Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise e Implantação de Métodos Ágeis: Um Estudo de Caso no Centro de Informática da Universidade de Brasília

Riane de Oliveira Torres Santos

Dissertação apresentada como requisito parcial
para conclusão do Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Rommel Novaes Carnalho

Brasília
2015

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

Santos, Riane de Oliveira Torres
S237a Análise e Implantação de Métodos Ágeis: Um Estudo de Caso no Centro de Informática da Universidade de Brasília / Riane de Oliveira Torres Santos; orientador Rommel Novaes Carvalho. -- Brasília, 2015.
98 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2015.

1. Métodos Ágeis. 2. Práticas Ágeis. 3. Desenvolvimento de Software. 4. Engenharia de Software. I. Carvalho, Rommel Novaes, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise e Implantação de Métodos Ágeis: Um Estudo de Caso no Centro de Informática da Universidade de Brasília

Riane de Oliveira Torres Santos

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Rommel Novaes Carvalho (Orientador)
CIC/UnB

Prof. Dr. Paulo Roberto Miranda Meirelles
FGA/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
CIC/UnB

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 26 de janeiro de 2015

Dedicatória

Este trabalho é dedicado a minha família: minha mãe Francenide, minha avó Didi, meu irmão Raminho, meu pai Anatólio, meu padrasto Juarez e meu marido Giovanni, pelo amor incondicional dispensado a mim, durante todos esses anos.

Agradecimentos

Concluir essa etapa talvez tenha sido um dos maiores desafios da minha vida, contudo, todas as dificuldades se tornaram mais leves com a participação de algumas pessoas, as quais agradeço de todo o coração nas breves linhas abaixo:

Primeiramente agradeço a Deus, pelo dom da vida, e por sempre estar nos iluminando e nos conduzindo.

Ao Prof. Dr. Marcelo Ladeira, pelo apoio, compreensão e pelas tantas conversas que me incentivaram a continuar. Não esquecerei de você. "Fazer o bem, sempre".

Ao Prof. Dr. Rommel Novaes Carvalho pela orientação, pelas revisões, pelos conhecimentos transmitidos e acima de tudo, pela paciência. Não foi uma tarefa fácil, mas deixo aqui registrada a minha admiração por você e o meu sincero agradecimento. Obrigada!

Ao Prof. Dr. Jacir Luiz Bordim, meu grande exemplo de líder, por ter acreditado no potencial dessa pesquisa e pelo grande incentivo durante esses quatro anos que trabalhamos juntos.

Aos amigos do Centro de Informática que fizeram parte dessa turma: Andrei, Jackson, Juvenal e Karam, pelo companheirismo e união durante o curso, e em especial ao Andrei, meu companheiro de área, por todo o apoio, pelos trabalhos em parceria e pelo auxílio na execução do protocolo de *Grounded Theory*. Fomos um verdadeiro time.

Aos também amigos do Centro de Informática que participaram dos estudos de caso realizados nesse trabalho, pela disponibilidade e por terem aceitado o desafio de encarar um novo processo de desenvolvimento.

Aos Professores do PPCA pelos ensinamentos e contribuições durante o curso.

Por fim, agradeço a todos que direta ou indiretamente contribuíram de alguma forma para a realização desse trabalho.

Resumo

Os métodos ágeis fazem parte de uma nova concepção de desenvolvimento de software, já que propõem abordagens de baixa complexidade e de menos rigidez que as utilizadas nas metodologias tradicionais. Dentro desse contexto, esse trabalho realizou a adoção desses métodos em uma instituição pública do governo federal, com o objetivo de minimizar os problemas enfrentados em um processo de desenvolvimento de software prescritivo que é utilizado há 5 anos. Esses problemas incluíam o atraso na entrega de projetos, a dificuldade de comunicação entre os envolvido e a falta de motivação e integração da equipe. A implantação foi realizada através de dois estudos de caso bem sucedidos utilizando a nova abordagem: para o primeiro foram customizados padrões ágeis utilizados como referência na indústria de software e para o segundo utilizou-se a experiência do primeiro estudo de caso para definição de um processo ágil para o órgão. Para apresentação dos resultados dessa pesquisa, aplicou-se um questionário nas equipes envolvidas, de forma a obter a percepção do grupo em relação à nova abordagem de desenvolvimento de sistemas. As respostas dos questionários foram avaliadas através do método de pesquisa qualitativa *Grounded Theory* e os resultados apontam que os métodos ágeis promoveram melhorias em relação à capacidade de trabalho em equipe, à comunicação e à manutenção de um ritmo de trabalho constante. Esta pesquisa contribui como mais uma referência para equipes de desenvolvimento que pretendem realizar a adoção de métodos ágeis, apresentando a experiência de uma instituição que passou por essa implantação.

Palavras-chave: métodos agéis, práticas agéis, desenvolvimento de software, engenharia de software

Abstract

Agile methods are part of a new approach of software development, it proposes low complexity and less stiffness approaches than those used in traditional methodologies. Within this context, this research constitutes the adoption of these methods in a public institution of the federal government, aiming to minimize the problems faced in the current methodology that has been used for 5 years. These problems included delays in the delivery of projects, the difficulty of communication among stakeholders and the development team and the lack of motivation and team integration. The new methodology was tested in two successful case studies using the new approach: the first has customized agile reference standards used in the software industry, while the second has used the experience of the first case study for defining an agile approach to the institution. To evaluate the results of this research, we applied a questionnaire in teams involved in the case studies to get the perception of the group about the new systems development approach. The answers to the questionnaires were evaluated using the qualitative research method Grounded Theory and the results show that agile methods promoted improvements in the ability to work in team, in communication, and in maintaining a constant pace of work. This research contributes as a reference for development teams that plan to adopt agile methods, bringing the experience of an institution that has gone through this process.

Keywords: agile methods, agile practices, software development, software engineer

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desenvolver sistemas é uma atividade reconhecidamente problemática quando se trata de aspectos relacionados a prazo, qualidade, escopo e custos. A indústria de software vem tentando resolver estes problemas através da criação de metodologias, modelos, processos e práticas que possam de alguma forma suportar esta atividade [16].

Os primeiros modelos criados, tal como o processo em cascata, partiram do princípio da produção fabril, ou seja, como em uma fábrica, com etapas fixas, segmentadas, na qual o início da etapa seguinte depende da conclusão da etapa anterior [53]. Ao longo dos anos muitos problemas foram identificados na utilização dessa abordagem tradicional, como o alto custo gerado na mudança dos requisitos inicialmente propostos, a extensa documentação produzida e a falta de participação do cliente ou usuário que demandou o projeto [16].

As metodologias leves surgiram durante os anos 90 em resposta ao desenvolvimento lento e burocrático que ainda eram vivenciados pelos métodos de desenvolvimento anteriormente utilizados. Em 2001, reuniram-se 17 (dezessete) representantes dessas novas abordagens, que ao final do encontro, produziram um manifesto com os princípios básicos para o desenvolvimento ágil de software, que incluíam a valorização das pessoas do projeto (clientes e equipe), da interação entre elas, do sistema entregue e em funcionamento, da colaboração com o cliente e da capacidade de responder a mudanças. Esse manifesto ficou conhecido como "Manifesto Ágil" e os métodos que se encaixavam em seus princípios passaram a ser conhecidos como métodos ágeis [27].

A disseminação dessa nova forma de desenvolvimento atingiu equipes de desenvolvimento em todo o mundo. No Brasil, o relatório técnico RT MAC-2012-03 [13] aponta o crescimento na adoção destes métodos por usuários que vislumbram aumento de produtividade, gerenciamento de mudanças de prioridade e aumento de qualidade de software. Dentro desse contexto, verifica-se ainda que algumas empresas de caráter público e privado temem essa adoção devido a fatores como falta de documentação, falta de previsibilidade

e planejamento prévio a longo prazo [13].

O Centro de Informática da Universidade de Brasília (CPD/UnB) é o orgão responsável pela tecnologia da informação da instituição, sendo uma de suas atribuições o desenvolvimento de sistemas institucionais. Nos últimos 5 (cinco) anos, na tentativa de estruturar a área de desenvolvimento de sistemas e promover a manutenção do conhecimento institucional, o Centro vem tentando realizar a implantação de um processo desenvolvimento de *software* baseado em uma metodologia tradicional, o *Rational Unified Process* [32].

Essa metodologia prevê ciclos curtos e contínuos de desenvolvimento, de forma iterativa e incremental e com entrega de uma versão funcional do produto ao final de cada ciclo. No caso do CPD/UnB, a má aplicação da metodologia tornou o processo similar ao modelo cascata, composto de um único ciclo de desenvolvimento e com entrega de versão apenas ao final do projeto. Assim, os projetos passaram a ter uma duração maior que a esperada, trazendo insatisfação para os clientes e desmotivação para a equipe, que demorava para obter resultados no trabalho executado. Além disso, a forma de implementação com a separação de quatro áreas distintas para o desenvolvimento de software (desenvolvimento de sistemas, manutenção de sistemas, análise de sistemas e estratégia de dados) promoveu o afastamento da equipe, dificultando a sua integração e a comunicação.

Assim, como forma de minimizar as dificuldades experimentadas na utilização desse processo e com base em experiências bem sucedidas de instituições que realizaram a implantação de métodos agéis [38][4][22][37], foi proposto através desse trabalho, a adoção desses métodos no CPD/UnB.

Dessa forma, o estudo busca nessa adoção definir uma abordagem ágil para o Centro, a ser utilizada em 2 (dois) projetos piloto. A implantação dessa nova abordagem será descrita através de estudo de caso.

Ao final da realização do estudo de caso, um questionário será aplicado nas equipes dos projetos, a fim de avaliar os impactos da nova implantação. Esses questionários serão avaliados através de um método de pesquisa qualitativa, a *Grounded Theory*[11].

1.1 Objetivo

O objetivo principal desta pesquisa é realizar um estudo sobre a adoção de métodos ágeis no Centro de Informática da Universidade de Brasília, de forma a verificar o impacto dessa adoção na área de desenvolvimento de sistemas do Centro. De forma a complementar o objetivo geral proposto, os objetivos específicos abaixo foram estabelecidos:

- Realizar uma revisão bibliográfica da literatura acerca dos métodos ágeis e tradicionais de desenvolvimento de *software*;

- Fazer um estudo sobre o CPD/UnB focando a área de desenvolvimento de sistemas e os processos de trabalho da área;
- Definir uma abordagem ágil a ser utilizada pelo CPD/UnB em 2 projetos pilotos do Centro;
- Avaliar os resultados da aplicação da nova abordagem através de um questionário a ser aplicado às equipes dos projetos que irão utilizar a abordagem ágil.

1.2 Organização do trabalho

O trabalho está dividido em 5 (cinco) capítulos, dos quais o primeiro é a introdução. O Capítulo 2 apresenta o referencial teórico dessa pesquisa, envolvendo conceitos sobre desenvolvimento de softwares, metodologias tradicionais de desenvolvimento, métodos agéis, temas em agilidade, trabalhos correlatos à esse estudo e experiências de implantação de métodos ágeis na Administração Pública Federal. No Capítulo 3 encontra-se o estudo de caso, que envolve uma pesquisa sobre o CPD/UnB e os projetos pilotos realizados. O Capítulo 4 apresenta os resultados obtidos na adoção através da apresentação da abordagem ágil definida, da análise descritiva do questionário e da análise qualitativa desse questionário, realizada através da *Grounded Theory*. Por fim, no Capítulo 5 é realizada uma discussão sobre os resultados obtidos e são apresentadas as limitações dessa pesquisa, as contribuições e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Esse capítulo apresenta os principais conceitos e teorias relacionados ao desenvolvimento de *software*. Na Seção 2.1 definimos os conceitos de *software*, a engenharia de *software* e suas camadas. Na Seção 2.2 detalhamos o modelo de processo prescritivo, no qual estão inseridas as metodologias tradicionais de desenvolvimento. Em seguida, na Seção 2.3, conceituamos o modelo de processo adaptativo, no qual abordamos os métodos ágeis, foco principal dessa pesquisa. A descrição desses modelos foi fortemente baseada nos conceitos apresentados por Pressman [41] e Sommerville [53].

De forma a investigar os temas que estão sendo discutidos recentemente sobre agilidade, a Seção 2.4 apresenta um estudo sobre alguns desses temas. A Seção 2.5 apresenta os trabalhos que estão relacionados a temática dessa pesquisa, ou seja, implantação de métodos ágeis em um órgão de uma Instituição Federal de Ensino. Finalizando, a Seção 2.6 traz algumas experiências desse tipo de implantação na Administração Pública Federal.

2.1 Desenvolvimento de software

No mundo globalizado e na atual era da informação ainda se tem como uma das principais tecnologias ofertadas o *software*. *Software* pode ser descrito como um conjunto de elementos estruturados para executar um método, um procedimento ou um controle, tendo como principal produto a informação [41]. A construção desta tecnologia é baseada na Engenharia de *Software*, que segundo definição do *Institute of Electric and Electronic Engineers* (IEEE) [9] é a "aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de *software*". É a disciplina que abarca os processos, métodos e ferramentas inerentes ao desenvolvimento de *software*, com o objetivo de entregá-lo com qualidade e dentro do prazo esperado [41].

Figura 2.1: Representação da engenharia de software em camadas.

A engenharia de *software* pode ser representada em camadas e a primeira delas é o foco na qualidade que sustenta todas as demais a fim de que tenham abordagens efetivas, conforme apresentado na Figura 2.1. Em seguida aparece a camada de processos que representa a forma como os projetos de *software* serão controlados e fornece a conjuntura na qual serão aplicados os métodos e ferramentas das camadas acima [41].

A camada de métodos é responsável por executar atividades técnicas que auxiliarão o desenvolvimento de software, como por exemplo análise de requisitos, modelagem do projeto, construção de *software* e testes. Já a camada de ferramentas fornece o apoio automatizado para as camadas de processos e métodos, ou seja, sistemas que automatizam as atividades desenvolvidas nessas camadas.

Um processo de *software* pode ser descrito como um conjunto de atividades, ações, tarefas e resultados que têm como objetivo a construção de um produto de *software*, dentro do prazo e atingindo a qualidade necessária à satisfação dos envolvidos[41][53]. Ao contrário da grande maioria dos processos de engenharias, a engenharia de *software* prevê além de abordagens de processos prescritivas e rígidas, abordagens adaptativas, nas quais um processo pode ser adaptado e customizado de acordo com o contexto no qual está sendo executado [41].

Um modelo de processo descreve e representa um processo de *software*, definindo assim as atividades necessárias à sua completude. De forma geral, há 4 (quatro) atividades comuns a todos os processos[41][53]. São elas[53]:

- Especificação: definição das funcionalidades, requisitos, limitações e restrições do *software* de acordo com o informado pelo cliente;
- Desenvolvimento: construção do *software* de acordo com as atividades de especificação;
- Validação: verificação e inspeção do produto construído com o objetivo de avaliar se está de acordo com a especificação fornecida pelo cliente;
- Evolução: atualização da especificação anteriormente definida de forma a evoluir o produto de acordo com uma necessidade de mudança apresentada pelo cliente.

Vale ressaltar que cada modelo de processo poderá organizar, bem como descrever cada uma dessas atividades de forma distinta [53]. Cada organização deverá escolher o modelo de processo mais adequado às suas características considerando o tipo de produto que irá desenvolver, de forma a atender os requisitos de prazo, qualidade e utilidade do produto a ser construído [53][41].

Os modelos de processo podem ser divididos basicamente em 2 (duas) categorias: modelos de processo prescritivos e modelos de processo adaptativos [41]. As próximas sessões apresentarão ambos os modelos de forma detalhada, bem como alguns exemplos desses modelos de processo.

2.2 Modelos de processo prescritivo

Também conhecidos como tradicionais, foram criados a partir do final da década de 60 com o objetivo de solucionar o caos que era vivenciado na indústria do *software* [41][4]. Esses modelos possuem a denominação de prescritivo, pois prescrevem de forma ordenada e inter-relacionada um conjunto de elementos estruturais, nos quais estão inclusos atividades, tarefas, papéis, ações e artefatos da engenharia de *software*.

Apesar da contribuição fornecida à história da engenharia de *software*, a utilização desses modelos não resolveu o caos vivido pela indústria. Bassi Filho ressalta em seu trabalho [4] a comunicação baseada em documentos na utilização desses modelos e enfatiza que na prática o excesso de documentação não substitui a necessidade de interação entre as pessoas envolvidas no projeto. Além disso, o autor aponta para a rigidez desses processos que de certa forma limita o poder criativo das partes envolvidas, já que são descritos em um formato mecânico, tornando-se assim uma operação determinística. Nas sub-seções abaixo detalharemos alguns modelos de processos prescritivo, como o modelo cascata, a prototipação, o modelo espiral e o *Rational Unified Process*(RUP).

2.2.1 O modelo cascata

Foi criado por Royce [43] no ano de 1970 e é também conhecido como modelo de vida clássico [41]. Sugere uma sequência linear de fases, em cascata, nas quais as saídas de cada fase tornam-se entradas para a próxima fase. As principais etapas desse modelo são análise e definição de requisitos, projeto de sistemas e de software, implementação e teste de unidades, integração e teste de sistemas e por último a operação e manutenção [53]:

- Análise e definição de requisitos: é a etapa de identificação das funcionalidades, requisitos e restrições do sistema. Essas informações devem ser obtidas junto ao cliente e posteriormente detalhadas para construção da especificação do sistema.
- Projeto de sistemas e de software: o foco do projeto de sistemas é identificar através da especificação produzida os requisitos de *hardware* e *software* para criação da arquitetura geral do sistema. O projeto de software visa produzir uma abstração da especificação do sistema, definindo assim como cada parte ou unidade do sistema será implementada.
- Implementação e testes de unidades: a implementação envolve a construção de um conjunto de programas ou unidades de programa de *software* de acordo com o projeto especificado. Já os testes de unidades visam verificar se o que foi construído, de forma unitária, atende à especificação de requisitos e ao projeto de sistemas e de software produzidos.
- Integração e teste de sistemas: consiste na junção de todas as unidades produzidas na etapa anterior, de forma a testar o produto de *software* completo. Após a integração, outras etapas de testes são executadas com o objetivo de verificar se houve alguma falha após a integração. Após essa etapa, o produto é entregue ao cliente.
- Operação e manutenção: na etapa de operação o sistema é instalado e disponibilizado para uso. A manutenção envolve corrigir os erros não identificados na fase de testes. De forma evolutiva, essa fase também contempla a mudança de requisitos, devendo assim voltar para alguma das fases do ciclo, seja para a inclusão de uma nova funcionalidade, ou para ajustes e melhorias no produto.

A Figura 2.2 representa o modelo cascata, com a definição das fases do ciclo de vida e a sequência linear entre elas.

Apesar da intenção de disciplinar a construção de software através de conceitos práticos da engenharia, o modelo em cascata não se mostrou eficiente na construção de projetos com requisitos que tendem a ter mudanças [53]. Isso se deve ao fato da inflexibilidade do modelo quando há necessidade de retornar a etapas anteriores. Há um alto custo para

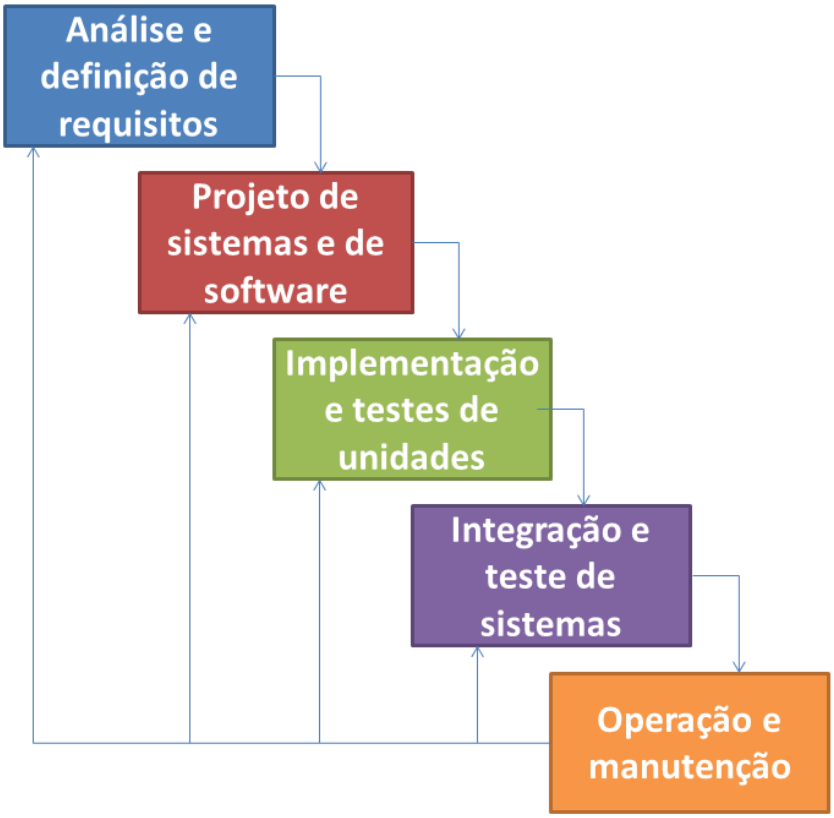

Figura 2.2: O modelo cascata [53].

produção e aprovação de documentos em todas as etapas, que ao serem refeitas, poderão onerar o tempo e o orçamento do projeto [53][22].

Há ainda críticas relacionadas a ociosidade da equipe do projeto, provocada pela dependência entre as etapas. Assim, um membro da equipe poderá ficar ocioso aguardando a conclusão de etapas anteriores a que ele executará [41].

2.2.2 O modelo prototipação

O modelo prototipação está inserido no contexto dos modelos de processo evolucionário que têm como premissa a ideia de desenvolver versões preliminares do produto e ir refinando até que o produto esteja concluído [53].

A Figura 2.3 apresenta a prototipação, detalhando as fases que envolvem o modelo, através de ciclos iterativos de desenvolvimento. A primeira fase é a comunicação, na qual são identificados os requisitos já definidos pelo usuário. A partir desses requisitos são realizadas as fases de projeto rápido e modelagem do projeto rápido. O projeto rápido foca na definição de elementos do produto que serão visíveis ao usuário, a exemplo do *layout* da *interface*, do posicionamento de informações na tela e padrão de cores [41].

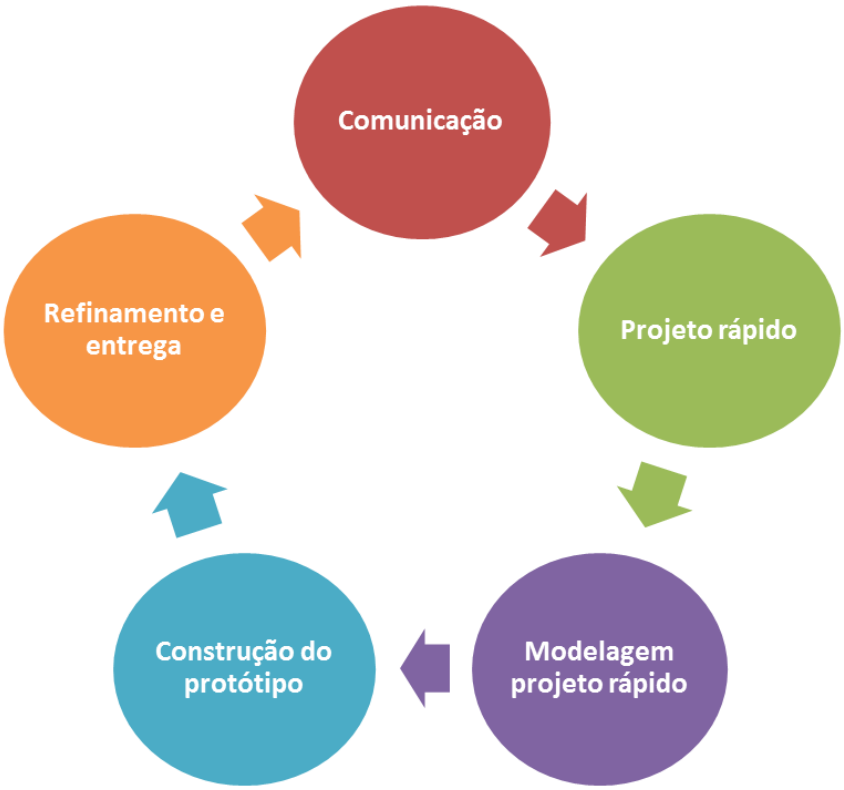

Figura 2.3: O modelo prototipação [41].

Esse projeto auxiliará na construção de uma versão preliminar do produto, ou seja, o protótipo, que é construído na fase de construção do protótipo.[41].

Com o protótipo definido, a próxima fase é a de refinamento e entrega, na qual é feita uma reunião com os envolvidos para obtenção do *feedback* do que foi construído. O resultado dessa reunião servirá para refinar o produto e decidir por executar ou não um novo ciclo de desenvolvimento. Poderão ser realizados quantos ciclos se julguem necessários para que o produto atenda às necessidades do cliente [41].

O modelo prototipação apesar de ter sido criado com o objetivo de ser um modelo de processo, muitas vezes é utilizado como uma técnica em outros modelos, pois consegue fornecer uma visão compreensiva do que será desenvolvido.

De forma geral, esse modelo atende as expectativas de quem o utiliza, porém, a equipe do projeto deverá ter ciência que a prototipação é um mecanismo para melhor compreensão e detalhamento dos requisitos. Do contrário, poderá causar falhas de entendimento ao cliente, que ao ver um protótipo acredita que já tem um produto operacional que necessita de poucas correções para que seja finalizado, quando na verdade o protótipo ainda precisa de um ou mais ciclos para estar pronto.

