UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE DESENVOLVIMENTO SOLO DE APLICAÇÕES UTILIZANDO PRINCÍPIOS ÁGEIS

ERICK VANSIM PREVIATO

ORIENTADOR: PROF. DR. DELANO MEDEIROS BEDER

São Carlos – SP

Fevereiro/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE DESENVOLVIMENTO SOLO DE APLICAÇÕES UTILIZANDO PRINCÍPIOS ÁGEIS

ERICK VANSIM PREVIATO

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

São Carlos – SP

Fevereiro/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Erick Vansim Previato, realizada em 06/04/2018:

Prof. Dr. Delano Medeiros Beder
UFSCar

Prof. Dr. Auri Marcelo Rizzo Vincenzi
UFSCar

Prof. Dr. Marcos Lordello Chaim
USP

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) *Marcos Lordello Chaim* e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

Prof. Dr. Delano Medeiros Beder

Dedico este trabalho aos meus pais, irmão, minha esposa Patrícia Sartori, minha afilhada e a todos os meus amigos que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa da minha vida.

AGRADECIMENTOS

Agradeço primeiramente à Deus que me deu forças para que eu seguisse até o fim dessa jornada. Agradeço também ao Professor Delano Medeiros Beder, pela orientação e companheirismo em todas etapas deste trabalho.

Aos colegas Danilo Coimbra e Maycon Santana pela ajuda que me deram na parte inicial da escrita, principalmente na etapa de qualificação, no qual não mediram esforços para que eu concluísse o que comecei.

À Professora Solange Oliveira Rezende pela pronta disposição em me ajudar sempre que precisei, tanto na parte emocional quanto na parte profissional.

Aos colegas de trabalho do ICMC-USP, em especial ao Leonardo José Martinussi que me ajudou com a correção da escrita, e ao Carlos Eduardo Favaro que ajudou na aplicação da proposta deste trabalho. Aos colegas da UFSCar que me auxiliaram nas disciplinas do programa, e me deram apoio para continuar sempre aprendendo mais.

Agradeço também a minha esposa, que além de ser uma excelente companheira, sempre acreditou em mim, e me ajudou em tudo que precisei.

Por fim agradeço a todas as pessoas que de algum modo fizeram parte dessa etapa decisiva da minha vida.

Gostaria de uma sociedade mais justa, menos corrupta, com menos hipocrisia, mais digna, com mais amor ao próximo, menos preconceito, menos rancor e principalmente mais paz na alma.

Albert Einstein

RESUMO

O mercado de desenvolvimento de aplicações vem crescendo com o passar dos anos, com isso, empresas buscam a todo tempo adotar metodologias de desenvolvimento que aumentem a produtividade e a interação da equipe, diminuam o gasto e o tempo de desenvolvimento, tudo isso sem perder a qualidade do produto final. Além disso, vários indivíduos tentam entrar nesse mercado com alguma ideia de um possível produto, porém, não possuem recursos necessários para contratação de uma equipe. Pensando nesse cenário, este trabalho propõe um estudo das metodologias de desenvolvimento solo que seguem princípios ágeis, e também propõe uma nova abordagem de desenvolvimento solo. Como resultado é apresentado se é possível adotar metodologias ágeis no desenvolvimento solo de software. Também apresenta-se uma nova abordagem de desenvolvimento com base nas metodologias ágeis *Scrum* e XP, porém adaptado para uso individual. Com o intuito de validar a proposta, dois projetos piloto foram elaborados para realizar a aplicação desta metodologia, e os resultados dessa aplicação serão explorados com o intuito de saber se tal abordagem é viável ou não.

Palavras-chave: Metodologias de desenvolvimento, Métodos ágeis, Desenvolvimento solo, Scrum

ABSTRACT

The application development market has been growing over the years, thereby, companies always seek to use development methodologies that increase productivity and team interaction, decrease the cost and development time, without losing the quality of the final product. In addition, several people try to work in this market, but does not have the necessary resources to hire a team. Thinking about this scenario, this paper proposes a study of the solo development methodologies that follow agile principles, and also proposes a new approach to solo development. As a result, is presented if it is possible to adopt agile methodologies in solo software development. It is also showed a new development approach based on agile Scrum and XP methodologies, but adapted for individual use. In order to validate a proposal, two pilot projects were developed for the implementation of the methodology, and the results of this application are explored in order to know whether such an approach is viable or not.

