizada a ligação entre os componentes arquiteturais do sistema e a implementação do programa principal.

O diferencial para a fase de montagem do sistema está ligado, principalmente, a três aspectos: (i) diferenciação entre os conceitos de componente arquitetural e componente de desenvolvimento; (ii) destaque dado aos conectores arquiteturais para a implementação dos requisitos não funcionais; e (iii) diretrizes para a implementação de técnicas de reconfiguração dos componentes do sistema e execução redundante.

2.11.8 FASE 8: Implantação (entrega ao cliente)

Diferentemente do método MDCE, o MDCE+ adota a abordagem de desenvolvimento iterativo, baseado em entregas sucessivas de versões do produto ao cliente. Essas entregas são definidas após a especificação das funcionalidades do sistema e contam com a determinante do cliente.

2.12 Resumo

Este capítulo apresentou os conceitos necessários para o entendimento do método proposto Scrum+CE.

Foram apresentadas as características gerais do Método Ágil na Seção 2.1, as quais foram definidas e documentadas, em 2001, por meio dos valores e princípios do Manifesto Ágil. Na sequência foi detalhada, entre as Seções 2.2 e 2.7, a Método Scrum que é um dos métodos que segue tais valores e princípios ágeis e que foi escolhido como referência para a solução proposta neste trabalho.

Também foram apresentados os conceitos de DBC, na Seção 2.8, de Desenvolvimento Centrado na Arquitetura, na Seção 2.9, e de Tolerância a Falhas e Tratamento de Exceções, na Seção 2.10, que são a base para o modelo MDCE+, descrito em detalhes na Seção 2.11, o qual proveu as técnicas para a estruturação do comportamento excepcional utilizadas no Scrum+CE.

O próximo capítulo descreve, em detalhes, todas as fases, artefatos, papéis e eventos do Scrum+CE bem como as técnicas utilizadas para expor o comportamento excepcional.

Capítulo 3

O Método Scrum+CE

Este capítulo apresenta o método Scrum+CE, suas principais características e atividades. O objetivo principal do Scrum+CE é o de adicionar as técnicas e as práticas do método MDCE+ (Seção 2.11) ao Scrum (Seção 2.2) para sistematizar a modelagem e a implementação do comportamento excepcional no desenvolvimento de sistemas baseados em componentes.

As mudanças principais se concentram nas fases Pré-jogo e Jogo, além de adicionar o papel de Dono da Arquitetura. A ênfase na arquitetura de software (Seção 2.9) possibilita, durante o Pré-jogo, uma análise mais apurada dos fluxos de exceções entre os componentes arquiteturais do sistema. Essa análise mais criteriosa permite a construção de tratadores mais eficientes, além de antecipar a correção de possíveis falhas de especificação.

Este capítulo inicia-se com uma visão geral do método Scrum+CE na Seção 3.1 e depois segue a mesma divisão utilizada para descrever o Scrum (fases, pilares, artefatos, equipe e eventos), porém descrevendo apenas as características e atividades adicionadas ao método, provenientes das técnicas do MDCE+. Este capítulo não trata detalhes das atividades relativas ao comportamento normal pois ele já é considerado dentro do fluxo convencional do próprio Scrum.

3.1 Visão geral do Método Scrum+CE

O Scrum+CE propõe mudanças nas fases Pré-jogo (Seção 2.3.1) e Jogo (Seção 2.3.2) no que diz respeito à identificação de Estórias Excepcionais, à adição de Testes de Aceitação Excepcionais e ao refinamento da arquitetura com elementos excepcionais explícitos. Além disso, técnicas ágeis que melhoram a confiabilidade como TDD (Seção 2.1.1.3) e Integração Contínua (Seção 2.1.1.4) são fortemente recomendadas, mas não formalmente obrigatórias, assim como no Scrum.

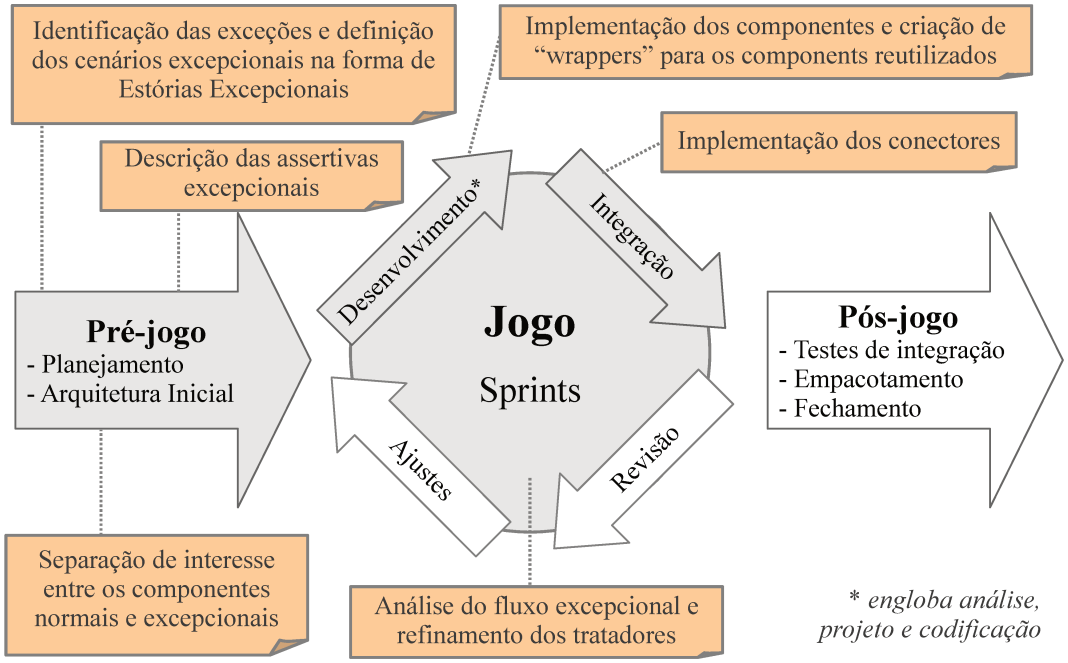


Figura 3.1: Interferências do método MDCE+ nas fases do Scrum.

A Figura 3.1 mostra as fases do Scrum com as atividades do MDCE+ (em cinza) que devem ser desenvolvidas nas respectivas fases. Note que as maiores modificações são na fase Pré-jogo, na qual o comportamento excepcional é documentado na forma de Estórias Excepcionais e de Testes de Aceitação nas Estórias de Usuário (Seção 2.1.1.1) que validem especificamente os fluxos excepcionais. Além disso, a Arquitetura Inicial passa a expor os componentes excepcionais conforme o modelo de componente tolerante a falhas ideal (Seção 2.10.3), ou simplesmente componente ideal, que é categorizado em dois tipos: normal, que produz respostas corretas; e excepcional (ou anormal), que é executado quando um erro é detectado.

Essa Arquitetura Inicial será definida pela Equipe de Desenvolvimento, respeitando o princípio do Manifesto Ágil (Seção 2.1) que dita que "as melhores arquiteturas, requisitos e projetos emergem de times auto-organizáveis". Porém, na proposta apresentada, terá a ajuda e será de responsabilidade do novo papel introduzido de Dono da Arquitetura.

Dentro da fase Jogo, teremos como entradas as Estórias de Usuário e as Estórias Excepcionais igualmente priorizadas no *Backlog* do Produto. Elas deverão entrar no *Backlog* do *Sprint* de acordo com suas prioridades e dependências. Como no Scrum, o Dono do Produto é o responsável por priorizar as estórias no *Backlog* do Produto por ordem de valor de negócio, as Estórias Excepcionais têm um aspecto não funcional e técnico, então é de responsabilidade do novo papel de Dono da Arquitetura ajudar o Dono

do Produto a priorizá-las de forma coerente de acordo com a criticidade e dependências com as outras Estórias de Usuário. Dessa forma, fica eliminado o risco das Estórias Excepcionais terem sempre prioridade baixa e por sua vez elas não serem selecionadas para implementação nos próximos *sprints*. Isto deve ser um ponto de atenção durante as Reuniões de Planejamento dos *Sprints*, que teve uma terceira parte adicionada justamente para revisar o impacto na Arquitetura.

A combinação com o MDCE+ adiciona o artefato de Arquitetura Inicial e afeta os artefatos *Backlog* do Produto, com a adição das Estórias Excepcionais, e o *Backlog* do *Sprint*, com as tarefas específicas de implementação do comportamento excepcional. Uma vez dentro dos *backlogs*, as estórias e tarefas excepcionais serão tratadas e implementadas normalmente, como um requisito funcional comum.

Como o Scrum+CE injeta as práticas do MDCE+ no Scrum, a seguir serão detalhadas tais práticas seguindo a mesma estrutura apresentada na Seção 2.2. Neste capítulo serão mostradas apenas as adições feitas ao Scrum.

3.2 Fases do Scrum+CE

O Scrum+CE não adiciona fases ao Scrum, portanto ficam mantidas as três fases: Pré-jogo, Jogo e Pós-jogo, como descrito na Seção 2.3. Conforme já mostrado na Figura 3.1, foram adicionadas às fases do Scrum atividades do MDCE+. Tais atividades serão detalhadas nas etapas do Pré-jogo de Planejamento e de Arquitetura e Projeto iniciais, bem como nas seções seguintes relativas aos papéis, artefatos e eventos.

A Tabela 3.1 mostra a relação entre as fases do Scrum e do MDCE+. Note que devido ao aspecto iterativo da fase Jogo, no formato de *sprints*, as fases equivalentes do MDCE+ podem ser repetidas.

Assim como as fases do MDCE+ foram mapeadas para as fases e eventos do Scrum, as atividades relacionadas ao comportamento excepcional de cada uma delas foram adaptadas com o objetivo de manter a agilidade e, ao mesmo tempo, aumentar a confiabilidade do processo.

A seguir, as atividades adicionadas às três fases são descritas em maiores detalhes.

3.2.1 Fase Pré-jogo

A fase Pré-jogo continua dividida em duas atividades: (i) planejamento; e (ii) arquitetura e projeto iniciais.

3.2.1.1 Planejamento

O planejamento continua contendo todas as atividades do Scrum, definidas na Seção 2.3.1.1. Além dessas, ainda lhe são adicionadas atividades relativas às técnicas de

Tabela 3.1: Relação entre as fases do MDCE+ e do Scrum.

Fases do Scrum	**Eventos do Scrum**	**Fases do MDCE+**
Pré-jogo	Planejamento	FASE 1: Especificação e análise dos requisitos FASE 2: Definição dos aspectos gerenciais
	Arquitetura Inicial	FASE 3: Projeto arquitetural
Jogo	Reunião de Planejamento do *Sprint*	FASE 1: Especificação e análise dos requisitos FASE 3: Projeto arquitetural FASE 4: Análise do sistema FASE 5: Projeto do sistema
	Sprint	FASE 6: Materialização dos componentes FASE 7: Integração dos componentes
	Reunião de Revisão do *Sprint*	FASE 1: Especificação e análise dos requisitos FASE 8: Implantação (demonstração ao cliente)
Pós-jogo	Testes de integração	FASE 7: Integração dos componentes
	Empacotamento	FASE 8: Implantação (entrega ao cliente)
	Fechamento	

(i) identificação das exceções e definição dos cenários excepcionais na forma de Estórias Excepcionais; e (ii) descrição das assertivas excepcionais, conforme identificado na Figura 3.1. Essa etapa equivale às fases (1) especificação e análise dos requisitos (Seção 2.11.1) e (2) definição dos aspectos gerenciais (Seção 2.11.2) do MDCE+, visto na Tabela 3.1, portanto todas as atividades relativas ao tratamento ao comportamento excepcional de tais fases serão adaptadas e adicionadas nesta à fase do Scrum+CE. Sendo assim, além das atividades já listadas na subseção do Seção 2.3.1, também devem ser executadas:

- A atividade de elicitação dos requisitos excepcionais que irão ser adicionados ao *Backlog* do Produto juntamente com os demais requisitos; e
- A atividade de revisão e ajustes dos itens do *backlog* adicionando novas assertivas que tratarão dos comportamentos excepcionais detectados anteriormente.

No caso de utilizar a técnica de Estória de Usuário (Seção 2.1.1.1), os requisitos excepcionais devem ser formalizados no mesmo formato, ou seja, conter uma breve descrição e a confirmação na forma de Testes de Aceitação. Uma vez documentadas as Estórias Excepcionais, elas entram no *Backlog* do Produto com as demais estórias. Porém, com um marcador de Estória Excepcional.

Durante a revisão do *Backlog* do Produto, as demais estórias devem ser revisadas com o objetivo de adicionar mais Testes de Aceitação, oriundos das Estórias Excepcionais, gerando assim uma melhor formalização das assertivas excepcionais e, consequentemente, testes mais criteriosos.

A Fase 2 do MDCE+, definição dos aspectos gerenciais, sugere a classificação das funcionalidades de acordo com a sua criticidade: (i) essenciais; (ii) importantes; e (iii)

desejáveis. No Scrum, e, por consequência, no Scrum+CE, isso ocorre quando o Dono do Produto prioriza e ordenada as estórias no *Backlog* do Produto. Como as Estórias Excepcionais podem não ressaltar o valor de negócio em comparação às demais estórias, é de responsabilidade do Dono da Arquitetura ajudar o Dono do Produto a entender a criticidade e impactos gerados pelas Estórias Excepcionais de forma que ele consiga priorizá-las de modo coerente.

Note que não ocorre uma priorização explícita das Estórias Excepcionais em detrimento das demais. Essa análise deve ser feita em conjunto respeitando o grau de confiabilidade desejado. Essa etapa é crítica ao processo pois, explicitar o comportamento excepcional com a adição de estórias no *Backlog* do Produto fará o escopo do sistema crescer, em tamanho, aos olhos do cliente e, por consequência, aumentará a estimativa de custo e prazo. Isso será visível pois as Estórias Excepcionais serão estimadas em Pontos de Estória como as demais, ou seja, com mais estórias, a soma dos pontos do *backlog* irá aumentar.

3.2.1.2 Definição de "*Pronto*"

A Definição de "*Pronto*" continua tendo um papel fundamental no Scrum+CE, seguindo exatamente o mesmo conceito do Scrum, descrita na Seção 2.3.1.2, e sendo o compartilhamento e entendimento coletivo do que deve ser executado para transformar um item do *Backlog* do Produto em algo demonstrável no Incremento do sistema.

Todavia, no Scrum+CE o tratamento do comportamento excepcional também deve ser explicitado nesta parte do processo. Portanto, seja qual for a definição proferida pela equipe, o Dono da Arquitetura tem a responsabilidade de adicionar à tal definição o seguinte trecho: **"...e todas as exceções foram devidamente tratadas..."**. Com isso, a ênfase da importância do comportamento excepcional ficará evidente e transparente para toda a equipe, além de disponível durante todo o ciclo de desenvolvimento, uma vez que tal definição é relembrada a cada *sprint*.

3.2.1.3 Arquitetura e projeto

Assim como no planejamento, essa etapa continua seguindo todas as atividades definidas pelo Scrum, descritas na Seção 2.3.1. Porém, no Scrum+CE foram adicionados um novo papel de Dono da Arquitetura e um artefato obrigatório de saída nesta fase, a Arquitetura Inicial. Tanto o novo papel quanto o artefato têm como objetivos explicitar e focar na arquitetura, trazendo os conceitos de Desenvolvimento Centrado na Arquitetura para dentro de um modelo ágil.

A arquitetura também deve seguir os conceitos de DBC, ao mesmo tempo que, deve seguir uma documentação mais ágil, ou seja, ser composta por, no mínimo, um diagrama

de arquitetura no nível de componentes e as relações entre os mesmos. Além do diagrama, é aconselhável criar um documento simples e resumido (normalmente na forma de um *Wiki*[1] para fomentar a colaboração) com as decisões arquiteturais mais importantes, tecnologia usada, infraestrutura, modelo de dados, entre outros. Como um dos princípios do Manifesto Ágil dita: "As melhores arquiteturas, requisitos e projetos emergem de times auto-organizáveis"; esse artefato deve servir como guia e ser acessível e flexível o suficiente para ser atualizado durante o ciclo de desenvolvimento.

Além de disponibilizar o diagrama de arquitetura, o Scrum+CE adiciona a atividade de **separação de interesse entre os componentes normais e excepcionais** como visto na Figura 3.1. Esta separação deve ser feita transformando a representação dos componentes críticos no modelo de estrutura do componente ideal (Seção 2.10.3).

Uma vez finalizado o planejamento, declarada a definição de *Pronto*, e definida e disponibilizada a Arquitetura Inicial, a fase Jogo está habilitada a ser iniciada.

3.2.2 Fase Jogo

A estrutura da fase Jogo não sofrerá modificações em relação ao Scrum, e continua sendo composta por *sprints*, de período de tempo fixo e predefinido que pode variar de uma a quatro semanas, conforme descrito na Seção 2.3.2. As quatro atividades de (i) desenvolvimento; (ii) empacotamento ou integração; (iii) revisão; e (iv) ajustes; continuam ocorrendo exatamente da mesma forma, apesar das três atividades destacadas na Figura 3.1, que implicarão as seguintes tarefas:

- **Implementação dos componentes e criação de "wrappers" para os componentes reutilizados**: deve ocorrer de forma natural durante o desenvolvimento das tarefas do *Backlog* do *Sprint*, uma vez que, as Estórias Excepcionais são quebradas em tarefas durante a Reunião de Planejamento do *Sprint*;

- **Implementação dos conectores**: também ocorre de forma transparente durante a Integração proveniente das tarefas do *Backlog* do *Sprint*, igualmente geradas durante a Reunião de Planejamento do *Sprint*;

- **Análise do fluxo excepcional e refinamento dos tratadores**: ocorre no início do *sprint*, durante a Reunião de Planejamento do *Sprint*, na qual o Dono da Arquitetura, juntamente com a Equipe de Desenvolvimento, fazem a análise das Estórias Excepcionais e dos Testes de Aceitação que abordam de qualquer comportamento anormal, produzindo assim, tarefas técnicas que implementam tais funcionalidades.

[1]Termo do dialeto havaiano, que pode ser traduzido como "rápido, ligeiro, veloz", utilizado para identificar um tipo específico de software colaborativo que permite a edição coletiva de documentos.

O Scrum+CE afeta diretamente os eventos de Reunião de Planejamento e Reunião de Revisão do *Sprint*, que serão mais detalhados nas Seções 3.6.2 e 3.6.4 respectivamente.

3.2.3 Fase Pós-jogo

No Pós-jogo, ocorre o fechamento que é a preparação para o lançamento da *release* em produção, ou seja, no ambiente no qual os usuários finais utilizarão o sistema. Como o Scrum+CE acrescenta práticas ao Scrum, essa fase não sofre nenhuma alteração, seguindo exatamente os mesmos passos descritos na Seção 2.3.3.

3.3 Pilares do Scrum+CE

O Scrum+CE segue à risca os pilares do Scrum, descritos na Seção 2.4, que aplica uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos. Os três pilares que sustentam a implementação de um controle do processo continuam sendo: (i) transparência, (ii) inspeção e (iii) adaptação. Como não houve modificação dos pilares, a descrição deles não será repetida nesta seção.

3.4 Artefatos do Scrum+CE

Os artefatos do Scrum representam o trabalho ou o valor que são úteis no sentido de oferecer máxima transparência e oportunidades para inspeção e adaptação. Eles são manipulados pela Equipe Scrum com o objetivo de entregar um Incremento *Pronto*. O Scrum+CE faz uso de todos os artefatos do Scrum, descritos na Seção 2.5, e ainda adiciona a Arquitetura Inicial descrita a seguir.

3.4.1 Arquitetura Inicial

Com o objetivo de introduzir um fator de visibilidade e importância maior para a arquitetura, esse artefato foi introduzido como obrigatório no Scrum+CE. Tal obrigatoriedade não transforma o Scrum+CE em um método de Desenvolvimento Centrado na Arquitetura, como descrito na Seção 2.9, uma vez que apenas traz um enfoque na sua definição mínima e inicial. A arquitetura deve ser de preferência componentizada (DBC) e manter um nível alto de abstração.