2.2.3 O modelo espiral

O modelo espiral foi proposto por Barry Boehm em 1988 [7] e engloba a proposta iterativa e incremental do modelo prototipação e o controle proposto no modelo cascata [41]. Além disso, esse modelo propõe uma avaliação a cada ciclo de desenvolvimento, através de uma etapa de análise de riscos, na qual os envolvidos poderão entender, mitigar e sanar as dificuldades que surgirem durante o desenvolvimento do produto [4][41].

Na execução do modelo espiral, cada iteração de desenvolvimento é formada por quatro etapas que envolvem o planejamento, a análise de riscos, a engenharia e a avaliação do cliente. Ao final de cada iteração espera-se que o produto seja aperfeiçoado e esteja o mais próximo do produto final a ser entregue. As etapas definidas no modelo espiral são [4]:

- Planejamento: engloba a fase de levantamento de requisitos, que posteriormente são refinados a cada iteração, através da avaliação do cliente.
- Análise de riscos: nesta etapa, deve-se verificar os riscos associados aos requisitos levantados na fase de planejamento. Esses riscos são analisados de forma a propor soluções para pelo menos minimizá-los. Nessa fase, poderá caber aos envolvidos algumas decisões chaves, como por exemplo, parar ou continuar o projeto.
- Engenharia: é a construção propriamente dita do produto através dos requisitos levantados nas fases de planejamento e das soluções propostas na fase de análise de riscos. Vale ressaltar que o produto é construído de forma incremental e iterativa, ou seja, serão executados quantos ciclos sejam necessários para conclusão do produto. Para a execução dessa etapa poderá ser escolhido tanto o modelo de prototipação, quanto o modelo cascata.
- Avaliação do cliente: o cliente avalia o produto entregue na etapa da engenharia e o resultado dessa avaliação servirá de insumo para o próximo ciclo de desenvolvimento.

A Figura 2.4 apresenta o modelo espiral, mostrando as etapas do modelo e o caminho percorrido a cada iteração. Ressalta-se que cada círculo representa uma iteração completa no modelo, e assim, serão percorridas quantas voltas no espiral sejam necessárias para a conclusão do produto.

Esse modelo consegue suprir as necessidades do desenvolvimento de *software* relacionadas às mudanças de requisitos, já que a cada ciclo os requisitos são planejados e avaliados junto ao cliente [41].

Uma dificuldade que pode ser impactante no uso desse modelo é a incerteza da quantidade de ciclos que serão necessárias para a conclusão do produto, mas de forma geral, é um problema presente também nos demais modelos apresentados [4][22].

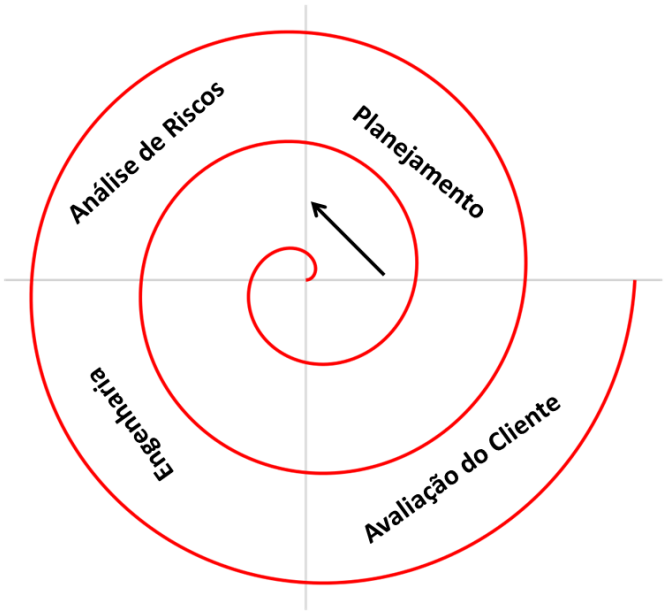

Figura 2.4: O modelo espiral [41][4].

2.2.4 *Rational Unified Process* (RUP)

O RUP [32] foi criado em 1998 pela empresa *Rational Corporation* e trata-se de uma versão do *Unified Process* (UP) [29] de desenvolvimento de sistemas [4].

O UP nasceu com o intuito de dar suporte à *Unified Modeling Language* (UML), já que a linguagem não dispunha da metodologia de processo necessária à aplicação da UML [41].

A proposta do RUP é ser um processo iterativo e incremental, ou seja, o sistema é construído em ciclos contínuos e evolutivos de desenvolvimento, permitindo assim uma compreensão crescente do produto a ser construído. Uma outra característica desse processo é o fato dele ser configurável, podendo ser ajustado e redimensionado de acordo com as necessidades do projeto que ele está inserido[33] .

Assim como no modelo prototipação, o RUP permite que os riscos sejam identificados e mitigados no início do processo de desenvolvimento, diminuindo assim as possibilidades de fracasso do projeto [8].

O processo é composto de 4 (quatro) fases [33]:

- Concepção: define uma visão do projeto, coletando os requisitos de alto nível, o escopo, os principais riscos e a estimativa de esforço e recursos necessários para a realização do projeto.
- Elaboração: os requisitos são compreendidos e amadurecidos. Os riscos identificados são tratados e as decisões arquiteturais do projeto são tomadas.

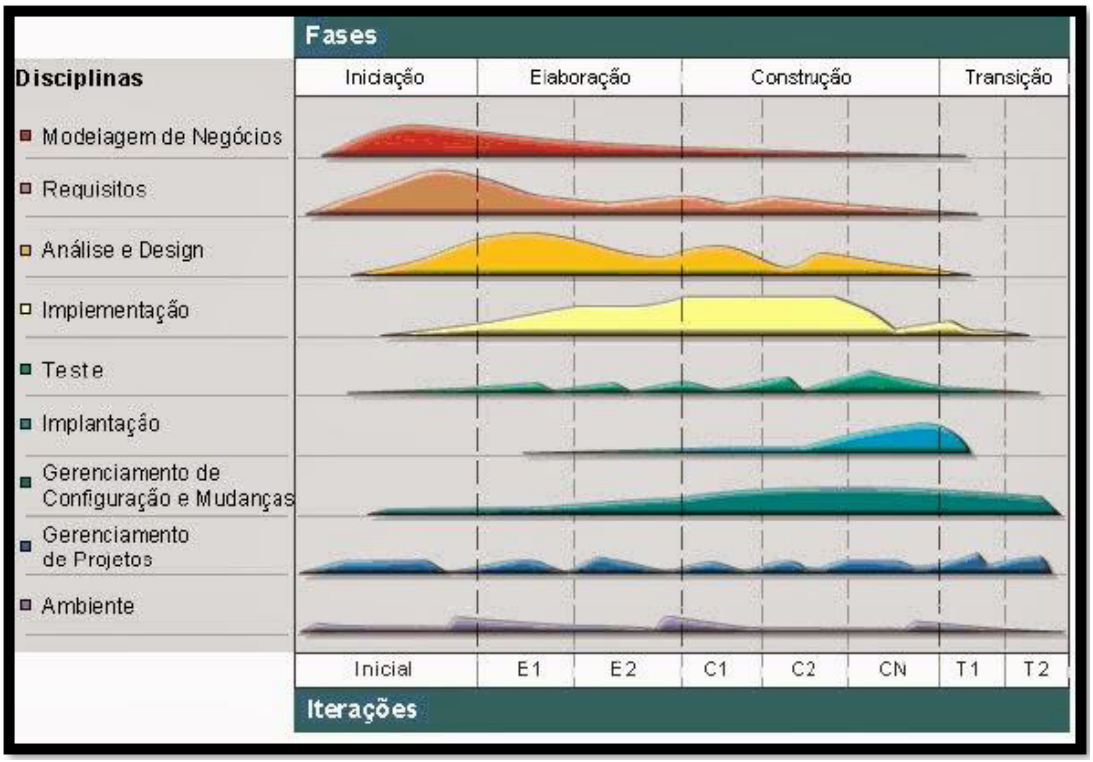

Figura 2.5: Detalhamento das fases do RUP. Extraída da página 7 de [38].

- Construção: a partir do que foi definido nas fases anteriores, o sistema é construído, testado e avaliado pelo cliente, de forma iterativa e incremental. Nessa fase poderão ser descritos os requisitos que não foram identificados na Elaboração, bem como critérios de aceitação do produto.
- Transição: nessa fase o produto será disponibilizado ao cliente e de acordo com o *feedback* recebido, poderá sofrer ajustes até que chegue à versão final. Espera-se que essa seja uma etapa curta, já que o produto foi avaliado na etapa de construção.

A Figura 2.5 apresenta as fases do RUP e o esforço despendido em cada uma delas. Cada fase é composta por uma ou mais iterações, que resultam em uma versão do produto que está em desenvolvimento. É possível observar a dinamicidade do tempo de cada fase no eixo horizontal, bem como as atividades inerentes a cada uma delas, através das disciplinas no eixo vertical [38][33].

O RUP é um modelo ainda bastante utilizado na indústria de *software*, porém, em muitos casos tem suas características perdidas durante a implementação. A proposta de ser iterativo e incremental muitas vezes não é seguida, transformando-se assim em um modelo sequencial, similar ao Cascata [22]. A implementação nesse formato faz com que o RUP promova entrega de versões apenas ao final do projeto, restrinja o *feedback* do

cliente, promova ociosidade na equipe, além de aumentar o tempo de desenvolvimento do produto.

2.3 Modelos de processo adaptativo

Os modelos de processo adaptativo têm como base experiências vivenciadas na prática efetiva do gerenciamento e do desenvolvimento de sistemas. São conhecidos como adaptativo por trazerem em sua essência a melhoria continua, ou seja, todo processo pode ser evoluído, melhorado e adaptado para melhor atender as partes envolvidas [4].

Diferentemente do prescritivo, o adaptativo não propõe um padrão rígido de tarefas e fases a serem seguidas. Normalmente baseiam-se em um conjunto de valores e/ou práticas que podem ser customizadas de acordo com a organização [41].

Como exemplo para esses modelos, podem-se destacar os métodos ágeis que serão descritos e exemplificados nas subseções seguintes, juntamente com o modelo de melhoria contínua.

2.3.1 O modelo de melhoria contínua

O modelo de melhoria contínua, também conhecido como PDCA (*Plan-Do-Check-Act*), foi proposto por Edwards Deming nos anos 50 [23] e tem como objetivo a melhoria de processos e produtos, através de ciclos de desenvolvimento sem intervalos ou interrupções.

A Figura 2.6 representa um ciclo de desenvolvimento PDCA, apresentado as 4 (quatro) fases de forma contínua e iterativa. Um ciclo do PDCA é composto por quatro etapas [38][4]:

- Planejar: estabelecer os objetivos, os caminhos e os métodos a serem seguidos.
- Executar: colocar o planejado em prática e coletar os dados dessa execução para serem verificados na etapa seguinte.
- Verificar: verificar se o planejado foi realmente executado e se o objetivo proposto no planejamento foi atingido. A análise dos dados visa também identificar as lições aprendidas durante a execução.
- Agir: são feitas as investigações e correções acerca dos problemas encontrados durante a verificação do processo. Em seguida, o ciclo é repetido, dando continuidade ao processo de melhoria contínua.

Figura 2.6: O modelo de melhoria contínua. Adaptado de [4][38].

2.3.2 O manifesto ágil

Em 2001 reuniram-se 17 (dezessete) representantes de metodologias não tradicionais de desenvolvimento de sistemas, na época conhecidas como "metodologias leves", e registraram o manifesto apresentado na Figura 2.7 [27].

Figura 2.7: O manifesto ágil. Extraída de [27].

Verifica-se nesse manifesto que os valores apresentados à esquerda devem ter maior prioridade em relação aos itens da direita. Assim, é reconhecida a importância de processos, ferramentas, vasta documentação, contratos e planos no desenvolvimento de sistemas, porém, deve-se ofertar maior valor às pessoas do projeto (clientes e equipe), à interação entre elas, ao sistema entregue e em funcionamento, à colaboração com o cliente e à capacidade de responder a mudanças.

Nessa reunião, foram estabelecidos ainda os seguintes princípios, que versam sobre satisfação do cliente, maximização do valor do produto, entregas frequentes, integração da equipe, motivação da equipe, comunicação direta, *software* em funcionamento, ritmo de trabalho constante e sustentável, foco na qualidade técnica e no *design* do produto, simplicidade, equipes auto organizáveis e retrospectivas com ênfase na melhoria contínua [27]:

1. "Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de *software* com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente *software* funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face à face.
7. *Software* funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom *design* aumentam a agilidade.
10. Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial.
11. As melhores arquiteturas, requisitos e *designs* emergem de equipes auto organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo."

2.3.3 *Scrum*

Scrum é um *framework* criado para gerenciar o desenvolvimento e manutenção de produtos complexos, como por exemplo, um *software*, de forma produtiva e criativa, e com

um alto grau de valor associado. A ideia por trás desse *framework* não é de especificar um processo, uma metodologia ou uma técnica, e sim de definir valores e práticas que podem ser empregados em conjunto com outros métodos [47]. O termo "*Scrum*" é originário do esporte *Rugby* em um contexto no qual todos os jogadores do time disputam uma reposição de bola. Para que o objetivo seja alcançado é necessário que todos os membros atuem em conjunto, e se um deles falhar, todos os outros do time falham. Baseado neste conceito de formação de equipes únicas e com um mesmo objetivo comum, Takeuchi e Nonaka em 1986 [54] associaram analogamente o *Scrum* ao desenvolvimento de produtos e publicaram o artigo "The New New Product Development Game" [54] abordando as seguintes características:

- Instabilidade embutida: a alta gerência cria um elemento de tensão na equipe do projeto proporcionando um elevado grau de liberdade para que eles conduzam um projeto estratégico para a empresa e com requisitos bastante desafiadores;
- Equipes de projetos auto-organizadas: as equipes passam a tomar iniciativas, mitigam riscos e desenvolvem a sua própria agenda. Características comuns destas equipes envolvem autonomia, autotranscendência que está relacionada a atingir metas e limites cada vez mais difíceis, e fertilização cruzada, que está associada a equipes compostas de membros com diferentes padrões comportamentais e especializações funcionais.
- Fases de desenvolvimento sobrepostas: projetos com fases sequenciais determinam que uma fase só se inicie após a conclusão de todos os requisitos da fase anterior. Problemas com este tipo de abordagem incluem desde a falta de integração entre as atividades do projeto, bem como o comprometimento do prazo do projeto nos casos de atrasos em uma das fases. Equipes auto-organizadas estabelecem uma dinâmica diferente e promovem a utilização de fases sobrepostas, na qual uma fase pode ser iniciada antes mesmo da conclusão da anterior. Normalmente os membros encontram-se em um nivelamento distinto, no que tange ao conhecimento do projeto e consequentemente no ritmo de produção, todavia, trabalham de forma a sincronizar este ritmo para atingir o prazo estabelecido.
- Múltiplo aprendizado: partindo da premissa de que os membros da equipe deverão manter um relacionamento estreito com as principais fontes de informação do projeto, é possível que eles possam responder rapidamente a mudanças e condições impostas pelo mercado. O termo múltiplo aprendizado surge da ideia de que o aprendizado da equipe se estabelece em duas dimensões: em vários níveis (individual, grupo e corporativo) e em várias funções. Este aprendizado é obtido através

de um processo contínuo de tentativa e erro, até que as alternativas possíveis se esgotem.

- Controle sutil: as equipes devem ser gerenciadas com controles menos rígidos, de forma a não comprometer a criatividade e espontaneidade de seus membros. Deve-se para tanto utilizar técnicas de controles através dos pares (próprios membros), do autocontrole, fornecendo maior autonomia ao profissional e do controle sutil ou controle "por amor", no qual a gerência conduz a equipe a uma determinada meta sem impor autoridade.
- Transferência de aprendizado organizacional: o aprendizado gerado em um determinado projeto deve ser compartilhado para todos os membros da organização, de forma a dar suporte e experiência para novos projetos. Esta transferência pode ser realizada através do rodízio de membros entre equipes e também com a tradução das lições aprendidas em práticas-padrões da organização.

Percebe-se que as características apresentadas focam no relacionamento interpessoal e o aprendizado e a transferência de conhecimento das equipes se convergem com o manifesto ágil no que diz respeito à valorização de pessoas e relações em detrimento a processos e ferramentas. Outro fator que pode ser destacado é a aquisição de conhecimento através da experiência. *Scrum* é fundamentado no empirismo, que define que o conhecimento deve ser adquirido enfatizando a descoberta de evidências a partir de experiências. Para apoio a esta teoria, o *framework* sustenta-se em três pilares: a transparência, que está relacionada a um vocabulário de entendimento comum entre os membros da equipe; a inspeção, de forma a verificar os artefatos produzidos e o andamento do projeto de acordo com o objetivo estabelecido; e por fim a adaptação, que promove a realização de ajustes necessários decorrentes de algum desvio encontrado na inspeção [47].

Papéis

O *Scrum* estabelece em sua estrutura três papéis distintos: o *product owner*, a equipe de desenvolvimento e o *scrum master*. Juntos, esses três papéis formam o time *Scrum* e são guiados pelas características de equipes auto-organizadas e multidisciplinares citadas anteriormente.

O *product owner* (PO) ou o dono do produto é o responsável por garantir que o trabalho executado pela equipe de desenvolvimento entregará o valor esperado pelo cliente. É também o responsável pelo gerenciamento do *product backlog*, que nada mais é que uma lista de requisitos do produto a ser desenvolvido. Este gerenciamento inclui atividades como definição clara dos requisitos, priorização dos mesmos, de acordo com as metas do cliente, e a ampla divulgação e transparência do *product backlog* para todo o time *Scrum*.

É facultado ao PO a delegação de suas atividades, porém, toda a responsabilidade de execução continua sob sua alçada.

Vale ressaltar que o papel de PO é desempenhado por apenas uma pessoa e não por um grupo, e é fundamental para o seu sucesso que toda a organização respeite as suas decisões, de forma que, qualquer nova necessidade a ser inserida no produto seja acordada com o mesmo. Assim, somente o PO poderá estabelecer, para a equipe de desenvolvimento, o que será feito e quando será feito.

A equipe de desenvolvimento é composta por desenvolvedores que irão realizar todas as iterações necessárias à construção do produto. Essas iterações são chamadas de *sprints*, geralmente duram de 2 a 4 semanas e na sua conclusão devem ser entregues funcionalidades de acordo com os requisitos priorizados a partir do *product backlog* selecionados e detalhados para aquela *sprint*. Sugere-se que a equipe de desenvolvimento tenha no mínimo três e no máximo nove pessoas. Poucas pessoas promovem baixa interatividade na equipe e muitas pessoas podem dificultar o trabalho de coordenação do grupo [47].

Uma característica relevante desta equipe é o fato de que ela não possui subgrupos para realização de um trabalho específico, como por exemplo, testes do sistema ou construção de componentes. Todo o grupo deve estar integrado em prol da finalização da *sprint* formando uma única equipe. Nesses termos, as características de equipes auto-organizadas (na qual a própria equipe define como executará o seu trabalho), e multifuncionais (a equipe deverá possuir todas as habilidades para realização do trabalho), se fazem bastante presente no grupo.

O *scrum master* é o responsável por garantir que tanto a equipe de desenvolvimento como o PO estão seguindo a teoria, as práticas e as regras presentes no *Scrum*. Uma das principais atribuições do *scrum master* é proteger o time *Scrum*, de forma a não permitir que eles se comprometam com mais itens do *backlog* do que possam entregar. Normalmente o papel do *scrum master* é desempenhado por um gerente de projetos, ou um líder técnico, porém, atuando como um facilitador, já que o time *Scrum* é auto-gerenciado [47].

É ainda atribuição do *scrum master* remover quaisquer obstáculos que impeçam o progresso da equipe de desenvolvimento, educar a equipe de desenvolvimento na criação de produtos de alto valor e nas características das equipes auto-organizadas e multidisciplinares, buscar técnicas para o gerenciamento do *backlog* e também liderar e treinar a organização na adoção do *Scrum* [47].

Eventos

O *Scrum* descreve em seu guia os eventos necessários para realização do trabalho de forma transparente e com duração máxima estipulada, criando assim uma rotina e

maximizando o tempo do time *Scrum*. Nestes eventos é possível inspecionar o produto que está sendo construído e adaptá-lo caso seja necessário.

Os eventos que compõe o *Scrum* são [47]:

- *Sprint*;
- Reunião de planejamento da *sprint*;
- Reunião diária;
- Revisão da *sprint*;
- Retrospectiva da *sprint*.

A *sprint* é uma iteração de um mês ou menos que contém além do trabalho de desenvolvimento, os eventos de planejamento, revisão e retrospectiva da *sprint*. Na *sprint* são definidos um conjunto de itens ou requisitos, extraídos do *backlog* e que devem estar finalizados ao final da iteração, atingindo assim uma meta proposta pelo PO. É desejável que na saída de cada *sprint* exista um incremento de produto pronto e também de valor agregado.

Após definido os requisitos para esta iteração, o *scrum master* deverá trabalhar de forma a garantir que não haverá mudanças que possam afetar o objetivo da iteração. Todavia caso as mudanças aconteçam e tenham interferência significativa na meta estabelecida, existe a possibilidade de cancelamento da *sprint*, no entanto é algo que deva ser executado somente em última instância.

A reunião de planejamento da *sprint* é feita em dois momentos: no primeiro momento o PO junto com a equipe de desenvolvimento escolhem os itens do *backlog* que serão executados na sprint e em seguida o PO define a meta a ser atingida. No segundo momento, a equipe de desenvolvimento transforma os itens em tarefas e planeja como irá atingir a meta proposta.

A reunião diária do *Scrum* é uma reunião de 15 (quinze) minutos na qual deve ser inspecionado o trabalho de desenvolvimento da *sprint*. Nesta reunião deve ser questionado o que a equipe de desenvolvimento já finalizou, o que será feito posteriormente e se há algum obstáculo que esteja impedindo o progresso da *sprint*.

Ao final de cada *sprint*, deve acontecer a reunião de revisão da *sprint*, na qual a equipe de desenvolvimento apresenta o resultado do trabalho executado. Nesta reunião, o PO definirá de acordo com o que foi apresentado se a equipe atingiu ou não a meta estabelecida.

A reunião de retrospectiva da *sprint* acontece após a reunião de revisão da *sprint* e tem como principal objetivo verificar se o desenvolvimento realizado na *sprint* está adequado ao processo de desenvolvimento e práticas da organização. A partir desta verificação, são

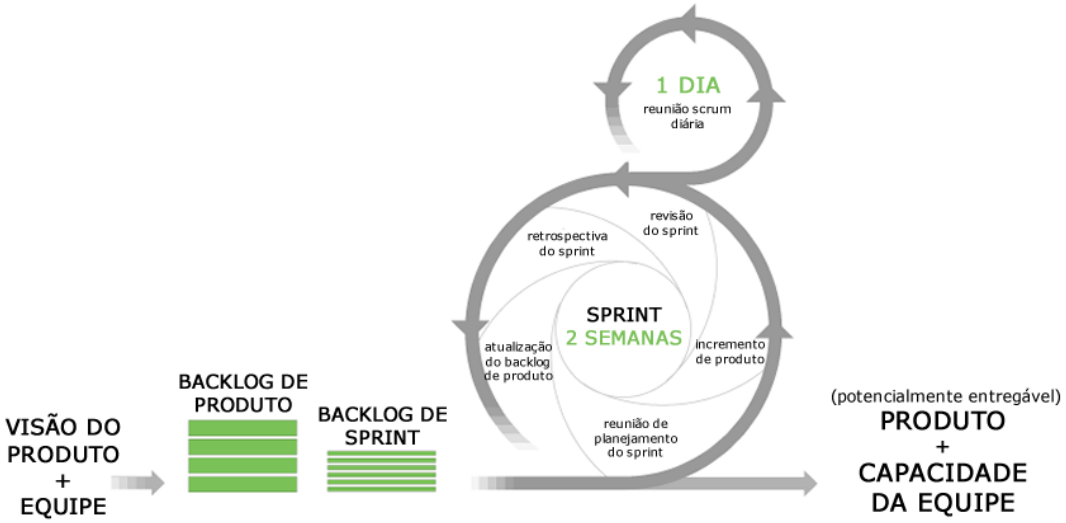

Figura 2.8: O ciclo de vida do *Scrum*. Extraída de [14].

propostas melhorias para que o processo seja seguido ou continue a ser seguido. Essas melhorias devem ser efetivadas preferencialmente na próxima *sprint*.

A Figura 2.8 apresenta o ciclo de desenvolvimento de produtos utilizando o *Scrum*. O ciclo começa com a definição do *product backlog* , que é construída junto ao PO. Esse *backlog* é dividido em partes menores e priorizadas pelo PO para a *sprint*, gerando o *sprint backlog*. Durante a *sprint*, os eventos do *Scrum* são executados e ao final espera-se que a equipe tenha produzido um incremento de produto entregável e com valor agregado.

Artefatos

O *Scrum* estabelece em seu guia os seguintes artefatos [47]:

- *Product backlog*: de responsabilidade do PO, é uma lista ordenada e priorizada que contém todas as características, funções, definições, melhorias e correções necessárias à conclusão do produto. O *backlog* é dinâmico e evolui de acordo com o produto e a organização.
- *Sprint backlog*: é composto dos itens selecionados do *product backlog* para a *sprint* e das informações de andamento da *sprint*, contendo a informação das tarefas relacionadas, o andamento destas tarefas e a estimativa de conclusão.
- *Sprint burndown*: é um gráfico que demonstra o andamento da *sprint*. É criado na reunião de planejamento e deve ser atualizado na reunião diária até o final da *sprint*. A intenção por trás desse gráfico é apresentar o quanto do trabalho já foi finalizado em comparação ao trabalho que foi previsto para a *sprint*. É comumente utilizado para medir os pontos de estórias concluídos, ao longo dos dias da *sprint*, definindo

assim o ritmo de trabalho da equipe[30]. Vale ressaltar que os pontos de estórias tem como base uma unidade de tamanho definida pelo time *Scrum*, que pode ser, por exemplo, a história de menor complexidade ou alguma outra medida definida pelo time.