Keywords: Development Methodologies, Agile Methods, Solo Development, Scrum

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO

AUP – *Agile Unified Process*

CQP – *Comissão de Qualidade e Produtividade*

DSDM – *Dynamic Systems Development Method*

FAQ – *Frequently Asked Questions*

FDD – *Feature-Driven Development*

ICMC – *Instituto de Ciências Matemáticas e de Computação*

PSP – *Personal Software Process*

PXP – *Personal Extreme Programming*

RUP – *Rational Unified Process*

TDD – *Test-Driven Development*

TI – *Tecnologia da Informação*

T&D – *Treinamento e Desenvolvimento*

USP – *Universidade de São Paulo*

XP – *Extreme Programming*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

Com os recentes avanços na última década, inúmeros aparatos tecnológicos têm sido desenvolvidos pela indústria e setores privados ou propostos pela academia científica. Em particular, a computação tem ajudado no desenvolvimento de tais soluções, tanto no domínio de hardware quanto no de software.

No contexto de software, as soluções envolvem aplicações na área da medicina como softwares para diagnósticos, para acompanhamento de tratamento de pacientes, entre outros (SOMMERVILLE, 2010); na engenharia civil com softwares e aplicações para cálculos e projeções de custos e riscos (RIBEIRO, 2016); na área financeira com sistemas de projeções de cálculo de inflação, juros, e outros (COSTA, 2012); até a recente área denominada *Big data*, na qual uma quantidade massiva de dados é gerenciada (HEWLETT PACKARD ENTERPRISE, 2016).

A grande maioria desses softwares passou por um processo de desenvolvimento, ou seja, surgiu de uma ideia ou necessidade de alguém e essa ideia foi transformada em um sistema computacional. O gerenciamento desse processo de criação pode ter sido realizado por uma ou várias pessoas, ou até mesmo por empresas especializadas nesses tipos de aplicações.

Segundo Neto (2015), no Brasil, no ano de 2016 cerca de 15.700 empresas que atuavam no setor de software e serviços foram identificadas e divididas em três categorias: desenvolvimento e produção, distribuição e comercialização e prestação de serviços. Dentre as dedicadas ao desenvolvimento e produção, que é o setor que cria tais aplicações, foram totalizadas 4.872 empresas, que por sua vez foram divididas por tamanho, sendo que $49,2\%$ delas possuem menos que 10 funcionários.

Embora existam grandes empresas que desenvolvem sistemas informatizados, nota-se que

há um número expressivo de micro empresas que atua nesse mercado. Dentre essas, existem algumas que criam aplicações com um único desenvolvedor, seja por opção, seja por não haver recursos para a contratação de uma equipe em um momento inicial da empresa.

Além das empresas específicas de desenvolvimento, existem várias outras empresas de diversos setores que contratam pessoas na área de desenvolvimento de software, pois em meio a crises, como a dos últimos anos, empresas tendem a investir em mais sistemas que otimizem processos e diminuam custos (FLORêNCIO, 2016).

Pensando nesse cenário, um principal ponto de partida para quem vai entrar no mercado de desenvolvimento solo de software, é encontrar uma metodologia que o ajude a aumentar a produtividade, diminuindo o gasto e o tempo de trabalho, sem perder a qualidade do produto final.

Assim como adotado em desenvolvimento de softwares, é necessário adotar uma abordagem eficiente e eficaz para o desenvolvimento solo de aplicações computacionais, ou seja, quando se tem uma única pessoa por trás do projeto.

Nesse contexto, este trabalho aborda algumas metodologias de desenvolvimento de software e sua utilização e aceitação no mercado atual. Feito isso, algumas metodologias voltadas ao desenvolvimento individual e suas principais contribuições para o mercado são apresentadas.

Na literatura existem inúmeras abordagens e metodologias descritas. Na engenharia de software é comumente estabelecido o modelo de desenvolvimento denominado cascata, também conhecido como modelo tradicional. Este modelo visa elaborar um projeto de uma forma sequencial, no qual as etapas são pré-definidas – levantamento de requisitos, projeto, implementação, testes e validação e, por fim, a manutenção (SOMMERVILLE, 2010).

Uma abordagem de desenvolvimento de software atualmente em evidência é conhecida como metodologia ágil. Surgiu em 2001 quando um grupo de especialistas trocaram experiências e estabeleceram alguns princípios e valores fundamentais para um melhor jeito de se desenvolver software, criando assim o manifesto ágil: "O Manifesto Ágil é uma declaração sobre os princípios que servem como base para o desenvolvimento ágil de software" (BECK et al., 2001).