Esse artefato deve ser formalizado por, no mínimo, um diagrama no nível de componentes e das relações entre eles. Além do diagrama, deve ser adicionas qualquer outra informação relevante ao projeto e que tenha valor para a equipe. Esse artefato deve servir

como guia e ser acessível e flexível o suficiente para ser atualizado durante o ciclo de desenvolvimento.

A criação inicial, durante o planejamento do Pré-jogo, bem como a atualização desse artefato, no decorrer dos *sprints*, são de responsabilidade do novo papel de Dono da Arquitetura. Todavia, como a colaboração entre todos os envolvidos é crucial em projetos ágeis, tal criação e atualização devem ser feitas em conjunto com o Dono do Produto e com a Equipe de Desenvolvimento. Vale ressaltar que, no Scrum, a mudança nos requisitos durante o ciclo de desenvolvimento é natural, portanto isso deve ser levado em conta no início da definição da arquitetura, pois ela deve ser flexível para acomodar as alterações futuras. Por este motivo, deve ser mantido um alto nível de abstração e algumas decisões que causam grande impacto em sua estrutura devem ser postergadas até um momento mais propício, quando os requisitos que demandam tal decisão, estiverem mais maduros.

O papel de Dono da Arquitetura será descrito na Seção 3.5.1.

3.4.2 *Backlog* do Produto

Continua sendo a lista única e ordenada de tudo que possa ser necessário para o desenvolvimento e para a entrega de um sistema de software, contendo todos os requisitos (funcionais e não funcionais), melhorias e defeitos, como detalhado na Seção 2.5.1. Cada item do *Backlog* do Produto possui, no mínimo, uma descrição, um número de ordem e uma estimativa.

O Scrum+CE adiciona atividades nas fases e eventos que criam, adicionam e revisam os itens do *Backlog* do Produto com o objetivo de explicitar o comportamento excepcional dos requisitos. Como consequência, um sistema desenvolvido utilizando Scrum+CE teria uma quantidade maior de itens no *Backlog* do Produto, em relação ao Scrum, uma vez que estaria-se documentando e trazendo para a discussão durante os eventos, requisitos que eram deixados a critério da Equipe de Desenvolvimento durante o decorrer dos *sprints*.

O *Backlog* do Produto continua sendo dinâmico e completo, evoluindo junto com o produto, mudando constantemente para melhor atender as necessidades de negócio, mantendo assim a competitividade do sistema. Tal dinamismo irá impactar diretamente a arquitetura, o que ressalta ainda mais o caráter flexível e adaptável que ela deve ter.

O Dono do Produto continua sendo o responsável pelo conteúdo, disponibilidade e ordenação do *Backlog* do Produto. Porém, com a introdução de requisitos excepcionais, que, na maioria das vezes serão mais técnicos do que de negócio, cabe ao Dono da Arquitetura colaborar fortemente com o Dono do Produto no sentido de orientá-lo para a importância de tais requisitos na confiabilidade do sistema. Esta colaboração será crucial, principalmente no planejamento do Pré-jogo e nas reuniões de planejamento dos *sprints*, para que os requisitos excepcionais sejam corretamente priorizados e ordenados.

A Equipe de Desenvolvimento ainda é responsável pelas estimativas dos itens do *Backlog* do Produto, e para isso é auxiliada tanto pelo Dono do Produto, que ajuda a detalhar os impactos no negócio; como pelo Dono da Arquitetura, que esclarece o grau de impacto de tal item na arquitetura.

3.4.3 Objetivo do *Sprint*

O Objetivo do *Sprint* continua sendo uma sentença declarada pela Equipe de Desenvolvimento juntamente com o Dono do Produto e o Dono da Arquitetura durante a Reunião de Planejamento do *Sprint*, da mesma maneira descrita na Seção 2.5.2.

3.4.4 *Backlog* do *Sprint*

O *Backlog* do *Sprint* também não sofreu alterações estruturais e continua sendo o conjunto de itens do *Backlog* do Produto selecionados para o *sprint*, durante a Reunião de Planejamento do *Sprint*, que define o trabalho que a Equipe de Desenvolvimento irá desempenhar para transformar os itens do *Backlog* do Produto em Incrementos *Prontos*, assim como a descrição da Seção 2.5.3.

Porém no Scrum+CE, como existirá uma visibilidade maior sobre a arquitetura, devido ao novo artefato obrigatório de Arquitetura Inicial e ao maior número de Testes de Aceitação, mais tarefas serão geradas para implementar os requisitos provenientes do comportamento excepcional explicitado pelas técnicas já mencionadas. Isso também será viabilizado pela alteração feita na Definição de *Pronto*, descrita na Seção 3.2.1.2, que obriga o tratamento de todas as exceções e da participação efetiva do Dono da Arquitetura na criação e manutenção deste *backlog*.

3.4.5 Incremento

O Incremento permanece sendo o produto final da implementação de todas as tarefas do *Backlog* do *Sprint*, descrito na Seção 2.5.4. Todavia, por estar mais completo no Scrum+CE, resultará em um Incremento mais confiável a cada *sprint*, pois ele respeita a Definição de *Pronto* mais criteriosa do Scrum+CE (Seção 3.2.1.2) estabelecida na fase de planejamento.

3.5 Equipe do Scrum+CE

A equipe do Scrum+CE teve a adição do Dono da Arquitetura, além dos papéis do Scrum de Dono do Produto, da Equipe de Desenvolvimento e do Scrum Master descritos

na Seção 2.6. A equipe se mantém auto-organizada e multifuncional, apenas destacando-se um papel mais específico para o arquiteto. Com esse modelo de equipe, mesmo que minimamente alterado, continua se alcançando melhor flexibilidade, criatividade e produtividade.

3.5.1 Dono da Arquitetura

O Dono da Arquitetura foi introduzido pelo Scrum+CE com o objetivo de salientar a importância da arquitetura no ciclo de desenvolvimento. Ele deve ser representado por apenas uma pessoa, com experiência em arquitetura, liderança, análise e projeto de software baseado em componentes. Ele participará ativamente de todas as fases do processo e será responsável pela criação e pela atualização do novo artefato de Arquitetura Inicial. Tais atividades já existiam no Scrum, mas ficavam a cargo da Equipe de Desenvolvimento e espalhada durante os *sprints*. No Scrum+CE, com esse novo papel, antecipa-se e obriga-se a criação e a documentação da arquitetura utilizando técnicas de DBC e expondo os componentes excepcionais seguindo o modelo de componente ideal.

3.5.2 Dono do Produto

O Dono do Produto não sofreu modificações, continua ainda sendo responsável por maximizar o valor de negócio do produto e da qualidade do trabalho pela Equipe de Desenvolvimento, por meio do gerenciamento do *Backlog* do Produto. As atividades desempenhadas por ele podem ser vistas na Seção 2.6.1 do Scrum.

3.5.3 Equipe de Desenvolvimento

O mesmo ocorre com a Equipe de Desenvolvimento, na qual as características estão descritas na Seção 2.6.2 do Scrum, que apenas teve o papel do Dono da Arquitetura destacado para um novo papel.

O tamanho ótimo de uma Equipe de Desenvolvimento recomendado se mantém entre 5 e 9 membros, excluindo os Donos da Arquitetura e do Produto, e o Scrum Master, a não ser que eles também executem o trabalho do *Backlog* do *Sprint*, o que pode ser mais provável, mesmo que não recomendado, no caso do Dono da Arquitetura.

3.5.4 Scrum Master

Como o Scrum+CE apenas adiciona práticas, um papel e um artefato ao Scrum, mantendo toda sua estrutura, o Scrum Master continua existindo da mesma forma sendo o responsável por garantir que o processo seja entendido e aplicado, obedecendo aos pila-

res, fases, práticas, eventos e regras. Portanto, suas características, atividades e relacionamento com os outros papéis, descritas na Seção 3.5.4, continuam sendo totalmente aplicáveis.

3.6 Eventos do Scrum+CE

O Scrum+CE mantém exatamente os mesmos eventos do Scrum, adicionando apenas uma terceira parte na Reunião de Planejamento do *Sprint* para analisar o impacto da implementação dos itens selecionados na arquitetura com o auxílio do Dono da Arquitetura, com o objetivo de gerar tarefas que tratem das mudanças necessárias.

A seguir, seguem os eventos na mesma ordem que foram descritos na Seção 2.7.

3.6.1 *Sprint*

O Scrum+CE segue iterativo no formato de *sprints* com duração fixa e predefinida de duas a quatro semanas, que se repete seguidamente na implementação dos requisitos do *Backlog* do Produto. Com isso, os eventos, estrutura e ordem das atividades continuam as mesmas do Scrum.

A Figura 3.2 mostra, em detalhes, o fluxo do *sprint* que ocorre dentro da fase Jogo, diferenciando os papéis, artefatos e cerimônias ou eventos. Em relação à Figura 2.3 do Scrum, foram adicionados:

- Um papel de Dono da Arquitetura;
- Um artefato de Arquitetura Inicial;
- Requisitos de comportamento excepcional no *Backlog* do Produto no formato de Estórias Excepcionais e Testes de Aceitação mais criteriosos;
- Participação obrigatória do Dono da Arquitetura nas reuniões de Planejamento e Reunião de Revisão do *Sprint*, e opcional no Scrum Diário e na Retrospectiva.

Os eventos de um *sprint* ainda são os mesmos descritos na Seção 2.7 e a seguir são salientadas as adições feitas pelo Scrum+CE.

3.6.2 Reunião de Planejamento do *Sprint*

A Reunião de Planejamento do *Sprint* é o primeiro evento de um *sprint* e visa criar um plano sobre o trabalho que será executado. Tal plano é feito de forma colaborativa e com

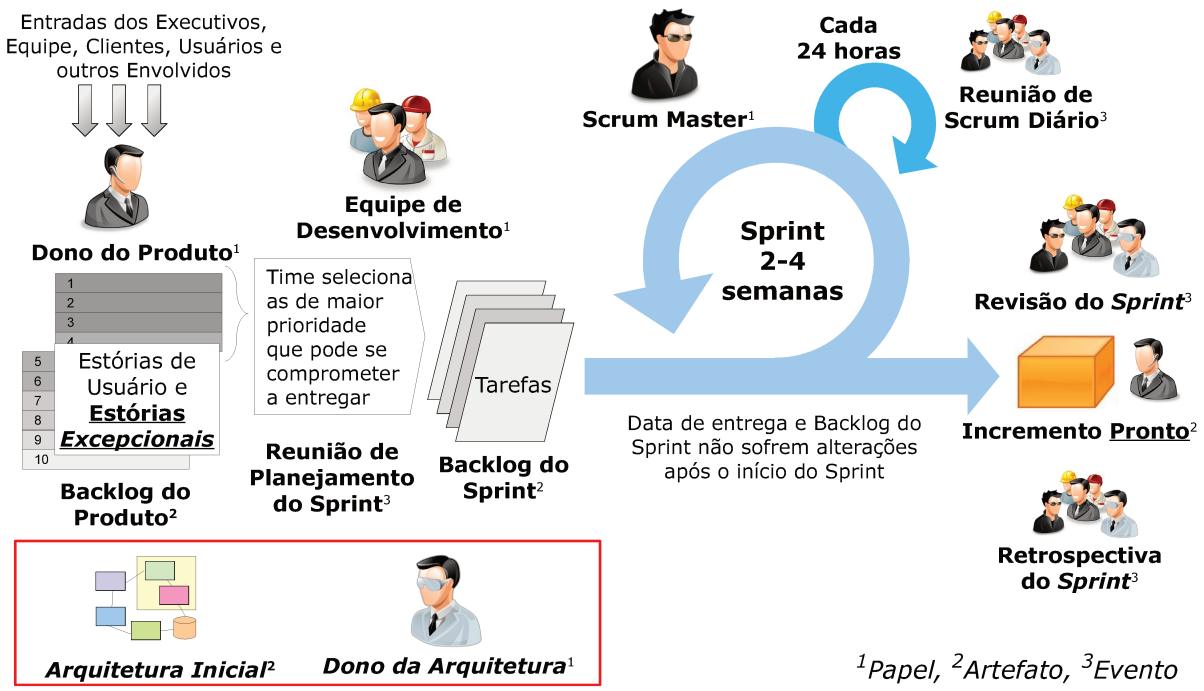


Figura 3.2: Detalhamento da fase Jogo do Scrum+CE.

a participação de toda equipe. Essa reunião tem duração fixa de oito horas para *sprints* de quatro semanas, sendo proporcionalmente menor para durações menores.

Esse evento tem, como artefatos de entrada, a Arquitetura Inicial, o *Backlog* do Produto e o Incremento atual do sistema; e de saída, o Objetivo do *Sprint* e o *Backlog* do *Sprint*.

A reunião de planejamento é dividida em três partes: (i) **O que** vai ser entregue como resultado do Incremento do próximo *sprint*; (ii) Qual o **impacto** de tais mudanças na arquitetura; (iii) **Como** será realizado o trabalho necessário para adaptar a arquitetura e entregar o Incremento.

A segunda parte, referente à arquitetura, foi inserida pelo Scrum+CE e posicionada entre a seleção dos itens para o *sprint* e a análise e planejamento da Equipe de Desenvolvimento. Como visto na Tabela 3.1, tal técnica foi inserida referente à FASE 3 Projeto arquitetural (Seção 2.11.3) do MDCE+, porém nesse caso, tal fase está dentro dos *sprint*, o que resulta em um aspecto totalmente iterativo e incremental também na arquitetura. O fato de ter sido posicionada entre as duas partes existentes no Scrum, e não ao final delas, foi estratégico para reforçar a importância da arquitetura e por sua vez do comportamento excepcional.

Cada uma das partes será descrita a seguir:

Parte 1: O que será *Pronto* neste *sprint*? Nesta parte, o Dono do Produto apresenta os itens do *Backlog* do Produto ordenados e o Dono da Arquitetura apresenta a Arquitetura Inicial para a Equipe de Desenvolvimento. Toda a equipe colabora no entendimento do trabalho do *sprint* levando em consideração o último Incremento do produto, a arquitetura atual, a capacidade projetada e o desempenho da Equipe de Desenvolvimento durante os *sprints* anteriores. A Equipe de Desenvolvimento então avalia o quanto consegue prever para ser implementado no próximo *sprint*. Uma vez feita essa previsão, a equipe de Scrum+CE produz o Objetivo do *Sprint*.

Parte 2: Como o trabalho escolhido afetará a arquitetura? Tendo selecionado o trabalho do *sprint*, a Equipe de Desenvolvimento faz uma análise, guiada pelo Dono da Arquitetura, sobre a existência de algum impacto na arquitetura atual gerado pela implementação de tais funcionalidades. Além disso, como a arquitetura expõe o comportamento excepcional por intermédio dos componentes ideias, essa será uma oportunidade para rever e para discutir os melhores formas de se modelar e implementar o tratamento de tais exceções, gerando assim mais tarefas na parte seguinte.

Parte 3: Como o trabalho escolhido será *Pronto*? Finalizada a parte 2, a Equipe de Desenvolvimento decide como irá transformar estas funcionalidades em um Incremento *Pronto* durante o *sprint*. Lembrando que no Scrum+CE, a Definição de *Pronto* considerada neste momento, adicionou a obrigatoriedade do tratamento de todas as exceções, e é de responsabilidade de toda a equipe, reler e relembrar tal definição. Sendo assim, continua sendo feita a decomposição dos requisitos dos *Backlog* do Produto em tarefas executáveis, ainda com duração máxima de dois dias de trabalho. Tais tarefas formam o *Backlog* do *Sprint* como resultado da análise e projeto dos requisitos, além do planejamento de como implementá-los. Com a exposição obrigatória da arquitetura, das Estórias Excepcionais, dos Testes de Aceitação mais criteriosos, de uma definição de *Pronto* que exige o tratamento de todas as exceções e de uma parte exclusiva para analisar o impacto das mudanças na arquitetura, é esperado que sejam geradas mais tarefas ao final da Reunião de Planejamento do *Sprint* em relação ao Scrum, considerando os mesmos itens selecionados do *Backlog* do Produto.

A participação do Dono do Produto permanece obrigatória na primeira parte e altamente recomendada nas duas seguintes. Para esse evento ainda podem ser convidados outras pessoas envolvidas, como especialistas no domínio do sistema ou na tecnologia, para colaborar no entendimento dos requisitos e no desenho da solução. Já o Dono da Arquitetura deve participar ativamente de todas as partes deste evento.

Ao final da Reunião de Planejamento do *Sprint*, a Equipe de Desenvolvimento e o Dono da Arquitetura devem estar aptos a explicar ao Dono do Produto e ao Scrum Master como pretendem trabalhar de maneira auto-organizada para criar o Incremento com o grau de confiabilidade desejado e atingir o Objetivo do *Sprint*.

3.6.3 Scrum Diário

A Reunião do Scrum Diário, não foi alterada, em relação ao descrito na Seção 2.7.3, e permanece sendo um evento com 15 minutos fixos, coordenado pelo Scrum Master, na qual a Equipe de Desenvolvimento sincroniza as atividades e cria um plano para as próximas 24 horas. A participação do Dono da Arquitetura é opcional. Continuam válidas as práticas de manter o horário e local da reunião, e seguir um roteiro simples, composto pelas três perguntas que devem ser respondidas por cada membro da Equipe de Desenvolvimento. As perguntas são:

1. O que conseguiu realizar desde o último encontro?
2. O que vai ser *Pronto* até o próximo encontro?
3. Quais são os obstáculos que estão no seu caminho?

O Scrum Master continua sendo o responsável por gerenciar os impedimentos verbalizados na terceira pergunta, e de endereçar suas resoluções fora da reunião. A Equipe de Desenvolvimento usa o Scrum Diário para avaliar o progresso na direção do Objetivo do *Sprint*, aumentando assim a probabilidade de atingi-lo no prazo planejado.

3.6.4 Reunião de Revisão do *Sprint*

A Reunião de Revisão do *Sprint* continua sendo executada no final do *sprint* para inspecionar o Incremento e adaptá-lo ao *Backlog* do Produto. As atividades a serem realizadas continuam as mesmas do Scrum, descritas na Seção 2.7.4, e o Dono da Arquitetura deve estar presente. A Equipe de Desenvolvimento demonstra o trabalho que foi *Pronto* ocorrendo a validação do Incremento produzido durante o *sprint* pelo Dono do Produto em relação ao Objetivo do *Sprint* declarado na reunião de planejamento.

Durante esta validação, ocorrem também a revisão e a adição de novos requisitos no *Backlog* do Produto. Da mesma forma que ocorrem no planejamento do Pré-jogo, descrito na Seção 3.2.1, as atividades técnicas relativas à (i) identificação das exceções e definição dos cenários excepcionais na forma de Estórias Excepcionais e (ii) descrição das assertivas excepcionais devem continuar sendo feitas com o mesmo critério, uma vez que em projetos ágeis os requisitos são dinâmicos. Essa é uma oportunidade de colaboração e de adaptação.

Finalizada a reunião, o *Backlog* do Produto está revisado, atualizado e reordenado levando-se em consideração o que foi entregue neste *sprint* e as novas demandas que possam ter surgido.

3.6.5 Retrospectiva do *Sprint*

A Retrospectiva do *Sprint* segue idêntica ao Scrum, conforme a Seção 2.7.5, ocorrendo após a Reunião de Revisão do *Sprint* e antes da próxima Reunião de Planejamento do *Sprint*. Fica mantido o propósito de inspecionar como foi o último *sprint* no que diz respeito a pessoas, relações, processos e ferramentas, visando aumentar a efetividade no uso do Scrum+CE e na qualidade do produto entregue, podendo-se até adaptar a definição de *Pronto* quando apropriado. Deve-se também discutir a efetividade do papel de Dono da Arquitetura e do artefato de Arquitetura Inicial introduzidos pelo Scrum+CE da mesma forma que a utilização da técnica de Estórias Excepcionais no aumento da confiabilidade do Incremento atual.

No final da Retrospectiva do *Sprint*, a equipe deve identificar melhorias que serão aplicadas no próximo *sprint*. Apesar de melhorias poderem ser adotadas a qualquer momento, esse evento oferece um momento exclusivo, dedicado e focado na inspeção e na adaptação do processo Scrum+CE.