- *Release burndown*: é um gráfico utilizado para acompanhar o progresso de uma entrega, que pode ser composta por uma ou mais *sprints* e deve ser atualizado ao final da execução da *sprint*.
- A definição de pronto: É o entendimento comum, para todo o time *Scrum*, de que uma tarefa ou trabalho foi concluído.

2.3.4 *Lean Software Development* (LSD)

O LSD é uma abordagem ágil para desenvolvimento de *software* baseada nos princípios do *Lean Manufacturing* ou sistema Toyota de produção [39] .

O sistema Toyota foi desenvolvido na manufatura, com o objetivo de eliminar o desperdício, aumentar a velocidade de produção e aprimorar a qualidade dos produtos desenvolvidos [42]. Em consequência dessas melhorias, é possível ainda obter um produto com menor custo associado [4].

A filosofia por trás desse sistema, segundo Poppendieck [40] é "dar aos clientes o que eles querem, no local e hora que eles quiserem, sem nenhum desperdício de tempo ou trabalho". Aplicando essa filosofia no desenvolvimento de *software*, Poppendieck determinou 7 (sete) princípios [40]:

1. Otimizar o todo: está relacionado com a compreensão de todo o trabalho que o cliente deseja e como essa solução poderá ser mediada via *software*. Deve-se considerar que a equipe e o processo de trabalho sejam seguidos em conjunto, de forma a otimizar toda a cadeia de valor e não apenas funções específicas.
2. Eliminar o desperdício: fundamenta-se na ideia de eliminar tudo aquilo que não gera valor ao produto, de acordo com a visão do cliente. Poppendiek [40] descreve alguns exemplos desse tipo de desperdício no desenvolvimento de sistemas, como código ou funcionalidade desnecessária, excesso de processos e documentação, trabalhos incompletos, antecipar funcionalidades que só serão utilizadas posteriormente, alternância de mais de duas tarefas ao mesmo tempo, espera pela homologação do cliente, defeitos e comunicação ineficiente.
3. Inclua qualidade no processo: associa-se à construção da qualidade em todo o processo de desenvolvimento e não apenas na finalização do produto. Nesse ponto

algumas técnicas poderão ser utilizadas, como a programação em par, integração contínua, refatoração e testes automatizados.

4. Aprender constantemente: para o LSD, desenvolvimento está relacionado à criação de conhecimento e à transformação desse conhecimento em produto. A experiência de outros projetos poderá fornecer subsídios para a escolha da melhor solução de um outro projeto. Assim como o *feedback* constante dos envolvidos poderá fazer com que a equipe aprenda continuamente sobre o produto.

5. Entregar rápido: fornecer ao cliente uma versão do produto pronta para utilização, o mais rápido possível. Iterações curtas promovem maior *feedback*, aumentam a satisfação do cliente e agregam maior conhecimento e experiência para a equipe.

6. Envolver todos: a participação de todas as pessoas que fazem parte da equipe do projeto é essencial para o desenvolvimento de um produto efetivo. Em alguns projetos de *software* cabe ao analista de sistemas projetar uma solução, enquanto o desenvolvedor apenas segue o que foi especificado pelo analista. Inserir o desenvolvedor, bem como outros papéis faz com que determinadas decisões sejam mais acertadas e que o trabalho realizado seja do time e não de uma pessoa específica.

7. Tornar-se cada vez melhores: esse princípio utiliza o conceito do modelo de melhoria contínua de Deming [23], já que as lições aprendidas na utilização do processo poderão ser incorporadas, aprimorando-o em cada ciclo de desenvolvimento.

Além desses princípios, o LSD recomenda a utilização de métricas e indicadores para auxiliar na incorporação de mudanças ao processo. Essas métricas poderão envolver desde o *feedback* do cliente à qualidade dos produtos desenvolvidos [4].

2.3.5 *Extreme Programming* (XP)

XP foi criado por Kent Beck [5] a partir da compilação de um conjunto de princípios e boas práticas de sucesso utilizadas na indústria do *software*. Apesar dessa compilação ter ocorrido no ano de 1996, os princípios e práticas que a compõem fizeram parte do dia a dia de trabalho de Kent Beck e Ward Cunningham [5] por mais de 10 (dez) anos, quando trabalharam em projetos utilizando *Smalltak* na empresa Tektronixs, Inc.

XP nasceu no contexto de um projeto crítico chamado C3 [6], com prazo e orçamento reduzido e que já havia sido executado por outras equipes, sem sucesso. Beck então aplicou a metodologia XP, obtendo sucesso com a entrega de um produto de excelente qualidade e construindo em menos tempo do que as tentativas que o antecederam.

O resultado desse trabalho foi publicado em 1999, no livro entitulado *"Extreme Programming Explained: Embrace Change"* [5], focando em equipes de tamanho reduzido

(até 12 pessoas). Em 2004, esse trabalho foi aperfeiçoado através da segunda edição do livro [6], dando maior flexibilidade ao XP e focando os valores da metodologia.

XP promete reduzir os riscos do projetos, melhorar a capacidade de resposta às mudanças, aumentar a produtividade ao longo da vida de um sistema e tornar o desenvolvimento de sistemas uma atividade prazerosa [4][6].

Nos tópicos a seguir serão detalhados o conjunto de valores, princípios e práticas que formam o XP.

Valores

O XP é pautado em 5 (cinco) valores, que servem como referência para implementação dos príncipios e práticas da metodologia [6]:

1. Comunicação: para Beck [6], um dos principais problemas que afetam o desenvolvimento de software é a dificuldade de comunicação. Assim, no XP, a comunicação é evidenciada através de atividades e práticas colaborativas, a exemplo do envolvimento real do cliente no desenvolvimento do produto e da programação pareada. XP enxerga todos os membros do projeto como uma única equipe, que trabalha em prol de um mesmo objetivo.

2. Simplicidade: assim como o LSD, XP encoraja o time a eliminar o desperdício. Isso inclui fazer rotinas desnecessárias ou que antecipem problemas que talvez nunca aconteçam. A simplicidade e a comunicação são valores que trabalham juntos, ou seja, quanto mais interação a equipe tiver, maior entendimento do problema ela terá e por consequência produzirão soluções mais simples.

3. *Feedback*: quanto mais cedo uma mudança for identificada, menos custosa ela será ao projeto. Assim, XP ressalta a necessidade de *feedback* constante em curtos ciclos de desenvolvimento. O *feedback* é observado na utilização de testes unitários, quando os clientes escrevem novas histórias, na priorização da construção de funcionalidades importantes, dentre outras atividades.

4. Coragem: é um valor essencial para que o time não seja resistente a mudanças e possa propor inovações. Isso não significa que os membros possam fazer o que bem entender, mas sim saber ousar, com base na experiência e com apoio mútuo. A coragem de forma isolada pode ser um valor bastante perigoso, mas aliada aos demais valores produz bons resultados.

5. Respeito: está relacionado ao reconhecimento das pessoas que compõem a equipe e à importância dessas pessoas para o sucesso do projeto. O respeito influencia diretamente a comunicação e o *feedback*, evita conflitos entre os membros e a tão

conhecida prática de "encontrar culpados". O trabalho em equipe, com respeito, gera oportunidades de aprendizado e colaboração mútua, considerando as limitações de cada um.

Princípios

Os princípios na programação extrema são vistos como a transformação dos valores, que são abstratos, em práticas concretas. Beck especifica 14 (quatorze) princípios [6][4]:

1. Humanidade: *software* é um produto desenvolvido por pessoas, assim, deve-se levar em conta o lado humano no desenvolvimento de sistemas, de forma a gerar oportunidades de crescimento, bom relacionamento e participação. A individualidade dos membros necessita ser considerada e equilibrada junto às necessidades da equipe e do negócio.

2. Economia: a equipe do projeto deve ter ciência do negócio ao qual estão inseridos, para que possam agregar valor ao *software* que está sendo construído. Esse princípio também está relacionado à priorização dos requisitos junto ao cliente, já que realizar as atividades mais importantes primeiro poderá gerar maior satisfação ao cliente, maximizando assim o valor do produto.

3. Benefício mútuo: as tarefas executadas em um projeto XP devem ser benéficas para todos os envolvidos, seja no presente ou no futuro. Produzir documentações extensas com foco na manutenção, por exemplo, é uma prática desencorajada, já que é algo que apenas será útil no futuro. Assim, XP propõe técnicas como testes automatizados, refatoração e utilização de metáforas que auxiliam o desenvolvedor durante o desenvolvimento do projeto, bem como facilitam as manutenções futuras.

4. Auto-semelhança: assim como em outras metodologias, a programação extrema enfatiza o ganho positivo em replicar boas soluções de desenvolvimento em projetos semelhantes. Manter uma base de conhecimento e lições aprendidas dos projetos facilita a aplicação desse princípio.

5. Melhoria: esse princípio ressalta a necessidade da equipe fazer o melhor sempre, de acordo com as condições do projeto naquele momento. A diferenciação dos conceitos de “bom” e “ótimo” é algo que os membros da equipe devem saber tratar, levando em conta a demanda constante por melhoria. Uma “boa” solução pode ser colocada em produção devido a urgência do projeto e posteriormente ser transformada em “ótima”.

6. Diversidade: a equipe deve reunir as habilidades necessárias à construção do projeto, de forma a se tornar multidisciplinar.

7. Reflexão: refletir sobre o trabalho realizado faz com que a equipe possa identificar os pontos de sucesso em um determinado projeto para que sejam replicados em outros, bem como buscar melhorar os pontos de falhas, buscando assim a melhoria contínua.

8. Fluxo: fazer pequenas entregas de valor agregado faz com que a equipe mantenha um fluxo de trabalho constante e aumente a satisfação do cliente. Além disso, minimiza os riscos do projeto, já que problemas com os requisitos podem ser identificados prematuramente.

9. Oportunidade: mudanças no projeto devem ser encaradas como oportunidades para melhoria e aprendizado.

10. Redundância: visa garantir a redução de defeitos e a produção de *software* de qualidade. Assim, muitas das práticas do XP são feitas de forma redundante, tais como a programação em par e os testes automatizados.

11. Falha: complementando os princípios da oportunidade e da redundância, o princípio da falha preconiza que toda falha deve ser vista como um aprendizado e que de posse de mais de uma solução para um determinado problema, a equipe deve executar todas, mesmo que alguma falhe. Dessa forma, XP incentiva o ganho de experiência e o aprendizado através de erros.

12. Qualidade: a qualidade é vista como um fator inegociável dentro de um projeto XP. A equipe deve buscar produzir sempre *software* de qualidade, aumentando a satisfação do cliente, a motivação e a produtividade da equipe

13. Passos pequenos: aliado ao princípio do fluxo, esse princípio propõe a implantação de novas versões de tamanho reduzido constantemente. Assim, o cliente poderá ofertar *feedback* constantemente e a equipe poderá verificar se está no caminho correto.

14. Aceitação da responsabilidade: as responsabilidades dentro do projeto devem ser aceitas e não impostas. Cada membro da equipe deve se sentir responsável pelo que faz, aumentando assim o seu comprometimento e envolvimento com a equipe com o projeto.

Práticas

Beck especificou um total de 12 (doze) práticas na primeira versão do livro [5], que têm como base os princípios e os valores da metodologia e podem ser vistas como as melhores técnicas para desenvolvimento de sistemas utilizadas pela indústria de software.

Para melhor compreensão, essas práticas foram divididas em 4 (quatro) categorias: práticas de codificação, práticas da equipe, práticas do processo e práticas do produto [35].

1. Práticas de codificação

    - Programação em pares: os desenvolvedores da equipe devem implementar soluções em conjunto com outro membro. O objetivo dessa técnica é promover maior colaboração, melhorar a comunicação, diminuir a incidência de erros, além da troca de experiências [5][4]. Essa prática requer que os pares sejam trocados regularmente, para que a interação possa ocorrer entre todos os membros da equipe.
    - Testes: construção de testes automatizados para validação das funcionalidades do sistema durante todo o ciclo de desenvolvimento. Esses testes devem ser escritos por desenvolvedores e clientes, ou por uma dupla composta por cada um deles [5][4].
    - Refatorações: visa realizar alterações estruturais no código, porém, de forma transparente ao cliente. Assim, o comportamento externo do sistema não é modificado. A refatoração tem como objetivo a melhoria do código, tornando-o mais limpo e preparado para abarcar novas funcionalidades.

2. Práticas da equipe

    - Padronização do código: a equipe deve definir padrões para construção do código-fonte do projeto, de forma a promover um entendimento comum entre os membros, melhorar a comunicação e facilitar a prática da propriedade coletiva do código [5].
    - Integração contínua: o código-fonte da aplicação deve ficar armazenado em um repositório único, ao qual todos os desenvolvedores possuem acesso. Quando uma tarefa é concluída, o par responsável deverá realizar a integração do novo código, gerando uma nova versão do projeto e executando os testes automatizados [5]. Assim, toda a equipe tem acesso às novas versões do sistema automaticamente e os riscos decorrentes de integrações complexas são minimizados, já que são feitas pequenas integrações frequentes a cada tarefa finalizada [4].
    - Propriedade coletiva do código: o código-fonte do projeto é propriedade de toda a equipe e não apenas de quem executou determinada tarefa [5]. Essa prática dissemina o conhecimento do projeto entre todos os membros e permite que todos os envolvidos possam executar modificações e melhorias em todo o código [4].

3. Práticas do processo

    - Metáforas: a utilização de metáforas proporciona à equipe um entendimento comum do sistema, através do comparativo de abstrações do projeto em construção com objetos do mundo real.
    - *Design* simples: o objetivo dessa prática é fazer com que a equipe trabalhe em prol das necessidades atuais do projeto [5]. Assim como no LSD, a intenção é evitar o desperdício, como por exemplo, antecipar requisitos que poderão não ocorrer ou construir estruturas demasiadamente complexas para as demandas do projeto.
    - Entregas curtas: desenvolvimento incremental e iterativo com ciclos de no máximo 3 (três) semanas. Entregas frequentes promovem maior *feedback* e minimiza os riscos e custos de mudança ao final do projeto [5].
    - Ritmo sustentável: também conhecida como "40 horas por semana", essa prática prevê que todo o trabalho deve ser realizado dentro da carga horária exercida pelo membro do projeto, sem a necessidade de horas extras. Assim, Beck enfatiza que o trabalho não deve interferir na vida pessoal dos envolvidos e que a sobrecarga do mesmo diminui a criatividade e interfere na qualidade do código [5].

4. Práticas do produto

    - Jogo do planejamento: o planejamento na programação extrema é feito através de estórias que detalham os requisitos do sistema. Essas estórias são escritas pelos clientes e a cada iteração devem ser priorizadas de acordo com o valor agregado ao projeto e com a estimativa de tempo para implementação, definida pela equipe [5].
    - Cliente presente: ter o cliente presente junto à equipe do projeto melhora a comunicação, maximiza o tempo de desenvolvimento e facilita o entendimento das regras de negócio junto aos desenvolvedores [5].

A Figura 2.9 apresenta as práticas do XP em uma visão de círculos. O primeiro círculo detalha as práticas de codificação, o segundo as da equipe, o terceiro as do processo e por fim, o quarto círculo apresenta as práticas relacionadas ao produto.

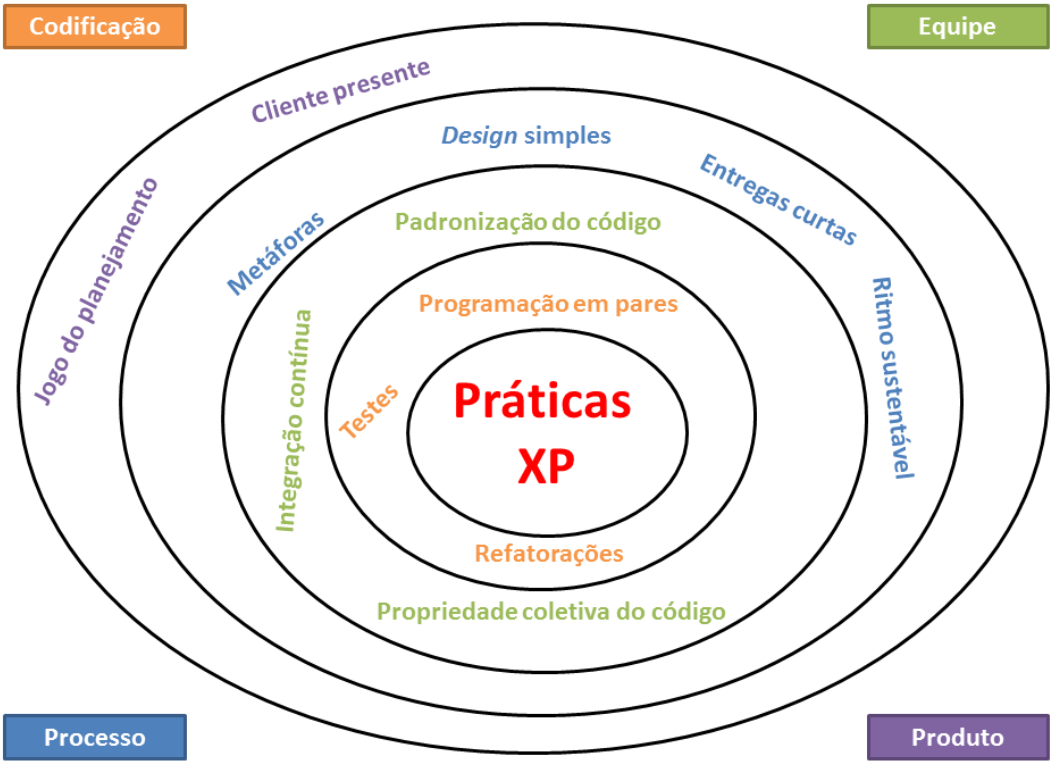

Figura 2.9: Os 4 círculos da programação extrema. Adaptada de [35].

2.4 Temas em Agilidade

O objetivo dessa seção é estudar alguns temas em agilidade que estão em discussão atualmente na comunidade ágil. Para isso, foram escolhidos dois temas: *Agile Modeling* e Dívida Técnica.

2.4.1 *Agile Modeling*

Agile Modeling (AM) é uma metodologia baseada em melhores práticas para modelagem efetiva de sistemas e guiada por um conjuntos de princípios e valores que podem ser aplicados na modelagem de sistemas [2]. O autor desse método afirma que:

> "AM não é um processo prescritivo, ela não define procedimentos detalhados de como criar um dado tipo de modelo, ao invés ela provê conselhos de como ser efetivo como modelado"

.

AM utiliza modelos, diagramas e desenhos como uma proposta colaborativa e centrada no projeto [42] e tem como missão [2]:

- definir e apresentar um conjunto de valores, princípios e práticas pertinentes à modelagem efetiva e leve;
- explorar como aplicar técnicas de modelagem ágil em métodos ágeis como *Scrum* e *XP*;
- explorar como também aplicar essas técnicas em processos prescritivos como RUP.

AM assina o manifesto ágil, e além dos valores do manifesto, apresenta 5 (cinco) valores próprios [2]:

- Comunicação;
- Coragem;
- *Feedback*;
- Simplicidade.
- Humildade;

Pode ser percebido que os 4 (quatro) primeiros valores apresentados são iguais aos valores do XP. Assim, apenas o valor "Humildade" é definido na AM.

2.4.2 Princípios e Práticas da AM

Os principais princípios da AM são [42][2]:

- Modele com um propósito: a essência desse princípio é direcionar a equipe para criar somente artefatos e modelos que de fato são necessários ao projeto. Isso se deve ao fato de que, em muitos projetos, as equipe produzem muita documentação desnecessária, por conta de um processo previamente estabelecido.
- *"Travel light"*: esse princípio incentiva a manter apenas o detalhamento necessário à compreensão do que se está querendo descrever. Assim, estimula-se a criação de poucos artefatos no projeto e com apenas informações estritamente essenciais presente.
- Modelos múltiplos: cada projeto possui suas especificidades e por isso, nem sempre precisam utilizar um mesmo conjunto de artefatos. Dessa forma, a AM preconiza que sejam utilizados múltiplos modelos, de acordo com o contexto que será aplicado.

O conjunto de práticas da *AM* visa usar a modelagem como ferramenta de análise e comunicação, fazendo com que os desenvolvedores modelem com um objetivo definido. Assim, as principais práticas descritas pelo AM são [42][2]:

- Modele em conjunto: através dessa prática AM encoraja a equipe de desenvolvimento a modelar em conjunto com o usuário, de forma a obter um melhor *feedback* do que se está modelando.
- Prove com código: a criação de modelos deve visar somente uma abstração de alto nível do que será implementado. Assim, provar com código significa dizer que o que foi modelado é passível de implementação. Essa prática evita criar especificações que poderão ter restrições técnicas para serem construídas.
- Use ferramentas simples: modelagem não significa utilizar ferramentas computacionais para gerar modelos. Assim, AM encoraja a utilização de ferramentas manuais como cartolinas, *flipcharts* e fichas pautadas.
- Modelos são públicos/propriedade coletiva: assim como o XP incentiva a propriedade coletiva do código, AM descreve como prática que todos os modelos sejam públicos,ou seja, todo artefato deve estar disponível para toda a equipe.

2.4.3 Dívida Técnica

Dívida Técnica é um termo criado por Cunningham [48] que faz alusão a uma dívida financeira. Segundo o autor do termo:

> "... A primeira vez que a qualidade do código é comprometida é como se estivesse incorrendo em Dívida Técnica. Uma pequena Dívida Técnica acelera o desenvolvimento até que seja paga através da reescrita do código. O perigo ocorre quando a Dívida Técnica não é paga. Cada minuto em que o código é mantido em inconformidade, juros são acrescidos na forma de reimplementação..."[48].

Seaman e Guo [48] definem dívida técnica como "uma metáfora para artefatos imaturos, incompletos ou inadequados no ciclo de vida de desenvolvimento de *software*". Assim, podemos exemplificar dívida técnica com um código que teve sua qualidade comprometida devido a outras variáveis, como tempo e recursos. Um outro exemplo é quando um órgão utiliza um *framework* próprio para o desenvolvimento de seus sistemas. Alguns projetos podem ter especificidades que para serem atendidas exigem adaptações ou a não utilização do padrão, fazendo assim com que o projeto tenha o seu código comprometido.

Adquirir uma dívida técnica pode promover benefício a curto prazo, como por exemplo, redução de tempo de desenvolvimento ou esforço. Porém, pensando a longo prazo, pode comprometer a qualidade e gerar impactos, como a dificuldade de manutenção do código produzido. Assim, o conceito de dívida técnica têm sido estudado no âmbito de tomada de decisão e da necessidade de gerenciamento das dívidas técnicas dos projetos. A equipe deve decidir se adquire ou não a dívida e se adquirir, quando pagar.

Conceitualmente, dívida técnica só pode incidir sob projetos que estão sendo implementados ou em produção [34]. Projetos que são cancelados devem ter as suas dívidas anuladas. Dívidas de projetos comuns a outros projetos, deverão ser atreladas a todas as equipes envolvidas [34].

Uma dívida técnica pode ser classificada em dois tipos [34]:

1. Dívida técnica sem intenção: esse tipo de dívida acontece sem que a equipe perceba. Podemos exemplificar essa dívida com um *software* que parou de funcionar ou teve seu funcionamento comprometido devido a mudança da versão do navegador.
2. Dívida técnica intencional: ocorre quando a equipe decide, conscientemente, adquirir a dívida técnica. Esse tipo de dívida é subdivida em mais (2) categorias:
    - Dívida técnica a curto prazo: é uma dívida que pode ser pagas em um curto espaço de tempo, seja pela baixa complexidade, ou pela disponibilidade de recursos para a sua quitação.
    - Dívida técnica a longo prazo: É uma dívida que não tem previsão de ser quitada rapidamente. Normalmente são dívidas que possuem alta complexidade e podem vir a causar um grande impacto no projeto.

A utilização da metáfora de dívida técnica tem contribuído para melhoria da qualidade de código em projetos de sistema, já que ao adquirir uma dívida, a equipe se compromete a "pagá-la", minimizando a incidência de códigos ruins [34]. No entanto, o grande desafio é o gerenciamento e a priorização dessas dívidas em projetos [55]. Além disso, ainda existem poucos estudos de técnicas para identificação de dívidas sem intenção e também para apoiar a tomada de decisão de adquirir ou não uma dívida [55].

2.5 Trabalhos correlatos

Alguns estudos relacionados à implantação de processos baseados em métodos ágeis foram realizados em Instituições Federais de Ensino (IFES) do país [56][4][38][51][46]. Pode ser observado que parte desses estudos tiveram foco em projetos acadêmicos, cuja principal mão-de-obra são alunos oriundos de cursos de graduação e pós-graduação dessas instituições. Esses estudos apontam ainda que a implantação de um processo ágil proporcionou melhoria na produtividade da equipe, na qualidade do produto, bem como promoveu a satisfação do cliente e a motivação da equipe.

Muzetti relata em seu trabalho [38] a implantação de um processo ágil, denominado BOPE, que mescla a utilização de eventos e artefatos do *Scrum* e práticas XP (testes automatizados, jogo do planejamento e entregas curtas) com algumas orientaçãoes do

Project Management Body of Knowledge (PMBoK). O objetivo desse estudo foi avaliar que na utilização de estudantes de graduação para o desenvolvimento de *software* se faz necessária a customização de um processo à essa realidade. O estudo foi realizado em laboratórios de pesquisa e inovação em computação da Universidade Federal de Ouro Preto (UFOP), de forma a atingir melhoria na produtividade desses laboratórios e na qualidade de seus produtos.