Na atualidade, várias empresas utilizam metodologias ágeis em seus projetos, pois a aceitação e o índice de sucesso aumentou nos últimos anos. De acordo com Hastie e Wojewoda (2015), um relatório denominado *Chaos Report* é publicado anualmente desde 1994 por Standish Group[1]. No relatório de 2015 foram analisados 50 mil projetos ao redor do mundo, e constatou-se que apenas 29% dos projetos tiveram sucesso nos últimos 5 anos. Ainda dentro dessa análise,

[1]Mais informações sobre Standish Group podem ser encontradas no site: https://www.standishgroup.com/about

destaca-se uma comparação feita com projetos que utilizaram metodologia tradicional e os que utilizaram metodologia ágil, no qual a porcentagem de sucesso com metodologia ágil é de 39%, e na metodologia tradicional apenas 11%. Existem vários fatores envolvidos nessa pesquisa, como tamanho e complexidade dos projetos, mas o resultado mostra que as metodologias ágeis vêm conquistando cada vez mais o mercado de desenvolvimento.

1.2 Objetivos

Com base nos dados apresentados anteriormente, no qual têm-se que as metodologias ágeis são utilizadas em grande parte do mercado, e também que existe uma necessidade de se desenvolver software de maneira solo, este trabalho apresenta um estudo sobre metodologias de desenvolvimento solo de software para atender os objetivos descritos a seguir.

O objetivo principal deste trabalho é **apresentar uma proposta de abordagem de desenvolvimento solo utilizando princípios ágeis** e descrever sua aplicação em um ambiente real de desenvolvimento. Tal abordagem é a adaptação da metodologia *Scrum* juntamente com algumas práticas e valores da metodologia *Extreme Programming* (XP). Para aplicação e avaliação da nova abordagem, dois **projetos piloto foram conduzidos**, sendo um deles na Seção Técnica de Informática do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC-USP) e outro para uma escola de futebol que realiza locação de campos na cidade de São Carlos.

Em segundo lugar, este trabalho também tem como objetivo **apresentar metodologias de desenvolvimento de software que podem ser utilizadas por uma única pessoa**, para que a mesma consiga se orientar e realizar todo o processo de criação de uma aplicação. Além disso, o trabalho busca **apresentar propostas que sigam abordagens ágeis de desenvolvimento**, mostrando que é possível utilizar metodologias ágeis sem necessariamente ter uma equipe de desenvolvimento.

Com esses objetivos traçados, o trabalho aqui apresentado busca responder às seguintes questões: É possível adaptar metodologias ágeis para desenvolver sistemas com somente uma pessoa no time de desenvolvimento? Quais opções existem para quem quer começar no mercado de desenvolvimento sem ter uma equipe? Existe metodologia certa ou errada para esse tipo de situação?

1.3 Resultados

De forma a cumprir o objetivo principal deste trabalho, apresenta-se um levantamento das metodologias de desenvolvimento, dando ênfase nas abordagens ágeis voltadas para uso individual.

Após isso, uma nova abordagem de desenvolvimento é apresentada, a qual mescla as metodologias ágeis *Scrum* e XP, porém adaptada para uso individual. Com o intuito de validar a proposta, dois projetos piloto foram elaborados para realizar a aplicação dessa metodologia, e os resultados dessas aplicações são demonstrados juntamente com a apresentação da abordagem.

1.4 Organização o texto

A dissertação segue com mais quatro capítulos.

No Capítulo 2 os conceitos fundamentais sobre desenvolvimento de software e uma discussão sobre a realidade no mercado de TI (Tecnologia da Informação) são apresentados. Também, mostra-se um breve histórico sobre as metodologias de desenvolvimento, dando enfoque na metodologia ágil.

O Capítulo 3 relata o que se tem encontrado na literatura no que tange ao uso de metodologias de desenvolvimento solo, apresentando dados de algumas das propostas elaboradas nos últimos 12 anos.

No Capítulo 4, uma nova abordagem de desenvolvimento solo é descrita. Além disso, apresenta-se dois projetos piloto para aplicação da mesma, no qual algumas aplicações foram desenvolvidas desde o começo de 2016.

E, finalmente, no Capítulo 5, contém a conclusão e as contribuições obtidas neste trabalho. Apresenta-se também, algumas ideias para trabalhos futuros.

Capítulo 2

CONCEITOS FUNDAMENTAIS

2.1 Considerações iniciais

Este capítulo trata de conceitos fundamentais e metodologias necessários para a realização deste projeto de pesquisa. A sequência deste capítulo segue com um estudo geral sobre a área de desenvolvimento de software e sua realidade atual no mercado. Em seguida, descreve-se a importância da utilização de uma metodologia de desenvolvimento de software e a evolução dos métodos nas últimas décadas. São apresentadas as metodologias tradicional, espiral, prototipação, incremental e iterativa e uma análise mais aprofundada das metodologias ágeis *Scrum*, *Extreme Programming* (XP) e a *Agile Unified Process* (AUP) são apresentadas. Por fim, as considerações finais são mostradas no final do capítulo.