3.7 Resumo

Esse capítulo apresentou o Scrum+CE, que é um processo baseado no Scrum com adição de práticas de tratamento e exposição do comportamento excepcional do MDCE+. O Scrum+CE adicionou um papel e um artefato focados na arquitetura e na explicitação do comportamento excepcional. Também afetou as fases Pré-jogo e Jogo no que diz respeito a elicitar Estórias Excepcionais e adicionar Testes de Aceitação Excepcionais.

O próximo capítulo descreve um experimento controlado, realizado com o objetivo de confrontar o uso do Scrum+CE com o Scrum.

Capítulo 4

Experimento controlado

Este capítulo mostra os principais passos da execução do experimento controlado quantitativo realizado com o intuito de avaliar a aplicabilidade prática do método Scrum+CE descrito no Capítulo 3. O experimento controlado, ou simplesmente experimento, consiste na implementação de um sistema de software do domínio de sistemas de informação com requisitos de confiabilidade relativos à consistência de dados. Trata-se de um sistema de controle de pontos de parada e horários dos itinerários de linhas de ônibus corporativos.

O conteúdo desse capítulo está exposto conforme mostrado a seguir.

A Seção 4.1 descreve o sistema que foi implementado durante o experimento. A Seção 4.2 apresenta alguns detalhes relevantes ao experimento, como por exemplo cronograma, preparação e infraestrutura utilizados para a sua condução. A Seção 4.3 relata os critérios empregados no planejamento do experimento e descreve as métricas utilizadas. As Seções 4.4, 4.5 e 4.6 mostram um resumo da execução do experimento, apresentando os principais artefatos produzidos para a especificação e a implementação de uma *release* do sistema em cada uma das fases do processo. A Seção 4.7 apresenta algumas conclusões que foram tomadas a partir dos resultados desse estudo. Finalmente, a Seção 4.8 lista as ameaças à validade do experimento, discutindo como cada uma delas foi anulada ou minimizada.

4.1 Descrição do problema

O sistema desenvolvido, no experimento, pertence ao domínio de sistemas de informação, mais especificamente é um sistema de apoio ao departamento de transportes e usuários de ônibus de transporte corporativo. Esse sistema tem sua visão definida na Tabela 4.1, os papéis ou atores dos dois tipos de usuários descritos na Tabela 4.2 e as Estórias de Usuário que formam o *Backlog* do Produto na Tabela 4.3.

Tabela 4.1: Visão do sistema de itinerários de transporte corporativo.

Para	usuários do serviço de fretados e funcionários do departamento de transportes.
Que	precisam saber qual linha de ônibus passa por determinado ponto, em um determinado horário.
O	*TransCorp* é uma plataforma móvel.
Que	controla os itinerários das linhas de fretados por meio do cadastro e manutenção dos pontos de parada dos ônibus.
Ao contrário	do sistema atual que apenas lista as avenidas e as ruas principais de cada linha e seu horário de partida no sentido de ida para a empresa.
Nosso produto	possibilitará a visualização dos itinerários em ambos os sentidos, bem como o horário previsto de parada em cada ponto, facilitando assim a busca das linhas pelos usuários e a manutenção e planejamento dos itinerários pelo setor de transportes.

Tabela 4.2: Papéis dos usuários.

Papel	**Descrição**
Cliente	Funcionário da empresa que utiliza o serviço de transporte de ônibus fretado.
Gestor	Funcionário do setor de transportes ou da empresa provedora responsável pelos itinerários e horários das linhas de ônibus.

A confiabilidade do sistema está no fato de que as informações guardadas e mostradas devem ser corretas, uma vez que orientar um funcionário para esperar um ônibus em um local errado ou em horário errado pode gerar graves problemas de segurança.

As estórias de 1 a 13 são críticas, ou seja, devem ser totalmente corretas para a validade desse sistema, uma vez que o mau funcionamento ou inconsistência nos dados irá resultar ou em uma localização errada ou em um horário errado de um ponto, causando transtorno para o usuário do fretado. Para efeito do experimento, pelo menos as estórias 1 e 2 devem, obrigatoriamente, ser implementadas para produzir um sistema minimamente funcional.

A próxima Seção descreve o ambiente de trabalho utilizado para a condução do experimento. Essa descrição do ambiente abrange tanto a descrição do cronograma de atividades, das ferramentas utilizadas, quanto o perfil dos executores.

4.2 Definição do experimento

O objetivo deste experimento é de analisar a efetividade do uso do método Scrum+CE (Capítulo 3) para o desenvolvimento de sistemas de informação com requisitos de confiabilidade conforme descrito na Tabela 4.4. O método Scrum+CE, assim como o Scrum, pode ser utilizado para o desenvolvimento de sistemas de software em diversas linguagens ou

Tabela 4.3: *Backlog* do Produto ordenado e estimado em Pontos de Estória.

Ordem	Estórias de Usuário	Pontos
1	Como um cliente, eu quero buscar uma linha que tenha um ponto de parada próximo a um endereço desejado, para que eu possa pegá-la para chegar a meu destino.	20
2	Como um cliente, eu quero buscar o ponto mais próximo de minha casa, do meu turno para que eu ande o mínimo possível e possa me programar para esperar o mínimo possível.	5
3	Como um cliente, eu quero adicionar o meu ponto à minha linha para que outros clientes que morem na mesma região e trabalhem no mesmo turno possam localizá-lo.	8
4	Como um gestor, eu quero ver a lista de todos os pontos de um itinerário, independente do seu estado, para que eu possa analisá-los.	3
5	Como um gestor, eu quero visualizar uma lista de todas as linhas com pontos pendentes para serem validados.	3
6	Como um gestor, eu quero validar um ponto para garantir que os itinerários estejam de acordo com o contratado.	2
7	Como um gestor, eu quero rejeitar um ponto para garantir que os itinerários estejam de acordo com o contratado.	1
8	Como um cliente, eu quero atualizar o horário de passagem de uma linha por um ponto para manter a informação atualizada.	3
9	Como um gestor, eu quero adicionar um ponto de caminho, em um itinerário, para que, quando em um mapa, a rota real feita pelo ônibus seja mostrada.	5
10	Como um gestor, eu quero cadastrar um turno no sistema.	2
11	Como um cliente, eu quero marcar um ponto como sendo o meu ponto.	2
12	Como um gestor, eu quero cadastrar uma linha em um determinado turno preexistente.	5
13	Como um gestor, eu quero atualizar o horário de um ponto qualquer.	2

plataformas, de diferentes tamanhos, e pertencente a outros domínios de negócio. Porém, devido à viabilidade de recursos humanos disponíveis, conhecimento técnico do autor, viabilidade do tempo de execução, o experimento, em questão, utilizou a plataforma Java Web.

A execução do experimento, apresentada entre as Seções 4.4 e 4.6, ocorreu em uma empresa multinacional americana de grande porte, situada em Hortolândia (região metropolitana de Campinas), na qual o autor desta dissertação trabalha desde 2005 como desenvolvedor de software, instrutor e arquiteto de aplicações.

Os profissionais selecionados para implementar o projeto do experimento trabalham há pelo menos 3 anos com desenvolvimento de software em Java, sendo a maioria deles certificado pela Sun/Oracle, no teste de programador Java (*Sun Certified Java Programmer*).

Além disso, todos os programadores passaram por um treinamento básico em Métodos Ágeis, com os conceitos descritos na Seção 2.1, como parte de um programa de disse-

Tabela 4.4: Objetivo do experimento.

Objetivo	Analisar a viabilidade do uso do Scrum+CE no desenvolvimento de sistemas de informação com requisito de confiabilidade sem perder a agilidade.
Foco da qualidade	Qualidade do código gerado por meio da diminuição do número de defeitos; Entrega de menor quantidade de Pontos de Estória.
Contexto	**Objeto**: sistema de software para plataforma Web, usando a linguagem Java. **Sujeitos**: desenvolvedores profissionais com pelo menos 3 anos de experiência em Java (se possível, certificados).
Hipótese nula	Maior quantidade ou mesmo número de defeitos. Maior quantidade ou mesmo número de Pontos de Estória entregues.
Fator principal	Práticas propostas pelo Scrum+CE em relação ao Scrum.
Outros fatores	Experiência dos desenvolvedores na plataforma JavaEE e no processo Scrum.
Variáveis dependentes	(i) Corretude do código entregue avaliada pela execução de um conjunto de testes de validação manuais; (ii) Quantidade de Pontos de Estória entregues ao final do *release*

minação do conhecimento sobre processos ágeis iniciado em 2008, no qual o autor deste trabalho foi um dos instrutores. Como constatado nas respostas da parte 2 do Formulário de Inscrição do Apêndice A, muitos deles atuam ou já atuaram em algum projeto real utilizando o método Scrum (Seção 2.2).

O fato de os participantes serem certificados em Java e já conhecerem, pelo menos, na teoria, o Scrum, ajuda a reforçar a validade interna do experimento, que é a confiança que se tem de que o efeito observado é realmente devido à manipulação feita e não a outros fatores. Ou seja, o resultado obtido será influenciado pelo processo usado no desenvolvimento do sistema e não pela experiência dos participantes na linguagem utilizada.

Preferiu-se a plataforma Java Web por ser ela conhecida dos profissionais selecionados, assim como o conceito da arquitetura em 3 camadas[1] usando o padrão arquitetural *Model-View Controler* (MVC) [29]. Essa escolha foi feita pois o objetivo do experimento é avaliar o processo em si e não a tecnologia usada. Portanto, o mesmo experimento poderia ser replicado, utilizando o mesmo projeto, com tecnologias diversas em um contexto com profissionais diferentes ou até mesmo estudantes de graduação e pós-graduação.

Mais especificamente, o *contexto* do experimento consiste em um *objeto* (P1), ou seja, um sistema de software em Java para a plataforma Web, descrito na Seção 4.1, de uma arquitetura predefinida, seguindo o padrão MVC, e de três *sujeitos*, ou seja, três grupos

[1] do inglês *Three Tiers*

Tabela 4.5: Modelagem do experimento.

Objeto	G1	G2	G3
P1	O1: Scrum	O2: Scrum+CE	O3: Scrum+CE

de profissionais (G1, G2 e G3) com experiência na plataforma Java Web que farão parte da Equipe de Desenvolvimento.

A Tabela 4.5 descreve qual método cada grupo utilizou para implementar o projeto P1. Para efeito de notação, será chamada de O1 a implementação de P1 feita pelo grupo G1 utilizando Scrum; de O2 a implementação de P1 feita pelo grupo G2, utilizando o Scrum+CE; e de O3 a implementação de P1 feita pelo grupo G3, utilizando o Scrum+CE. Com esse formato, foi definido que o grupo G1 seria o grupo de controle e os grupos G2 e G3 seriam os grupos experimentais. Essa separação e escolha dos grupos serão detalhadas e justificas na Seção 4.4.1.2.

A seção a seguir descreve como o experimento foi planejado para sua execução.

4.3 Preparação do experimento

O experimento seguiu um formato chamado **Experimento em Ambiente Sintético**[2] descrito por Zelkowitz e Wallace [35], no qual uma versão reduzida do Scrum, e também do Scrum+CE, foram executadas. Essa redução ocorreu por questões de viabilidade de tempo e recursos para a execução do experimento. Os fatores sugeridos pelo Scrum e os realmente utilizados no experimento, devido às restrições já apresentadas, estão descritos na Tabela 4.6.

Tabela 4.6: Fatores de redução para viabilizar simulação.

Fator	**Sugerido**	**Utilizado**
Tamanho da Equipe de Desenvolvimento	entre 5 e 9	4
Quantidade de *sprints* na *release*	1 ou mais	2
Duração total do *sprint*	entre 2 e 4 semanas	1 semana
Duração da Reunião de Planejamento do *Sprint*	entre 4 e 8 horas	2 horas
Duração do Scrum Diário	15 minutos	3 minutos
Duração do dia de trabalho	8 horas	2 horas
Duração da Reunião de Revisão do *Sprint*	entre 2 e 4 horas	1 hora
Duração da Retrospectiva do *Sprint*	entre 2 e 4 horas	1 hora

O experimento foi executado no formato de um curso simulado de "Scrum na Prática" dentro da empresa em que o autor desta dissertação trabalha. A duração total foi de 40

[2]do inglês *Synthetic Environment Experiments*

horas, distribuídas em 10 manhãs com 4 horas cada, durante duas semanas (com uma semana de intervalo). Cada *sprint* teve duração de 1 semana, o equivalente há 20 horas, resultando em 2 *sprints* no total.

Os participantes não foram informados de que se tratava de um experimento até a Reunião de Revisão do *Sprint* do segundo *sprint*, quando todos os grupos foram reunidos, na mesma sala para a apresentação do sistema desenvolvido por cada equipe e foi-lhes revelado que havia diferenças nos processos utilizados.

Uma semana antes do início do curso, ocorreu uma palestra de 2 horas de introdução teórica ao processo Scrum para nivelar o conhecimento dos participantes, bem como das ferramentas utilizadas e da configuração do ambiente de desenvolvimento. Logo, as 40 horas do treinamento foram exclusivamente utilizadas para a fase Jogo do processo, isto é, para a execução da implementação do projeto em *sprints*.

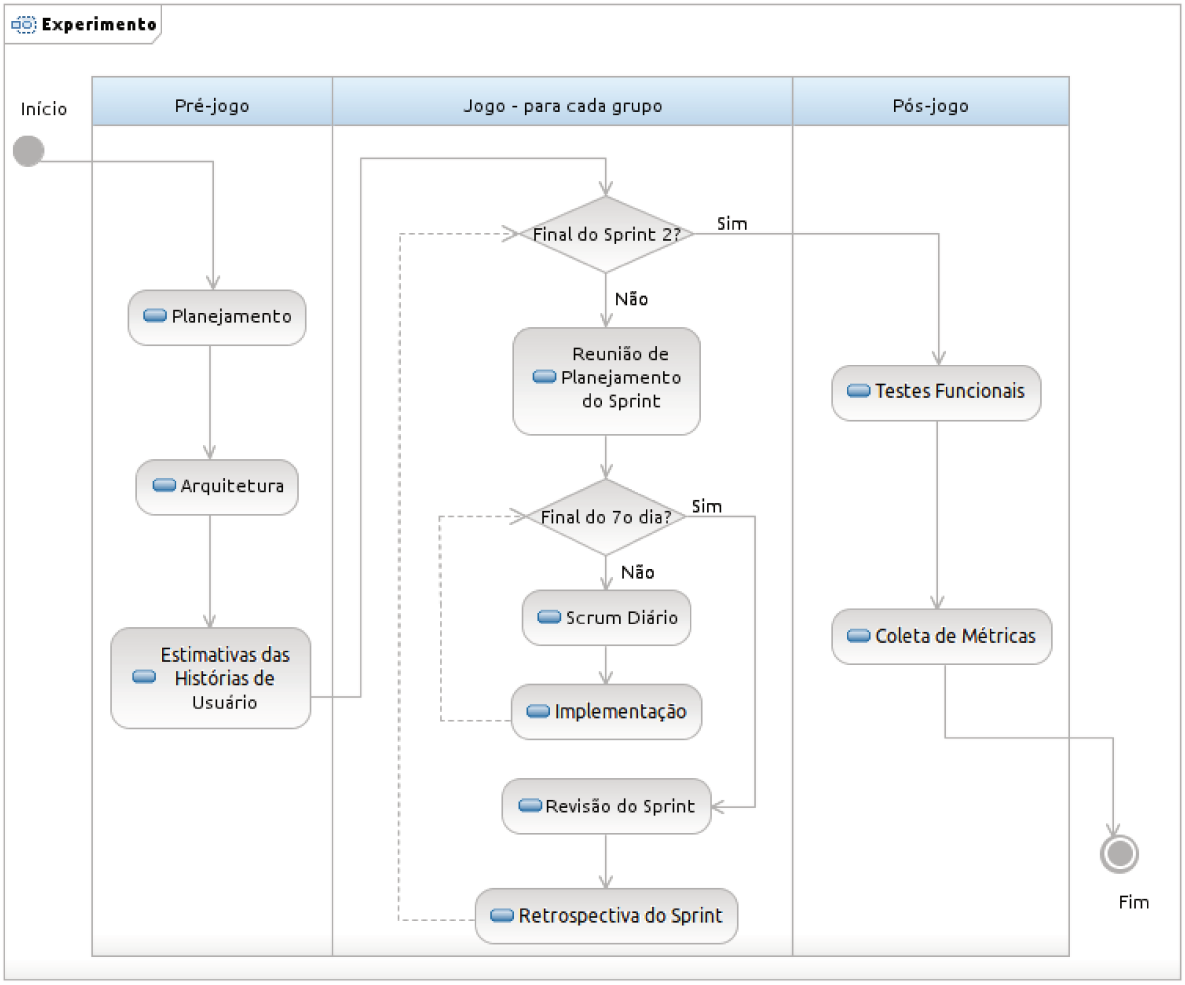


Figura 4.1: Diagrama das atividades do experimento.

A Figura 4.1 mostra o fluxo das atividades efetuadas no experimento. As atividades da fase Pré-jogo foram executadas pelo pesquisador no momento do planejamento deste experimento, o que está descrito na Seção 4.1. A arquitetura foi apresentada para os grupos na Reunião de Planejamento do *Sprint* conforme detalhado na Seção 4.5. As estimativas baseadas em Pontos de Estória foram definidas para cada Estória de Usuário e mostradas na última coluna da Tabela 4.3.

As atividades da fase Jogo foram executadas por cada grupo consistindo na implementação real do projeto proposto em dois *sprints*, sendo que cada *sprint* foi composto por sete dias virtuais (de 2 horas cada), seguindo rigorosamente o processo Scrum. Os detalhes de como cada atividade foi executada estão descritos nas Seções de 4.4 a 4.6.

A fase Pós-jogo, assim como a fase Pré-jogo, foram executadas pelo pesquisador, após o final dos dois *sprints* que de posse do código-fonte criado por cada uma das equipes, executou os testes funcionais e coletou as métricas planejadas e listadas na Seção 4.7.

A seguir, as Seções de 4.4 a 4.6 mostram as principais atividades executadas em cada uma das fases da execução do processo adaptado. Como mostrado no Capítulo 3, o processo é composto, assim como o Scrum, por três fases, sendo na fase Jogo (Seção 4.5) que o experimento em si foi executado.

4.4 Execução do Pré-jogo

A fase Pré-jogo foi executada pelo pesquisador antes do início do experimento, seguindo as atividades mostradas na Figura 4.1 e descritas da Seção 3.2.1.

4.4.1 Planejamento

As atividades de planejamento consistiram na criação dos requisitos do sistema, que foram implementados na fase Jogo, por meio da definição da Visão (Tabela 4.1), e do *Backlog* do Produto (Tabela 4.3) devidamente estimado por Pontos de Estória e priorizado, conforme definidos pela Seção 3.2.1.1.

Além dos artefatos listados na Seção 4.1, as atividades relativas tanto à definição (Seção 4.2) quanto à preparação (Seção 4.3) desse experimento também fizeram parte dessa fase do processo.

4.4.1.1 Atribuição dos papéis

Também dentro dessa fase, foram mapeados os papéis para os participantes do experimento, listados na Tabela 4.7. O pesquisador fez os papéis de Dono do Produto e de Scrum Master nos grupos que utilizaram tanto o Scrum quanto o Scrum+CE, assim como os participantes do experimento formaram a Equipe de Desenvolvimento em ambos

Tabela 4.7: Mapeamento dos papéis aos participantes do experimento.

Papel	**Scrum**	**Scrum+CE**
Dono do Produto	pesquisador	pesquisador
Dono da Arquitetura	-	pesquisador
Scrum Master	pesquisador	pesquisador
Equipe de Desenvolvimento	participantes do experimento	participantes do experimento

os processos. O papel de Dono da Arquitetura, adicionado pelo Scrum+CE, foi realizado pelo pesquisador para os grupos que seguiram tal processo. O fato de o pesquisador atuar em tais papéis se fez necessário para controlar as variáveis, os requisitos, os artefatos e a adesão aos processos pelos grupos, conseguindo assim, manter a validade do experimento.