O estudo realizado por Torres Filho [56] teve como objetivo mostrar que a utilização de métodos ágeis no desenvolvimento de Ambientes Virtuais de Aprendizagens (AVA) podem aumentar a qualidade e a usabilidade desses ambientes. Além disso, o autor se propôs a demonstrar que o comprometimento e a felicidade da equipe são fundamentais para a maximização dos resultados da atividade de desenvolvimento. Para a execução do estudo de caso foi utilizado o *Scrum* e algumas práticas do XP (testes automatizados, integração contínua e programação em par) e do *Kanban*. A implantação de métodos ágeis dessa pesquisa foi realizada no projeto SOLAR 2.0, que é o AVA da Universidade Federal do Ceará (UFC) e ao final da pesquisa é relatado que a utilização desses métodos fez com que a equipe se tornasse comprometida, mantivesse um ritmo sustentável de desenvolvimento e reduzisse a dependência de especialistas externos à equipe.

2.6 Experiências na Administração Pública Federal

A utilização de métodos ágeis no contexto da Administração Pública Federal (APF) vem ganhando espaço nos últimos anos [22][37][52][1]. Através da recente publicação do Acórdão TC 010.663/2013-4 [22] que versou sobre o conhecimento acerca da utilização de métodos ágeis nas contratações para desenvolvimento de *software*, verificou-se que é possível alinhar a utilização desses métodos aos preceitos legais que regem a APF. Nesse mesmo Acórdão são relatadas diversas experiências de órgãos da APF na contratação de desenvolvimento de sistemas utilizando esses métodos, bem como os riscos envolvidos nessa contratação.

A experiência da Agência Nacional de Cinema (ANCINE) foi descrita no trabalho "Adaptação na Prática de um Setor Público às Metodologias Ágeis" [52]. O estudo de caso foi desenvolvido a fim de verificar e provar a aplicabilidade dos métodos ágeis no contexto da organização e teve como foco a utilização do *Scrum* em um projeto cuja equipe de desenvolvimento era externa ao órgão. Os autores citam que o resultado dessa implantação foi bastante positivo seja para a equipe técnica, como para os patrocinadores dos sistemas desenvolvidos. Ressalta-se ainda que após a finalização do estudo, foi proposta a continuidade do processo ágil em novos projetos.

Após anos de experiência com métodos tradicionais de desenvolvimento, o Banco Central realizou a adoção de métodos ágeis através de 2 (dois) projetos piloto utilizando todas as práticas do XP e *Scrum* [37]. O objetivo do estudo foi identificar se os métodos ágeis, no contexto da organização, aceleram o aprendizado de novas tecnologias, conceitos e padrões e se aumentam a qualidade do código do sistema, a produtividade do time e a satisfação do cliente. A implantação durou 18 (dezoito) meses e ao final dos projetos foi considerado que os métodos ágeis podem acelerar o aprendizado de novas tecnologias e alavancar a satisfação dos clientes, além de um discreto aumento na qualidade do código e na produtividade dos times estudados.

Capítulo 3

Estudo de Caso

A utilização de abordagens qualitativas em engenharia de software vem ganhando espaço na academia, principalmente pelo fato de muitos modelos de processo e métodos em geral possuírem influência de fatores humanos [24][49][44]. Um exemplo são os métodos ágeis, objeto de estudo desse trabalho, que priorizam as pessoas e suas interações na atividade de desenvolvimento de sistemas [27].

Uma abordagem qualitativa foca na identificação de características, situações, eventos, organizações e pessoas. O foco dessa abordagem é estudar as complexidades do comportamento humano, como por exemplo, motivação, comunicação e compreensão [49]. O ambiente de aplicação da pesquisa é a fonte de dados para o estudo, sendo o seu caráter essencialmente descritivo [49].

Esse tipo de abordagem tem o pesquisador como peça chave, pois os dados coletados por esses estudos estão sob o ponto de vista do pesquisador. Como relatado no trabalho [3], em um estudo qualitativo, o pesquisador tem o papel fundamental na aquisição dos dados, na preparação desses dados para avaliação e na interpretação final do resultado obtido. Aniche [3] também cita que o pesquisador é o responsável por identificar valores pessoais, pressuposições e vieses do estudo.

Pesquisas exploratórias segundo [50] "são todos aquelas que buscam descobrir ideias e soluções, na tentativa de adquirir maior familiaridade com fenômeno de estudo", ou seja, o objetivo nessa pesquisa é tornar o problema de estudo mais explícito ou construir hipóteses ao seu respeito.

Assim sendo, essa pesquisa é de caráter exploratório e usará abordagens qualitativas na coleta dos dados pesquisados. A técnica de estudo de caso será utilizada por ser uma técnica bastante utilizada em pesquisas qualitativas e capaz de reunir informações detalhadas a respeito de determinado problema [44]. Além disso, ao final do estudo será aplicado um questionário, como forma de obter a percepção do grupo envolvido na pesquisa. A análise desse questionário será feita através do método de pesquisa qualitativa

Grounded Theory [11], uma metodologia sistemática das ciências sociais que envolve a construção de uma teoria a partir de uma análise fundamentada nos dados obtidos na pesquisa.

Esse estudo não tem a intenção de proporcionar uma pesquisa abrangente sobre o tema "implantação de métodos ágeis". Pelo contrário, o estudo será desenvolvido apenas no Centro de Informática da Universidade de Brasília, o que sugere que os resultados aqui encontrados não podem ser generalizados.

3.1 Questão de pesquisa

O objetivo principal desta pesquisa é realizar um estudo sobre a adoção de métodos ágeis no CPD/UnB, verificando assim os impacto da implantação desses métodos na área de desenvolvimento de sistemas do Centro. Para atingir esse objetivo, tentaremos responder a seguinte questão de pesquisa, sob a percepção dos envolvidos na adoção:

- Quais os impactos da utilização de uma abordagem ágil no Centro de Informática da Universidade de Brasília?

3.2 Projeto da pesquisa

Os passos estabelecidos para o estudo seguiram a seguinte especificação:

1. Execução de projetos piloto: detalhamento através de pesquisa documental, entrevistas com os colaboradores do Centro e observação. Para cada projeto piloto, definiu-se como estrutura básica para a sua apresentação, a descrição de informações a respeito da preparação para o projeto, a descrição do projeto, a execução do projeto e as lições aprendidas. Uma versão preliminar da abordagem ágil é apresentada nessa execução. A riqueza de detalhes na descrição desses projetos contribui para a qualidade do estudo, além de possibilitar a repetição por outros pesquisadores [3].

2. Abordagem Ágil: Após a execução dos projetos piloto, será apresentado o modelo final da abordagem ágil aplicada. Essa versão é uma versão validada pela execução da abordagem nos dois projetos piloto e conta com as lições aprendidas durante esses projetos.

3. Aplicação de questionário: Ao final da execução dos projetos piloto é aplicado um questionário composto de perguntas abertas e fechadas (com um campo de justificativa aberta).

4. Análise descritiva do questionário: Nessa fase é realizada uma descrição das respostas obtidas nesse questionário. É uma primeira leitura dos dados coletados, que servirá de insumo para a aplicação da *Grounded Theory*.

5. Análise qualitativa do questionário: Para análise das respostas do questionário, será utilizado o método qualitativo *Grounded Theory*. O detalhamento do método e a aplicação do protocolo encontra-se na subseção 4.3.

3.3 Ambiente da Pesquisa: O Centro de Informática da Universidade de Brasília

O Centro de Informática (CPD) trata-se de um órgão complementar da Universidade de Brasília (UnB), subordinado ao Decanato de Planejamento e Orçamento (DPO), responsável pela tecnologia da informação da Universidade.

Dentre os objetivos principais do órgão encontra-se a atribuição de desenvolver, implantar e manter sistemas em mainframe e em microcomputadores. Além disso, é objetivo estratégico do CPD/UnB promover atualização tecnológica dos sistemas e da infraestrutura de TI da UnB.

O CPD tem como missão viabilizar soluções de tecnologia da informação que promovam a disponibilidade, integridade, confiabilidade e autenticidade das informações dos ativos relacionados aos sistemas informatizados da Universidade de Brasília. O princípio norteador da organização é a visão, que atualmente está descrita como "Ser referência nacional, como Centro de Informática para as Universidades Federais Brasileiras e para o Ministério da Educação, reconhecido como um Centro sólido pela excelência dos produtos e serviços tecnológicos oferecidos".

3.3.1 Estrutura organizacional

A Figura 3.1 apresenta a estrutura organizacional do CPD/UnB, que é composta de 1 (uma) direção geral e 6(seis) gerências, sendo 2 (duas) delas compostas de subáreas. Para fins da realização dessa pesquisa, iremos focar nas gerências relacionadas ao desenvolvimento de sistemas do Centro.

- Gerência de Estratégia de Dados: Compete proporcionar a disponibilidade, confiabilidade, integridade e guarda dos bancos de dados sob custódia do Centro.

- Gerência de Desenvolvimento de Sistemas: Compete atender as demandas para desenvolvimento de sistemas com base em normas e padrões estabelecidos no Governo Federal.

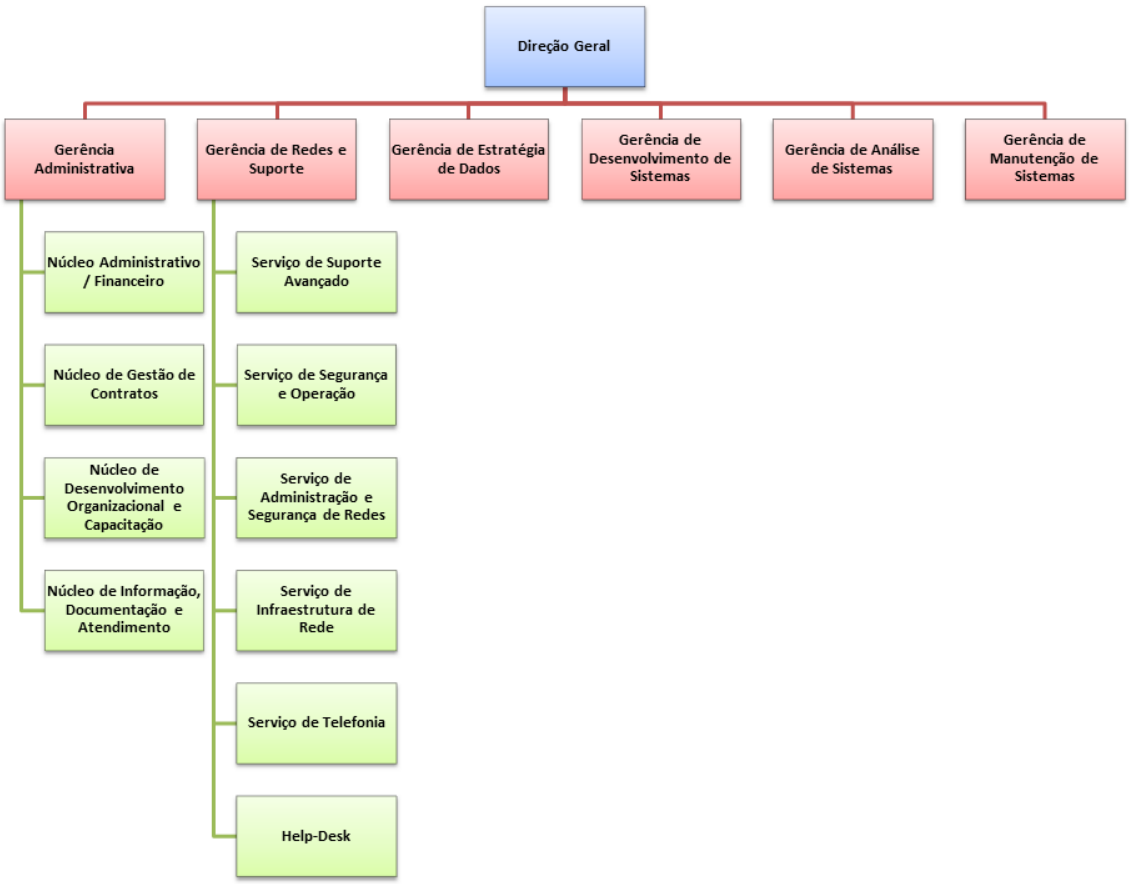

Figura 3.1: Estrutura organizacional do Centro de Informática da Universidade de Brasília.

- Gerência de Análise de Sistemas: Compete receber e analisar demandas de soluções de software de caráter institucionais enquadradas como adaptativa, perfectiva, evolutiva ou projetos novos.
- Gerência de Manutenção de Sistemas: Compete proceder a manutenção corretiva, adaptativa ou evolutiva nos sistemas corporativos em produção.

3.3.2 Estudo preliminar do desenvolvimento de *software* no CPD/UnB

Conforme proposta dessa pesquisa, realizaremos nessa seção o estudo preliminar sobre desenvolvimento de *software* no CPD/UnB, incluindo o histórico da área de desenvolvimento e suas respectivas gerências e as tecnologias e processos utilizados atualmente.

Histórico

De acordo com os especialistas que atuam no Centro de Informática, a maior parte dos sistemas corporativos atualmente em produção foram desenvolvidos na década de 90. O advento do *bug* do milênio, problema previsto para ocorrer na virada do ano de 1999 para o ano 2000, fez com que os sistemas fossem convertidos da plataforma COBOL/XGEN para Visual Basic ao final daquela década.

A estratégia utilizada nessa conversão envolvia a necessidade de um rápido desenvolvimento com a disponibilidade de poucos desenvolvedores. Assim, muitos dos projetos tornaram-se personificados, de forma a existir no Centro um determinado desenvolvedor associado a um sistema.

Finalizada essa fase de conversão, o Centro passou muitos anos fazendo apenas manutenções no que foi construído nessa época, sem novos projetos ou mudança de plataforma, seja pela equipe reduzida ou pela grande necessidade de manutenção nos sistemas vigentes.

Em 2008, após a posse de um novo diretor, foi contratada uma consultoria para definição de um processo para desenvolvimento de sistemas no CPD/UnB. A proposta desse processo era disseminar o conhecimento dos projetos desenvolvidos, de forma a minimizar os problemas experimentados nos projetos personificados: a dependência do desenvolvedor em um projeto específico, a falta de motivação do desenvolvedor por estar há bastante tempo trabalhando em um mesmo projeto, sem novos desafios e a perda do conhecimento institucional, já que boa parte do conhecimento encontrava-se armazenado nos indivíduos do Centro. Além disso, o processo pretendia aumentar a agilidade na entrega das demandas, através da organização e padronização das atividades de desenvolvimento de sistemas.

Como fruto dessa consultoria foram definidas as atuais gerências de sistemas e dados, além dos processos de negócios de desenvolvimento de sistemas, testes, estratégia de dados e gerência de projetos, detalhados nas seções a seguir.

Processo de Desenvolvimento de Software (PDS)

O PDS foi customizado a partir do RUP e define integralmente as fases constantes na metodologia, conforme especificado abaixo:

- Concepção: tem como objetivo definir o escopo inicial do projeto de forma a permitir o planejamento macro para iniciar a realização das atividades.
- Elaboração: tem como objetivo, a partir de várias iterações, analisar e projetar o(s) produto(s).

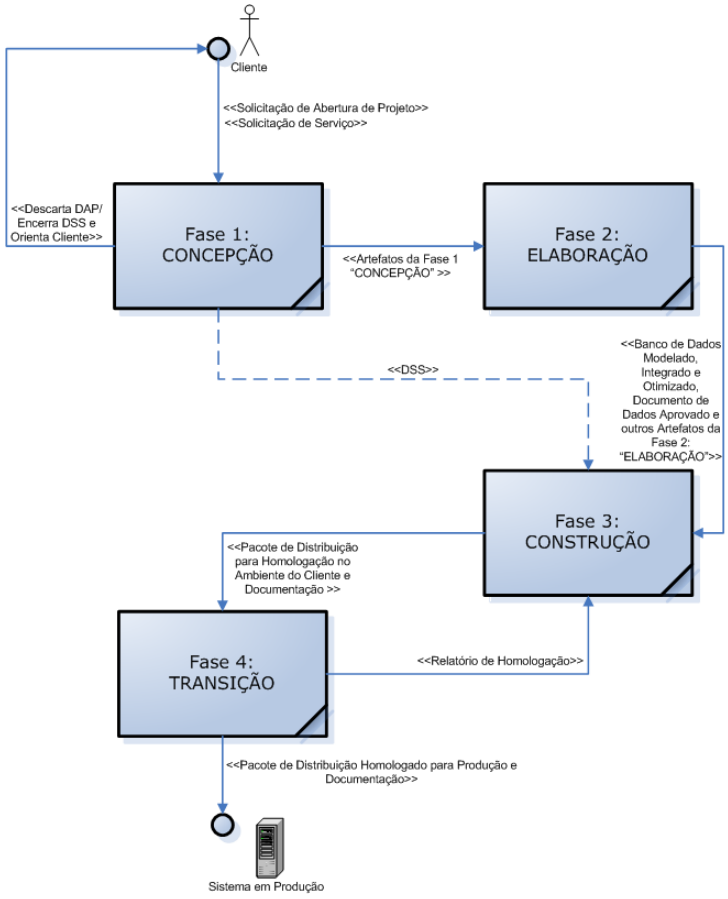

Figura 3.2: O PDS do CPD/UnB. Extraída de [19].

- Construção: tem como objetivo, a partir de várias iterações, construir e testar o(s) produto(s). A partir desta perspectiva, o sistema terá *releases* constantes, ou seja, versões ou incrementos do produto entregues ao cliente.
- Transição: tem como objetivo implantar o sistema no ambiente do cliente para avaliar de forma a obter o aceite do sistema.

A Figura 3.2 apresenta o PDS do CPD/UnB. O desenvolvimento de um sistema inicia-se com uma solicitação de abertura de projeto, que é realizada através do preenchimento do Documento de Abertura do Projeto (DAP). No caso de manutenções é preenchido o Documento de Solicitação de Serviço (DSS). Durante a fase de concepção é realizado o estudo de viabilidade do projeto e o entendimento do problema, através de uma lista preliminar de requisitos e casos de uso.

Na fase de elaboração a solução para o projeto é definida e especificada. Os casos de uso são detalhados, o banco de dados é modelado e a técnica de prototipação é utilizada com o objetivo de validar as prováveis telas para a fase de construção.

O produto ou incremento de software é programado na fase de construção, que envolve ainda as tarefas de integração e testes do produto.

Por fim, a fase de transição engloba a homologação pelo cliente e a disponibilização do que foi construído em produção.

Processo de Testes de Software (PTS)

O PTS instituído foca apenas na abordagem de testes funcionais, alinhados às etapas estabelecidas no PDS. Essa abordagem concentra-se em todos os requisitos de testes que possam ser diretamente associados a casos de uso ou funcionalidades e regras de negócios. A meta do teste funcional é verificar a adequada aceitação, o processamento e a recuperação dos dados, e a implementação apropriada das regras de negócios. Esse tipo de teste baseia-se em técnicas de caixa preta, ou seja, verificar o projeto e seus processos internos interagindo com o aplicativo através da Interface Gráfica do Usuário (GUI) e analisar a saída ou os resultados [53][41]. Na Figura 3.3 pode ser observado o fluxo construído para o PTS. Apesar do processo permear todas as fases do RUP, na concepção apenas são apresentados os manuais de padronização de testes. A realização do processo em si, de acordo com o caráter dos testes funcionais, somente ocorre nas etapas de construção e na transição, durante a homologação pelo cliente.

Processo de Estratégia de Dados (PED)

O PED foi elaborado alinhado ao PDS e dessa forma adequado às fases propostas nesse processo. As tarefas do PED envolvem o entendimento da demanda na fase de concepção, análise, integração e criação de modelos de dados e *scripts* na fase de construção e por fim a homologação da solução na fase de transição.

A Figura 3.4 demonstra o PED, através das fases do PDS. Apesar de envolver a fase de elaboração no fluxo, o PED não especifica atividades nessa fase.

Avaliação dos processos

Apesar da proposta de um desenvolvimento iterativo e incremental, com papéis e áreas bem definidos, os processos em questão não conseguiram atingir esse objetivo. Após 5 (cinco) anos de implantação o Centro conseguiu desenvolver 4 (quatro) projetos: o Sistema do Restaurante Universitário (SISRU), o Sistema de Ouvidoria (SISOUV), o Sistema de Tabelas (SITAB) e o Sistema de Extensão (SIEX), sendo os dois últimos conversão de sistemas construídos na plataforma anterior.

Os registros da ferramenta de acompanhamento [20] desses projetos apontam que o processo acabou seguindo o fluxo do modelo cascata, já que todo o sistema era especificado

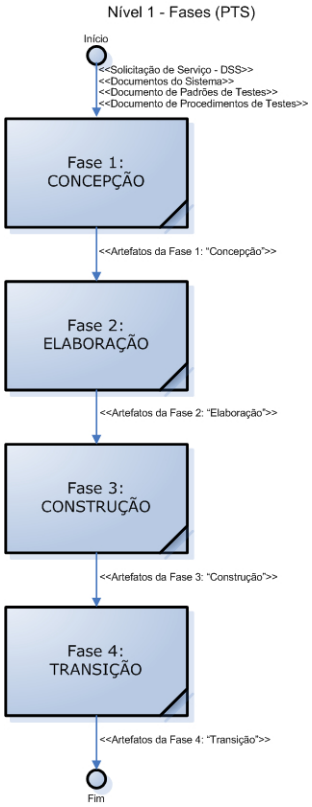

Figura 3.3: O PTS do CPD/UnB. Extraída de [17].

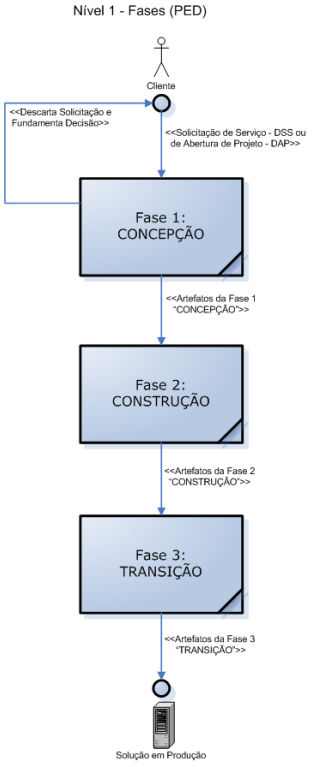

Figura 3.4: O PED do CPD/UnB. Extraída de [18].

na gerência de análise de sistemas, para somente após toda a especificação chegar à gerência de desenvolvimento de sistemas. É possível também observar através desses registros que alguns sistemas demoraram em torno de 3 (três) vezes mais tempo especificando do que desenvolvendo e que todos os projetos tiveram atrasos na entrega.

A gerência de manutenção de sistemas permaneceu no mesmo cenário, pois cada desenvolvedor continuou focado no seu sistema, sem utilização do processo, e sem interação com a área de análise de sistemas, que deveria realizar o entendimento das demandas de sistemas do Centro.

A falta de envolvimento da gerência de estratégia de dados na fase de elaboração, provocou retrabalho para a gerência de análise de sistemas em alguns projetos, já que problemas no modelo de dados somente eram detectados por essa gerência na fase de construção.

O PTS não foi implantado, devido à falta de pessoal para compor uma área específica de testes e assim o trabalho de testes, quando realizado, era feito pela equipe de análise de sistemas.

Os processos construídos acabaram tornando-se ineficazes para a construção de sistemas, já que os projetos permaneceram em atraso, havia uma queixa permanente sobre a baixa produtividade do setor, a equipe reclamava da falta de comunicação e integração entre as gerências e havia uma desmotivação e descontentamento geral do grupo em relação à forma de trabalho.

Assim, os métodos ágeis surgiram como uma alternativa viável para a solução dos problemas apresentados nos processos do CPD/UnB, principalmente por propor os valores e princípios necessários à motivação da equipe, à valorização dos indivíduos, bem como a interação entre eles. Além disso, alguns integrantes da equipe já demonstravam interesse na utilização desses métodos, o que facilitou a direção do Centro a aceitar as novas práticas de desenvolvimento.

Nas subseções a seguir detalharemos a adoção desses métodos no CPD/UnB, através do estudo de caso de dois projetos que foram construídos utilizando essa nova abordagem.

3.4 Projeto piloto 1: O projeto SINUP

Em 25/09/2013 foi solicitada pela Direção do Centro uma reunião com toda a equipe da área de desenvolvimento de sistemas (gerências de análise de sistemas, desenvolvimento de sistemas e estratégia de dados) para propor melhorias nos processos da área, bem como na rotina de trabalho dos envolvidos.

A reunião começou com a exposição das diversas dificuldades enfrentadas por essa equipe, as quais se destacam:

- Dificuldade de comunicação entre as gerências: houve uma separação distinta entre áreas e papéis com a implantação dos novos processos, que dificultou a interação entre as gerências e mais especificamente entre os analistas e desenvolvedores. A falta de comunicação gerava divergências no entendimento dos artefatos e no negócio do projeto, provocando aumento do retrabalho tanto em documentação, quanto no produto em desenvolvimento. Em consequência desse retrabalho, os projetos eram pronlogados além do necessário.
- Quantidade de documentação dos processos: a grande maioria dos projetos contavam com apenas um ou dois analistas de sistemas, para preenchimento de uma extensa documentação, principalmente nas fases de concepção e elaboração. Muito desses artefatos sequer eram utilizados nas fases seguintes, o que fez a equipe questionar sobre a real necessidade desses documentos, já que havia um gasto considerável de tempo no preenchimento desses artefatos.
- Falta de atualização dos processos: desde que foram entregues em 2008, os processos não sofreram atualização, o que os tornaram inadequados às necessidades de alguns projetos. A exemplo do setor de manutenção, que não conseguiu se inserir nos processos e da área de análise de sistemas, que absorveu o processo de testes.
- Desmotivação do grupo: muitos alegaram estar desmotivados com o formato de trabalho, especialmente pelo fato de não conseguirem ter produção. Alguns desenvolvedores ainda alegaram que não tinham poder de decisão, já que recebiam uma mini-especificação pronta para programar. Isso limitava o seu poder criativo e fazia com que ele trabalhasse como uma máquina que recebe instruções em um sistema de produção.