2.2 Realidade do mercado de TI

Nos dias atuais é inegável que o uso de sistemas de informação dos mais variados tipos têm uma grande importância para o sucesso de qualquer instituição, seja ela pública, seja privada. Essa crescente busca pelo uso de sistemas informatizados, tem exigido cada vez mais resultados positivos com menos investimento e mais retorno (ROBERTS; SIKES, 2011).

Vários fatores tornam o desenvolvimento de software uma tarefa não trivial. As principais dificuldades encontradas na entrega do produto podem ser, por exemplo, compreender e encontrar uma maneira concisa e clara para representar o que o cliente realmente deseja, quais tecnologias e metodologias a utilizar, como estimar tempo e custo, mudanças de normas e legislações, entre outros fatores.

Baseando-se em três pilares para avaliar o sucesso do produto (tempo, orçamento e quali-

dade), o relatório *Chaos Report* é amplamente utilizado na literatura. De acordo com Hastie e Wojewoda (2015), que analisaram os dados de 2015, foram avaliados cerca de 50.000 projetos ao redor do mundo entre 2011 e 2015. Os resultados mostram que apenas 29% foram concluídos com sucesso, 52% tiveram alguma mudança de requisitos ou de funcionalidades, ou tiveram alterações de prazo ou custo, e 19% não chegaram a se tornar um produto de fato.

Como visto, uma boa parte dos projetos apresentam algum tipo de dificuldade em alguma parte do processo de desenvolvimento. Geralmente, essas dificuldades transformam-se em prejuízos tanto para o cliente quanto para o empreendedor. Vale ressaltar que, atualmente é comum encontrar equipes multidisciplinares envolvidas em um mesmo projeto, o que pode tornar-se um problema se não houver comunicação clara e objetiva entre elas.

A necessidade de eliminar os fatores que de alguma maneira influenciam negativamente no desenvolvimento de um projeto torna-se indispensável na área de engenharia de software. Nesse sentido, destaca-se a importância da utilização de uma metodologia de desenvolvimento que norteie o processo de construção do projeto de software, que por sua vez, contém um conjunto de diretrizes, conceitos e padrões a serem seguidos.

Na próxima seção apresenta-se uma discussão sobre as metodologias e sua evolução no desenvolvimento de software em geral.

2.3 Metodologias de desenvolvimento

Como mencionado na Seção 2.2, o uso de uma metodologia de trabalho para o desenvolvimento de software tem como objetivo ordenar e melhor gerenciar o processo de construção, tornando essa tarefa menos complicada (SOMMERVILLE, 2010; PRESSMAN, 2005).

Na literatura, o termo metodologia é amplamente utilizado e várias definições podem ser encontradas. Segundo Pressman (2005), metodologia é um conjunto de atividades que são necessárias para desenvolver engenharia de software com qualidade. Já Aveson e Fitzgerald (2006) definem como sendo um conjunto de técnicas, ferramentas, procedimentos, regras e documentações que auxiliam o processo de desenvolvimento de software. De maneira geral, esse termo consiste em obedecer um conjunto de passos e procedimentos previamente estabelecidos de forma a alcançar o produto final de forma organizada.

O uso de metodologias de desenvolvimento de software surgiu em meados da década de 1970, quando a metodologia tradicional, também conhecida como modelo cascata, foi primeiramente descrita por Royce (1970). Desde então, uma variedade de abordagens vêm sendo

aprimoradas e/ou criadas, cada uma com suas vantagens e desvantagens dependendo da natureza do projeto e da equipe envolvida. Antes disso, o foco de desenvolvimento de software era apenas em superar a ausência de recursos computacionais e em programação, ou seja, em entregar o sistema funcionando sem se importar com muitos aspectos importantes como satisfação do cliente, funcionalidades realmente necessárias, prazos, etc (AVESON; FITZGERALD, 2006).

A metodologia tradicional citada acima é conhecida como linear e sequencial, consistindo em etapas bem definidas na qual uma fase tem início logo após o término da outra. Contudo, essa rigidez entre as etapas pode ser considerada um problema, já que muitos sistemas em desenvolvimento frequentemente são suscetíveis a mudanças – se um requisito, por exemplo, for alterado na etapa final do projeto, todo o trabalho feito nas etapas anteriores relacionados a este requisito pode sofrer alterações (PRESSMAN, 2005). No entanto, essa abordagem serviu de base para o nascimento de muitas outras metodologias e ainda serve de inspiração para novos processos.