4.4.1.2 Formação das equipes de desenvolvimento

Os participantes foram convidados a participar do experimento no formato de um treinamento de "Scrum na Prática" e, como parte do convite, tiveram que preencher um formulário de inscrição, apresentado na Seção A.1 do Apêndice A. Ao todo foram recebidas 18 respostas ao formulário de inscrição e, baseado nos critérios técnicos e da disponibilidade para as duas semanas do experimento, foram selecionados 12 participantes. A Tabela A.1 lista todas as respostas a cada pergunta, bem como se o participante foi selecionado ou não.

Uma vez selecionados os 12 participantes, foi criado um índice técnico, descrito na Seção A.3, para dividi-los em 3 grupos, da forma mais equilibrada possível, considerando a soma dos índices de cada participante.

O objetivo de balancear os grupos pelo conhecimento técnico se deu pelo fato de que o objetivo do experimento é validar o método e não a tecnologia, por esse motivo, todos os participantes tinham como pré-requisito o conhecimento da tecnologia a ser usada, no caso, Java para sistemas Web. A Seção A.3 descreve como foi calculado esse índice e a Tabela A.3 mostra os valores correspondentes para as perguntas bem como o índice para cada um dos participantes.

Tendo os índices calculados, a distribuição dos participantes foi feita de forma a equilibrar, ao máximo, a soma dos índices em cada um dos grupos. A Tabela 4.8 mostra os participantes ordenados, de forma decrescente, pelo índice técnico e para qual grupo eles foram alocados.

A Figura 4.2 mostra que a distribuição foi feita no sentido de equilibrar a experiência técnica dos participantes, pois em porcentual, a distribuição ficou entre 32% e 36% do total da soma dos índices. Cada parte do gráfico é representada pelo número do grupo, pelo valor do índice total e pela porcentagem relativa a esse índice.

Tabela 4.8: Distribuição dos participantes em grupos.

Participante	Índice	Grupo
JRJ	78	G2
WFC	77	G3
RV	74	G1
GPB	71	G1
KRC	70	G3
LNZ	68	G2
FB	66	G2
LG	64	G3
MYFM	60	G1
GV	48	G1
JL	22	G3
CSN	13	G2

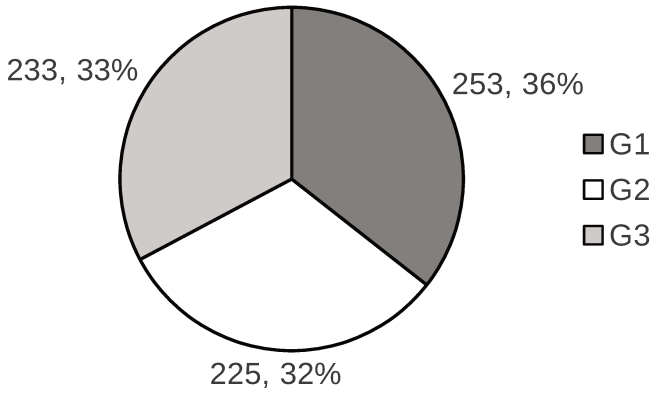


Figura 4.2: Índice total dos grupos.

Com o objetivo de minimizar a ameaça de "Regressão à Média" [33], que é a seleção do grupo experimental por meio da aplicação de um pré-teste e a escolha do grupo com pior média no teste, foi selecionado, como grupo de controle, o grupo com os membros mais experientes na tecnologia e nos métodos ágeis, ou seja, o grupo G1. Dessa forma, conseguiu-se medir efetivamente se o processo sugerido para os grupos experimentais (G2 e G3) realmente gerou alguma melhoria, uma vez que tentar melhorar o grupo que, na teoria já é melhor (G1), não agregaria valor aos resultados do experimento.

Essa escolha do grupo de controle foi crucial para o andamento e para a avaliação do experimento, pois, como o objetivo é analisar a melhoria de um processo, escolher o grupo com maior índice técnico, como experimental, isso não geraria resultados significativos, uma vez que, melhorar algo que já é melhor não possibilitaria a comparação com os outros grupos. Dessa forma, como G1 é o grupo com maior índice técnico, pode-se pressupor

que seus resultados, isto é, o sistema desenvolvido por ele será de melhor qualidade, por isso ele foi escolhido como grupo de controle.

4.4.1.3 Sessão introdutória

No dia 9 de fevereiro de 2012, foi realizada uma sessão introdutória, por teleconferência, com duração de uma hora e com todos os participantes do experimento. Foi explicado como funcionariam as atividades em termos de agenda, horários de início e fim, localização física das salas, tecnologia usada, ambiente de desenvolvimento e ferramentas. Para nivelar o conhecimento dos participantes com relação ao método que seria utilizado no desenvolvimento, foram apresentados, em trinta minutos, os conceitos básicos do Scrum, explicadas e tiradas as dúvidas de cada pilar, papel, evento e artefato, pois neste caso, o processo ditaria a dinâmica do desenvolvimento do sistema.

Também foram apresentadas quais ferramentas cada um precisaria configurar em seu próprio computador para executar a implementação do projeto.

A ferramenta de ambiente integrado de desenvolvimento (IDE) requerida foi o *Eclipse IDE for Java EE Developers* versão 3.3 ou *Indigo Service Release 1* disponível para *download* no site da Fundação Eclipse[3].

O servidor de aplicação escolhido foi o Apache Tomcat[4], versão 7.0.12, cujo *download* e configuração se deram dentro do próprio Eclipse.

O banco de dados utilizado foi o IBM DB2[5], versão 9.5.0.2, e tanto estrutura, esquema, tabelas e dados de testes, quanto usuário e senha para se conectar ao banco foram fornecidos pelo pesquisador no primeiro dia do experimento, não necessitando assim que cada participante tivesse a sua própria instância local do banco e dados de teste.

Para controle de versão e repositório de código-fonte, foi usado o *Concurrent Versions System* (CVS) da empresa, ao qual os participantes já tinham acesso ou foram instruídos em como criá-lo durante esta sessão.

Uma vez o projeto definido e planejado; os requisitos elicitados, priorizados e estimados; a arquitetura e o modelo de dados definidos; os participantes selecionados e separados em grupos; e a infraestrutura de desenvolvimento instalada e configurada tanto nos servidores quanto nos equipamentos dos participantes; pôde-se dar início à fase Jogo, ou seja, à implementação efetiva do sistema, utilizando o processo proposto.

[3]Disponível em: http://www.eclipse.org/webtools/. Acessado em: 8 fev. 2012.
[4]Disponível em: http://tomcat.apache.org. Acessado em: 8 fev. 2012.
[5]Disponível em: http://www.ibm.com/software/data/db2/. Acessado em: 8 fev. 2012.

4.4.1.4 Definição de *Pronto*

Como definido no Scrum (Seção 2.3.1.2) e complementado no Scrum+CE (Seção 3.2.1.2), a Definição de *Pronto* é fundamental para o andamento da fase Jogo uma vez que, estabelece um consenso entre a Equipe de Desenvolvimento e o Dono do Produto sobre o que deve ser implementado durante o *sprint* e apresentado na Reunião de Revisão do *Sprint* para validação. No caso do experimento, uma estória foi considerada pronta se codificada, testada e disponível para teste funcional por meio de uma chamada do método GET do protocolo HTTP[6] via um navegador de internet padrão com passagem de parâmetros e que forneça uma resposta em texto no formato JSON[7], conforme descrito na arquitetura definida em seguida.

Para os grupos utilizando o Scrum+CE, cada estória também deve ter todos os parâmetros de entrada validados quanto à sua presença e ao formato correspondente, além de apresentar uma resposta padronizada para qualquer erro que ocorra durante a requisição ao sistema.

4.4.2 Arquitetura Inicial

A arquitetura do sistema a ser implementado foi apresentada para os três grupos durante a reunião de planejamento do primeiro *Sprint* e seguiu um modelo bem simples, de 4 camadas, sugerido pelo *UML Components* e mostrado na Figura 2.5. As camadas sugeridas são: Interface de usuário, Sessão do usuário, Serviços de sistema, e Serviços de negócio. Neste caso, foram unificadas as camadas de Sessão do usuário e Serviços de sistema e, além disso, as camadas do lado do servidor seguiram o padrão de projeto[8] MVC [29]. Do ponto de vista de DBC, cada uma das camadas foi tratada como um componente, e o objetivo da fase Jogo será a implementação de tais componentes.

A Figura 4.3 ilustra essa arquitetura básica além das fronteiras com o cliente, que acessa a aplicação utilizando um navegador padrão de internet, e um o servidor de banco de dados, que já havia sido previamente configurado. Tal diagrama foi utilizado no Scrum.

Já a Figura 4.4, ilustra a mesma arquitetura explicitando os dois componentes excepcionais, nas camadas de serviço com papéis específicos para apresentação e persistência, os quais devem ser responsáveis por tratar a validação dos dados de entrada e de saída, bem como as consultas, alterações e disponibilidade de acesso do banco de dados, respectivamente. Essa arquitetura foi utilizada no Scrum+CE.

Além das camadas e o relacionamento entre elas e fronteiras, nessa fase também foram tomadas algumas decisões arquiteturais para guiar o desenvolvimento do sistema. Tais

[6] do inglês *Hypertext Transfer Protocol*, ou Protocolo de Transferência de Hipertexto
[7] do inglês *JavaScript Object Notation*. Formato leve para intercâmbio de dados computacionais.
[8] do inglês *Design Pattern*

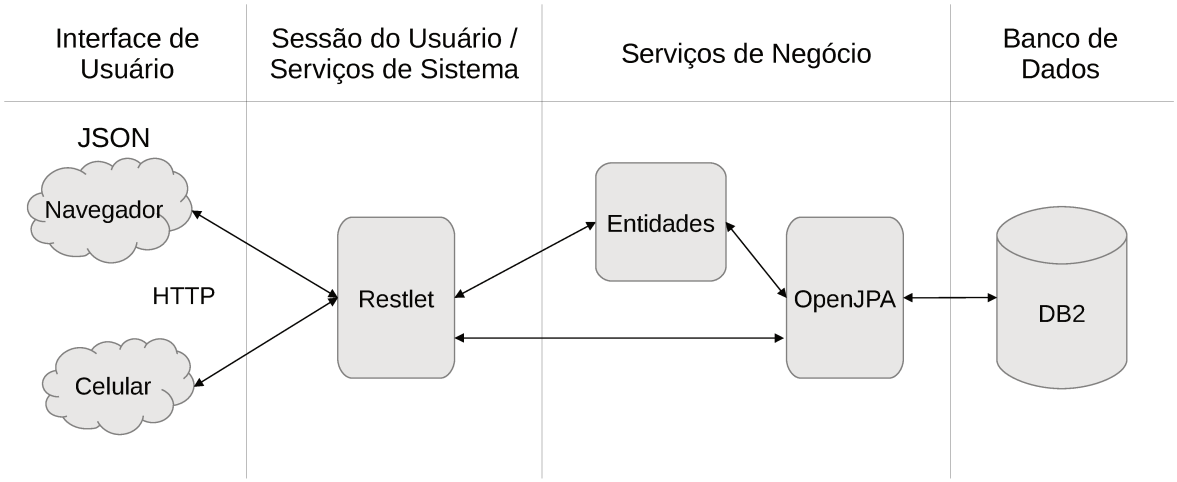


Figura 4.3: Arquitetura base utilizada pelo grupo de controle (G1).

decisões podem ser vistas nas Figuras 4.3 e 4.4, e são elas:

- Toda comunicação feita entre o cliente e a aplicação deve ser efetuada por meio de chamadas HTTP;
- Os parâmetros de entrada devem ser passados como atributos da própria chamada do método GET via URL;
- Todas as saídas do sistema devem estar no formato JSON;
- A camada de apresentação deve implementar o estilo arquitetural *REpresentational State Transfer* (REST), reutilizando o arcabouço Restlet[9];
- A camada de persistência deve ser implementada reutilizando o arcabouço Apache OpenJPA[10];
- As entidades serão objetos Java simples, conhecidos como *Plain Old Java Objects* (POJO)[11], marcados com anotações[12] do OpenJPA.

As entidades devem seguir o modelo apresentado na Figura 4.5 que já foi criado no servidor do banco de dados. Foi fornecida uma cópia do modelo de dados para cada grupo do experimento, todos iniciados com a mesma massa de dados para testes.

[9] implementação REST para Java. Disponível em: http://www.restlet.org/. Acessado em: 12 out. 2012.

[10] implementação da especificação da *Java Persistence API* (JPA). Disponível em: http://openjpa.apache.org. Acessado em: 12 out. 2012.

[11] traduzido como "Os Singelos Clássicos Objetos Java"

[12] conceito de *Annotations* da linguagem Java

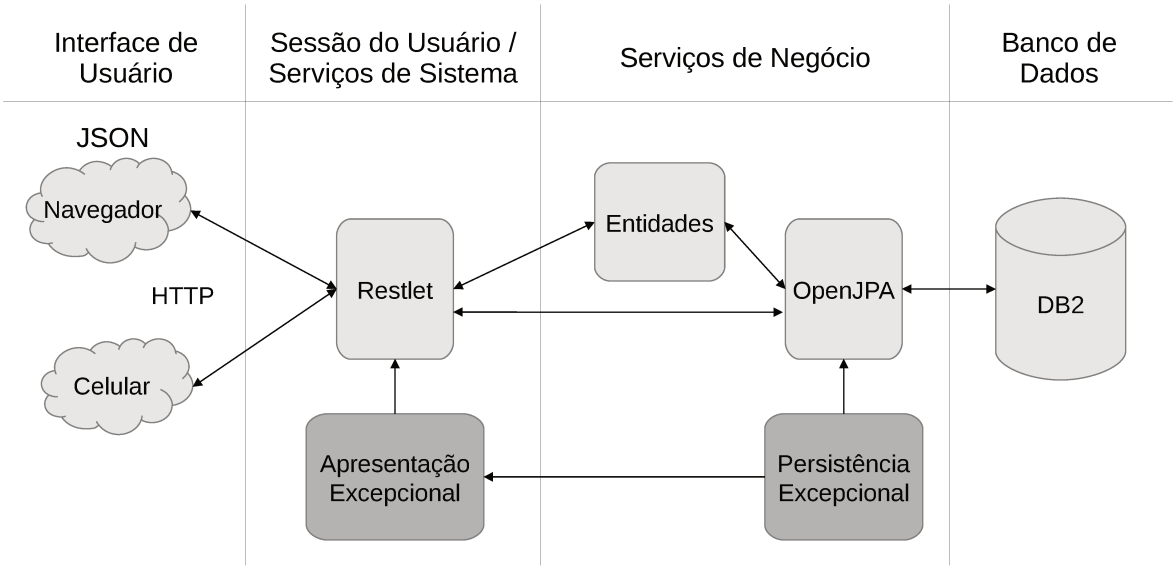


Figura 4.4: Arquitetura, utilizada pelos grupos experimentais (G2 e G3), expondo os componentes excepcionais.

Portanto, o artefato de Arquitetura Inicial criado e disponibilizado foi composto pelo diagrama de arquitetura, a pela descrição das decisões arquiteturais e pelo diagrama entidade-relacional.

Uma vez terminados o planejamento e a Arquitetura Inicial, a fase Jogo foi iniciada, conforme descrito a seguir.

4.5 Execução do Jogo

O experimento em si ocorreu na fase Jogo, que seguiu estritamente o cronograma definido na Seção 4.3. As atividades dos três grupos foram realizadas paralelamente com uma diferença de aproximadamente dez minutos entre um grupo e outro devido ao deslocamento do pesquisador entre as salas.

A seguir, será descrito, resumidamente, o que ocorreu em cada equipe, durante o andamento dos dois *sprints*, sendo que cada *sprint* foi realizado em cinco dias corridos, no período da manhã, e foi composto por uma Reunião de Planejamento do *Sprint*, por sete dias virtuais de implementação de duas horas de duração cada (sempre iniciados com uma reunião de Scrum Diário de três minutos), por uma Reunião de Revisão do *Sprint* e, apenas no *Sprint* 2, por uma Retrospectiva do *Sprint*.

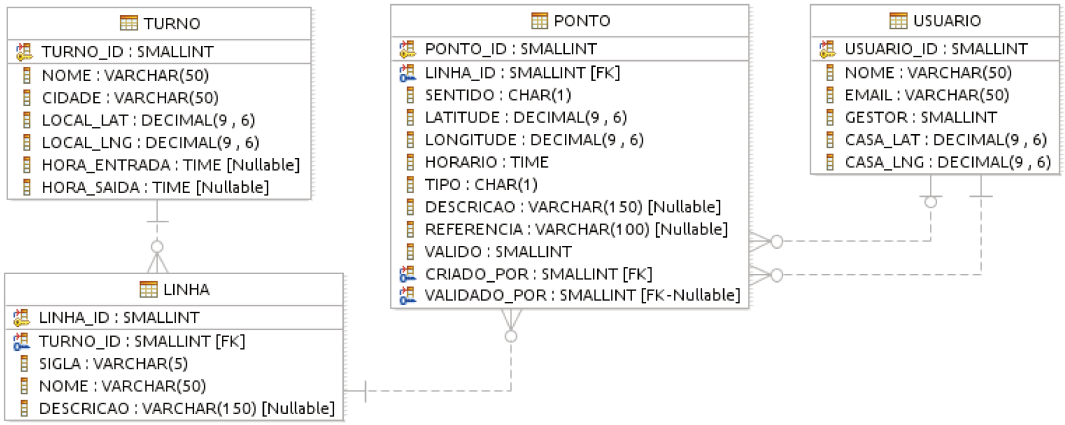


Figura 4.5: Modelo Entidade-Relacional (MER).

4.5.1 *Sprint* 1

No início do experimento, os 12 participantes se reuniram na mesma sala onde foram separados nos seus respectivos grupos, conforme definido na Tabela 4.8, e direcionados para uma sala de reunião exclusiva e totalmente isolada uma das outras, separando assim, o contato entre os grupos durante a implementação do projeto.

A Tabela 4.9 apresenta, em relação ao *Sprint* 1, todas as atividades ocorridas em cada um dos dias do experimento com seus respectivos tempos de duração.

4.5.1.1 Dia 1: Recepção e organização

Cada participante foi orientado a ligar seu computador, terminar de configurar o ambiente de desenvolvimento apresentado na sessão introdutória e aguardar a chegada do pesquisador para o início da Reunião de Planejamento do *Sprint*.

4.5.1.2 Dia 1: Reunião de Planejamento do *Sprint* 1

Foi apresentado, para cada um dos grupos, o *Backlog* do Produto impresso (uma estória por página) e completo, como mostrado na Tabela 4.3 e detalhado no Apêndice B, juntamente com a ordem das estórias, estimativas, testes de aceitação e testes excepcionais para os grupos G2 e G3. Além disso, também foi entregue o diagrama de arquitetura correspondente para os grupos, ou seja, G1 recebeu o diagrama da Figura 4.3 pois utilizou o Scrum, e G2 e G3 receberam o diagrama da Figura 4.4 já que utilizaram o Scrum+CE. O MER (Figura 4.5), igualmente impresso, também foi entregue aos três grupos. Os dois diagramas foram colados, na parede de cada sala, para reforçar a visibilidade desses artefatos.

Tabela 4.9: Atividades de cada dia do *Sprint* 1.

Dia	Atividade	Duração
1	Recepção e organização	2 horas
	Reunião de planejamento	2 horas
2	Scrum diário	3 minutos
	Implementação dia 1	2 horas
	Scrum diário	3 minutos
	Implementação dia 2	2 horas
3	Scrum diário	3 minutos
	Implementação dia 3	2 horas
	Scrum diário	3 minutos
	Implementação dia 4	2 horas
4	Scrum diário	3 minutos
	Implementação dia 5	2 horas
	Scrum diário	3 minutos
	Implementação dia 6	2 horas
5	Scrum diário	3 minutos
	Implementação dia 7	2 horas
	Reunião de Revisão	2 horas

Parte 1: O que será *Pronto* neste *sprint*? Antes de iniciar esta parte, foi relembrada e igualmente coladas nas paredes, a Definição de *Pronto* de cada grupo. Lembrando que estar *Pronto* para os grupos de Scrum+CE contém a adição do trecho "e todas as exceções foram devidamente tratadas", conforme descrito na Seção 3.2.1.2.