Bassi Filho ressalta em seu trabalho [4] a necessidade dos modelos de desenvolvimento de *software* estarem focados nas pessoas, já que são elas as responsáveis pelo desenvolvimento de um projeto de software. O autor enfatiza ainda que muito além do aspecto técnico, esse desenvolvimento está sob influência de fatores psicológicos e motivacionais. Dentro dessa linha, percebe-se que o desenvolvimento no CPD/UnB focalizava apenas os aspectos técnicos, em um modelo pré-definido de tarefas, sem considerar as questões inerentes ao perfil da equipe e até mesmo do caráter do projeto.

Ao final dessa reunião, ficou definido o primeiro estudo de caso desse trabalho, o Projeto do Sistema de Números Únicos de Processos (SINUP), que deveria utilizar uma abordagem derivada dos métodos agéis. A escolha desse projeto se deu pelo fato de ser considerado um projeto de fácil execução dentre os que se encontravam na fila de espera para desenvolvimento e que, de acordo com a avaliação do grupo, poderia ser construído no prazo de 30 dias.

Foram escolhidos para utilização no projeto o *Scrum* e algumas práticas do XP (detalhadas na subseção 3.4.1), por serem considerados os métodos agéis com maior índice de adoção no país [13]. Como os problemas mais enfatizados pela equipe estavam ligados à melhoria do processo, ao envolvimento dos membros da equipe e ao prazo de entrega, foram também utilizados todos os princípios descritos no LSD para essa primeira abordagem.

3.4.1 Preparação para o projeto

Foi definida uma equipe multidisciplinar composta de 1 (um) analista de sistemas, 4 (quatro) desenvolvedores, 1 (um) estagiário desenvolvedor para a equipe de desenvolvimento e 1 (um) arquivista desempenhando o papel do *product owner*. O arquivista era um funcionário do órgão cliente e detentor do conhecimento do negócio envolvido no SINUP. O papel de *scrum master* foi desempenhado por um dos desenvolvedores, que possuía experiência prévia em projetos utilizando o *framework Scrum*.

A Tabela 3.1 apresenta o perfil da equipe do projeto SINUP. Observa-se que apenas o *scrum master* tem experiência com métodos ágeis. A experiência com a plataforma de desenvolvimento varia de 3 (três) meses a 2 (dois) anos. E a experiência na função exercida varia de 1 (um) a 15 (quinze) anos.

Tabela 3.1: Perfil dos colaboradores do projeto SINUP.

Colaborador	Faixa etária	Exp. função	Exp. *framework*	Exp. métodos ágeis
Analista de Sistemas	36-40 anos	10 anos	2 anos	Iniciante
Desenvolvedor 1	26-30 anos	5 anos	2 anos	Iniciante
Desenvolvedor 2	31-35 anos	8 anos	2 anos	Avançada
Desenvolvedor 3	31-35 anos	1 ano	3 meses	Iniciante
Desenvolvedor 4	31-35 anos	7 anos	2 anos	Iniciante
Estagiário	20-25 anos	1 ano	6 meses	Iniciante
Arquivista	36-40 anos	15 anos	Não se aplica	Iniciante

Como alguns membros ainda não haviam tido contato com os métodos agéis, o guia *scrum* [47] foi passado como leitura obrigatória e posteriomente foi executado um pequeno treinamento com foco nesse guia, nas práticas XP e no LSD. O responsável por esse treinamento foi o *scrum master* da equipe.

Para o início do projeto foram definidas as seguites condições:

- Da documentação inicialmente prevista nos processos do CPD, foi mantido o DAP, que já havia sido elaborado por 2 (dois) analistas de negócios da gerência de análise de sistema.

- O projeto seria construído em 4 (quatro) *sprints* de 1 (uma) semana, como forma de melhor avaliar a abordagem utilizada, já que seriam feitas retrospectivas semanais.
- Foram definidas reuniões de 15 (quinze) minutos diárias, sempre às 12h, de forma a compatibilizar o horário dos membros da equipe (alguns trabalhavam de 7h às 13h e outros de 13h às 19h)
- Foi solicitada à direção um quadro branco, pincéis de marcação e *post-it* para criação de um mural de acompanhamento das tarefas da *sprint*. Esse mural constava de 4 (quatro) colunas: não iniciada, para as tarefas que ainda não começaram, iniciada, para as tarefas que estavam em andamento, teste, para as tarefas que encontravam-se em teste e, pronto, para as tarefas que já haviam sido concluídas, testadas e estavam prontas para serem entregues ao *product owner*. Essa técnica é baseada no *Kanban*, que nada mais é que um sinal visual (como por exemplo, o quadro branco), para acompanhar um determinado trabalho (nesse caso, a *sprint*), à medida em que ele é executado e em várias etapas de um fluxo de valor previamente determinado (a exemplo das colunas definidas). A utilização do *Kanban* promove transparência ao processo utilizado no desenvolvimento, bem como no fluxo prescrito, já que sempre apresenta a situação real do projeto [31].
- Foi mantida a plataforma de desenvolvimento atual, baseada em *Java* [1] e algumas tecnologias como *Hibernate* [2], o *JavaServer Faces* (JSF)[3] e o *PrimeFaces*[4]. Essa plataforma foi construída em conjunto com uma consultoria externa e vêm sendo utilizada e mantida pelo Centro desde 2010. Durante a execução do projeto, havia um consultor externo contratado para manutenção e evolução da plataforma.
- As reuniões de planejamento, revisão e retrospectiva foram registradas em um relatório, por *sprint*, como forma de manter a memória do projeto. Os relatórios foram enviados logo após a conclusão da respectiva reunião, por e-mail, para todas as gerências de sistemas e direção do Centro.
- A técnica de prototipação, utilizando a plataforma de desenvolvimento, foi definida para validação das *interfaces* do sistema junto ao cliente.

Além das condições acima, as seguintes práticas XP foram selecionadas:

[1] *Java* é uma linguagem de programação orientada a objetos desenvolvida na década de 90, na empresa *Sun Microsystems*.

[2] *Hibernate* é um *framework* para o mapeamento objeto-relacional escrito na linguagem *Java*.

[3] JSF é um *framework* baseado em *Java* para a construção de *interfaces* de usuário baseadas em componentes para aplicações Web.

[4] *PrimeFaces* é uma suíte de componentes de código aberto, composta de biblioteca de componentes de *interface* de usuários, para aplicações baseadas em JSF.

- Programação em pares, para desenvolvimento do produto, de forma a melhorar a comunicação e promover troca de experiências entre os membros;
- Padronização do código, prática já utilizada no CPD/UnB;
- Propriedade coletiva do código, também já utilizada pelo Centro;
- Integração contínua, envolvendo todos os desenvolvedores, pois nos moldes anteriores apenas um único desenvolvedor poderia gerar novas versões de um produto em desenvolvimento;
- *Design* simples, com vistas a evitar o desperdício no desenvolvimento;
- Ritmo sustentável, apesar de não ser comum a realização de horas-extras no CPD/UnB, essa prática foi reforçada, pois o prazo estipulado de 30 (trinta) dias poderia proporcionar a quebra dessa prática;
- Entregas curtas, semanais, conforme previsto;
- Jogo do planejamento, conforme detalhado na subseção 2.3.5, para auxiliar na priorização e na estimativa das atividades;
- Cliente presente, para melhor entendimento do negócio e *feedback*.

3.4.2 Abordagem ágil preliminar

Conforme relatado na seção anterior, a primeira versão da abordagem ágil envolveu a utilização do *Scrum* e grande parte das práticas XP. As práticas excluídas dessa abordagem foram a "Testes"e "Refatorações"devido à inexperiência da equipe com esses temas e à impossibilidade de ser ofertado um treinamento para capacitação do time. *Scrum e XP* foram escolhidos devido a experiências bem-sucedidas de outras implantações utilizando esses métodos [37][56][38][4][25]. Nessa seção detalharemos a abordagem ágil definida, focando nas especificidades, já que os eventos e práticas, em sua maioria, foram executados conforme sugerido no método.

O projeto se iniciou com o recebimento das informações do DAP, que serviram de insumo para a produção do *Product Backlog*. Ressalta-se que o *PO* trabalhou sempre com uma *sprint* à frente da equipe de desenvolvimento. Assim, enquanto a equipe estava desenvolvendo as estórias da *sprint*, o *PO* estava trabalhando no detalhamento das estórias que estavam no *backlog*.

Na reunião de Planejamento, o *PO* apresentou o *backlog* priorizado e foram selecionados alguns itens para a *sprint*. Esses itens selecionados comporam o *Sprint Backlog*. A equipe de desenvolvimento, juntamente com o *scrum master* estimaram as estórias presentes no *sprint backlog*, através da aplicação do *planning poker*. A estimativa utilizada

baseou-se na apresentada no livro“Scrum e XP Direto das Trincheiras"[30], na qual o autor afirma que a unidade de pontos de estória geralmente corresponde a mais ou menos a “relação homem/dias”. Assim, foi computada a disponibilidade em dias de cada membro da equipe, que após totalizadas, definiram a quantidade de pontos de estória que puderam ser executadas na *sprint*.

O *planning poker* é uma técnica na qual cada membro da equipe de desenvolvimento recebe um baralho de 13 cartas. A cada item do *sprint backlog*, os membros selecionam uma carta do baralho que representa a sua estimativa de tempo e coloca-a virada para baixo. Quando todos já tiverem selecionado suas cartas, pede-se para que elas sejam reveladas, simultaneamente [30]. Se houver uma grande divergência entre as estimativas, a equipe discute as diferenças, de forma a chegar numa visão comum.

Existem ainda algumas cartas especiais que podem ser utilizadas nessa técnica. São elas:

- 0: informa que a estória já está feita ou que levará um tempo bastante reduzido para ser executada;
- ?: indica que o membro não possui conhecimento suficiente para efetuar a estimativa;
- Xícara de café: indica a solicitação de uma pausa no jogo.

Ao final da reunião de planejamento, um relatório composto das principais decisões tomadas foi produzido. As estórias foram disponibilizadas individualmente em um *post-it*, no quadro branco, na coluna "não iniciada". Cada *post-it* contém a descrição da estória e a quantidade de pontos que ela possui. Nesse quadro também foram informadas as datas de início e fim da *sprint*, das reuniões de revisão e retrospectiva e também o horário e local estabelecido para a reunião diária.

O início da *sprint* normalmente se deu no dia seguinte após a reunião de planejamento. Os membros escolheram as estórias que foram disponibilizadas no quadro branco e as moveram para a coluna "Iniciada". Atrás do *post-it* foi assinalado o nome de quem tá executando a estória. Como estávamos utilizando a técnica de programação em par, foi preenchido o nome de uma dupla no *post-it*. Os membros foram movendo a estória no quadro branco à medida que a situação foi mudando.

A reunião diária foi executada durante todos os dias da *sprint*, com a presença do *scrum master* e da equipe de desenvolvimento. Nessa reunião o *scrum master* fazia 3 (três) perguntas: “o que você fez?, o que você irá fazer?, há algum obstáculo?”.

Todo o trabalho da equipe de desenvolvimento foi integrado na mesma frequência com que as estórias foram desenvolvidas, ou seja, a prática XP de Integração Contínua foi

utilizada na abordagem. Para tal, foi utilizada a ferramenta *Hudson*[5], por já fazer parte da infraestrutura do Centro.

O *Hudson* basicamente verifica os possíveis erros de compilação dos programas. Os *builds*[6] foram realizados de forma automática, com frequência de 4 (quatro) em 4 (quatro) horas. Quando ocorreram erros de compilação, esses foram identificados e um e-mail foi enviado para toda a equipe de desenvolvimento do projeto. O *Hudson* possui uma *interface* para que os desenvolvedores possam acompanhar o estado dos *builds* dos projetos. Além disso, o *Hudson* indica o estado do projeto de forma gráfica e intuitiva, utilizando círculos com cores que indicam os possíveis problemas durante o processo de compilação.

Finalizada a *sprint*, as reuniões de revisão e de retrospectiva da *sprint* foram realizadas. Elas aconteceram sempre no mesmo dia, sendo a retrospectiva a última e sem a presença do *PO*, já que é uma reunião focada para a melhoria do time *Scrum*. Essas reuniões também apresentaram um relatório, para melhor acompanhamento do projeto.

Quando ainda tinham itens no *backlog*, uma nova reunião de planejamento foi realizada, assim foram realizadas quantas *sprints* forem necessárias à conclusão do produto.

A Figura 3.5 apresenta a modelagem da abordagem ágil definida para o projeto SINUP, de acordo com o fluxo descrito nessa seção.

[5]Ferramenta de integração contínua escrita em *Java*. Disponível em: http://hudson-ci.org/.

[6]É uma versão "compilada" de um *software* ou parte dele que contém um conjunto de recursos que poderão integrar o produto final.

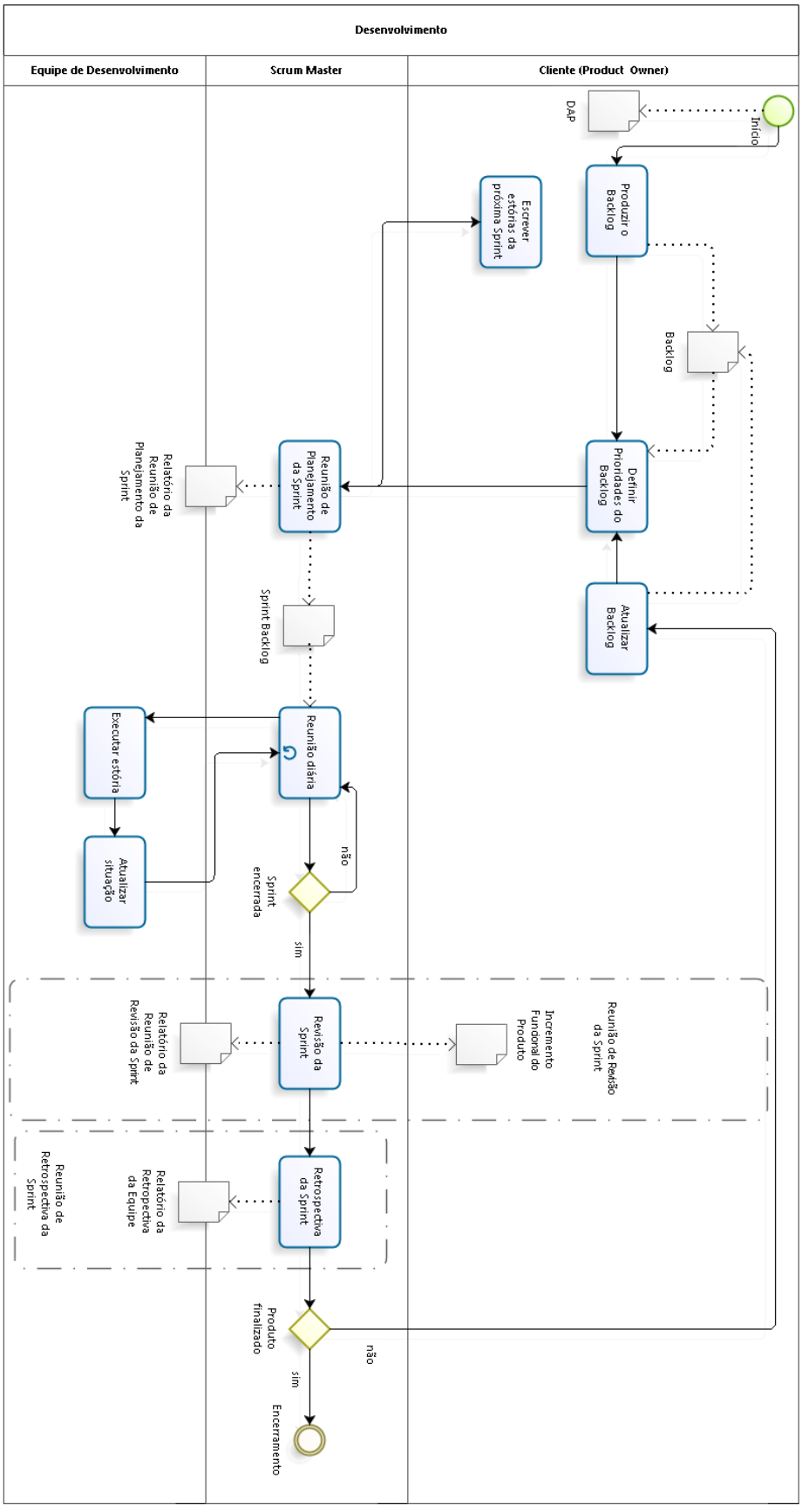

Figura 3.5: Abordagem Ágil definida para o Projeto SINUP.

3.4.3 Descrição do projeto

O objetivo do projeto era desenvolver uma solução sistêmica para controle e utilização de números únicos de processos (NUP). O NUP é um código numérico que identifica, de forma única e exclusiva, cada processo, produzido, recebido ou autuado no âmbito da Administração Pública Federal (APF).

Como na UnB diversos órgãos faziam a emissão do NUP, existia a necessidade de saber qual foi a unidade responsável pela emissão de determinado NUP, evitar a geração de NUP em duplicidade e fazer associação do NUP com o sistema de tramitação de documentos da UnB, no caso o UnBDOC.

Com a implantação desse sistema, a UnB poderia aumentar o número de unidades autuadoras, de forma descentralizada, ofertar maior segurança nas autuações de processos, além de reduzir o custo com impressão de etiquetas, já que para evitar a emissão de NUP em duplicidade, o órgão estava emitindo etiquetas previamente e muitas vezes uma quantidade além da necessária.

3.4.4 Execução do projeto

O projeto iniciou em 07/10/2013, data na qual ocorreu a primeira reunião de planejamento da equipe. Foi apresentando o DAP já construído, que continha informações gerais do projeto, como o escopo, o não-escopo, as premissas e os riscos inerentes ao projeto.

De posse dessas informações, foi construído junto ao *product owner* a primeira versão do *backlog* do produto e posteriomente foram selecionados 5 (cinco) itens para o *backlog* da primeira sprint. Nessa reunião foi utilizada a prática do jogo do planejamento para definição das estimativas de tempo de desenvolvimento de cada item, bem como para priorização desses itens de acordo com o valor de negócio associado.

Todas as reuniões diárias aconteceram conforme previsto e durante esses encontros alguns problemas começaram a ser detectados, como a dedicação parcial de um dos membros que estava envolvido em outro projeto e a dificuldade de aceitação de um modelo de dados incremental pela equipe da estratégia de dados.

Ao final da primeira *sprint*, 2 (dois) dos itens previstos não foram finalizados. A reunião de entrega aconteceu em 14/10/2013 e em comum acordo, os itens não finalizados foram inseridos na próxima *sprint*. A reunião de retrospectiva aconteceu na mesma data, logo após a reunião de entrega, e a equipe avaliou a experiência da primeira *sprint*.

Nessa avaliação, a equipe identificou que houve um aumento considerável da integração dos membros, o que tornou a comunicação mais fluída. Outro ponto avaliado foi a transparência das informações do projeto, já que todos os membros puderam desde o início obter conhecimento acerca do projeto. Além disso, todos tinham acesso ao que

cada membro estava executando, tornando-os mais próximos um dos outros. A experiência de desenvolvimento em par foi bem avaliada, principalmente pelos desenvolvedores iniciantes, que puderam absorver a experiência dos desenvolvedores mais experientes.

Como pontos negativos, a equipe considerou que a divisão do tempo de um dos desenvolvedores impactou na entrega do projeto, e que para as próximas *sprints* essa situação deveria ser evitada. Um outro problema identificado foi a indisponibilidade do *product owner*, o que fez por muitas vezes que dúvidas fossem respondidas por e-mail, adiando o desenvolvimento de algumas tarefas. Em relação à plataforma de desenvolvimento, houve uma atualização sem prévio aviso que comprometeu 1 (um) dia de trabalho de todo o time. Por fim, o fato de equipe de estratégia de dados estar fora do time *scrum* e o desconhecimento dessa equipe acerca da metodologia, fez com que a compreensão do caráter iterativo e incremental não fosse bem aceito. Esse foi inclusive um dos motivos para não conclusão de uma das tarefas, pois envolvia a necessidade da criação de algumas tabelas no banco de dados, e a estratégia de dados somente queria criar essas tabelas com todo o banco concluído.

Ao final da primeira retrospectiva, a equipe levantou a necessidade de interação do consultor responsável pela plataforma de desenvolvimento, bem como da equipe de estratégia de dados, com o time do projeto, já que mudanças na plataforma de desenvolvimento influenciaram negativamente na *sprint*. Além disso, o fato das tabelas não terem sido criadas a tempo, prejudicou a conclusão de uma das tarefas da *sprint*. Foi ratificado com o cliente o comprometimento ao projeto, que afirmou que o ocorrido foi uma situação ocasional e que participaria ativamente das próximas *sprints*.

A reunião de planejamento da segunda *sprint* ocorreu em 15/10/2013 e foram selecionados além dos dois itens da primeira *sprint*, mais um item do *backlog* priorizado pelo *product owner*. A reunião de entrega ocorreu em 23/10/2013, um dia a mais do que o previsto no planejamento, devido a uma atividade externa envolvendo dois membros do time *scrum*. Apesar de todos os itens estarem em estado de pronto, um dos itens não foi aceito pelo *product owner*, por não atender as necessidades inicialmente previstas. O problema com esse item foi associado a uma limitação da plataforma de desenvolvimento.

Na reunião de retrospectiva com a equipe de desenvolvimento, o *scrum master* sugeriu criar no quadro branco uma área de tarefas não planejadas, para poder verificar quais membros estão envolvidos em tarefas de outros projetos e mensurar o impacto dessas tarefas ao projeto. Um outro ponto observado nessa reunião foi que os itens do *backlog* estavam bastante abrangentes e que poderiam ser divididos em itens menores. Assim, ficou definido que para a próxima reunião de planejamento os itens seriam revistos junto ao *product owner*.

A indisponibilidade do *product owner* foi avaliada mais uma vez como um ponto negativo da *sprint* e ficou definido que o analista de sistemas deveria promover a melhor interação do mesmo com a equipe de desenvolvimento. Como solução ao problema apresentado na plataforma de desenvolvimento, ficou definida a construção de uma solução específica para o SINUP e que posteriormente, caso seja realizada alguma alteração na plataforma que contemple essa solução, que o código seria refatorado para adequação a essa alteração.

Ainda na segunda *sprint* pode ser observado o interesse de alguns funcionários das gerências de sistemas que não faziam parte do time *scrum* na nova abordagem. Foi permitido que essas pessoas assistissem a essas reuniões, porém apenas como ouvintes. O *scrum master* relatou que houve uma mudança de postura na equipe de desenvolvimento em relação à responsabilidade das tarefas, já que observou o comprometimento dos membros com toda a *sprint* e não apenas com os itens aos quais estavam associados.

O envio dos relatórios sobre as *sprints* aconteceu conforme planejado, ao final das *sprints*. Houve contribuições e questionamentos do gerente de análise de sistemas e da direção do Centro, relacionadas ao cronograma de desenvolvimento e as estimativas das tarefas, que foram esclarecidas pelo *scrum master*. A direção ainda questionou sobre a falta de participação do gerente de desenvolvimento de sistemas, que optou pelo não envolvimento no projeto.

No planejamento da terceira *sprint* levou-se em conta o ponto observado de ter tarefas específicas e detalhadas como itens do *sprint backlog*. Foi identificada a necessidade de envolvimento de um funcionário externo à equipe, especialista na plataforma de desenvolvimento, para auxílio na inserção de um novo componente criado na plataforma. Para o quadro de itens não planejados, previsto na *sprint* anterior, foi computada a participação do analista de sistemas em um curso previamente acordado, assim, foi considerada a participação parcial do analista na *sprint*.

Como forma de manter o cliente mais próximo do grupo, foi agendada uma reunião no meio da *sprint* com o objetivo de sanar as dúvidas que porventura viessem a surgir.

Na reunião de entrega da terceira *sprint*, a equipe apresentou o incremento construído, que foi aceito integralmente pelo *product owner*. O *feedback* dado pelo cliente foi bastante positivo, pois o incremento entregue foi considerado como um produto pronto para uso. A reunião de retrospectiva aconteceu no dia seguinte e nela foram observadas a melhoria na construção dos itens do *backlog*, por conta da inclusão de tarefas detalhadas e específicas e a importância da reunião com o cliente durante a *sprint*, que evitou que as dúvidas do produto fossem sanadas apenas na reunião de entrega. Segundo o registro da retrospectiva, a reunião com o cliente foi fundamental para a entrega de um produto satisfatório. O último item observado nessa reunião foi o fato dos desenvolvedores estarem integralmente

dedicados ao projeto, o que foi considerado também como um fator de sucesso da *sprint*.

A última *sprint* aconteceu com os 9 (nove) itens finais do *backlog*, e na reunião de entrega foi entregue uma versão final do sistema. Esse resultado já era esperado pelo time, pois na *sprint* anterior já havia uma sinalização do *product owner* de que o produto estava pronto para uso. Na reunião de retrospectiva final foi aberta uma discussão sobre a nova abordagem, a qual, por unanimidade, foi aceita como positiva pelo time. Foi decidida ainda nessa reunião uma apresentação formal, para todo o CPD/UnB, como forma de apresentação do trabalho realizado e da nova abordagem.