A busca por uma abordagem mais dinâmica, que flexibilizasse e facilitasse as alterações de interesse do cliente, impulsionou a criação de diversas metodologias de desenvolvimento ao longo dos anos. Dentre elas, pode-se citar as metodologias: incremental, espiral, prototipagem e RUP (Rational Unified Process) (SOMMERVILLE, 2010; PRESSMAN, 2005). Em 2001, após o Manifesto Ágil, começaram a surgir as metodologias conhecidas como ágeis, proporcionando mais flexibilidade ao desenvolvedor e aproximando a equipe envolvida no projeto com o cliente. Alguns dos modelos ágeis encontrados são o *Scrum* (SCHWABER, 1997), XP (BECK, 2000) e *Agile Unified Process* (AMBLER, 2018).

Um fator determinante para o êxito do projeto é sem dúvidas a escolha de qual metodologia de desenvolvimento mais adequada deve ser adotada. E isso não é diferente quando fala-se em desenvolvimento solo, o uso adequado de uma metodologia pode garantir o sucesso ou o fracasso de quem se arrisca sozinho nesse mercado competitivo.

Dessa maneira, no desenvolvimento solo, a utilização de uma única metodologia ou uma combinação de metodologias pode ser um dos caminhos a se seguir, mesmo que algumas destas metodologias tenham sido criadas pensando no desenvolvimento coletivo no qual, geralmente, as pessoas envolvidas têm responsabilidades diferentes no processo.

O capítulo segue com um breve histórico da evolução das metodologias e, logo após, apresenta-se detalhadamente algumas metodologias ágeis conhecidas e citadas na literatura que servem como base para os próximos capítulos.

2.3.1 Histórico das metodologias

A metodologia tradicional de desenvolvimento de software, também denominada na literatura como ciclo de vida clássico, é o paradigma mais antigo conhecido na engenharia de software (PRESSMAN, 2005). No início da década de 1970, essa abordagem foi proposta por Royce no artigo denominado "*Managing the development of large software systems*" (ROYCE, 1970), e tornou-se a primeira tentativa de formalizar o processo de desenvolvimento de um software.

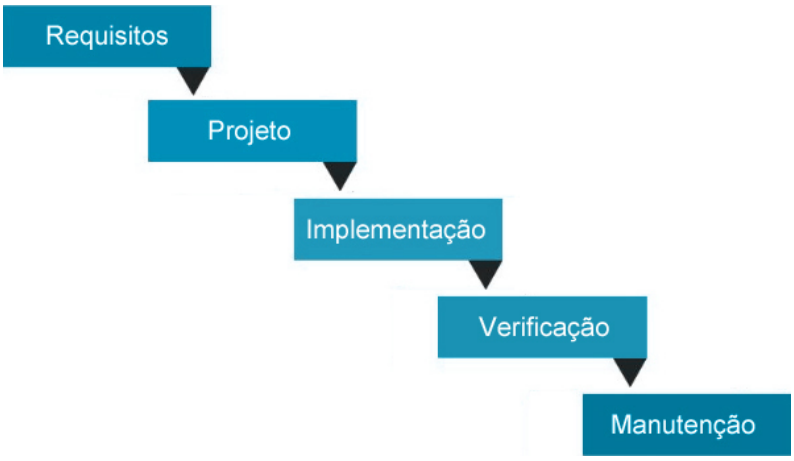
Com o avanço computacional e o aumento da complexidade dos softwares desenvolvidos, Royce definiu uma metodologia sequencial, estruturada e direcionada a manter documentação das tomadas de decisões, diferentemente de como vinham sendo desenvolvidos softwares antes disso (FRANCO, 2007).

Segundo Sommerville (2010), a metodologia tradicional é uma abordagem de desenvolvimento sequencial e seu ciclo de vida gera um encadeamento entre uma fase e outra, na qual uma fase tem início quando a anterior é finalizada. A Figura 2.1 representa essa dependência entre as fases constituindo uma estrutura em cascata – esse fato levou essa metodologia a ser também conhecida como modelo cascata. As fases, também apresentadas na Figura 2.1, são definidas da seguinte forma:

- Requisitos - O usuário ou solicitante do sistema é consultado a fim de se obter as funcionalidades, metas e restrições do produto esperado.
- Projeto - Define uma arquitetura geral do sistema, envolve tanto especificações de software quanto hardware, além de definir as abstrações fundamentais do sistema e seus relacionamentos.
- Implementação - O sistema começa a ganhar vida. Nesta etapa é realizado o desenvolvimento do programa, a partir do qual podem ser realizados testes unitários para garantir que o programa está sendo criado de acordo com os requisitos.
- Verificação - São realizados testes do software por completo com o objetivo de conferir se todos os requisitos estão de acordo com o que foi solicitado.
- Manutenção - Nesta etapa, o sistema é instalado e colocado em uso. A manutenção é então realizada assim que são descobertos erros que não foram identificados em outras fases do projeto. Esta fase também pode ser adaptativa, onde uma manutenção pode ser realizada para adequar o sistema a alguma mudança externa ao projeto como leis

e normas. Para finalizar, tem-se a manutenção evolutiva que é realizada para atender funcionalidades não previstas no projeto e que acarretará em melhor qualidade do produto final.

Figura 2.1: Modelo cascata adaptado de Sommerville (2010)



A rigidez imposta por essa sequência entre as fases traz algumas desvantagens. Uma delas é que o solicitante do software só terá uma visão de como o sistema funciona somente na fase final do projeto, e caso tenha mudança em algum dos requisitos ou em alguma regra de negócio, o mesmo pode acarretar em atraso do projeto (PRESSMAN, 2005). Isso pode até gerar um comprometimento no orçamento inicial do projeto, já que uma mudança em uma etapa tardia pode resultar em um esforço elevado da equipe de desenvolvedores.

Outra desvantagem é que essa natureza linear da abordagem faz com que alguns membros da equipe fiquem ociosos até que um outro membro finalize sua parte, ocasionando assim um "estado de bloqueio". Esse estado ocorre com mais frequência na transição entre uma etapa e outra (Bradac *apud* Pressman (2005)).

O modelo cascata originalmente proposto por Royce (1970) garante alguns "ciclos de realimentação", no qual o processo prevê algumas mudanças no andamento do ciclo. Em contrapartida, na maior parte da literatura da área, o modelo cascata é visto como estritamente sequencial, sendo interpretado assim como um modelo rígido.

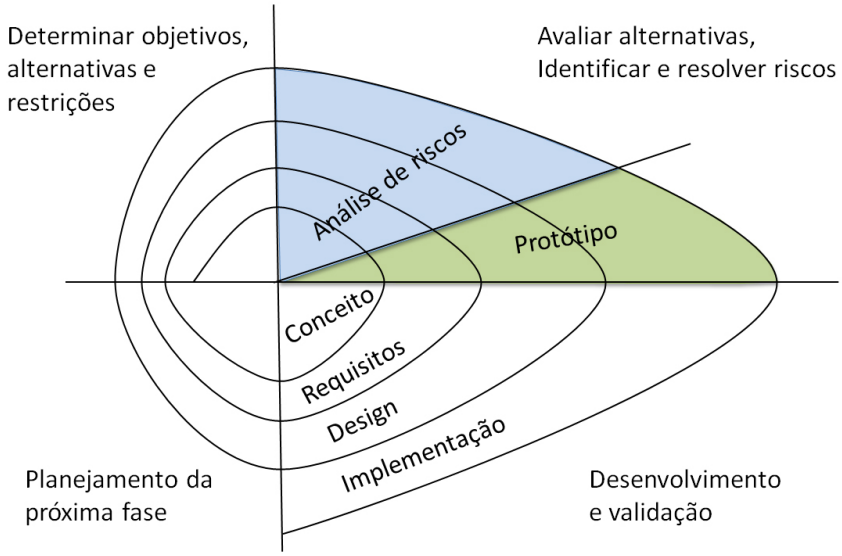
Segundo Leite (2016), nos dias atuais o mercado vem se tornando cada vez mais exigente e, com isso, respostas rápidas a mudanças de funcionalidades, características e informações são essenciais. Geralmente nesses tipos de aplicações o modelo cascata não é indicado.

Apesar dos problemas citados, a metodologia tradicional funciona muito bem quando os requisitos estão fortemente estabelecidos ou o produto final é uma atualização de um software já

em funcionamento. Nesse último caso somente a tecnologia irá mudar e não as funcionalidades e características do produto.

Como já mencionado, a natureza linear da metodologia tradicional é um dos problemas dessa abordagem, visto que não se está claro como responder aos imprevistos ou mudanças que acontecem no meio do processo. Para melhorar esse ponto, Boehm et al. (1987) apresenta a metodologia espiral, na qual cada fase da metodologia tradicional passa por uma tarefa de análise de risco e também de prototipação, como mostra a Figura 2.2.