Feito isso, foi dado um tempo, de aproximadamente vinte minutos, para os grupos estudarem todas as estórias do *Backlog* do Produto. Passado esse tempo, o pesquisador, no papel de Dono do Produto, passou em cada uma das salas questionando os grupos quais estórias seriam entregues no final do *sprint*. Foram tiradas dúvidas e reforçada a importância da ordem das estórias.

Foi declarado e acordado o Objetivo do *Sprint* para todos os grupos: **implementar e testar a arquitetura, e implementar pelo menos a estória 1**. Com isso, também já foram selecionadas as estórias desse *sprint*, conforme a Tabela 4.10:

- G1: sem a preocupação dos aspectos excepcionais, estava bem otimista de que conseguiria entregar as duas primeiras estórias (1 e 2), uma vez que ambas são relativas à busca (linha na 1 e ponto na 2);

- G2 e G3: mais conservadores e seguiram estritamente o Objetivo do *Sprint*, comprometendo-se apenas em entregar a estória 1.

Tabela 4.10: Estórias selecionadas durante o planejamento do primeiro *sprint*.

Estória	Pontos	G1	G2	G3
1	20	•	•	•
2	5	•		
Total de pontos estimados		25	20	20

Parte 2: Como o trabalho escolhido afetará a arquitetura? Como essa parte apenas existe no Scrum+CE, ela foi realizada somente pelos grupos G2 e G3. Neste primeiro *sprint*, não havia código algum, portanto a arquitetura em si teria que ser implementada. Com isso, o pesquisador, agora no papel de Dono da Arquitetura, salientou para os grupos relativos que os dois componentes excepcionais, mostrados no diagrama de arquitetura, deveriam ser implementados explicitamente.

Parte 3: Como o trabalho escolhido será *Pronto*? Uma vez declarados a Definição de *Pronto* e o Objetivo do *Sprint*, selecionadas as estórias na Parte 1, definidos e estudados a arquitetura e o MER, cada grupo começou a definir as tarefas necessárias para cumprir o Objetivo do *Sprint*. Nessa parte ocorreu uma interação intensa com o pesquisador, no papel de Dono do Produto, para elicitar os detalhes dos requisitos, como por exemplo, as definições dos formatos de entrada e saída das chamadas REST e dos arquivos JSON.

Ao final dessa reunião, todos os grupos estavam com seu *Backlog* do *Sprint* pronto e com as tarefas colocadas no quadro de tarefas, que foi utilizado para gerenciar o trabalho da equipe dentro dos dias de implementação do *sprint*.

4.5.1.3 Dia 2: Implementação dias 1 e 2

Este dia foi iniciado pela primeira reunião de Scrum Diário. Todas as equipes e todos os participantes planejaram iniciar as tarefas de finalização do ambiente de desenvolvimento e criação dos projetos. E assim todos fizeram durante as duas horas restantes desse primeiro dia virtual.

No segundo Scrum Diário, que marcou o início do segundo dia virtual de duas horas, ocorreu um fato interessante nas equipes: os participantes se dividiram naturalmente em duplas, ficando uma responsável por baixar e estudar o *framework Restlet* da camada de apresentação; e a outra, por baixar e estudar o *framework OpenJPA* da camada de persistência.

4.5.1.4 Dia 3: Implementação dias 3 e 4

Nestes dois dias virtuais, as três equipes continuaram a implementação da arquitetura, configurando, estudando e implementando exemplos para testes dos dois *frameworks* relativos às camadas de apresentação e persistência. As duplas que estavam focadas na camada de persistência, já tinham testado o acesso ao banco de dados e, por conta disso já haviam começado a criar as entidades da camada de negócio que mapeiam diretamente as tabelas do banco.

Nesse momento, os dois grupos, que estavam utilizando o Scrum+CE (G2 e G3), criaram as classes de exceções dos dois componentes explicitados na arquitetura.

4.5.1.5 Dia 4: Implementação dias 5 e 6

Com os projetos criados, *frameworks* estudados e configurados, construção dos componentes inicias, e arquitetura testadas, as equipes iniciaram, efetivamente, a implementação da primeira Estória de Usuário. Seguindo a mesma divisão natural em duplas, uma focou na formatação e nos parâmetros de entrada, bem como no arquivo JSON de saída; a outra, no acesso e na consulta ao banco de dados. Um dos participantes de cada grupo também ficou com a tarefa de pesquisar a forma de implementar o cálculo da distância entre dois pontos.

4.5.1.6 Dia 5: Implementação dia 7

Na reunião de Scrum Diário foi comunicado o término da implementação da estória 1 pelos grupos G1 e G3. A equipe G2 ainda estava implementando o código de cálculo de distância entre os pontos.

A equipe G1 declarou que dois membros iriam trabalhar na implementação da estória 2 e os outros iriam fazer testes na estória 1. Uma particularidade deste grupo é que um dos participantes era muito experiente e orientou os outros a escrever testes de unidade para as regras de validação dos parâmetros de entrada.

A equipe G3 teve uma peculiaridade, eles implementaram um cliente Web, utilizando *JavaScript*, para simular as chamadas REST e para formatar o arquivo JSON de resposta, facilitando assim a execução dos testes de validação, e declaram que iriam fazer mais testes, incluindo os excepcionais.

A equipe G2 estava um pouco atrasada em relação às outras duas. Eles conseguiam fazer as chamadas e analisar as respostas da funcionalidade da estória 1, mas ainda não estavam retornando resultados corretos.

4.5.1.7 Dia 5: Reunião de Revisão do *Sprint* 1

Esta reunião foi realizada pelo pesquisador, agora no papel de Dono do Produto, individualmente, com cada um dos grupos em suas respectivas salas. Com isso, o tempo total de duas horas, mostrado na Tabela 4.9, foi dividido em quarenta minutos para cada equipe. Nessa reunião, cada equipe fez uma demonstração da implementação das estórias selecionadas na Reunião de Planejamento do *Sprint*.

Conforme planejado na Tabela 4.10, a equipe G1 apresentou com sucesso as estórias 1 e 2, porém cada uma falhou em um teste de aceitação – de que o ponto encontrado deveria ser de parada – que teve que ser corrigido no *sprint* seguinte. A equipe G2 demonstrou a implementação da Estória 1, porém falhou no teste de validação da distância máxima de 500 metros, retornando sempre o ponto mais próximo. Esse defeito foi corrigido no *Sprint* 2, enquanto a G3 também demonstrou apenas a Estória 1, conforme planejado, e obteve total sucesso em todos os testes.

Tabela 4.11: Estórias demonstradas durante a revisão do primeiro *sprint*.

Estória	**Pontos**	**G1**	**G2**	**G3**
1	20	•	•	•
2	5	•		
Porcentagem de pontos entregues		100%	100%	100%

Mesmo com falhas nas entregas dos grupos G1 e G2, tarefas para corrigi-las foram adicionadas no *sprint* seguinte e as estórias mostradas na Tabela 4.11, foram consideradas concluídas. Com isso, ao final desse *sprint*, o *Backlog* do Produto dos grupos foi atualizado, removendo as Estórias 1 e 2 de G1 e a Estória 1 de G2 e G3.

Devido ao formato de treinamento, e restrições de tempo do experimento, não ocorreu o evento de Retrospectiva do *Sprint* nesse primeiro *sprint*.

4.5.2 *Sprint* 2

Após uma semana de pausa, foi iniciado o segundo *sprint*. Da mesma forma que o primeiro, ele aconteceu durante cinco dias consecutivos, no período da manhã. A Tabela 4.12 apresenta todas as atividades ocorridas, bem como suas durações, em cada um dos dias do experimento.

Um fato isolado, mas que deve ser relatado, foi que um dos participantes do grupo G3, o KRC, abandonou o experimento. Mesmo notificando previamente que não poderia participar de todo o *sprint*, não foi possível providenciar um substituto. Esse fato deixou a equipe com um membro a menos, o que será considerado e apropriadamente discutido na Seção 4.7 de Análise dos resultados.

Tabela 4.12: Atividades de cada dia do *Sprint* 2.

Dia	Atividade	Duração
1	Reunião de planejamento	2 horas
	Scrum diário	3 minutos
	Implementação dia 1	2 horas
2	Scrum diário	3 minutos
	Implementação dia 2	2 horas
	Scrum diário	3 minutos
	Implementação dia 3	2 horas
3	Scrum diário	3 minutos
	Implementação dia 4	2 horas
	Scrum diário	3 minutos
	Implementação dia 5	2 horas
4	Scrum diário	3 minutos
	Implementação dia 6	2 horas
	Scrum diário	3 minutos
	Implementação dia 7	2 horas
5	Reunião de Revisão	3 horas
	Retrospectiva	1 hora

4.5.2.1 Dia 1: Reunião de Planejamento do *Sprint* 2

O início deste segundo *sprint* já foi mais natural, uma vez que os participantes já entendiam quais os passos do processo que seriam repetidos nesta iteração. Por consequência das entregas do primeiro *sprint*, cada grupo agora tem um *Backlog* do Produto diferente para iniciar as atividades de planejamento. A equipe G1 eliminou as estórias 1 e 2, porém cada uma tem um defeito para ser corrigido neste *sprint*. Já G2, eliminou apenas a primeira estória, e também tem um defeito a ser corrigido. Enquanto G3, entregou a estória 1 com sucesso. Com isso, os grupos não ficaram mais sincronizados e seguiram independentes as etapas dessa reunião.

Parte 1: O que será *Pronto* neste *sprint*? Com seus *Backlogs* do Produto atualizados, os três grupos iniciaram a revisão das estórias faltantes. Também relembraram as respectivas Definições de *Pronto* e discutiram as dúvidas com o Dono do Produto, no caso, o pesquisador. Como as estórias restantes continuaram ordenadas por importância para o negócio, os grupos naturalmente foram selecionando-as na sequência e calibrando com a capacidade de Pontos de Estória entregues no *sprint* anterior, de 25 pontos para G1 e 20 pontos para G2 e G3.

Como a arquitetura já estava implementada e testada, os três grupos foram otimistas em relação às estimativas da quantidade de pontos que seriam capazes de implementar.

Tanto as estórias selecionadas quanto as estimativas das equipes estão listadas na Tabela 4.13. Note que G1 foi mais otimista quanto ao número de estórias, uma vez que não tem preocupação explícita com o comportamento excepcional.

Tabela 4.13: Estórias selecionadas durante o planejamento do segundo *sprint*.

Estória	**Pontos**	**G1**	**G2**	**G3**
1	20	*Sprint* 1	*Sprint* 1	*Sprint* 1
2	5	*Sprint* 1	•	•
3	8	•	•	•
4	3	•	•	•
5	3	•	•	•
6	2	•	•	•
7	1	•	•	•
8	3	•	•	•
9	5	•	•	•
10	2	•		
11	2			
12	5	•		
13	2			
Total de pontos estimados		32	30	30

Parte 2: Como o trabalho escolhido afetará a arquitetura? Novamente, como essa parte apenas existe no Scrum+CE, somente foi realizada pelos grupos G2 e G3. Neste segundo *sprint* a arquitetura, bem como os dois componentes excepcionais já haviam sido implementados e testados pelos dois grupos experimentais. O pesquisador no papel de Dono da Arquitetura salientou para os grupos que os dois componentes excepcionais deveriam ser atualizados com o comportamento excepcional derivado das novas estórias. Isto ocorreu devido ao desenvolvimento ser incremental. Como as estórias desse *sprint* não adicionaram nenhum componente novo, apenas novas funcionalidades aos existentes, foi alinhado que o trabalho a ser implementado não afetaria a arquitetura naquele momento.

Parte 3: Como o trabalho escolhido será *Pronto*? A Definição de *Pronto* foi relembrada por todos os grupos e o Objetivo do *Sprint* foi definido como sendo "a implementação das funcionalidades de busca, listagem e aprovação de pontos". Com a colaboração do pesquisador, no papel de Dono do Produto, foram quebradas em tarefas todas as estórias selecionadas na Parte 1, incluindo as tarefas de tratamento de exceções para os grupos G2 e G3. Também foram adicionadas as tarefas para corrigir os defeitos encontrados na Reunião de Revisão do *Sprint* anterior para os grupos G1 e G2.

Ao final dessa reunião, todos os grupos estavam com seu *Backlog* do *Sprint* pronto e com as tarefas colocadas no quadro de tarefas que foi novamente utilizado para gerenciar o trabalho dentro dos dias de implementação.

4.5.2.2 Dia 1: Implementação dia 1

Após uma pausa, o primeiro dia virtual de implementação começou com a reunião de Scrum Diário. As equipes G1 e G2 planejaram atacar primeiro os defeitos do primeiro *sprint*, já a equipe G3 planejou começar a implementação da estória 2. O desenvolvimento e o teste seguiram normalmente de forma empírica entre os membros da equipe, uma vez que eles já estavam mais entrosados entre si e mais familiarizados com o domínio do sistema e com a tecnologia.

4.5.2.3 Dia 2: Implementação dias 2 e 3

Este dia começou com as reuniões de Scrum Diário e G1 declarou que havia resolvido o defeito do *sprint* 1 e que começaria a implementar as tarefas das estórias de seu *Backlog* do *Sprint*. G2 ainda estava trabalhando no defeito de retornar o ponto a menos de 500 metros do ponto desejado, então ficou a cargo de um membro finalizar a correção deste defeito enquanto os outros três iriam começar a implementar as tarefas das outras novas estórias. G3 já havia terminado as tarefas da estória 2 e seus membros iriam continuar trabalhando nas tarefas de seu *Backlog* do *Sprint*.

No terceiro dia virtual, G2 finalmente havia corrigido o defeito da estória 1 e já havia implementado as tarefas da segunda estória. G1 e G3 já estavam seguindo o fluxo empírico de implementação e de testes das tarefas de seus *backlogs*.

4.5.2.4 Dia 3: Implementação dias 4 e 5

O quarto e quinto dias virtuais começaram com as reuniões de Scrum Diário e todos os grupos seguiram normalmente o ciclo de codificação e testes das suas tarefas.

4.5.2.5 Dia 4: Implementação dias 6 e 7

O sexto e sétimo dias também começaram com as reuniões de Scrum Diário e todos os grupos seguiram normalmente o ciclo de codificação e testes das suas tarefas. Porém, como já estavam chegando ao final do *sprint* e no desenvolvimento ágil não tem uma fase explícita de integração e testes, todos os grupos voltaram uma atenção especial para testar as funcionalidades implementadas em decorrência das estórias selecionadas. Os grupos G2 e G3, por utilizarem o Scrum+CE, concentram-se ainda mais nos testes de validação dos parâmetros de entrada das chamadas REST.

O sétimo dia terminou com todos os grupos enviando suas mudanças de código para CVS e o pesquisador declarou o final do *sprint*, salientando que o estado dos sistemas versionados, naquele momento, seria o demonstrado na reunião de revisão do dia seguinte. O pesquisador então sincronizou os códigos-fonte dos três grupos em sua máquina, que seria a utilizada durante a demonstração, garantindo assim que nenhuma mudança posterior iria interferir na medição dos resultado, uma vez que todos os grupos deveriam ter exatamente o mesmo tempo de implementação.

4.5.2.6 Dia 5: Reunião de Revisão do *Sprint* 2

Este último dia do experimento começou com todos sendo direcionados para a mesma sala. O pesquisador relembrou o roteiro da reunião e configurou o ambiente na sua máquina para rodar os três sistemas. Como foi utilizado o mesmo banco de dados, o pesquisador rodou um *script* para limpar e inserir sempre os mesmos dados a cada demonstração, logo, as mudanças do sistema de um grupo não influenciariam os resultados do grupo seguinte.

Cada grupo teve vinte minutos para apresentar a implementação das estórias planejadas na Reunião de Planejamento do *Sprint*. A Tabela 4.14 mostra as estórias entregues por cada grupo marcadas com um círculo; e as planejadas mas não entregues, com um traço. Esta tabela também apresenta a porcentagem de Pontos de Estória entregues em relação aos planejados, mostrando que o grupo G1 deixou de entregar duas estórias, o que somou a entrega de 75% do planejado. Já o G2 também entregou duas estórias a menos, totalizando 83%, e G3 deixou apenas uma para trás, realizando 90% do planejado. Pode-se analisar essa diferença pelo fato de os grupos G1 e G2 ainda terem defeitos do *sprint* herdados e que foi demandado tempo para corrigi-los neste *sprint*, afetando assim um pouco a produtividade.

Como o pesquisador, no papel de Dono do Produto validou cada estória durante a demonstração do sistema usando como base os Testes de Aceitação, e o Scrum+CE formaliza testes mais criteriosos, a validação de G2 e G3 foi mais detalhada. Portanto, foi nesse momento, antes de o primeiro grupo apresentar, que foi revelado pelo pesquisador que se tratava de um experimento controlado e as diferenças entre os grupos foi explicada.

Os detalhes de quais métricas foram utilizadas bem como suas análises serão descritos nas próximas seções.

4.5.2.7 Dia 5: Retrospectiva do *Sprint*

Logo após a Reunião de Revisão do *Sprint*, ainda na mesma sala, com todos os presentes, o pesquisador relembrou o roteiro da retrospectiva que tem como objetivo avaliar e propor melhorias no processo de desenvolvimento. Nesse caso, foi discutida entre os participantes a dificuldade de se trabalhar com tempo fixo e de variar a quantidade de funcionalidades

Tabela 4.14: Estórias demonstradas durante a revisão do segundo *sprint*.

Estória	**Pontos**	**G1**	**G2**	**G3**
1	20	*Sprint* 1	*Sprint* 1	*Sprint* 1
2	5	*Sprint* 1	•	•
3	8	•	•	•
4	3	•	•	•
5	3	•	•	•
6	2	•	-	•
7	1	•	•	•
8	3	-	-	-
9	5	•	•	•
10	2	•		
11	2			
12	5	-		
13	2			
Porcentagem de pontos entregues		75%	83%	90%

a ser entregue. De modo geral, os participantes gostaram do formato e salientaram que a colaboração e a sinergia entres os membros de uma equipe é o fator fundamental para o sucesso da entrega do sistema, uma vez que o processo não dita como as coisas devem ser feitas, mas apenas o que deve ser feito e quando se deve entregar.

Um dos participantes relatou que o ritmo de trabalho fora bem mais intenso do que nos projetos que já havia trabalhado, e atribuiu isso à duração curta do *sprint* e também à dinâmica durante o desenvolvimento que favoreceram a manutenção do foco, no que deve ser demonstrado na Reunião de Revisão do *Sprint*.

No geral, os participantes salientaram o valor do Scrum Diário como forma de sincronizar as atividades desempenhadas por cada membro durante o dia, e também como uma forma de detectar se alguém está com dificuldades. Essa reunião ajudou a correção rápida desses desvios durante os dias de implementação.

O pesquisador também questionou os participantes dos grupos, que usaram o Scrum+CE, se os componentes excepcionais na arquitetura e os testes de aceitação excepcionais, ajudaram a aumentar a confiabilidade durante o desenvolvimento. As respostas, no geral, foram que tais técnicas não penalizaram o desenvolvimento e ajudaram a testar melhor as funcionalidades.

Como essa reunião seguiu um formato informal de discussão, não havendo portanto, possibilidade de ser descrita na íntegra.

4.6 Execução do Pós-jogo

Ao final das duas semanas de experimento e dos dois *sprints*, o pesquisador executou as atividades da fase Pós-jogo conforme mostrado no fluxograma da Figura 4.1. Entre as atividades estão a execução dos testes funcionais nos três projetos entregues sobre as Estórias de Usuário implementadas, e a coleta das métricas utilizadas para a análise dos resultados.

As métricas foram divididas em três categorias: (i) entrega de requisitos; (ii) qualidade dos requisitos entregues; e (iii) qualidade do código.

4.6.1 Métricas de entrega dos requisitos

A entrega das Estórias de Usuário prontas, que foram demonstradas pelos três grupos, nas reuniões de revião, estão consolidadas na Tabela 4.15.

Tabela 4.15: Estórias entregues por grupo do experimento.