3.4.5 Lições aprendidas

Na reunião de apresentação estiveram presentes a equipe de desenvolvimento e o *scrum master* do projeto, os gerentes e membros das áreas de manutenção, análise de sistemas, desenvolvimento de sistemas e estratégia de dados, além da direção do Centro. O trabalho foi apresentado pelo time *scrum* e ao final foram observados os seguintes pontos:

- A interferência de projetos externos prejudicou a produtividade da equipe nas primeiras *sprints*, além de intervir no poder de autonomia do *scrum master* junto a equipe.
- O *sprint backlog* composto de tarefas mais específicas facilitou o trabalho da equipe de desenvolvimento.
- Ter o cliente disponível para o projeto é fundamental para esclarecer dúvidas do *sprint backlog* e ter *feedback* antes da reunião de entrega.
- Há necessidade de envolvimento da estratégia de dados no projeto, bem como do consultor da plataforma de desenvolvimento, mesmo que mediado pelo *scrum master*.
- A equipe observou que precisa aprimorar os testes na abordagem, já que algumas falhas não foram detectadas durante a *sprint*, deixando transparecer apenas na reunião de entrega.
- O pareamento de um desenvolvedor mais experiente, com outro menos experiente ajudou a nivelar o conhecimento acerca da plataforma de desenvolvimento.
- A utilização da plataforma de desenvolvimento para prototipar as interfaces do sistema foi considerada uma solução lenta. Além disso, a ideia inicial era poder evoluir esse protótipo para que ele se tornasse funcional, mas o trabalho para que isso pudesse acontecer era o mesmo de começar a construir do zero, segundo os desenvolvedores do projeto.

- De forma geral, a experiência foi considerada positiva e satisfatória por todo o time *scrum*, inclusive pelo *product owner*, que analisou que o produto entregue atendeu as suas expectativas e em um curto prazo de tempo.

Após a apresentação, o gerente de desenvolvimento de sistemas pediu a palavra, e apesar de concordar sobre o ganho de produtividade da equipe, alegou que a forma de trabalho não respeitava a hierarquia, já que as decisões eram tomadas sem o seu envolvimento. Informou ainda que até então ele não tinha conhecimento de como estava sendo executado o trabalho. O gerente de análise de sistemas então interviu, informando que o mural de atividades estava exposto em local de acesso a todos, que os relatórios estavam sendo enviados semanalmente e que as reuniões do *scrum* eram abertas a todos. A direção ainda sinalizou para o fato do gerente de desenvolvimento de sistemas ter declarado que não participaria da nova abordagem.

Assim, apesar do apoio da direção do Centro, houve um conflito hierárquico gerado com o gerente de desenvolvimento de sistemas. Contudo, diante do resultado positivo e da satisfação do grupo e do cliente, foi autorizada a realização de mais um projeto utilizando a nova abordagem. Foi solicitado para que nesse novo projeto houvesse o envolvimento de todos os gerentes, inclusive para a formação do novo time.

A Tabela 3.2 apresenta o resultado da utilização de práticas, princípios e técnicas especificadas para a abordagem ágil definida para o SINUP, bem como sugere adaptações para os pontos que foram candidatos à melhoria para o próximo projeto.

Tabela 3.2: Utilização de práticas, princípios e técnicas na abordagem ágil do SINUP e considerações para o novo projeto.

Prática / Princípio / Técnica	**Utilização**	**Considerações para o próximo projeto.**
Kanban	Considerado positivo para melhor controle e acompanhamento de tarefas.	Mantida para o próximo projeto.
Framework	Mudanças sem interação com o time Scrum prejudicou o andamento do projeto.	Necessidade de inserção de um ou mais membros da plataforma de desenvolvimento na equipe do projeto.

continua na próxima página

Tabela 3.2: Utilização de práticas, princípios e técnicas na abordagem ágil do SINUP e considerações para o novo projeto. (continuação)

Prática / Princípio / Técnica	**Utilização**	**Considerações para o próximo projeto.**
Relatório das reuniões de planejamento, entrega e retrospectiva	Promoveu melhor acompanhamento e transparência, principalmente para as gerências que não participavam dos eventos *Scrum* do projeto.	Mantida para o próximo projeto
Técnica de prototipação com o framework	Considerada improdutiva, devido ao desperdício gerado pelo não-aproveitamento do protótipo.	Necessidade de adaptação para a próxima fase, através da utilização de ferramentas de desenho ou rascunhos.
Programação em par	Considerada positiva, pois colaborou na disseminação do conhecimento sobre o *framework* de desenvolvimento.	Mantida para o próximo projeto.
Padronização do código e propriedade coletiva do código	Práticas já utilizadas pelo Centro, foram consideradas positivas para o projeto.	Mantida para o próximo projeto.
Integração contínua	Promoveu maior dinamismo e comprometimento ao time, já que todos os desenvolvedores passaram a estar envolvidos na utilização dessa prática.	Mantida para o próximo projeto.
Ritmo sustentável	Seguida à risca pela equipe, que se manteve num ritmo de trabalho constante e sem necessidade de horas-extras.	Mantida para o próximo projeto.

continua na próxima página

Tabela 3.2: Utilização de práticas, princípios e técnicas na abordagem ágil do SINUP e considerações para o novo projeto. (continuação)

Prática / Princípio / Técnica	**Utilização**	**Considerações para o próximo projeto.**
Entregas curtas	*Sprints* com duração de 1 (uma) semana foram consideradas curtas demais, pois o time sentiu a necessidade mais dias para realização de testes.	Ampliação do número de dias das *sprints*, incluindo dias específicos para realização de testes.
Cliente presente	A falta de envolvimento do *product owner* impactou na execução de algumas tarefas, bem como na falta de *feedback* antes da reunião de entrega.	Criação de um novo papel que possa ser uma espécie de espelho do *product owner* durante o projeto, garantindo maior presença de um perfil cliente no projeto.
Planning Poker	De fácil entendimento para a equipe, foi considerada uma boa prática para estimar o tamanho das tarefas.	Mantida para o próximo projeto.

continua na próxima página

Tabela 3.2: Utilização de práticas, princípios e técnicas na abordagem ágil do SINUP e considerações para o novo projeto. (continuação)

Prática / Princípio / Técnica	**Utilização**	**Considerações para o próximo projeto.**
Princípios do LSD	O princípio da eliminação do desperdício foi impactado pela falta da presença do cliente, bem como pela utilização do framework na construção de protótipos. Os demais princípios foram seguidos e considerados positivos no projeto. A inclusão de qualidade no processo se deu através da inserção de práticas do XP, como por exemplo a programação em par e os novos moldes da integração contínua. O princípio de "aprendizado contínuo"e o de "tornar-se cada vez melhores"foi reforçado nas retrospectivas do *Scrum*, bem como no registro das lições aprendidas ao final do projeto. As entregas curtas e constantes e o cumprimento do curto prazo de 30 (trinta) dias foram relacionadas ao princípio "entregar rápido". O envolvimento e o comprometimento da equipe na execução das *sprints* estiveram relacionados ao princípio "envolver todos".	Mantidos para o próximo projeto, com as adaptações necessárias à eliminação do desperdício.

3.5 Projeto piloto 2: O projeto SIADD

Com o aval da direção para criação de mais um projeto utilizando a abordagem ágil, reuniram-se as gerências de desenvolvimento e análise de sistemas para seleção do próximo projeto. Dentre os projetos na fila para construção, escolheu-se o SIADD (Sistema de Acompanhamento de Desempenho Docente), por ser considerado um projeto estratégico

para a UnB, desafiador para o Centro e de escopo e tamanho superior ao SINUP. Assim como o SINUP, esse projeto também já possuía o DAP elaborado, assinado e validado pelo cliente.

A equipe formada foi composta de 2 (dois) desenvolvedores sênior que participaram do projeto SINUP, 1 (um) desenvolvedor iniciante, 1 (um) analista de sistemas sênior. O papel de *scrum master* foi desempenhado pelo gerente de análise de sistemas, que também atua como analista de sistemas da área e recebeu acompanhamento do *scrum master* do projeto SINUP durante toda a execução.

Esse acompanhamento se deu durante as primeiras *sprints* do projeto, através da participação do *scrum master* do projeto SINUP como observador dos eventos *Scrum*, de forma a orientar o novo *scrum master* do projeto SIADD na condução do projeto. O papel de *product owner* foi desempenhado por um professor da UnB, com experiência na área de sistemas, nomeado pela Vice-Reitoria para acompanhamento do projeto.

A Tabela 3.3 apresenta o perfil da equipe do projeto SIADD. Observa-se que todos os membros são iniciantes com métodos ágeis. A experiência com a plataforma de desenvolvimento varia de 3 (três) meses a 2 (dois) anos. E a experiência na função exercida varia de 2 (dois) meses a 32 (trinta e dois) anos.

Tabela 3.3: Perfil dos colaboradores do projeto SIADD.

Colaborador	Faixa etária	Exp. na função	Exp.*framework*	Exp. métodos ágeis
Analista de Sistemas 1	51-60 anos	32 anos	Sem experiência	Iniciante
Analista de Sistemas 2	31-35 anos	13 anos	1 ano	Iniciante
Desenvolvedor 1	26-30 anos	5 anos	2 anos	Iniciante
Desenvolvedor 2	31-35 anos	7 anos	2 anos	Iniciante
Desenvolvedor 3	31-35 anos	2 meses	3 meses	Iniciante
Professor	45-50 anos	14 anos	Não se aplica	Iniciante

Foi ofertado novamente um treinamento para toda a equipe, nos mesmos moldes do projeto SINUP e especificamente o gerente de análise de sistemas recebeu um treinamento focado na formação de *scrum master*.

3.5.1 Preparação para o projeto

Com base na experiência anterior, algumas práticas foram mantidas e outras revisadas. Além disso, novas técnicas foram adicionadas para experimentação do grupo.

- Como forma de minimizar os problemas vivenciados com a ausência do cliente, o analista de sistemas assumiu o papel de *product owner proxy* (POP). O POP é um profundo conhecedor do negócio do projeto que tem como objetivo representar o cliente e os seus interesses [26]. Esse papel é comumente utilizado em projetos

onde o *product owner* não é acessível, como o caso do SIADD. Assim, na ausência do *product owner* o POP é o responsável por fornecer informações, dar suporte às decisões do time e facilitar a modelagem de requisitos [15][10].

- A prototipação utilizando a plataforma de desenvolvimento foi suspensa e o time *scrum* acatou a utilização de rascunhos em papel ou quadro branco. A técnica de rascunho é considerada eficaz em abordagens ágeis, por promover maior aproximação entre o cliente e a equipe de desenvolvimento e possibilitar a construção de protótipos com a participação do próprio cliente [42].
- As *sprints* foram ampliadas para 10 (dez) dias úteis com o objetivo de termos o mínimo de 2 (dois) dias úteis para integração contínua e testes.
- A programação em par foi mantida e como a equipe de desenvolvimento era composta de 2 (dois) programadores experientes e 1 (um) iniciante, ficou definido que os experientes fariam revezamento com o iniciante a cada *sprint*.
- Foi acordado entre as gerências que a equipe de desenvolvimento deveria estar integralmente focada no projeto SIADD, para que não houvessem interferências de projetos externos e consequentemente baixa de produtividade do time.
- As reuniões diárias foram agendadas para 14h30, pois todos os membros do time possuíam disponibilidade para o turno da tarde.
- Durante as reuniões de planejamento foi sugerida a utilização de um projetor para exibição do produto, para melhor visualização das tarefas definidas;
- Foi inserida uma restrição de realização de reuniões de entrega e retrospectiva às sexta-feiras, devido à necessidade de ausência de um membro do time nesse dia da semana.
- Para maior aproximação dos gerentes na abordagem ágil, foi solicitado a participação de cada um deles nas reuniões de entrega e retrospectiva de cada *sprint*.
- O *product owner* sugeriu que uma das funcionalidades previstas no sistema fosse implementada por um aluno que estava sob sua orientação e posteriormente integrada ao produto em desenvolvimento. A sugestão foi acatada pela equipe e ficou definido que assim que essa funcionalidade estivesse pronta o *product owner* comunicaria à equipe e solicitaria a integração no planejamento da *sprint* seguinte.

3.5.2 Descrição do projeto

O projeto SIADD tem como objetivo o cadastro e o acompanhamento do processo de avaliação de desempenho docente, para fins de progressão funcional. O sistema permite a automatização da coleta de informações com o preenchimento da solicitação de avaliação de desempenho, via Internet, e a importação das informações existentes nas bases de dados da UnB e no Lattes[7]. Além disso, o sistema permite a visualização do histórico de progressões do docente, a simplificação de documentos comprobatórios, a segurança das informações prestadas e a geração de informações gerenciais relacionadas.

3.5.3 Execução do projeto

A primeira *sprint* iniciou em 16/01/2014 e teve a primeira reunião de entrega em 30/01/2014. Para essa primeira *sprint* foram selecionados 8 (oito) itens do *backlog*, priorizados pelo *product owner*. Na reunião de entrega 2 (dois) desses itens não haviam sido finalizados, devido a uma atualização na plataforma de desenvolvimento, porém, foi entregue um incremento de produto utilizável. Ficou acordado que os dois itens restantes seriam encaixados na próxima *sprint*. Na reunião de retrospectiva um ponto bastante comentado foi o fato da equipe ter se mobilizado como um todo para a entrega dos itens do *backlog*. Essa postura minimizou os problemas causados pela mudança na plataforma, que poderiam ter prejudicado a *sprint* integralmente. Os desenvolvedores que haviam participado do projeto anterior consideraram como positiva a experiência do POP, já que esteve integrado ao time, participou das reuniões diárias e esteve disponível para esclarecer dúvidas a respeito das tarefas. Os dias específicos para testes não puderam ser utilizados, devido a necessidade de inclusão das tarefas nesses dias. Não houve participação do gerente de desenvolvimento de sistemas, conforme previsto.

A segunda *sprint* iniciou em 31/01/2014 e foram selecionados 7 (sete) itens do *backlog*. Nessa segunda *sprint* o POP passou a detalhar os itens do *backlog* para a *sprint* seguinte. Em 03/02/2014 foi anunciada a substituição da direção do Centro, o que causou uma certa tensão no time, já que no passado, mudanças de gestão promoveram a suspensão de alguns projetos em andamento. A reunião de entrega aconteceu em 17/02/2014 e a equipe conseguiu entregar todas as tarefas previstas. Na reunião de retrospectiva foi enfatizada a melhora da comunicação do time e o ritmo de trabalho como pontos positivos. Houve uma cobrança por parte da equipe de desenvolvimento, ao *scrum master*, sobre a continuidade do projeto. O *scrum master* se comprometeu a agendar uma reunião com a direção para obter informações a respeito dos planos para o projeto, mas informou que a

[7]É uma plataforma do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) que representa a integração de suas bases de dados de currículos, grupos de pesquisa e instituições em um único sistema de informações.

equipe deveria manter o acordado com a direção anterior, já que havia a expectativa de recebimento do projeto por parte do *product owner* e do orgão que demandou o projeto, no caso a Vice-Reitoria.

A terceira *sprint* iniciou em 18/02/2014 e devido a um recesso já previsto, foi programada para finalizar em 26/02/2014. Os dias de testes foram novamente prejudicados, mas a equipe conseguiu entregar todas as tarefas e o incremento entregue foi considerado satisfatório. O *scrum master* comunicou o compromisso da nova direção em manter o projeto e do interesse em expandir a abordagem para todos os projetos do Centro.

A reunião de planejamento da quarta *sprint* aconteceu em 10/03/2014 e nessa data realizou-se uma discussão sobre a greve que aconteceria em 17/03/2014. Ficou acordado que seria feita uma tentativa de trabalho à distância, caso a greve fosse aprovada e para isso foi inserido no *backlog* as atividades para disponibilização do ambiente de desenvolvimento para acesso externo. A direção aprovou o trabalho à distância, bem como a equipe de desenvolvimento manteve o compromisso de continuidade do projeto durante a greve, desde que não tivessem necessidade de estarem fisicamente presentes. Com base na experiência relatada por Igor Muzzetti [38], realizamos a aquisição insitucional e gratuita da ferramenta *PivotalTracker*[8], como forma de manter um quadro virtual semelhante ao quadro físico utilizado para acompanhamento do projeto.

A greve inicou em 17/03/2014 e a primeira reunião de entrega, virtual, estava prevista para 31/03/2014. As reuniões diárias foram suspensas e a comunicação entre o time passou a ser realizada por e-mail. Ao final dessa *sprint*, apenas 2 (duas) das 6 atividades previstas foram concluídas, o que tornou a *sprint* fracassada. A equipe relatou a dificuldade do trabalho à distância, que envolviam a lentidão do ambiente e a falta do contato face à face. Além disso, houve uma mobilização dos servidores das áreas de sistemas, para que os servidores desse projeto suspendessem suas atividades, o que acabou ocorrendo na reunião de entrega dessa *sprint*.

Após um pouco mais de 3 (três) meses, a greve foi encerrada e uma reunião de planejamento foi agendada para 01/07/2014. Ao resgatar o último *backlog*, houve um esforço considerável para que a equipe de desenvolvimento lembrasse de regras de negócio anteriormente detalhadas. As tarefas dessa *sprint* foram as mesmas da *sprint* anterior, com exceção da configuração de uma máquina virtual para o trabalho à distância e a liberação de acesso aos servidores de desenvolvimento e homologação, que foram as únicas tarefas anteriormente concluídas. A entrega dessa sprint aconteceu em 15/07/2014 e mais uma vez foi considerada satisfatória. Os dias específicos de testes foram utilizados nessa *sprint*, contando com a colaboração do POP, que validou as regras de negócios solicitadas. A

[8]Ferramenta para acompanhamento e gestão de projetos. Disponível em http://www.pivotaltracker.com/.

equipe ainda relatou a dificuldade de retomada ao trabalho após o período de greve e o *scrum master* reforçou a disponibilidade do POP para auxiliar na compreensão dos requisitos e regras de negócio que porventura tenham sido esquecidos.

Durante a greve, o *product owner* solicitou ao aluno que encaminhasse o projeto desenvolvido para que pudéssemos realizar a integração e essa atividade foi prevista para a próxima *sprint*, que se iniciou em 16/07/2014. Na reunião de entrega, todas os itens do *backlog* foram concluídos, exceto a integração, devido ao fato da funcionalidade externa ter sido construída em um padrão diferente da plataforma de desenvolvimento do Centro. O POP enfatizou que na entrega, com exceção dessa integração, todos os requisitos já haviam sido finalizados, restando no *backlog* atividades de correção, de aumento de performance e de melhoria de interface. Na reunião de retrospectiva considerou-se que o desenvolvimento externo, sem o acompanhamento do time, contribuiu para a construção de uma funcionalidade que não atendeu as expectativas do cliente e do time e ficou combinado que na próxima *sprint* seria realizada uma tentativa de adaptação desse código para a plataforma de desenvolvimento.

A sétima *sprint* teve início em 30/07/2014 e além da adaptação prevista, foram selecionados alguns itens do *backlog* que promoveram a utilização da prática de refatoração, a exemplo do carregamento de dados em uma determinada tela que teve o seu tempo reduzido de 26 (vinte e seis) segundos para menos de 1 (um) segundo. Ao final dessa *sprint* optou-se pelo descarte da funcionalidade externa. Essa decisão baseou-se no LSD [40], que sugere que defeitos são considerados desperdício, pois o custo para corrigí-lo poderá ser maior que o de construir uma nova funcionalidade.

Para a *sprint* seguinte ficou definido que a equipe como um todo deveria trabalhar na construção dessa nova funcionalidade e com o auxílio do POP, foram definidas algumas tarefas no *backlog* com o objetivo de concluí-la. A reunião de entrega aconteceu em 27/08/2014 e a versão entregue foi aceita com êxito. Houve necessidade de integração de mais um membro na equipe de desenvolvimento, para auxiliar na utilização de um componente necessário para conclusão da funcionalidade.

A última *sprint* teve menor duração que as demais, devido à finalização do *backlog*. Em 02/09/2014 foi realizada a apresentação da versão final do sistema, com a participação da Vice-Reitora, que realizou o aceite do produto.

3.5.4 Lições aprendidas

Com base na experiência anterior, muitos problemas puderam ser minimizados, a exemplo da dedicação exclusiva do time, que evitou a queda de produtividade da equipe.

Na execução desse projeto, as seguintes lições foram incorporadas para melhoria da abordagem:

- Grandes intervalos de tempo comprometem a execução, devido à dificuldade de retomada do ritmo da equipe, bem como da necessidade de reaprendizado do negócio do projeto.
- Há necessidade de disseminação do conhecimento acerca da plataforma de desenvolvimento para toda a área de sistemas.
- O trabalho à distância promoveu resultados insatisfatórios, já que a equipe além de entregar menos da metade das tarefas, não conseguiu produzir um incremento com valor agregado.
- A dependência entre integrações que serão desenvolvidas por outras equipes deverão ser acompanhadas pelo time do projeto. Do contrário, poderão gerar desperdício e retrabalho à equipe de desenvolvimento.
- O papel do POP e a sua disponibilidade integral ao projeto promoveram melhor entendimento do negócio, bem como facilitaram a modelagem de requisitos. Vale ressaltar que o POP atuou planejando a *sprint* seguinte e, assim, o próximo *backlog* já encontrava-se detalhado quando da sua execução.
- A utilização de rascunhos no desenvolvimento, envolvendo a participação do POP, foi considerada uma dinâmica produtiva e que auxilia no aprendizado do negócio do projeto. Na Figura 3.6 apresentamos um exemplo de rascunho do SIADD, na qual são definidas algumas *interfaces* do sistema, bem como o seu fluxo de execução através das setas que referenciam essas interfaces.

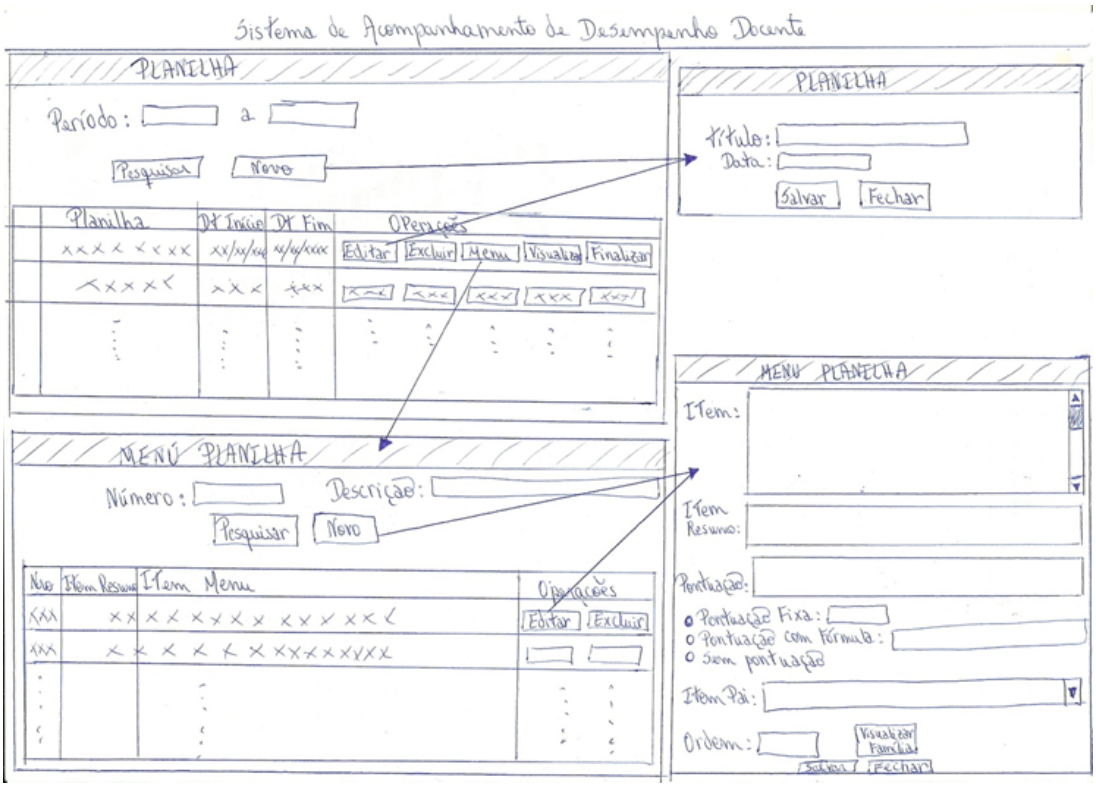

Figura 3.6: Rascunho de *interfaces* projeto SIADD.

Capítulo 4

Resultados

Após a conclusão dos projetos piloto, a abordagem ágil preliminar foi atualizada, de forma a contemplar as experiências obtidas durante a execução desses projetos. Além disso, foi aplicado um questionário com parte das respostas fechadas e parte abertas, tendo as respostas fechadas um campo de justificativa para a pergunta. A utilização desse questionário seguiu padrão similar ao utilizado em [25] no que tange ao formato das perguntas e ao utilizado por [56] em relação ao conteúdo do questionário. Ressalta-se que questionário foi aplicado de forma individual, com o objetivo de responder a questão de pesquisa do estudo de caso.

Todos os membros dos projetos que estão lotados no CPD/UnB responderam ao questionário, com exceção do estagiário do projeto SINUP, que havia concluído o seu contrato de estágio, devido à formatura.

Esse capítulo apresenta o modelo da abordagem ágil na seção 4.1, faz uma análise descritiva do questionário aplicado na seção 4.2 e em seguida realizada a aplicação do método de pesquisa qualitativa *Grounded Theory* na seção 4.3.

4.1 Abordagem ágil

A grande diferença da versão preliminar da abordagem ágil foi a inserção do papel do POP. O POP na abordagem compartilha das tarefas destinadas ao PO e diferentemente dele, pode participar da reunião de retrospectiva da *sprint*. A inserção desse novo papel foi modelada na abordagem ágil e apresentada na Figura 4.1.