Estória	Pontos	G1	G2	G3
1	20	•	•	•
2	5	•	•	•
3	8	•	•	•
4	3	•	•	•
5	3	•	•	•
6	2	•		•
7	1	•	•	•
8	3			
9	5	•	•	•
10	2	•		
11	2			
12	5			
13	2			

No gráfico da Figura 4.6, estão totalizados o número de estórias entregues e seus respectivos Pontos de Estória disponíveis e por grupos.

4.6.2 Métricas de qualidade dos requisitos

Com o código dos três projetos disponíveis para o pesquisador, foi possível realizar testes funcionais sobre as estórias entregues. Já que todas as estórias tinham interfaces de entrada e de saída bem definidas, foi possível testar cada uma delas com uma chamada HTTP passando os parâmetros de entrada na própria URL, e validar o conteúdo de saída recebido no formato JSON.

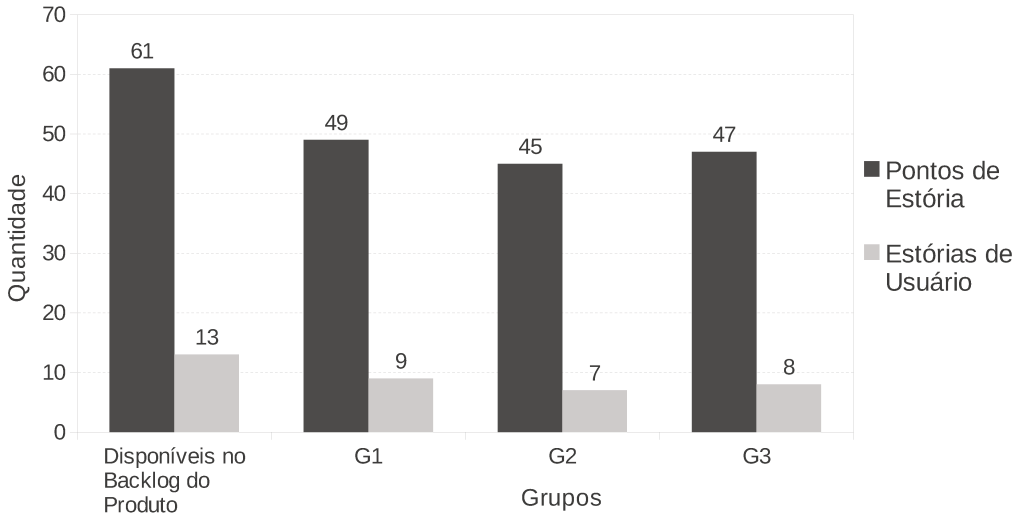


Figura 4.6: Total de Estórias e Pontos entregues.

O pesquisador então, escreveu uma aplicação de suporte automatizada para executar uma chamada para cada teste, tanto de aceitação como excepcional, listados no Apêndice B, sobre as estórias entregues por cada grupo.

Tabela 4.16: Testes realizados e defeitos encontrados por grupo experimento.

Métrica	**G1**	**G2**	**G3**
Testes realizados	189	149	161
Testes falhos	66	44	21
Taxa de insucesso	35%	30%	13%

A Tabela 4.16 consolida os resultados desses testes, mostrando o número de testes executados sobre o código de cada grupo, a quantidade de defeitos encontrados, ou seja, o número de testes que falharam, e a porcentagem de defeitos sobre os testes efetuados. Note que a quantidade de testes realizados variou entre os grupos, pois a quantidade de estórias entregues não foi a mesma, portanto apenas estórias entregues, listadas na Tabela 4.15, foram testadas.

4.6.3 Métricas de qualidade de código

Da mesma forma que os testes funcionais, os três projetos entregues foram submetidos a uma análise estática da qualidade do código implementado. Essa análise foi realizada de forma automática, utilizando-se a ferramenta de código aberto Sonar[13].

[13]Disponível em: http://www.sonarsource.org/. Acessado em: 16 de out. 2012.

A Tabela 4.17 mostra, de forma consolidada, algumas das métricas extraídas pelo Sonar.

Tabela 4.17: Métricas de qualidade de código.

Métrica	**G1**	**G2**	**G3**
Linhas de código	2232	1984	1950
Número de classes	27	47	38
Número de exceções	1	21	2
Blocos *catch* de exceções nativas	53	18	33
Blocos *catch* de exceções criadas	9	12	6
Complexidade Ciclomática (CC) [19] total	446	305	288
CC por classe	15.9	4.5	7.2
CC por método	5.0	2.4	2.5

A seguir será feita a análise dos resultados do experimento, tomando com base as métricas coletadas nesta fase.

4.7 Análise dos resultados

Fazendo uso das métricas coletadas na fase Pós-jogo do experimento (Seção 4.6), pode-se fazer uma análise comparativa dos resultados obtidos pelas três equipes. Lembrando que, como descrito na Tabela 4.5, o grupo G1 utilizou o processo Scrum e os grupos G2 e G3 utilizaram o processo Scrum+CE na implementação do projeto.

Devido à complexidade e à demanda de recursos humanos, o experimento só foi realizado uma vez, portanto não foi possível fazer uma análise estatística dos dados coletados. Dessa forma, foi efetuada uma análise quantitativa comparativa entre o grupo de controle (G1) e cada um dos grupos experimentais (G2 e G3), resultando em dois pares de resultados: G2/G1 e G3/G1.

As métricas do experimento foram divididas em três categorias: (i) entrega de requisitos; (ii) qualidade dos requisitos entregues; e (iii) qualidade do código. Contudo esta análise focará apenas na entrega (i) e qualidade (ii) dos requisitos, pois elas bastam para discutir as duas hipóteses experimentais. As métricas de qualidade do código (iii) serão utilizadas para suportar os argumentos das outras duas categorias.

4.7.1 Requisitos entregues

A Figura 4.7 mostra a comparação entre as estórias e os pontos entregues pelos grupos experimentais G2 e G3 em relação ao grupo de controle G1. Essa mesma lógica será utilizada para a comparação das métricas de qualidade de requisitos na Figura 4.8.

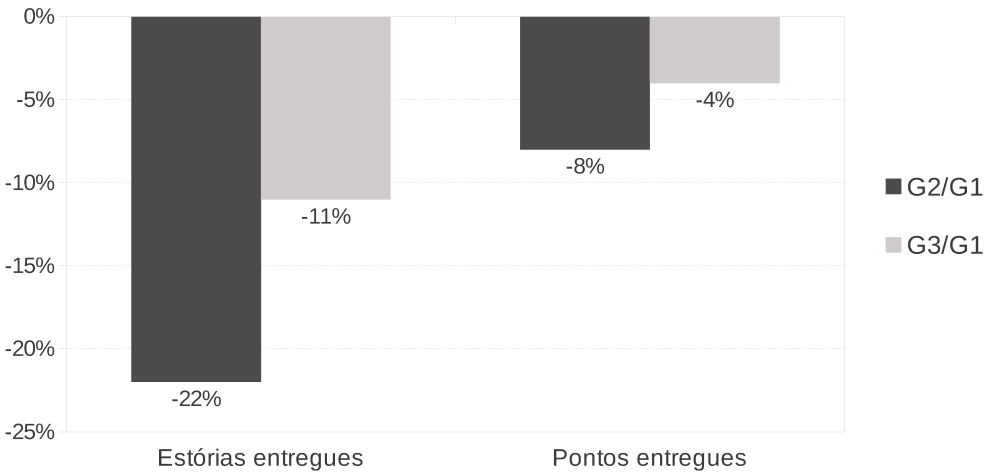


Figura 4.7: Comparação das entregas entre G1 e G2, e entre G1 e G3.

Ainda mostra os dados de G2/G1 que significa que G2 entregou 22% menos estórias que G1, o que equivale a 8% menos Pontos de Estória estregues. Analogamente, o grupo G3/G1 mostra que G3 entregou 11% estórias a menos que G1, ou 4% menos pontos de estória.

Analisando esses números, em relação a uma das hipóteses apresentadas na Tabela 4.4, a qual o uso do Scrum+CE em detrimento ao Scrum geraria uma "Entrega de menor quantidade de Pontos de Estória", percebe-se nitidamente a validade de tal hipótese, uma vez que os dois grupos experimentais entregaram menos estórias (22% e 11%, ou 16.5% em média) e, por consequência, menos Pontos de Estória (8% e 4%, ou 6% em média).

Esses números mostram que utilizar o Scrum+CE, um processo mais rigoroso que o Scrum, impacta a quantidade de entrega de requisitos negativamente, – o que já era esperado – e que vai ser compensado com um código de melhor qualidade, como será apresentado a seguir.

Outra métrica que apoia esta questão é a quantidade de linhas de código de cada sistema entregue, apresentada na Tabela 4.17. Essa métrica mostrou que o código entregue por G1, que implementou mais estórias, ficou maior que os outros dois. Utilizando a mesma forma de comparação anterior, o tamanho de G2 em relação a G1 foi 11% menor; e o de G3, 13% menor, mostrando assim, que realmente o tamanho do sistema em termos de linha de código é diretamente proporcional à quantidade de funcionalidades entregues.

4.7.2 Qualidade do sistema

Com relação às métricas de qualidade dos requisitos entregues, apresentadas na Tabela 4.16, podem ser visualizadas e comparadas as porcentagens no gráfico da Figura 4.8. Como a quantidade de testes realizados está diretamente proporcional à quantidade de

requisitos entregues, uma vez que apenas estórias entregues foram testadas, os grupos experimentais G2 e G3, que entregaram menos funcionalidades, foram submetidos a uma quantidade menor de testes. Esse número foi utilizado para fazer a ponderação da métrica de taxa de insucesso.

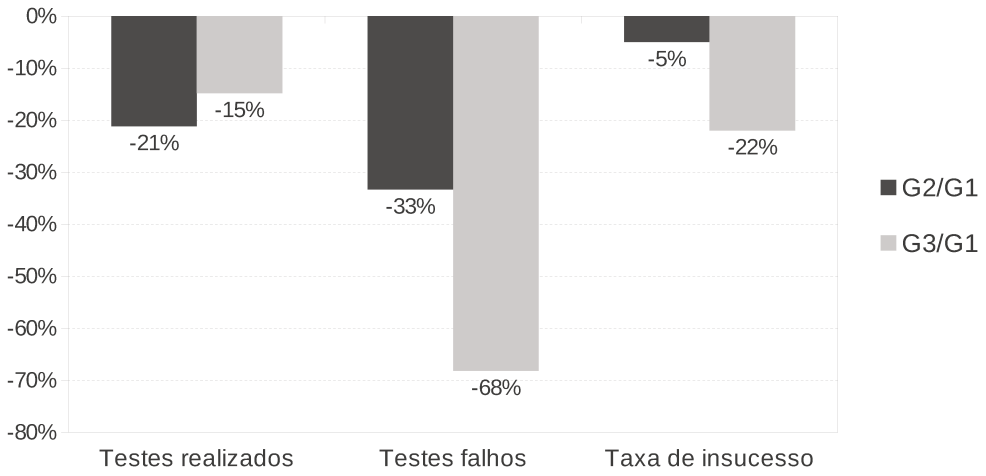


Figura 4.8: Comparação dos testes entre G1 e G2, e entre G1 e G3.

A Figura 4.8 mostra três conjuntos de dados que comparam G2 com G1, e G3 com G1. O primeiro bloco mostra que G2 (149 testes) foi submetido a 21% menos testes que G1 (189 testes), assim como G3 (161 testes) foi testado 15% menos que G1. Essa informação apenas confirma que, efetivamente, apenas foram testadas as estórias entregues, e como G2 e G3 entregaram menos que G1, foram submetidos a uma quantidade menor de testes. O segundo bloco, apenas corrobora o fato de que se são executados menos testes, são encontrados menos defeitos. Dos 149 testes executados sobre G2, foram encontrados 44 defeitos, enquanto dos 189 testes de G1, 66 falharam, ou seja, G2 teve 33% menos testes com insucesso que G1. Da mesma forma, dos 161 testes executados no código de G3 apenas 21 não obtiveram sucesso, ou seja, 68% menos que os 66 de G1.

Porém, a porcentagem dos testes falhos não tem valor por si só, pois compara números brutos, e como G1 foi submetido a mais testes, falhou mais, então, relativamente, G2 e G3 teriam uma quantidade menor de testes falhos, o que, efetivamente, acabou ocorrendo. Portanto a taxa de insucesso é a métrica que mais agrega valor aos resultados do experimento e ajuda a validar a hipótese, também descrita na Tabela 4.4, a qual o uso do Scrum+CE em detrimento ao Scrum resultaria no aumento da "Qualidade do código gerado por meio da diminuição do número de defeitos". A taxa de insucesso de cada grupo foi de 35%, 30% e 13%, respectivamente para G1, G2 e G3, conforme mostrado na Tabela 4.16, ou seja, no caso de G1, 35% dos testes realizados no sistema entregue não retornaram a resposta esperada.

Considerando que um teste falho gera um defeito, essas três porcentagens já nos indicam uma validação da hipótese, pois os dois grupos experimentais tiveram uma taxa de insucesso menor que o grupo de controle, portanto, como ocorreu a diminuição relativa no número de defeitos, o uso do Scrum+CE provocou a criação de códigos de melhor qualidade. Relativamente a G1, G2 teve uma taxa de insucesso 5% menor, e G3 melhorou em 22% essa métrica, ou seja, utilizar o Scrum+CE diminuiu em média 13.5% o número de defeitos gerados.

Uma métrica que chamou a atenção e que corrobora com este argumento é a Complexidade Ciclomática total, apresentada na Tabela 4.17. Essa métrica mostrou que o código entregue por G1 tem uma complexidade de 446, enquanto G2 e G3 apresentaram complexidade de 305 (32% menor que G1) e 288 (35% menor que G1) respectivamente. O fato de os dois códigos desenvolvidos com Scrum+CE serem, na média, 33.5% menos complexos que o criado com Scrum mostra que a preocupação com o comportamento excepcional na arquitetura e uma maior visibilidade sobre os requisitos e cenários excepcionais resultaram, para esse caso, em códigos com um melhor projeto, evidenciado por uma menor Complexidade Ciclomática, o que, por consequência, gerou um menor número de defeitos.

4.7.3 Abandono de um participante

Um fato relatado no início do *Sprint* 2, na Seção 4.5.2, relativo ao abandono de um participante da equipe G3, causou um desequilíbrio entre o número de participantes dos grupos, o que poderia ter causado efeitos colaterais nas métricas. Porém, no decorrer das duas semanas de experimento, houve faltas esporádicas de participantes em todos os grupos.

Para avaliar o impacto dessas ausências, foi feita uma contagem do número de horas-homem (HH) de cada grupo no decorrer de todo o experimento. Foram levadas em consideração apenas as horas de implementação, ou seja, cada *sprint* foi composto por sete dias virtuais de duas horas, ou 14 horas por participante, totalizando 56 horas-homem por *sprint*. Portanto, se nenhum participante faltasse durante o experimento, cada grupo teria uma capacidade produtiva de 112 horas-homem.

A Tabela 4.18 mostra a quantidade real de horas-homem por grupo, a porcentagem em relação ao número total de 112 horas, a relação entre a quantidade de Pontos de Estória (PE) entregues e o número de horas-homem efetivamente realizado, indicando assim a produtividade da equipe.

Analisando os números da Tabela 4.18, nota-se que, curiosamente, as frequências de G1 e G2 foram exatamente a mesma, de 88%, e que realmente o abandono em G3 causou um impacto negativo, derrubando a frequência para 82%. Se se considerar apenas a frequência, pode-se concluir que o abandono influenciaria as métricas de G3 negativa-

Tabela 4.18: Quantidade de horas-homem e produtividade das equipes.

Métrica	**G1**	**G2**	**G3**
Horas-homem (HH)	98	98	92
Frequência (sobre 112)	88%	88%	82%
Pontos de Estória por Horas-homem (PE/HH)	0.50	0.46	0.51

mente, principalmente as de entrega, porém se comparados os valores da produtividade de PE/HH, percebe-se que o impacto não foi relevante, pois o G3 implementou mais pontos por hora que as outras duas equipes.

Portanto, conclui-se que o abandono do participante de G3 se comparado com as faltas esporádicas em G1 e G2, quando a quantidade de funcionalidades entregues por capacidade produtiva foi muito próxima entre as três equipes, não foi relevante a ponto de impactar as métricas coletadas, nem de causar a invalidade do experimento.

4.8 Ameaças à validade do experimento

Esta seção contém a lista de possíveis ameaças [33] e quais ações foram tomadas para minimizá-las quando identificadas alguma interferência no experimento.

4.8.1 Instrumentação

A instrumentação se baseia na ideia de que talvez a diferença entre os grupos seja devido a um erro na medição. Por exemplo, os testes são suficientemente diferentes, ou as observações são feitas por pessoas diferentes?

Como todos os grupos implementaram o mesmo projeto, tendo como subsídio os mesmos recursos: material teórico, *Integrated Development Environment* (IDE), banco de dados, dados de teste, duração total e requisitos (Seção 4.1), e o sistema gerado foi testado pelo pesquisador utilizando o mesmo conjunto de testes, esse experimento não foi exposto a problemas de instrumentação.

4.8.2 Testagem

Testagem é uma ameaça que ocorre em desenhos experimentais com dois projetos, e após a implementação do primeiro, mostram-se os erros encontrados aos participantes, fazendo com que eles fiquem cientes deles e não os cometam novamente no segundo projeto.

As três implementações ocorreram em paralelo e não houve troca entre os membros dos grupos. Além disso, os testes foram realizados apenas no Pós-jogo e sem a participação dos programadores, portanto não ocorreu interferência por testagem neste experimento.

4.8.3 Maturação

Ela ocorre quando os sujeitos dos experimentos tornam-se mais capazes com o tempo, independentemente da intervenção.

O experimento teve duração fixa de 2 semanas, para todos os grupos, e cada um implementou apenas um projeto, ou seja, mesmo que tenha havido maturação tanto do conhecimento dos requisitos, quanto da arquitetura, tecnologia e processo, todos os grupos sofreram os efeitos no mesmo grau.

4.8.4 História

Uma outra ameaça é que pode ocorrer alguma iniciativa externa ao experimento para diminuir o número de erros entre os grupos e que tal iniciativa melhore os resultados.

Neste caso, o projeto é fictício mas baseado em requisitos reais, portanto tal interferência não se aplica. Além do fato de os participantes não saberem que a implementação do sistema não era um experimento, e sim um treinamento prático, até a entrega do código final.

4.8.5 Mortalidade seletiva

Ela acontece quando os participantes abandonam o experimento, desbalanceando os grupos e interferindo nos resultados.

Como o experimento todo teve uma duração relativamente curta (2 semanas), essa ameaça não gerou impacto nos resultados. Isso foi demonstrado anteriormente na Seção 4.7.3, que relata o abandono de um dos participantes na segunda metade de experimento e como isso não impactou a carga total de trabalho dos grupos.

4.8.6 Contaminação

Ela ocorre quando os membros do grupo experimental ensinam aos membros do grupo de controle algumas das técnicas às quais eles estão sendo submetidos.

Essa ameaça pode ter acontecido, minimamente, se os membros dos grupos experimentais (que usaram Scrum+CE) ensinaram, ou citaram, para o grupo de controle (que usou Scrum) as técnicas de explicitação e tratamento do comportamento excepcional. Isso foi minimizado não deixando com que os participantes soubessem que se tratava de um experimento, portanto eles não tinham conhecimento de que o processo utilizado pelos demais grupos tinha diferença.

4.8.7 Comportamento competitivo

Os membros do grupo de controle podem se sentir preteridos frente aos dos grupos experimentais, e isso poderia motivá-los a competir com os grupos experimentais para demonstrar que, mesmo sem as melhorias propostas, eles ainda assim continuariam melhores.

O fato de os três grupos terem desenvolvido o experimento ao mesmo tempo, pode ter gerado uma competição inconsciente, mesmo que o pesquisador tenha-lhes enfatizado que não era uma competição. O formato de treinamento também ameniza essa interferência.

4.8.8 Comportamento compensatório

Ela acontece quando alguma autoridade externa sente que o grupo de controle foi preterido e crie medidas compensatórias para o grupo. Isso não ocorreu, por se tratar de um projeto fictício, executado como laboratório de um treinamento e apenas o pesquisador tinha contato com os participantes.

4.8.9 Regressão à média

A regressão à média se baseia em usar um pré-teste para selecionar o grupo experimental, utilizando os elementos com menor média. Fazendo isso, qualquer interferência feita irá melhorar a média desse grupo, pois eles já eram os piores.

A separação das equipes foi feita sem um teste prévio, mas sim, com uma análise criteriosa da experiência de cada participante. Os efeitos foram minimizados, conforme detalhado na Seção 4.4.1.2.

4.8.10 Efeito de expectativa do sujeito, efeito placebo ou Hawthorne

O fato de os sujeitos saberem que estão sendo estudados ou observados pode melhorar sua produtividade ou a qualidade do software gerado.

No caso desta pesquisa, como foram executadas as implementações em paralelo, esse fator influenciou todos os grupos na mesma intensidade, uma vez que foi omitido dos sujeitos quais grupos eram de controle ou experimentais, além do próprio fato de ser um experimento.

4.8.11 Efeito de expectativa do experimentador

Ela acontece quando o pesquisador interage intensamente com o sujeito, e as crenças do experimentador causam um efeito no sujeito.

Essa ameaça pode ter ocorrido, visto que, inevitavelmente aconteceu interação. A anulação dessa questão seria possível somente se outra pessoa, que não o pesquisador, executasse o experimento. Mas isso não foi possível. Para minimizar esse efeito, o experimento foi executado no formato de um treinamento, sendo a implementação do projeto o seu laboratório prático.

4.8.12 Avaliação subjetiva do pesquisador

As métricas coletadas foram objetivas, porém os testes foram executados pelo pesquisador o que poderia levá-lo a ignorar defeitos ou a atribuir menor severidade à implementação dos grupos experimentais com o objetivo de obter melhores resultados.

Isso foi minimizado fazendo uso de testes automatizados predefinidos e executando exatamente o mesmo conjunto de testes em todas as implementações independentemente do método utilizado no desenvolvimento. Dessa forma, pôde-se garantir que o mesmo rigor foi aplicado a todos os grupos, reduzindo-se assim o grau de subjetividade.

4.9 Resumo

Esse capítulo apresentou o procedimento de avaliação do método Scrum+CE, que consistiu na execução de um experimento controlado com a finalidade de verificar a eficácia do método no desenvolvimento de sistemas com requisitos críticos de confiança no funcionamento.

A escolha de um sistema com esses requisitos visou destacar o papel da arquitetura para o tratamento efetivo das situações excepcionais. A execução do experimento consistiu na utilização do processo apresentado no Capítulo 3, para a análise e a implementação de um sistema de informação.

De uma maneira geral, a expectativa em relação ao método Scrum+CE foi atendida, validando-se, assim, as duas hipóteses, pois foram entregues 6% Pontos de Estória a menos que no Scrum, e ainda foi gerada uma diminuição de 13.5% no número de defeitos encontrados.

Porém, devido à ausência de novas oportunidades para a repetição do experimento com mais equipes, equipes com mais membros ou tecnologias diferentes não houve massa de dados grande o suficiente para uma análise estatística mais sofisticada.

Por fim, foram listadas as ameaças à validade do experimento, discutindo como cada uma foi anulada ou minimizada.

Capítulo 5

Conclusões e trabalhos futuros

Este trabalho apresentou o Scrum+CE, um método ágil, baseado no Scrum que adiciona técnicas do MDCE+ para auxiliar na modelagem e documentação dos requisitos relativos ao comportamento excepcional de sistemas confiáveis baseados em componentes. O método Scrum+CE segue exatamente a mesma estrutura de fases, papéis, eventos e artefatos do Scrum e melhora a confiança no funcionamento do sistema desenvolvido, pois oferece uma maneira sistemática de modelar e documentar as exceções. Essa maneira estruturada de explicitar as exceções, no contexto de ocorrência de falhas, é particularmente relevante para sistemas que possuem requisitos de confiabilidade.

O Scrum+CE adiciona práticas de documentação do comportamento excepcional na forma de Estórias Excepcionais e Testes de Aceitação mais criteriosos nas Estórias de Usuário. Essa técnica, por se tratar de uma forma sistemática de estruturar e organizar os requisitos excepcionais, melhora a confiabilidade, pois auxilia na prevenção de falhas, e evita a inserção de erros.

A adição de tais práticas afetou diretamente as fases Pré-jogo e Jogo, já que é durante essas fases que ocorrem a elicitação de requisitos, análise, projeto, implementação e testes.

Em relação ao Scrum, ainda foram adicionados o papel de Dono da Arquitetura (descrito na Seção 3.5.1) e um artefato obrigatório de Arquitetura Inicial (detalhado na 3.4.1). Com isso, a ênfase na arquitetura de software possibilita uma melhor análise dos fluxos de exceções que circulam entre os componentes arquiteturais do sistema. Essa análise mais criteriosa permite a construção de tratadores mais eficientes, além de antecipar a correção de possíveis falhas de especificação. Enfim, se consegue melhorar a confiabilidade nas dimensões de tolerância a falhas e avaliação de falhas, além de dar um aspecto de centralização da arquitetura ao método.

Para avaliar e verificar a eficácia do Scrum+CE no desenvolvimento de sistemas com requisitos críticos foi realizado um experimento controlado. A execução do experimento consistiu na utilização dos processos Scrum, apresentado nas Seções 2.2 a 2.7, e Scrum+CE,

definido no Capítulo 3, para a análise e a implementação de um mesmo sistema de informação por três equipes distintas paralelamente. Para usar o Scrum+CE, em detrimento ao Scrum, foram testadas duas hipóteses: (i) entrega de uma menor quantidade de Pontos de Estória, uma vez que o Scrum+CE é mais criterioso no detalhamento dos requisitos; e (ii) software entregue com melhor qualidade, ou seja, com um menor número de defeitos, já que os cenários excepcionais, sendo explicitados, seriam igualmente testados.

Como resultado do experimento, pôde-se demonstrar que a expectativa em relação ao método Scrum+CE foi atendida validando as duas hipóteses pois foram **entregues 6% menos Pontos de Estória** que na utilização do Scrum, e ainda foi constatada uma **diminuição de 13.5% no número de defeitos** encontrado.

Portanto, usando o Scrum+CE, foram entregues menos Pontos de Estória porém com uma melhor qualidade (menos defeitos) em relação ao desenvolvimento que utilizou o Scrum.

Feito isso, foi comprovado que é possível desenvolver software confiável com um método ágil, no caso o Scrum+CE, e que, ao adicionar a documentação e análise do comportamento excepcional, isso não afeta a agilidade do processo, já que o Scrum+CE mantém as características do Scrum.

A seguir, a Seção 5.1 apresenta as principais contribuições deste trabalho. Finalmente, a Seção 5.2 aponta alguns direcionamentos futuros para pesquisa.

5.1 Contribuições

As contribuições deste trabalho se situam, principalmente, no campo da engenharia de software, em particular nas áreas de processos de desenvolvimento, métodos ágeis, tratamento de exceções, desenvolvimento baseado em componentes, arquitetura de software e desenvolvimento de sistemas tolerantes a falhas.

A seguir, são listadas as principais contribuições deste trabalho:

1. **O método Scrum+CE.** A principal contribuição desse trabalho foi a definição do método Scrum+CE, apresentado no Capítulo 3. Durante a condução da pesquisa, foram identificadas algumas outras contribuições secundárias:

   (a) **Estudo dos principais mecanismos e técnicas para tratamento de exceções.** Durante a definição das atividades do método Scrum+CE, foi necessário estudar alguns dos principais mecanismos de tratamento de exceções descritos no MDCE+ [8].

   (b) **Requisitos no formato de Estórias Excepcionais.** Um das técnicas adaptadas do MDCE+ para explicitar o comportamento excepcional foi documentá-lo na forma de Estórias Excepcionais. Esse tipo de estória segue exatamente o

mesmo formato das Estórias de Usuário, com uma breve descrição e Testes de Aceitação, porém são exclusivas para os requisitos que fogem ao fluxo normal de uma funcionalidade.

(c) **Adoção de uma arquitetura voltada para sistemas confiáveis.** Com a adoção da Arquitetura Inicial, apresentada na Seção 3.4.1, foi possível explicitar a estruturação dos elementos básicos necessários para que o tratamento de exceções possa ser feito de forma eficiente.

2. **Realização de um Experimento Controlado.** Com o objetivo de validar a aplicabilidade e a efetividade do método Scrum+CE, descrito no Capítulo 4, foi realizado um experimento controlado no qual um mesmo sistema de informação com requisitos de confiabilidade foi implementado simultaneamente por três equipes, de quatro membros cada. Uma das equipes – o grupo de controle – utilizou o Scrum, no desenvolvimento do projeto, enquanto as outras duas – grupos experimentais – utilizaram o Scrum+CE. A duração total do experimento foi de duas semanas, sendo dividas em dois *sprints*. Devido à disponibilidade de recursos humanos, o tamanho e o tempo do experimento (12 participantes por 2 semanas) não foram muito elevados, porém suficientes para a coleta e análise significativa de dados. Por demandar muitas pessoas, por um período considerável de tempo, e por gerar duplicidade de software, visto que todas as equipes implementaram os mesmos requisitos, ficaria muito difícil escalar tal experimento a ponto de prover uma quantidade maior de dados o que permitiria uma análise estatística deles.

5.2 Trabalhos futuros

Com a conclusão desse trabalho, foram identificados alguns direcionamentos para pesquisas futuras. Essas sugestões visam, principalmente, ao aperfeiçoamento do método Scrum+CE, no sentido de suprir algumas de suas limitações e de aplicá-lo em diferentes contextos com outras tecnologias. Algumas sugestões de melhoria seriam:

1. **Repetir o experimento mais vezes.** Repetir o mesmo experimento, com maior número de equipes, mantendo o número de membros por equipe, o mesmo projeto e a mesma duração com o objetivo de obter mais dados e tornar possível uma análise estatística mais refinada.

2. **Executar o experimento com outro projeto.** Repetir o mesmo experimento, mantendo o mesmo número de equipes, a mesma quantidade de membros por equipe e, a mesma a duração. Porém, com outro projeto na mesma tecnologia, ou em outra

linguagem, que suporte o tratamento de exceções de forma nativa, com o objetivo de validar o Scrum+CE independentemente da tecnologia utilizada.

3. **Executar o experimento por um período mais longo.** Repetir o mesmo experimento, com o mesmo número de equipes, a e mesma quantidade de membros por equipe e o mesmo projeto (com mais Estórias de Usuário) por mais de dois *sprints*, com o objetivo de validar o impacto do Scrum+CE na implementação de uma maior quantidade de requisitos.

4. **Realizar o Estudo de Caso em um projeto real.** Efetuar uma análise da utilização do Scrum+CE em um projeto real na forma de um estudo de caso qualitativo.

5. **Adicionar *Test Driven Development* (TDD).** Tornar obrigatório o uso da técnica de TDD, durante a fase Jogo do Scrum+CE, para melhorar o projeto e, ao mesmo tempo, aumentar a cobertura de testes de unidade, gerando um código de melhor qualidade e mantendo um conjunto de testes automatizados.

5.3 Publicações

Durante a condução deste trabalho, foi produzida uma publicação em um congresso nacional.

- **Método Ágil aplicado ao Desenvolvimento de Software Confiável baseado em Componentes** WBMA 2011 - II Workshop Brasileiro de Métodos Ágeis, 2011 [7] (Alan Braz, Cecília M. F. Rubira e Marco Vieira): Esse trabalho apresentou a primeira versão do método Scrum+CE. Apesar de, na sequência do trabalho, ele ter evoluído, a maioria dos conceitos apresentados, nesse artigo, permanece constante até a escrita e a conclusão desta dissertação.

Referências Bibliográficas

[1] Anderson, T., and Lee, P. A. *Fault tolerance, principles and practice*, 2 ed. Springer-Verlag, 1990.

[2] Bass, L., Clements, P., and Kazman, R. *Software architecture in practice*. Addison-Wesley, Boston ; Munich [u.a.], 2005.

[3] Beck, K. *Extreme Programming Explained: Embrace Change*, US ed. Addison-Wesley Professional, Oct. 1999.

[4] Beck, K., and Cunningham, W. A Laboratory For Teaching Object-Oriented Thinking. In *OOPSLA'89 Conference Proceedings* (Oct. 1989), vol. 24.

[5] Beck, K., et al. Manifesto para Desenvolvimento Ágil de Software. http://agilemanifesto.org/iso/ptbr/, 2001. Acessado em 18 de setembro de 2011.

[6] Boehm, B. W. A spiral model of software development and enhancement. *Computer 21*, 5 (May 1988), 61–72.

[7] Braz, A., Rubira, C. M. F., and Vieira, M. Método Ágil aplicado ao Desenvolvimento de Software Confiável baseado em Componentes. In *Anais do II Workshop Brasileiro de Métodos Ágeis (WBMA)* (Fortaleza, CE, Brasil, June 2011), pp. 15–23.

[8] Brito, P. H. S. Um Método para Modelagem de Exceções em Desenvolvimento Baseado em Componentes. Master's thesis, IC, Unicamp, Oct. 2005.

[9] Cheesman, J., and Daniels, J. *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001.

[10] Cohn, M. *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004.

[11] Cohn, M. *Agile Estimating and Planning*. Prentice Hall, 2005.

[12] D'Souza, D. F., and Wills, A. C. *Objects, Components, and Frameworks with UML: The Catalysis Approach*, 2nd ed. Addison-Wesley, 1999.

[13] Far, B. Software Reliability Engineering for Agile Software Development. *20th IEEE Canadian Conference on Electrical and Computer Engineering CCECE* (2007), 694–697.

[14] Ferreira, G. R. M. Tratamento de exceções no desenvolvimento de sistemas confiáveis baseados em componentes. Master's thesis, IC, Unicamp, Dec. 2001.

[15] Gold, R., Hammell, T., and Snyder, T. *Test Driven Development: A J2EE Example*. Apress, 2005.

[16] Highsmith, J. *Agile software development ecosystems*. Addison-Wesley, Longman Publishing Co., Inc. Boston, MA, USA, 2002.

[17] Kruchten, P. *The Rational Unified Process: An Introduction*, 3 ed. Addison-Wesley, Boston, MA, 2003.

[18] Laprie, J.-C. Dependable computing and fault tolerance: concepts and terminology. In *Proceedings of 15th International Symposium on Fault-Tolerant Computing (FTSC-15)* (1985), pp. 2–11.

[19] McCabe, T. J. A Complexity Measure. *IEEE Trans. Software Eng. 2*, 4 (1976), 308–320.

[20] McIlroy, M. D. Mass-Produced Software Components. *Software Engineering: Report of a conference sponsored by the NATO Science Committe* (Oct. 1968), 138–155.

[21] Nord, R. L., and Tomayko, J. E. Software Architecture-Centric Methods and Agile Development. *IEEE Software 23*, 2 (Mar. 2006), 47–53.

[22] Poppendieck, M., and Poppendieck, T. *Lean Software Development: An Agile Toolkit*. Addison-Wesley, 2003.

[23] Radinger, W., and Goeschka, K. M. Agile software development for component based software engineering. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (Oct. 2003).

[24] Royce, W. W. Managing the development of large software systems: concepts and techniques. In *Proc. IEEE WESTCON, Los Angeles* (Aug. 1970), IEEE Press, pp. 1–9. Reprinted in *Proceedings* of the Ninth International Conference on Software Engineering, March 1987, pp. 328–338.

[25] Rumbaugh, J., Jacobson, I., and Booch, G. *The Unified Modeling Language reference manual*. Addison-Wesley, 1999.

[26] Schwaber, K. SCRUM Development Process. In *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)* (1995), pp. 117–134.

[27] Schwaber, K., and Beedle, M. *Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River, New Jersey, 2002.

[28] Sommerville, I. *Engenharia de Software*, 8 ed. Addison Wesley, São Paulo, SP, 2007.

[29] Stelting, S., and Maassen, O. *Applied Java Patterns*. Prentice Hall Professional, 2002.

[30] Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[31] Takeuchi, H., and Nonaka, I. The New New Product Development Game. *Harvard Business Review* (Jan. 1986).

[32] VersionOne. State of Agile Development Survey. http://www.versionone.com/state-of-agile-survey-results/. Acessado em 10 de abril de 2013.

[33] Wainer, J. *Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação*. Sociedade Brasileira de Computação e Editora PUC-Rio, 2007.

[34] You-Sheng, Z., and Yu-Yun, H. Architecture-based software process model. *Software Engineering Notes 28*, 2 (Mar. 2003).

[35] Zelkowitz, M. V., and Wallace, D. Experimental validation in software engineering. *Information and Software Technology 39* (1997), 735–743.

Apêndice A

Formulário de inscrição no experimento

A.1 Formulário de inscrição

1. **Conhecimento em Java**

   1.1. Qual é o seu conhecimento teórico da linguagem Java?
   Sendo 1 se apenas teórico, e 5 se já ministrou cursos ou é certificado.
   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

   1.2. Qual é o seu conhecimento teórico da plataforma JavaEE?
   Sendo 1 se apenas teórico, e 5 se já ministrou cursos ou é certificado.
   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

   1.3. Qual é a sua experiência em projeto reais na plataforma JavaEE?
   Sendo 1 se nunca trabalhou em projetos reais, 3 se já trabalhou em apenas 1 projeto durante aproximadamente 1 ano, e 5 se já trabalhou em mais de 2 projetos de no mínimo 1 ano de duração cada.
   ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

   1.4. Possui alguma certificação Java (Sun ou Oracle)?
   Selecione quantas forem necessárias.
   ( ) 1. Programmer
   ( ) 2. Developer
   ( ) 3. Web Component Developer
   ( ) 4. Business Component
   ( ) 5. Enterprise Architect
   ( ) 6. Não possui certificação

1.5. Anos de experiência em Java?
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 ( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10

2. **Conhecimento de Scrum**

2.1. Qual é o seu conhecimento em Scrum?
Sendo 1 se apenas já ouviu dizer, 3 se conhecer bem a teoria, mas nunca ter praticado, e 5 se já participou de um projeto real.
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

2.2. Você já leu o Scrum Guide? http://www.scrum.org/scrumguides/
( ) Sim ( ) Não

2.3. Você já participou de algum treinamento formal?
( ) 1. Workshop de Métodos Ágeis de 2 dias
( ) 2. Certified Scrum Master
( ) 3. Palestra introdutória de 1 a 2 horas
( ) 4. Nunca participou de treinamento formal

2.4. Anos de experiência em Scrum?
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5 ( ) 6 ( ) 7 ( ) 8 ( ) 9 ( ) 10

2.5. Descreva brevemente sua experiência com Métodos Ágeis em geral e Scrum.
[texto livre]

3. **Ferramentas**
Avalei seu grau de conhecimento e experiência em cada uma das ferramentas a seguir, sendo 1 não ter conhecimento e 5 ter experiência prática e utilização diária.

3.1. Eclipse: *(ou derivados como RAD e RSA)*
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

3.2. Servidor de Aplicação Web: *Tomcat, WebSphere ou outros*
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

3.3. Banco de dados relacional: *DB2, MySQL, ou outros*
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

3.4. Ferramenta de controle de verão de código: *CVS, SVN, Git ou outros*
( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

Todas as perguntas eram de preenchimento obrigatório.

A.2 Respostas recebidas

Foram recebidos 18 formulários de inscrição, porém apenas 12 foram selecionados devido à disponibilidade de agenda dos próprios candidatos.

A Tabela A.1 mostra os dados brutos das inscrições em ordem de recebimento. Os valores das respostas 1.4 e 2.3 representam a numeração das alternativas possíveis nas respostas.

Tabela A.1: Respostas enviadas pelos candidatos ao experimento.