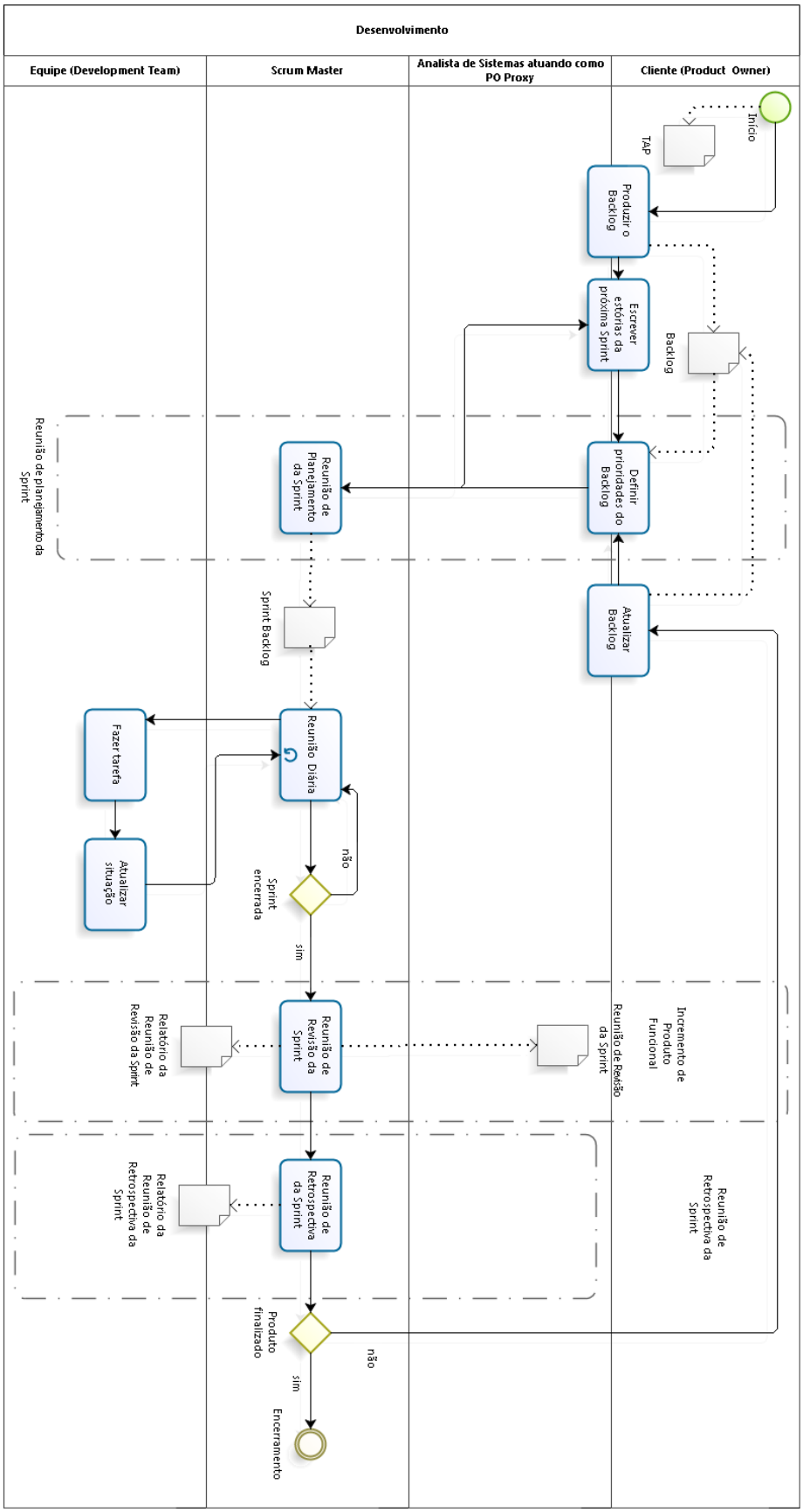

Figura 4.1: Abordagem Ágil do CPD/UnB.

De forma a auxiliar o entendimento das práticas e métodos utilizados, foi criado um diagrama conceitual da abordagem, disposto na Figura 4.2 O diagrama apresenta os métodos ágeis utilizados na abordagem ágil proposta. Os valores e princípios do LSD estão dispostos de forma a permear toda a abordagem. Essa visão está representada na cor vermelha.

O *framework* para o gerenciamento dos projetos é o *Scrum* e está apresentado no diagrama no círculo azul. Nesse círculo estão expostos os eventos, artefatos e papéis selecionados.

Por fim, no círculo verde estão as práticas do XP selecionadas na abordagem. A associação dessas práticas aos elementos do *Scrum* estão representadas nos círculos tracejados.

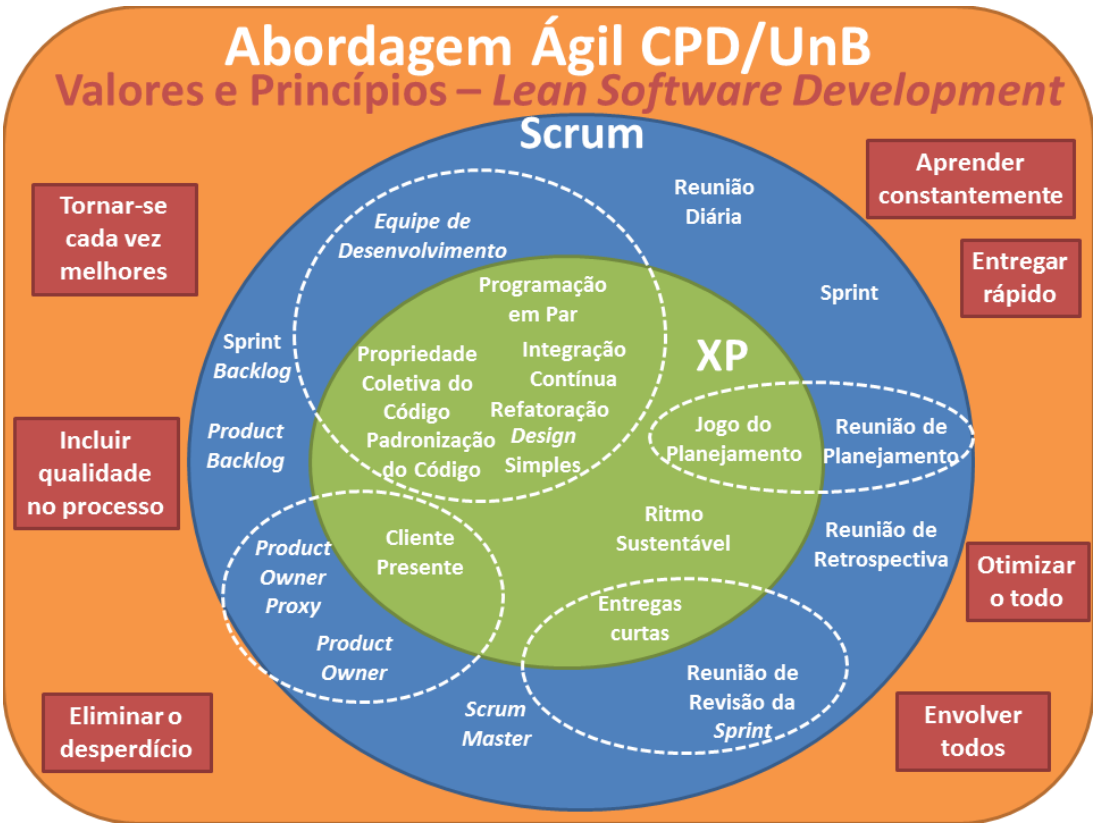

Figura 4.2: Diagrama conceitual da Abordagem Ágil.

4.2 Análise descritiva do questionário

A análise descritiva do questionário foi realizada sob a ótica do pesquisador, utilizando-se da resposta dos entrevistados e da correlação com a literatura sobre o assunto. O formato da análise é a apresentação da pergunta realizada, a análise das respostas e a exemplificação do resultado dessa análise com trechos das respostas dos entrevistados.

1. De acordo com os valores propostos no manifesto ágil, qual você julga de maior importância?

   O objetivo dessa pergunta era entender qual valor ágil tinha maior importância para o entrevistado, complementando assim o seu perfil. O valor considerado mais importante foi "Indivíduos e interações mais que processos e ferramentas". Há o reconhecimento do valor das pessoas no desenvolvimento de sistemas. Essa percepção vai de encontro ao observado em [4] ao afirmar que "Processos não irão produzir *software*, mas os indivíduos irão. Portando o modelo de desenvolvimento deve ser baseado nos indivíduos, o que significa considerar as características humanas durante o processo". O entrevistado 8 (E8) fez a seguinte colocação: "...acredito que toda execução de projetos deve ter o foco principal nas pessoas, independente de hierarquias, processos, linguagens de programação. São os profissionais os únicos que são capazes de fazer a diferença num projeto..."

2. A nova abordagem melhorou a sua capacidade de trabalho em equipe?

   Todos os entrevistados afirmaram que a nova abordagem melhorou a capacidade de trabalho em equipe. Foram observadas as seguintes respostas que associam essa melhora significativa a comunicação, a melhoria do ambiente organizacional e a praticidade da abordagem:

   - Entrevistado 2 (E2): "...a comunicação entre a equipe melhorou e os integrantes da equipe tiveram um comprometimento maior e não ficaram estagnados..."
   - Entrevistado 4 (E4): "...pouca teoria para muitos resultados..."
   - Entrevistado 5 (E5): "... me comuniquei melhor..."
   - Entrevistado 6 (E6): "...prefiro trabalhar em equipe, pois melhora o ambiente organizacional, no meu ponto de vista..."
   - E8: "...Ao haver mais interação com as pessoas, torna-se um compromisso tentar apresentar alguma coisa nova a cada reunião, esse sentimento faz com que você queira compartilhar seu trabalho com os outros. Isso gera um dinamismo orgânico de trabalho em equipe..."

3. Em relação a sua satisfação com a forma de trabalhar na abordagem ágil em comparação com o processo utilizado anteriormente no CPD/UnB, você pode afirmar que:

   A maioria dos entrevistados afirmou que a nova abordagem fez aumentar a satisfação com o formato de trabalho. Apenas um dos entrevistados afirmou que a sua satisfação não modificou comparada ao forma de trabalho anterior. Os pontos comentados

nessa resposta associam esse resultado ao poder de decisão de todos os envolvidos, ao caráter democrático da abordagem, a melhoria da comunicação, a melhoria da distribuição do trabalho, ao comprometimento de entrega de um produto com valor agregado, ao modelo iterativo e incremental e a disseminação do conhecimento do negócio e ao comprometimento da equipe.

- E2: "...gostei muito por ser um método que possibilita que todos participem e tenham voz sobre como deveriam ser feitas todas as tarefas do projeto. É uma maneira mais democrática aonde existe uma pessoa atuando como 'líder' e não 'chefe'..."
- E3: "... melhorou a comunicação e a distribuição do trabalho. Também aumentou o compromisso com a entrega de um produto funcional em cada *sprint*..."
- E4: "...o modo de desenvolvimento iterativo e incremental me deu a possibilidade de desenvolver um projeto sem que a complexidade do mesmo inibisse os trabalhos..."
- E5: "... melhorou a comunicação da equipe, os membros participavam das decisões e tinham uma noção de todo o sistema..."
- E6: "...as decisões são tomadas em equipe, levando em consideração o ponto de vista de cada envolvido no projeto, evitando assim as decisões de forma unilateral..."
- E8: "...Os métodos ágeis tendem a fazer com que as equipes interajam mais e que os indivíduos se comprometam com a meta coletiva, criando uma cumplicidade que se torna importante para que a equipe foque no objetivo do projeto, que passa a ser o objetivo de todos e não apenas do cliente ou gerente de projeto..."

4. De forma geral, como você se sente utilizando a prática de programação em par?

   Para essa pergunta, a totalidade dos entrevistados afirmou estar satisfeito com a prática. Foi comentado que a programação em par promoveu a troca de experiências e o nivelamento de negócio entre os pares, além do aumento da produtividade, da concentração, e da qualidade no código:

   - E8: "...Aumento da concentração e produtividade (incrível como a gente perde tempo com distrações do trabalho); Melhor interação; Qualidade do código (os gatos/gambiarras são praticamente excluídos dos códigos, por que 'alguém tá vendo')..."

5. Qual a sua satisfação com a abordagem ágil proposta?

Os entrevistados afirmaram estar satisfeitos com a abordagem e aliam esse fator ao resultado satisfatório do produto, a melhoria da comunicação na equipe e ao grau de motivação proporcionado pela finalização de uma *sprint*, que era encarada como uma vitória pela equipe.

- E2: "...fiquei bem satisfeito com o processo de trabalho e com os resultados obtidos tanto no produto como em relação a comunicação da equipe..."
- E8: "...cada iteração era encarada com uma vitória do projeto, causando um fator positivo dentro da equipe..."

6. Você acredita que o CPD/UnB deva adotar a abordagem ágil para a área de sistemas? Quando perguntados sobre a possibilidade de adoção dessa abordagem, de forma definitiva no CPD/UnB, todas as respostas foram positivas. Nesse ponto foi comentado que a abordagem facilita a interação e aumenta a produtividade da equipe.

   - E1: "...este método facilita a interação da equipe e o trabalho progride com mais rapidez..."
   - E2: "...as metodologias ágeis tiraram o SINUP de um 'limbo' de 1 ano devido a vários fatores, inclusive escrita de casos de usos. Antes o sistema deveria ser totalmente analisado para depois ser escrito. Com os métodos agéis possibilitaram a entrega e o entendimento do que o cliente queria em cada parte do sistema..."
   - E5: "... todos os sistemas construídos nessa metodologia ágil foram entregues no prazo, com satisfação do cliente..."
   - E6: "... dá uma dinâmica não só na metodologia de trabalho, mas também na capacidade de iniciativa dos envolvidos no projeto..."
   - E7: "...a abordagem ágil promoveu melhorias importantes no Centro, principalmente em relação ao desenvolvimento da equipe. Analistas e programadores trabalhavam como se fossem dois times distintos e quando um resultado insatisfatório acontecia (que não era difícil), era um tal de achar culpado que não acabava mais. Com a nova abordagem o projeto passou a ser de um único time e todos se envolveram e se comprometeram tornando-se responsáveis pelo que estavam fazendo..."
   - E8: "...Pelo dinamismo que se faz necessário na execução de projetos e pelo histórico de projetos abortados, uma abordagem ágil irá auxiliar o CPD/UnB a conseguir gerenciar melhor seus projetos e além de 'impor' a constante inspeção deste e assim determinar os melhores caminhos a serem seguidos..."

7. A nova abordagem ágil influenciou na utilização da plataforma de desenvolvimento (*framework*) do Centro?

   O objetivo dessa pergunta era saber se a abordagem ágil teve alguma influência na plataforma de desenvolvimento. Algumas respostas enfatizaram as mudanças que ocorreram na plataforma durante as *sprints*, que foi um problema já detectado nas lições aprendidas dos projetos piloto. Houve uma resposta relacionada à melhoria da qualidade do código, outra em relação às interações entre os profissionais e outra que acredita que não houve mudanças.

   - E1: "...sem diferenças marcantes..."
   - E2: "...houve problemas com alguns componentes que não estavam prontos e tiveram que ser feitos pela equipe. O resultado final, avaliando pelo funcionamento do sistema e pelo que foi dito pelo cliente foi satisfatório..."
   - E3: "...o fato de tentar usar novos componentes dificultou demais a flexibilidade de desenvolver algo que foge da plataforma..."
   - E4: "...a qualidade do código, seguindo o modelo incremental, recebeu melhorias e evoluiu, apesar de todo o engessamento herdado do *framework*..."
   - E5: "...não tínhamos tanto tempo para aprender o novo padrão imposto. Tivemos que voltar a forma de desenvolvimento livre para poder entregar a *sprint* dentro do prazo..."
   - E6: "...constantes evoluções e não propagações do modelo desejado..."
   - E7: "...a forma de trabalho isolada da equipe de construção da plataforma de desenvolvimento, por muitas vezes, comprometeu o trabalho do time ágil. Mudanças e atualizações foram realizadas sem o conhecimento da equipe, prejudicando assim as tarefas da *sprint*. Acredito que o fato de ter inspeção constante nessa abordagem tornem esse tipo de problema mais visível. Devemos de alguma forma melhorar a integração com os desenvolvedores da plataforma..."
   - E8: "...mudou principalmente interação dos profissionais que tinha o hábito de trabalhar isolados e no enfoque dos objetivos..."

8. Algumas *sprints* não atingiram o objetivo esperado. O que você considera que tenha influenciado nesse resultado?

   As demandas provenientes de outros projetos, bem como as dificuldades proporcionadas pela plataforma de desenvolvimento, foram os itens mais citados, confirmando as lições aprendidas de ambos os estudos de caso. Outro problema relatado na execução do estudo de caso foi o conflito com o gerente de desenvolvimento, que segundo

dois dos entrevistados influenciou na execução da *sprint*. Foi ainda comentado sobre a greve que atrasou o projeto SIADD, o fato de não ter uma tarefa específica para contemplar o tempo gasto na passagem de conhecimento entre os pares, a falta de conhecimento prévio sobre a metodologia, ausência do cliente e capacidade técnica do time.

- E1: "...envolvimento com outras tarefas por parte dos colaboradores, novo *framework*, greve..."
- E2: "...chefe não apoiou o uso da metodologia. Criação de componentes que demoraram para ficar pontos e tivemos que tentar outra alternativa..."
- E3: "... demandas externas. Também o fato de não contemplar o tempo para transferir conhecimento para os mais inexperientes..."
- E4: "...conhecimento prévio sobre a metodologia, que foi adquirido durante o processo. Falta de conhecimento sobre a plataforma de desenvolvimento..."
- E5: "...problemas com a nova plataforma e a utilização de membros da equipe em outros projetos..."
- E6: "...permissões de acesso, incompatibilidade da agenda do cliente, *framework*, boicote da chefia imediata, mudança da direção"
- E7: "...demandas de outros projetos, mudanças na plataforma de desenvolvimento durante a execução do projeto, falta de apoio da chefia de desenvolvimento..."
- E8: "...cada *sprint* tem sua peculiaridade. Quando seu objetivo não é alcançado ou alcançado parcialmente deve procurar qual foram os problemas (e não os culpados) e tentar sanar a adversidade para a próxima interação. De modo geral, posso identificar que estamos muito aquém em recursos humanos (deve haver treinamento técnico), arquitetura de sistemas e comprometimento dos atores envolvidos (desenvolvedores, setores, cliente)..."

9. Como você avalia a abordagem utilizada?

   Em relação à avaliação da abordagem foram abordados como positivos o envolvimento diário da equipe, a integração da equipe, a satisfação do usuário, a melhoria do formato de trabalho, atendimento dos prazos, passagem do conhecimento, otimização das atividades, diminuição dos conflitos entre os colaboradores, inspeção constante e comprometimento da equipe. Como pontos negativos ou a melhorar a equipe considerou a dependência do *scrum master*, seja pela necessidade de conduzir o grupo, seja pela disseminação da metodologia. A falta de um treinamento formal, também foi considerado um ponto negativo, além da documentação, a resistência a

mudanças, e a falta de dias específicos para testes. Esse último, foi minimizado no segundo projeto piloto, com a ampliação da *sprint* e dias específicos para testes. Foi comentado também sobre a necessidade de ter disponível um conhecedor do negócio, fato que foi aperfeiçoado também no segundo projeto, através da inclusão do papel do POP. Em relação a iniciar um projeto desde o início com a metodologia, isso se deve ao fato de ambos os projetos terem iniciados com o DAP já elaborado.

- E1: "...são vários os benefícios da utilização da metodologia. O envolvimento diário da equipe permitiu um rápido entendimento do sistema e geração de soluções conjuntas, enfim, um maior comprometimento da equipe. Com relação a problemas, é uma metodologia que exige um *scrum master* dinâmico e com liderança. Também, se não houver pelo menos um grande conhecedor do negócio a equipe pode ficar perdida..."
- E2: "... a metodologia em si é ótima, o que pode ser considerado um contra é que ao se focar na agilidade de desenvolvimento, se perde bastante em documentação. De resto, é tudo muito bom. A integração da equipe, ao meu ver, é a grande vantagem das metodologias ágeis..."
- E3: "...faltou iniciar um projeto desde o início utilizando a metodologia. Também sinto falta de treinamento com a participação de uma consultoria para guiar o processo como um todo..."
- E4: "...a metodologia se mostrou um sucesso e em pouco tempo teve o efeito de agregar as pessoas e os resultados fluiram de modo satisfatório. O projeto conseguiu elogio dos usuários aos quais eram endereçados..."
- E5: "...a experiência foi muito válida, pois mudou totalmente a forma de trabalho para melhor. Os projetos foram desenvolvidos de forma mais ágil e prazos foram atingidos e as demandas do cliente foram satisfeitas. O contra foi só a não definição de um tempo para testes dentro da *sprint*..."
- E6: "...prós: trabalho em equipe; maior envolvimento entre os membros da equipe; passagem de conhecimento; otimização de atividades; diminuição de ruídos que atrapalham o andamento das atividades. Contras: resistência a mudanças, falta de treinamento da equipe, vinculação extrema ao *scrum master*. Avaliação: aprovo a manutenção da metodologia."
- E7: "... como ponto positivo avalio a diminuição dos conflitos, melhoria na comunicação do grupo, maior integração do time e o contato diário, que proporciona inspeção constante das tarefas e do produto desenvolvido. Considero ainda que para melhor condução do processo todos os colaboradores da unidade de sistemas sejam inseridos em um treinamento focado nessa abordagem."

- E8: "...a abordagem ágil propiciou a constante avaliação das atividades e trouxe o foco da equipe para os objetivos do projeto, além de unir a equipe numa finalidade..."

4.3 Aplicação da *Grounded Theory*

A *Grounded Theory* (GT) é um método indutivo de pesquisa qualitativa, desenvolvida por Glaser e Strauss que utiliza um conjunto de dados coletados para gerar uma teoria fundamentada nesses conjunto de dados [11]. Strauss e Corbin definem GT como um conjunto sistemático de procedimentos para desenvolver indutivavemente uma teoria fundamentada nos dados a respeito de um determinado fenômeno.

O caminho percorrido pela GT é o de ler e interpretar dados, com o objetivo de descobrir categorias de conceitos e os seus relacionamentos, agrupar esses dados em um nível maior de abstração e em seguida gerar teorias a respeito das abstrações encontradas [36][45]. Apesar de não haver limitações dos tipos de dados aplicados à metodologia, a grande maioria de suas aplicações envolvem dados de questionários e entrevistas.

Toda a análise de dados no GT deve ser realizada por mais de um pesquisador, para evitar viés [45]. Assim, o resultado de cada etapada deve partir de um consenso entre os pesquisadores envolvidos.

A escolha da GT se deve à abordagem qualitativa desse estudo, por oferecer validade científica e o rigor necessário à análise dos dados e por fornecer teorias objetivadas pelo ambiente da pesquisa e interpretadas pelo próprio pesquisador [28].

Existem alguns trabalhos publicados na área de métodos ágeis que utilizam GT. Viviane Santos e Alfredo Goldman aplicaram essa metodologia em busca de melhorias em um curso de *Extreme Programming*, denominado Laboratório XP [45].

Marum Filho *et al.* realizou um estudo sobre *Scrum* em pequenas equipes, combinando a técnica de estudo de caso com GT [25]. O objetivo do uso da GT no estudo foi o de identificar relacionamentos entre os vários aspectos que envolveu a experiência da utilização do *Scrum*.

A estrutura da GT é formada por três etapas: a codificação aberta, a codificação axial e a codificação seletiva e serão detalhadas nas seções a seguir, em conjunto com a aplicação de cada fase nos dados dessa pesquisa.

4.3.1 Codificação aberta

Nessa fase são identificados os códigos baseado nos termos utilizados pelos entrevistados. O objetivo é a busca de categorias conceituais que possam ajudar a explicar ou entender o que os dados informam. Uma característica importante dessa etapa é que o

pesquisador é induzido a analisar cada frase, linha e parágrafo do questionário [21]. Essa análise aumenta a sensibilidade do pesquisador, auxiliando no entendimento do significado dos dados [21].

Conforme o protocolo estabelecido pela GT, é necessária a análise por mais de um pesquisador, e assim foi convidado um outro pesquisador, ex-aluno do PPCA, para auxiliar na utilização do método [45][11].

Após a análise de todos os questionários e consenso entre os pesquisadores, foram identificadas 37 (trinta e sete) categorias conceituais.

A Tabela 4.1 apresenta as categorias conceituais encontradas, um identificador sequencial e a quantidade de ocorrência nas entrevistas, representada pela coluna com o símbolo sustenido. As categorias estão apresentadas na tabela de acordo com a ordem de aparição nas entrevistas.

4.3.2 Codificação axial

A codificação axial pode ser definida como "um conjunto de procedimentos no qual os dados são colocados de volta, em um novo formato, através da realização de conexões entre as categorias". Essa fase se inicia através da identificação de relacionamentos entre as categorias conceituais. A identificação é realizada através de uma leitura minuciosa da resposta dos entrevistados. Como exemplo da identificação desses relacionamentos podemos citar o seguinte trecho de um questionário: "O envolvimento diário da equipe permitiu um rápido entendimento do sistema e geração de soluções conjuntas, enfim, um maior comprometimento da equipe". Nesse trecho podemos identificar as seguintes categorias conceituais relacionadas: envolvimento da equipe, entendimento do sistema e comprometimento.

Após o reconhecimento dessas relações, pode ser definida uma redução conceitual dessas categorias, de forma a reorganizá-las em uma categoria de maior abstração, denominada categoria nocional. Os códigos existentes que estão inter-relacionados e ligados a um mesmo tema são agrupados dentro dessa nova categoria. As categorias nocionais estabelecidas podem ser visualizadas na Tabela 4.2.