Iniciais	1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	2.4	3.1	3.2	3.3	3.4	Selecionado?
LNZ	4	4	5	6	8	4	Sim	3	3	5	5	5	5	Sim
MM	2	2	2	6	2	1	Não	2	2	3	2	2	1	Não
AL	5	5	5	1,2,3	6	4	Sim	1	2	5	5	5	5	Não
FPR	3	3	3	6	1	2	Não	1	1	3	3	3	3	Não
IP	4	4	4	3	8	5	Sim	1,3	4	5	5	5	4	Não
RB	5	4	4	1	4	5	Sim	3	0	5	5	5	5	Não
JL	2	2	2	6	2	3	Não	4	0	2	3	3	3	Sim
DV	5	5	5	1,3	8	3	Não	1	0	5	4	5	5	Não
GPB	5	5	5	1,3	8	4	Não	3	1	5	5	5	5	Sim
MYFM	5	5	5	1,3,4	6	2	Não	4	0	5	5	5	5	Sim
WFC	5	5	5	1,3,5	7	3	Sim	1	0	5	5	5	5	Sim
RV	5	5	5	1,3,4	8	3	Não	1	0	5	5	5	5	Sim
KRC	5	5	5	6	10	2	Não	4	2	4	4	4	4	Sim
LG	4	4	4	1,3	8	4	Não	4	1	4	4	4	4	Sim
GV	5	4	4	1	5	3	Não	4	0	5	5	5	5	Sim
FB	5	5	4	1	7	5	Não	1	1	5	5	3	5	Sim
CSN	2	1	1	6	1	1	Não	4	0	2	2	3	2	Sim
JRJ	5	5	5	1,3,4	9	4	Não	3	1	5	5	5	5	Sim

As respostas da pergunta 2.5 estão listadas na Tabela A.2 com a transcrição das respostas preenchidas pelos candidatos, adaptadas por questões de confidencialidade de nomes de clientes e projetos.

Tabela A.2: Respostas a pergunta 2.5.

Iniciais	**Resposta**
LNZ	Já participei de diversos eventos sobre metodologias ágeis, estudo dede 2004 metodologias ágeis. Trabalho com Scrum utilizando *Rational Team Concert* (RTC) desde 2008.
MM	Possuo interesse em me certificar como Scrum Master e faço parte do Centro de Competência de Requisitos do meu departamento. Esta seria uma boa oportunidade de ampliar os contatos para este Centro de Competência e de troca de conhecimentos.
AL	Utilização diária de ferramentas e processos ágeis em projetos JavaEE.
FPR	Não tenho experiência ainda, apenas aprendi um pouco na pós, queria aprender na prática.
IP	Instrutor de Ágil e Scrum.
RB	Trabalhei durante 2.5 anos no departamento desenvolvimento de produtos, os dois anos utilizando metodologias Ágeis em conjunto com o RTC e outras ferramentas de auxílio ao processo.
JL	Somente fiz alguns cursos oferecidos pela empresa.
DV	Já li textos e assisti a apresentações, discutindo sua forma de trabalho, estimativas e objetivos.
GPB	Possuo experiência razoável na elaboração de *sprints* baseados em Estórias de Usuário. Possuo boa experiência no planejamento e execução de Testes de Unidade utilizando métodos como TDD. Trabalho a quase 2 anos em projetos que se baseiam em metodologias ágeis e Scrum para o gerenciamento das entregas do projeto. Entretanto, os projetos são projetos pequenos e com 1 ou 2 desenvolvedores, o que facilita muito o acompanhamento. Gostaria de ter mais experiência e conhecimento de aplicação de metodologias ágeis em projetos com times maiores e de maior complexidade.
MYFM	Já cheguei a estudar metodologias ágeis e Scrum, mas não possuo experiencia prévia em nenhum dos dois. O meu projeto atual chegou até a cogitar a adoção do Scrum, mas nunca foi introduzido no projeto de fato. Acredito que com o treinamento, o cliente do meu projeto possa se sentir mais seguro para dar talvez o primeiro passo rumo a adoção do Scrum.
WFC	Tive somente experiências das abordagens práticas em treinamento.
RV	Já li diversos artigos e participei de cursos de metodologias Ágeis, mas nunca trabalhei em nenhum projeto que aplicasse as metodologias de fato.
KRC	Eu estudei e pratiquei bastante a metodologia XP. Estudei um pouco Scrum, e estou usando agora de forma moderada.
LG	Trabalhei 1 ano em um projeto com Scrum.
GV	Na outra empresa, praticávamos o Scrum em linhas gerais. Apenas tínhamos nossos *sprints* com duração de três semanas e alguns dias na semana fazíamos as Reuniões de Scrum Diário de 15 a 20 minutos.
FB	Meu conhecimento sobre a metodologia vem de um curso de 1 dia na própria empresa, e de participação de projetos que usam esta metodologia (total de 3 projetos até agora). Porém, a maior parte dos requerimentos vem do gerente de projetos. Como desenvolvedor, acabo mais seguindo a metodologia do que implementando-a.
CSN	Durante dois anos trabalhei para um departamento internacional e eles usavam um método ágil próprio, uma mistura de muitas metodologias, mas não era puramente Scrum.
JRJ	Utilizei Scrum (criação de *sprint* e gráfico burndown) na minha célula de desenvolvimento do projeto de uma associação de entidades financeiras. Estou começando a usar a mesma estratégia no projeto de uma grande companhia aérea.

A.3 Cálculo do índice técnico

Com base nos valores da Tabela A.1 foi criado um índice técnico com o objetivo de ordenar os participantes por experiência.

Para cada pergunta foi dado um peso diferente afim de ressaltar o aspecto técnico de Java (perguntas da parte 1). Os pesos, bem como os valores já multiplicados, estão na Tabela A.3.

As perguntas 1.4, 2.2 e 2.3 necessitaram de conversões especiais:

- Na 1.4 foram somados 1 para as certificações de 1 a 4, e 3 para a 5. Enterprise Architect. Foi atribuído 0 para a opção 6.
- Já na 2.2, foi designado 1 para resposta Sim e 0 para a Não.
- No caso da 2.3, foram somados 5 para a alternativa 1. Workshop de Métodos Ágeis de 2 dias, e 2 para a alternativa 3. Palestra introdutória de 1 a 2 horas. Foi atribuído 0 para a opção 4. A opção 2 não foi contabilizada pois o único participante que a marcou não foi selecionado para o experimento.

As quatro perguntas da parte (3) Ferramentas foram consolidadas em apenas um valor calculado pela média simples dos valores.

No total, os pesos por parte do formulário foram respectivamente 11, 5 e 1.

Tabela A.3: Valores usados para o cálculo do índice técnico.

Iniciais	1.1	1.2	1.3×2	1.4×3	1.5×4	2.1	2.2	2.3	2.4×2	$\frac{3.1+3.2+3.3+3.4}{4}$	**Índice**
LNZ	4	4	10	0	32	4	1	2	6	5	68
JL	2	2	4	0	8	3	0	0	0	2.75	22
GPB	5	5	10	6	32	4	0	2	2	5	71
MYFM	5	5	10	9	24	2	0	0	0	5	60
WFC	5	5	10	15	28	3	1	5	0	5	77
RV	5	5	10	9	32	3	0	5	0	5	74
KRC	5	5	10	0	40	2	0	0	4	4	70
LG	4	4	8	6	32	4	0	0	2	4	64
GV	5	4	8	3	20	3	0	0	0	5	48
FB	5	5	8	3	28	5	0	5	2	4.5	66
CSN	2	1	2	0	4	1	0	0	0	2.25	13
JRJ	5	5	10	9	36	4	0	2	2	5	78

Apêndice B

Estórias de Usuário com Testes de Aceitação

Este foi o *Backlog* do Produto utilizado durante o Experimento Controlado deste trabalho.

B.1 Estória 1

Descrição: Como um cliente, eu quero buscar uma linha que tenha um ponto de parada próximo a um endereço desejado, para que eu possa pegá-la para chegar a meu destino.

Estimativa: 20 pontos de estória.

Testes de Aceitação:

1. Escolho o turno e sentido;
2. O endereço da busca tem formato latitude/longitude;
3. Retorna o Nome da Linha;
4. O ponto pertencente a linha retornada é de parada;
5. O ponto está à menos de 500 metros de distância em linha reta do endereço de entrada.

Testes Excepcionais:

1. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
2. Mostrar mensagem de erro se não houverem parâmetros de entrada;
3. O parâmetro Turno não pode ser nulo;
4. O parâmetro Turno deve ser um número inteiro positivo;
5. Mostrar mensagem de erro caso não existam linhas para o Turno desejado;
6. O parâmetro Sentido não pode ser nulo;
7. O parâmetro Sentido deve ser um único caractere;
8. O valores válidos de Sentido são: T, t, C, c;
9. Os parâmetros de Latitude e Longitude não podem ser nulos;
10. Os valores de Latitude e Longitude devem ser numéricos, com casas decimais, negativos ou positivos.

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.2 Estória 2

Descrição: Como um cliente, eu quero buscar o ponto mais próximo de minha casa, em meu turno, para que eu ande o mínimo possível e possa me programar para esperar o mínimo possível.

Estimativa: 5 pontos de estória.

Testes de Aceitação:

1. Posição de minha casa é representada pelo par latitude/longitude;
2. Retorna nome da linha, ponto, e distância em metros da casa ao ponto;
3. O ponto é de parada;
4. O sentido da linha é de ida para o trabalho.

Testes Excepcionais:

1. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
2. Mostrar mensagem de erro se não houverem parâmetros de entrada;
3. O parâmetro Turno não pode ser nulo;
4. O parâmetro Turno deve ser um número inteiro positivo;
5. Mostrar mensagem de erro caso não existirem linhas para o Turno desejado;
6. Os parâmetros de Latitude e Longitude não podem ser nulos;
7. Os valores de Latitude e Longitude devem ser numéricos, com casas decimais, negativos ou positivos;
8. O formato do horário de parada de um ponto deve ser HH:mm.

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.3 Estória 3

Descrição: Como um cliente, eu quero adicionar o meu ponto à minha linha para que outros clientes que morem na mesma região e trabalhem no mesmo turno possam localizá-lo.

Estimativa: 8 pontos de estória.

Testes de Aceitação:

1. Pode-se adicionar em qualquer sentido;
2. Ponto adicionado como tipo parada;
3. Não existe outro ponto na mesma linha, no mesmo sentido e, no mesmo horário;
4. Não existe outro ponto com mesma posição no mesmo turno.

Testes Excepcionais:

1. O cliente deve ser identificado pelo seu e-mail como parâmetro de entrada;
2. O E-mail do cliente não pode ser nulo;
3. O E-mail do cliente deve estar no formato válido;
4. O E-mail de entrada não pode ser de um usuário do tipo Gestor;
5. O parâmetro Linha não pode ser nulo;
6. O parâmetro Linha deve ser no formato texto e deve representar a sigla da linha;
7. Mostrar mensagem de erro caso não exista uma linha com a sigla desejada;
8. O parâmetro Sentido não pode ser nulo;
9. O parâmetro Sentido deve ser um único caractere;
10. O valores válidos de Sentido são: T, t, C, c;
11. Os parâmetros de Latitude e Longitude não podem ser nulos;
12. Os valores de Latitude e Longitude devem ser numéricos, com casas decimais, negativos ou positivos;
13. O Horário de parada do ponto não pode ser nulo;
14. O Horário de parada do ponto deve estar no formato HH:mm;
15. O campo Referência deve ser no formato texto e pode ser nulo;
16. O campo Descrição deve ser no formato texto e pode ser nulo;
17. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
18. Mostrar mensagem de erro se não houverem parâmetros de entrada.

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.4 Estória 4

Descrição: Como um Gestor, eu quero ver a lista de todos os pontos de um itinerário, independente do seu estado, para que eu possa analisá-lo.

Estimativa: 3 pontos de estória.

Testes de Aceitação:

1. Itinerário é uma linha em um sentido;
2. Lista ordenada por horário;
3. Contém pontos tanto de parada quando de caminho.

Testes Excepcionais:

1. O Gestor deve ser identificado pelo seu e-mail como parâmetro de entrada;
2. O E-mail do gestor não pode ser nulo;
3. O E-mail do gestor deve estar no formato válido;
4. O E-mail de entrada não pode ser de um usuário do tipo Cliente;
5. O parâmetro Linha não pode ser nulo;
6. O parâmetro Linha deve ser no formato texto e deve representar a sigla da linha;
7. Mostrar mensagem de erro caso não exista uma linha com a sigla desejada;
8. O parâmetro Sentido não pode ser nulo;
9. O parâmetro Sentido deve ser um único caractere;
10. O valores válidos de Sentido são: T, t, C, c;
11. Mostrar mensagem de erro caso não existam pontos na linha e sentido;
12. A resposta deve ser uma lista de pontos;
13. O Horário de parada dos pontos devem estar no formato HH:mm;
14. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
15. Mostrar mensagem de erro se não houverem parâmetros de entrada.

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.5 Estória 5

Descrição: Como um gestor, eu quero visualizar uma lista de todas as linhas com pontos pendentes para serem validados.

Estimativa: 3 pontos de estória.

Testes de Aceitação:

1. Lista ordenada por sigla da linha;
2. Para cada linha, são exibidos ambos sentidos;
3. Apenas linhas com pontos pendentes devem ser exibidas.

Testes Excepcionais:

1. O Gestor deve ser identificado pelo seu e-mail como parâmetro de entrada;
2. O E-mail do gestor não pode ser nulo;
3. O E-mail do gestor deve estar no formato válido;
4. O E-mail de entrada não pode ser de um usuário do tipo Cliente;
5. Mostrar mensagem caso não existam pontos pendentes em todas as linhas;
6. Todas as linhas, independente do turno, devem ser mostradas, caso tenham pelo menos um ponto pendente;
7. A resposta deve ser uma lista de linhas;
8. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
9. Mostrar mensagem de erro se não houverem parâmetros de entrada.

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.6 Estória 6

Descrição: Como um gestor, eu quero validar um ponto para garantir que os itinerários estejam de acordo com o contratado.

Estimativa: 2 pontos de estória.

Testes de Aceitação:

1. Ponto marcado como validado.
2. Opcionalmente posso atualizar o horário, descrição e referência.

Testes Excepcionais:

1. O Gestor deve ser identificado pelo seu e-mail como parâmetro de entrada;
2. O E-mail do gestor não pode ser nulo;
3. O E-mail do gestor deve estar no formato válido;
4. O E-mail de entrada não pode ser de um usuário do tipo Cliente;
5. O Ponto a ser validado deve ser identificado pelo seu identificador numérico;
6. O identificador do ponto não pode ser nulo;
7. O identificador do ponto deve estar no número inteiro positivo;
8. Mostrar mensagem caso não exista ponto com o identificador;
9. Mostrar mensagem de erro caso o ponto já seja válido.
10. Se for fornecido o Horário de parada do ponto para ser atualizado, o formato deve ser HH:mm;
11. O E-mail do gestor deve estar gravado no campo *validado_por* do respectivo ponto;
12. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
13. Mostrar mensagem de erro se não houverem parâmetros de entrada.

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.7 Estória 7

Descrição: Como um gestor, eu quero rejeitar um ponto para garantir que os itinerários estejam de acordo com o contratado.

Estimativa: 1 ponto de estória.

Testes de Aceitação:

1. Ponto excluído do itinerário;
2. Apenas pontos pendentes são elegíveis.

Testes Excepcionais:

1. O Gestor deve ser identificado pelo seu e-mail como parâmetro de entrada;
2. O E-mail do gestor não pode ser nulo;
3. O E-mail do gestor deve estar no formato válido;
4. O E-mail de entrada não pode ser de um usuário do tipo Cliente;
5. O Ponto a ser rejeitado deve ser identificado pelo seu identificador numérico;
6. O identificador do ponto não pode ser nulo;
7. O identificador do ponto deve estar no número inteiro positivo;
8. Mostrar mensagem caso não exista ponto com o identificador;
9. Mostrar mensagem de erro caso o ponto já seja válido.
10. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
11. Mostrar mensagem de erro se não houverem parâmetros de entrada.

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.8 Estória 8

Descrição: Como um cliente, eu quero atualizar o horário de passagem de uma linha por um ponto para manter a informação atualizada.

Estimativa: 3 pontos de estória.

Testes de Aceitação:

1. Apenas pontos de parada podem ser atualizados;
2. Apenas pontos validados podem ser atualizados;
3. Novo horário deve ser único no itinerário;
4. Novo horário deve estar entre o mesmo intervalo dos pontos anteriores;
5. Ponto continua válido após a atualização.

Como esta estória não foi implementada por nenhuma equipe, não foram documentados os Testes Excepcionais.

B.9 Estória 9

Descrição: Como um gestor, eu quero adicionar um ponto de caminho em um itinerário para que, quando em um mapa, a rota real feita pelo ônibus seja mostrada.

Estimativa: 5 pontos de estória.

Testes de Aceitação:

1. Entrada: latitude, longitude, linha, sentido, horário;
2. Ponto adicionado como tipo Caminho;
3. O Horário do ponto deve estar entre os horários de dois pontos de parada já existentes;
4. Ponto de caminho sempre é Válido.

Testes Excepcionais:

1. O Gestor deve ser identificado pelo seu e-mail como parâmetro de entrada;
2. O E-mail do gestor não pode ser nulo;
3. O E-mail do gestor deve estar no formato válido;
4. O E-mail de entrada não pode ser de um usuário do tipo Cliente;
5. O parâmetro Linha não pode ser nulo;
6. O parâmetro Linha deve ser no formato texto e deve representar a sigla da linha;
7. Mostrar mensagem de erro caso não exista uma linha com a sigla desejada;
8. O parâmetro Sentido não pode ser nulo;
9. O parâmetro Sentido deve ser um único caractere;
10. O valores válidos de Sentido são: T, t, C, c;
11. Os parâmetros de Latitude e Longitude não podem ser nulos;
12. Os valores de Latitude e Longitude devem ser numéricos, com casas decimais, negativos ou positivos;
13. O Horário de parada do ponto não pode ser nulo;
14. O Horário de parada do ponto deve estar no formato HH:mm;
15. Mostrar mensagem de erro caso o banco de dados esteja inacessível;
16. Mostrar mensagem de erro se não houverem parâmetros de entrada;
17. Mostrar mensagem de erro se já existir um ponto na linha e sentido com o mesmo horário;

18. Mostrar mensagem de erro se jé existir um ponto no mesmo turno e sentido com a mesma posição (latitude/longitude).

Os Testes Excepcionais apenas foram disponibilizados para as equipes G2 e G3, que utilizaram o processo Scrum+CE.

B.10 Estória 10

Descrição: Como um gestor, eu quero cadastrar um turno no sistema.

Estimativa: 2 pontos de estória.

Testes de Aceitação:

1. Turno tem um nome, horário de entrada, horário de saída, posição e nome da localidade de trabalho;
2. Pode ser adicionado mais de um turno para mesma localidade;
3. Não se pode ter mais de um turno com mesmos horários nem mesmo nome.

Como esta estória foi implementada apenas pela equipe G1, que utilizou o processo Scrum, não foram documentados os Testes Excepcionais.

B.11 Estória 11

Descrição: Como um cliente, eu quero marcar um ponto como sendo o meu ponto.

Estimativa: 2 pontos de estória.

Testes de Aceitação:

1. Apenas pontos validados podem ser marcados;
2. Cada cliente poderá marcar apenas um ponto em cada sentido;
3. As linhas de ida e volta podem ser diferentes.

Como esta estória não foi implementada por nenhuma equipe, não foram documentados os Testes Excepcionais.

B.12 Estória 12

Descrição: Como um gestor, eu quero cadastrar uma linha em um determinado turno preexistente.

Estimativa: 5 pontos de estória.

Testes de Aceitação:

1. Linha tem uma sigla, um nome e uma descrição opcional;
2. Se turno tiver horário de entrada, deve ser criado automaticamente um ponto final no sentido trabalho;
3. Se turno tiver horário de saída, deve ser cariado um ponto inicial no sentido casa; Tais pontos já deve estar validados.

Como esta estória não foi implementada por nenhuma equipe, não foram documentados os Testes Excepcionais.

B.13 Estória 13

Descrição: Como um gestor, eu quero atualizar o horário de um ponto qualquer.

Estimativa: 2 pontos de estória.

Testes de Aceitação:

1. Qualquer tipo de ponto pode ser atualizado;
2. Indiferente ser válido ou não;
3. Novo horário deve ser único no itinerário;
4. Novo horário deve estar entre o mesmo intervalo dos pontos anteriores;
5. Validade não muda após a atualização.

Como esta estória não foi implementada por nenhuma equipe, não foram documentados os Testes Excepcionais.