Com vistas ao melhor entendimento das categorias nocionais, podem ser escritas narrativas que envolvam as relações estabelecidas entre as suas respectivas categorias conceituais:

- O *Scrum* através do "desenvolvimento iterativo e incremental" da "*sprint*" proporciona um maior "dinamismo", "inspeção","fluidez do trabalho" e uma melhor "distribuição do trabalho" fazendo com que o "incremento funcional do produto" tenha "entrega no prazo" e promovendo "satisfação ao cliente"

Tabela 4.1: Categorias conceituais.

ID	Categoria Conceitual	#
1	Programação em par	10
2	Produtividade	4
3	Qualidade do código	6
4	Trabalho em equipe	8
5	Comunicação	3
6	Distribuição do trabalho	2
7	*Framework*	8
8	Ambiente organizacional	2
9	Comprometimento	3
10	Capacidade de iniciativa	2
11	Disponibilidade do cliente	3
12	Envolvimento da equipe	3
13	Transferência de conhecimento	3
14	Resistência a mudanças	2
15	Treinamento adequado da equipe	4
16	Dependência do *scrum master*	3
17	Abordagem ágil	6
18	*Scrum*	9
19	Incremento funcional do produto	2
20	Demandas externas	3
21	Colaboração	2
22	Desenvolvimento iterativo e incremental	2
23	Interação entre as pessoas	6
24	Satisfação do cliente	4
25	Troca de experiências	2
26	Greve	2
27	Testes	2
28	Reunião diária	2
29	Tomada de decisões em equipe	4
30	Entendimento do sistema	6
31	Concentração	1
32	Dinamismo	7
33	Inspeção	1
34	Fluidez do Trabalho	2
35	Entrega no prazo	3
36	Mudança da Direção do CPD/UnB	2
37	Satisfação	8

Tabela 4.2: Categorias conceituais agrupadas em categorias nocionais.

ID	Categoria Nocional	Categoria Conceitual	#
10	Abordagem ágil	Capacidade de iniciativa	2
11	Abordagem ágil	Disponibilidade do cliente	3
14	Abordagem ágil	Resistência a mudanças	2
15	Abordagem ágil	Treinamento adequado da equipe	4
16	Abordagem ágil	Dependência do *scrum master*	3
17	Abordagem ágil	Abordagem ágil	6
20	Abordagem ágil	Demandas externas	3
21	Abordagem ágil	Colaboração	2
23	Abordagem ágil	Interação entre as pessoas	6
26	Abordagem ágil	Greve	2
27	Abordagem ágil	Testes	2
29	Abordagem ágil	Tomada de decisões em equipe	4
36	Abordagem ágil	Mudança da Direção do CPD/UnB	2
37	Abordagem ágil	Satisfação	8
1	Comunicação	Programação em par	10
4	Comunicação	Trabalho em equipe	8
5	Comunicação	Comunicação	3
8	Comunicação	Ambiente organizacional	2
9	Comunicação	Comprometimento	3
12	Comunicação	Envolvimento da equipe	3
13	Comunicação	Transferência de conhecimento	3
25	Comunicação	Troca de experiências	2
28	Comunicação	Reunião diária	2
30	Comunicação	Entendimento do sistema	6
31	Comunicação	Concentração	1
2	Fatores Tecnológicos	Produtividade	4
3	Fatores Tecnológicos	Qualidade do código	6
7	Fatores Tecnológicos	*Framework*	8
6	*Scrum*	Distribuição do trabalho	2
18	*Scrum*	*Sprint*	9
19	*Scrum*	Incremento funcional do produto	2
22	*Scrum*	Desenvolvimento iterativo e incremental	2
24	*Scrum*	Satisfação do cliente	4
32	*Scrum*	Dinamismo	7
33	*Scrum*	Inspeção	1
34	*Scrum*	Fluidez do Trabalho	2
35	*Scrum*	Entrega no prazo	3

- A “abordagem ágil” promoveu maior “satisfação", “interação entre as pessoas”, “colaboração”, apoiou a “tomada de decisões em equipe” e incentivou a “capacidade de iniciativa”, mas foi prejudicada por “resistência a mudanças”, por “demandas externas”, pela “disponibilidade do cliente”, pela “mudança de direção do CPD/UnB”, pela falta de “testes” e pela falta de um “treinamento adequado da equipe” que culminou na dependência do “*scrum master*”
- O “envolvimento da equipe” na “reunião diária” e na “programação em par” favorecem a “comunicação”, o “comprometimento”, a “troca de experiências”, um melhor “entendimento do sistema” e a “transferência do conhecimento” corroborando assim para o bom andamento do “trabalho em equipe”.
- Problemas com o “*framework*”’ ocasionaram na perda de “qualidade no código” e influenciou na “produtividade” da equipe.

Com as categorias nocionais definidas, a técnica de comparação constante é aplicada, cruzando os dados dos questionários e estabelecendo as dimensões de similaridades e diferenças dessas categorias.

A Figura 4.3 apresenta os dados identificados para as dimensões das categorias nocionais. As similaridades estão apresentadas na caixa de cor verde e dizem respeito a um padrão ou consenso encontrado nas respostas do questionários. As diferenças, apresentadas na caixa de cor laranja, estão ligadas às informações que apesar de estarem presentes na resposta de uma determinada pergunta, não tem rastreamento em outras respostas dessa mesma pergunta.

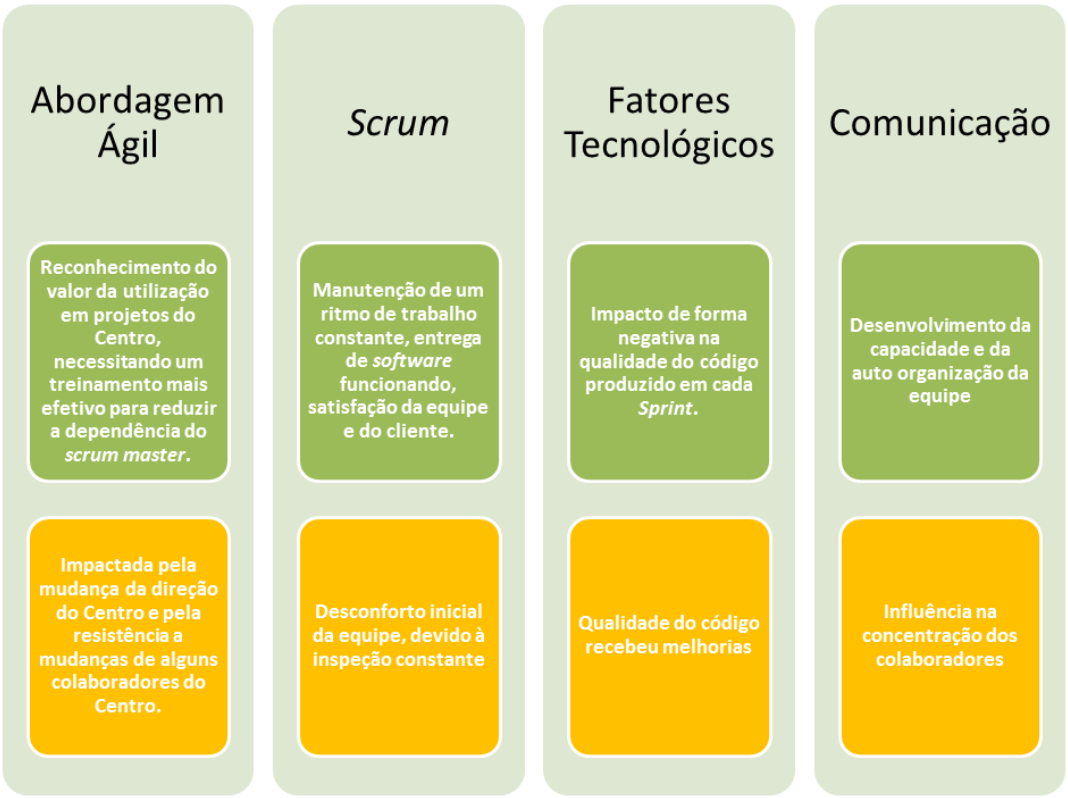

Figura 4.3: Dimensões das categorias nocionais.

4.3.3 Codificação seletiva

A fase de codificação seletiva tem como meta definir, de forma indutiva, uma categoria central, que possa integrar todas as categorias nocionais identificadas. Basicamente, o pesquisador deverá identificar qual o tema principal da pesquisa [36].

Assim, as categorias nocionais apontadas no estudo "Abordagem Ágil", "*Scrum*", "Fatores Tecnológicos", "Comunicação" convergem para um conceitor maior, que de forma indutiva, pode ser definido como "Elementos metodológicos e fatores de sucesso da implantação de métodos ágeis no CPD/UnB".

De maneira similar ao estudo [45], essa técnica de codificação foi aplicada para obter uma visão geral da implantação de métodos ágeis no CPD/UnB. Sendo assim, a visão geral da implantação pode ser expressada da seguinte forma:

- **A abordagem ágil implantada teve sua importância reconhecida pelas equipes envolvidas nos projetos piloto, pois proporcionou colaboração e interação entre as pessoas e favoreceu a tomada de decisões em equipe. Destaca-se nessa abordagem a utilização do *Scrum*, que através do caráter dinâmico de suas curtas iterações, trouxe satisfação ao cliente, cumpri-**

mento dos prazos acordados e a entrega de um incremento de produto funcional a cada ciclo. Contudo, a equipe requer capacitação em um treinamento nessa abordagem, com vistas a diminuir a dependência do *scrum master* no processo.

Por fim, de acordo com os critérios apresentados por Glaser [57][12], a utilização do protocolo da GT, baseada em métodos de comparação constantes, assegura que apenas conceitos válidos são apresentados ao final da execução do estudo. De acordo com o autor, a sugestão de triangulação dos resultados, ou seja, validação desses resultados por mais dois outros métodos de pesquisa, poderá ser necessária quando há necessidade de generalização, o que não é o caso desse estudo. Assim sendo, as teorias apresentadas nas fases de codificação axial e codificação seletiva possuem validade, haja vista a execução completa do protocolo GT.

Capítulo 5

Conclusão e trabalhos futuros

O objetivo principal desse trabalho foi realizar a adoção de métodos ágeis no CPD/UnB, uma instituição que utilizava um processo de desenvolvimento de *software* baseado em uma metodologia tradicional. Para alcançar esse objetivo, foram estudados métodos tradicionais e ágeis de desenvolvimento, foi realizada uma pesquisa a respeito do Centro de Informática e da área de desenvolimento de sistemas desse Centro.

Ademais, foi definida uma abordagem ágil de desenvolvimento, que foi aplicada através do estudo de caso envolvendo dois projetos piloto do Centro. O estudo de caso foi apresentação no Capítulo 3 e a abordagem ágil validada através desses projetos foi apresentada na Seção 4.1.

Após a execução dos projetos piloto, um questionário foi aplicado com o objetivo de avaliar a aplicação dessa abordagem ágil sob a percepção da equipe envolvida. A análise descritiva desse questionário está disponibilizada na Seção 4.2 e a avaliação qualitativa através da *Grounded Theory* encontra-se na Seção seguinte, a 4.3.

Os resultados obtidos através da aplicação da GT demonstram que, sob a ótica da equipe, a nova abordagem aumentou a capacidade de trabalho em equipe, melhorou a comunicação, aumentou a satisfação da equipe e proporcionou satisfação ao cliente. A satisfação do cliente esteve amplamente relacionada ao cumprimento de prazos e à entrega de um produto funcional a cada ciclo.

Além disso, de acordo com as teorias produzidas como resultado do protocolo GT, percebe-se que a equipe demonstra dependência do papel do *scrum master* para execução da abordagem e por isso almeja um treinamento específico nos métodos ágeis que a conceituam.

Percebe-se que essa dependência do *scrum master* relatada nos questionários pode vir a ser um problema, se formos considerar a característica de equipes auto organizadas do *framework Scrum*. O papel do *scrum master* deve ser de facilitador, orientando a equipe

sobre o processo e garantindo que ele aconteça. Como o trabalho será executado é uma atribuição da equipe de desenvolvimento

Um outro ponto a ser observado é que a dinâmica proposta pela abordagem ágil requereu que a equipe de desenvolvimento não esperasse pela equipe da plataforma de desenvolvimento na solução dos problemas técnicos apresentados na plataforma. Assim, em algumas situações, o desenvolvimento foi realizado fora dos padrões de código dos projetos do Centro, o que prejudicou a qualidade do código dos projetos.

Por fim, esse trabalho objetivou também responder a seguinte questão de pesquisa, sob o ponto de vista da equipe: "Quais os impactos da utilização de uma abordagem ágil no Centro de Informática da Universidade de Brasília?". Essa questão pode ser respondida com o auxílio das categorias e teorias produzidas pela *GT*:

- A abordagem ágil teve seu valor reconhecido e gerou satisfação à equipe na sua utilização.
- Interação entre as pessoas, colaboração, tomada de decisões em equipe, capacidade de iniciativa foram abordados como fatores de melhoria na equipe.
- A dependência do papel do *scrum master* foi citado como um fator negativo na abordagem, mas que segundo o resultado da análise, pode vir a ser minimizada com uma capacitação focada no LSD, *Scrum* e XP.
- A utilização do *framework Scrum* promoveu um ambiente mais dinâmico, dando fluidez ao trabalho da equipe. Além disso, a entrega de um incremento de produto funcional, no prazo previsto, a cada ciclo, promove satisfação ao cliente do projeto.
- O evento da reunião diária, bem como a prática XP de programação em par são reconhecidos pelo envolvimento da equipe e se relacionam com a melhora da comunicação, do comprometimento e da transferência do conhecimento no projeto.
- A plataforma de desenvolvimento de sistemas do CPD/UnB impactou na qualidade do código dos projetos desenvolvidos no estudo de caso.

Todos esses impactos gerados na utilização da abordagem podem ser confirmados no estudo de caso, através do *feedback* da equipe e do cliente relatados nos eventos do *Scrum* e nas lições aprendidas dos projetos piloto.

5.1 Contribuições

Essa pesquisa contribuiu para que o CPD/UnB pudesse experimentar uma nova abordagem para desenvolvimento de sistemas, que foi considerada uma experiência positiva

pelas equipes envolvidas nos projetos que a utilizaram. Contribui ainda com o conhecimento gerado acerca da implantação de métodos ágeis, principalmente em instituições que possuem um processo baseado em métodos tradicionais em vigor, através do relato dessa experiência no estudo de caso.

Além disso, a realização dessa pesquisa possibilitou a inserção do CPD/UnB como participante do grupo que irá definir uma estratégia ágil para o Sistema de Administração dos Recursos de Tecnologia da Informação (SISP). Esse trabalho é coordenado pela Secretaria de Logística e Tecnologia da Informação do Ministério do Planejamento, Orçamento e Gestão (SLTI/MPOG) e visa apresentar um guia de boas práticas de metodologia ágil, baseando-se na experiência de instituições que já utilizaram esses métodos.

5.2 Limitações

De acordo com o objetivo proposto, o estudo foi realizado em apenas uma instituição, com uma amostra pequena de envolvidos, o que limita a possibilidade de generalização da solução adotada. Contudo, por relatar com detalhes a experiência da implantação, o estudo de caso, bem como a abordagem ágil definida, poderão servir como referência para outras instituições que pretendam fazer a adoção de métodos ágeis, minimizando as dificuldades que podem ser encontradas nessa adoção.

5.3 Trabalhos futuros

Com o crescimento da adoção de métodos ágeis, sugere-se a replicação da abordagem ágil aplicada em instituições públicas e privadas, para que seja realizado um estudo comparativo sobre a implantação nessas instituições e no CPD/UnB.

Focalizando o CPD/UnB a pesquisa pode ser ampliada, de forma a abarcar a relação entre os times *Scrum* e a equipe da plataforma de desenvolvimento. Um outro ponto a ser colocado é a utilização de outras práticas XP na abordagem, como a de testes automatizados, que não pôde ser inserida na abordagem, devido à limitação técnica da equipe.

Além disso, de forma a complementar a pesquisa qualitativa realizada, novos estudos com implantação de métodos ágeis poderão ser realizados, com vistas a uma abordagem quantitativa. Esse novo estudo poderá ter ênfase, por exemplo, na comparação com projetos desenvolvidos em outras abordagens, com base em variáveis extraídas a partir do código-fonte dessas aplicações.

Referências

[1] V.S. Alves dos Santos e E. Dias Canedo. Agile methodology in the software development: Case study: Electoral justice of brazil. In *Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on*, pages 1–6, June 2014. 32

[2] Scott Ambler. Agile modeling. http://www.agilemodeling.com//, 2014. Acesso em: 2014-10-18. 28, 29

[3] Maurício Finavaro Aniche. Como a prática tdd influencia o projeto de classes em sistemas orientados a objetos. Mestrado, IME, Instituto de Matemática e Estatística, USP, 2012. 34, 35

[4] Dairton Luiz Bassi Filho. Experiências com desenvolvimento ágil. Mestrado, IME, Instituto de Matemática e Estatística, USP, 2008. x, 2, 6, 10, 11, 13, 14, 21, 22, 23, 24, 26, 31, 43, 46, 68

[5] Kent Beck. *Extreme Programming Explained: Embrace Change,*. Addison-Wesley, 1 edition, 1999. 22, 25, 26, 27

[6] Kent Beck e Cynthia Andres. *Extreme Programming Explained: Embrace Change,*. Addison-Wesley, 2 edition, 2004. 22, 23, 24

[7] B. W. Boehm. A spiral model of software development and enhancement. In *Computer (Volume: 21, Issue: 5)*, pages 61–72, 1988. 10

[8] Grady Booch et al. *UML: guia do usuário*. Elsevier, 2 edition, 2005. 11

[9] P. Bourque e R.E. Fairley. Guide to the software engineering body of knowledge, 2014. 4

[10] Rafael Buzon. Escalando o papel do product owner. http://blog.kudoos.com.br/agile/escalando-o-papel-do-product-owner/. Acesso em: 2014-08-22. 59

[11] J. Corbin e A.C. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, Inc, 3 edition, 1974. 2, 35, 74, 75

[12] J. Corbin e A.C. Strauss. *Basics of grounded theory analysis: Emergence vs. forcing*. Glaser, B. G., 3 edition, 1992. 80

[13] Hugo Corbucci et al. Métodos ágeis no brasil: estado da prática em times e organizações. In *Relatório Técnico RT MAC 2012-03*, May 2012. 1, 2, 44

[14] Leonardo Costa. O ciclo de vida do framework scrum. http://www.semeru.com.br/blog/o-ciclo-de-vida-do-framework-scrum/. Acesso em: 2014-08-12. x, 20

[15] Mike Cottmeyer. Product owner by proxy. http://www.leadingagile.com/2009/03/product-owner-by-proxy/, 2009. Acesso em: 2014-09-26. 59

[16] Ana Brasil Couto. *CMMI – Integração dos Modelos de Capacitação e Maturidade de Sistemas*. Moderna, 2007. 1

[17] CPD/UnB. http://www.cpd.unb.br/images/processos/pts/. http://www.cpd.unb.br/images/processos/pts//, 2008. Acesso em: 2014-09-18. x, 41

[18] CPD/UnB. Ped - processo de estratégia de dados. http://www.cpd.unb.br/images/processos/ped/, 2008. Acesso em: 2014-09-18. x, 41

[19] CPD/UnB. Pts - processo de teste de software. http://www.cpd.unb.br/images/processos/pds/, 2008. Acesso em: 2014-09-18. x, 39

[20] CPD/UnB. Ferramenta de acompanhamento de projetos: Web2project. https://www.projetoscpd.unb.br, 2010. Acesso em: 2014-09-15. 40

[21] Genésio Cruz Neto, Alex Gomes, e Natália Oliveira. Aliando grounded theory e re-formulações de conceitos da teoria da atividade para o melhor entendimento de práticas humanas. In *Workshop on perspectives, challenges and opportunities for Human-Computer Interaction in Latin American*, Sep 2007. 75

[22] Tribunal de Contas da União. Levantamento de auditoria realizado pela secretaria de fiscalização de tecnologia da informação – sefti. conhecimento acerca da utilização de "métodos ágeis" nas contratações para desenvolvimento de software pela administração pública federal. acórdão 2314-33/13-p, 2013. 2, 8, 10, 12, 32

[23] W. Edwards Deming. *Out of the Crisis*. Productivity Press, 1986. 13, 22

[24] Tore Dybå e Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. In *Information and Software Technology*, page 833–859, 2008. 34

[25] S. Filho Marum, M. Celso, G. Nauber, e A.A. Bessa. Using scrum in small teams: Combining case study with grounded theory. In *Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on*, pages 1–6, June 2014. 46, 65, 74

[26] Carlos Flores e Marta Bez. Proxy product owner - a função do gerente de projetos de software utilizando métodos ágeis em equipes geograficamente distribuídas. In *Computer on the Beach 2014*, pages 1–10, March 2014. 58

[27] M. Fowler e V. Highsmith. The agile manifesto. In *Software Development Magazine*, pages 29–30, August 2001. x, 1, 14, 15, 34

[28] Kátia R. Hopfer e Sandra M. Maciel-Lima. Grounded theory: avaliação crítica do método nos estudos organizacionais. In *Revista da FAE*, 2008. 74

[29] Ivar Jacobson, Grady Booch, e James Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999. 11

[30] Henrik Kniberg. *Scrum e XP direto das Trincheiras: Como fazemos Scrum*. C4Media Inc, 2007. 21, 47

[31] Henrik Kniberg e Mattias Skarin. *Kanban e Scrum - obtendo o melhor de ambos*. C4Media Inc, 2009. 45

[32] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison Wesley, 3 edition, 2003. 2, 11

[33] Craig Larman. *Utilizado UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo*. Bookman, 3 edition, 2007. 11, 12

[34] Steve McConnell. Managing technical debt. In *10x Software Development*, 2012. 31

[35] J.D. Meier. The four circles of xp (extreme programming). http://blogs.msdn.com/b/jmeier/archive/2010/04/04/the-four-circles-of-xp-extreme-programming.aspx. Acesso em: 2014-09-25. x, 26, 28

[36] R. Mello e C. Cunha. Operacionalizando o método da ground theory nas pesquisas em estratégia: técnicas e procedimentos de análise com apoio do software atlas/ti. In *I Encontro de Estudos em Estratégia (3Es), 2003 ANPAD*, May 2003. 74, 79

[37] Claudia Melo e Gisele Ferreira. Adoção de métodos ágeis em uma instituição pública de grande porte - um estudo de caso. In *Brazilian Workshop for Agile Methods (WBMA 2010)*, pages 112–125, 2010. 2, 32, 33, 46

[38] Igor Pereira Muzetti. Desenvolvendo software inovador em universidades públicas: Adaptando processos ágeis para a realidade de laboratórios de pesquisa e desenvolvimento. Mestrado, ICEB, Instituto de Ciências Exatas e Biológicas, UFOP, 2014. x, 2, 12, 13, 14, 31, 46, 61

[39] Taiichi Ohno. *Toyota Production System: Beyond Large-Scale Production*. Productivity Press, 1988. 21

[40] M. Poppendieck e M.A Cusumano. Lean software development: A tutorial. *Software, IEEE*, 29(5):26–32, Sept 2012. 21, 62

[41] Roger Pressman. *Engenharia de software: uma abordagem profissional*. AMGH, 7 edition, 2011. x, 4, 5, 6, 7, 8, 9, 10, 11, 13, 40

[42] Rafael Prikladnicki et al. *Métodos ágeis para desenvolvimento de software*. Bookman, 2014. 21, 28, 29, 59

[43] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *ICSE '87 Proceedings of the 9th international conference on Software Engineering*, pages 328–338, 1987. 7

[44] Per Runeson e Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009. 34

[45] Viviane A. Santos, Alfredo Goldman, e Carlos D. Santos. Uncovering steady advances for an extreme programming course. *CLEI Electron. J.*, 15(1), 2012. 74, 75, 79

[46] Danilo Toshiaki Sato. Uso eficaz de métricas em métodos Ágeis de desenvolvimento de software. Mestrado, IME, Instituto de Matemática e Estatística, USP, 2007. 31

[47] Ken Schawber e Jeff Sutherland. Um guia definitivo para o scrum: As regras do jogo. Technical report, July 2013. 16, 17, 18, 19, 20, 44

[48] Carolyn Seaman e Yuepu Guo. Measuring and monitoring technical debt. In *Advances in Computers, 82*, pages 25–46, 2011. 30

[49] C.B. Seaman. Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, 25(4):557–572, Jul 1999. 34

[50] C. Selltiz, D. Jahoda, e M. Deutsch. *Métodos de Pesquisas nas Relações Sociais*. EDUSP, 1974. 34

[51] Alexandre Freire da Silva. Ensino de programação extrema na academia, indústria e governo. Mestrado, IME, Instituto de Matemática e Estatística, USP, 2007. 31

[52] Bruno Siqueira Silva, Gustavo Robichez Carvalho, e Otávio Albuquerque Ritter Santos. Adaptação na prática de um setor público às metodologias Ágeis. Graduação, DI, Departamento de Informática, PUC-RIO, 2013. 32

[53] Ian Sommerville. *Engenharia de software*. Pearson Education, 9 edition, 2011. x, 1, 4, 5, 6, 7, 8, 40

[54] H. Takeuchi e I. Nonaka. The new new product development game. In *Harvard Business Review*, February 1986. 16

[55] Graziela S. Tonin et al. Uma análise de dívida técnica em uma empresa de tecnologia com desenvolvimento baseado em scrum. In *3rd Brazilian Workshop on Agile Method*, 2012. 31

[56] Ari do Amaral Torres Filho. Análise da utilização de métodos ágeis no desenvolvimento de ambientes virtuais de aprendizagem: um estudo de caso do solar 2.0. Mestrado, CC, Centro de Ciências, UFC, 2014. 31, 32, 46, 65

[57] Elizabeth Whitworth. Agile experience: Communication and collaboration in agile software development teams. Mestrado, Department of Psichlogy, Carleton University, Ottawa, Ontario, Canada, 2006. 80