Figura 2.2: Modelo espiral adaptado de (BOEHM et al., 1987)



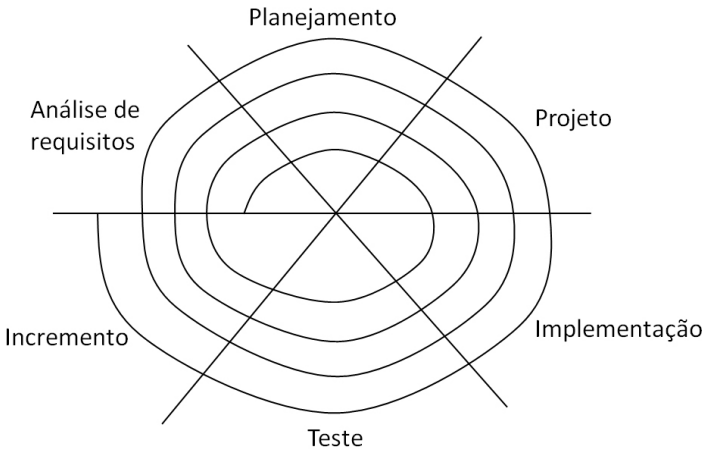
O modelo espiral divide os ciclos ou camadas em 4 etapas. A primeira etapa é a determinação dos objetivos, alternativas e restrições daquela camada, e logo após, tem-se a etapa de avaliação de riscos. Em seguida, prossegue-se para a etapa de desenvolvimento e validação, e por fim para o planejamento da próxima camada. Todas as camadas passam por estas etapas, para se ter um melhor controle do fluxo, e avaliar os riscos do processo a cada momento.

Apesar da significante melhoria apresentada pelo modelo espiral, no qual é possível detectar algum problema no decorrer do projeto, devido a sua divisão em camadas, o processo como um todo ainda continuava linear. A parte de implementação só acontece na camada de implementação, os requisitos só são trabalhados na camada de requisitos e assim por diante, ou seja, cada fase no seu tempo (SCHWABER, 1997).

Na intenção de deixar o processo mais flexível, o modelo iterativo foi especificado no documento de padrões militares do Departamento de Defesa dos Estados Unidos da América (DEFENCE, 1989). Neste modelo, em cada ciclo do processo todas as fases do modelo cascata são executadas, porém com um número reduzido de funcionalidades. Isso permitiu particio-

nar o software em vários pedaços, dando flexibilidade até para o cliente decidir o que deseja ter funcionando em um primeiro momento. No fim de cada ciclo, esse pedaço do software é adicionado ao projeto inteiro. A Figura 2.3 apresenta as fases do modelo iterativo.

Figura 2.3: Modelo iterativo adaptado de (SCHWABER, 1997)



Mesmo com essa divisão do produto, para cada iteração tem-se a mesma sequência linear do modelo cascata, ou seja, são vários pedaços de software que seguem uma certa ordem sem muita flexibilidade. Por outro lado, essa divisão apresentou um ótimo ganho, principalmente no ponto de vista do cliente. A partir dessa ideia, visando melhorar ainda mais a flexibilização do processo, os métodos ágeis foram definidos. Mais detalhes são abordados na Seção 2.3.2.

Antes de falar sobre métodos ágeis, apresenta-se o PSP (*Personal Software Process*), que é utilizado para melhorar o processo de pequenas empresas e pequenos times. O principal objetivo é auxiliar os indivíduos a terem disciplina para melhorar a eficiência, a performance e a qualidade dos projetos em que participam (HUMPHREY, 1995).

Segundo Humphrey (1995), existem alguns princípios em que o PSP se baseia para prover qualidade e melhor planejamento. São eles:

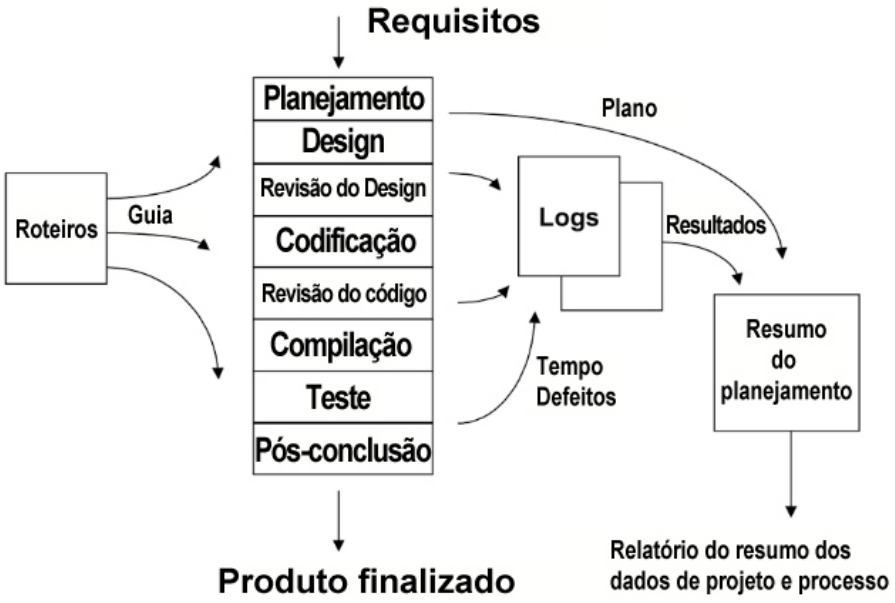
- Para ser mais efetivo, cada indivíduo deve planejar seu trabalho de modo a gerenciar seus dados pessoais de efetividade;
- Para melhorar a performance, os indivíduos precisam utilizar processos bem estruturados e mensuráveis;
- O indivíduo precisa ter consciência da responsabilidade sobre a qualidade dos produtos por ele desenvolvidos;
- Consertar erros o quanto antes é mais barato;

- Mais eficiente ainda é prever os defeitos antes de ter que consertá-los;
- O caminho mais rápido e mais barato é o correto.

A Figura 2.4 apresenta a estrutura do PSP, que por sua vez, tem início na fase de requisitos e vai até o produto finalizado. A primeira etapa desse caminho é a de planejamento. Existe um roteiro para esta etapa que é utilizado para guiar o trabalho durante a mesma. Com isso, são anotados o tempo de execução e os defeitos encontrados em formulários e todo o resultado é armazenado em uma base de planejamento.

O mesmo ocorre nas demais etapas – o trabalho é executado com base nos roteiros e toda etapa gera uma documentação de *logs* e dados sobre tempo gasto e defeitos encontrados durante o processo. Na última fase, denominada pós conclusão, todos os dados coletados são analisados para apresentar melhorias ou novas diretrizes para o processo, com o objetivo de se obter uma melhor qualidade e efetividade nos próximos projetos (HUMPHREY, 2000).

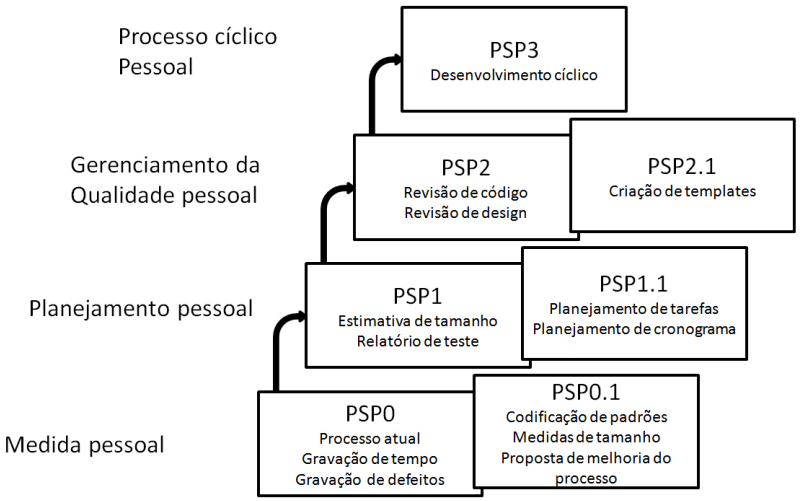
Figura 2.4: Processo PSP adaptado de (HUMPHREY, 2000)



O PSP possui um conjuntos de sete níveis de maturidade que, de forma gradativa, vão se complementando com objetivo de se obter uma melhoria contínua na qualidade do processo de desenvolvimento. Tais níveis são escalonados de PSP0 até PSP3 conforme mostra a Figura 2.5. Cada nível possui um conjunto de registros, formulários, roteiros e padrões que vão se complementando em sua evolução.

A seguir, tem-se uma breve descrição dos sete níveis de maturidade (HUMPHREY, 1995):

Figura 2.5: A evolução do PSP adaptado de (HUMPHREY, 1995)



- PSP0 - A base do processo pessoal: o desenvolvedor estabelece padrões iniciais no seu processo de produção de software, incluindo medições básicas do progresso pessoal durante o processo e definição de relatórios;
- PSP0.1: é adicionado um padrão de codificação, medição de tamanho, além de armazenar os problemas, experiências e sugestões para melhoria do processo;
- PSP1 - Planejamento pessoal: inclui etapas de planejamento, utiliza métricas para estimar o tamanho e recursos do projeto, e também relatório de testes;
- PSP1.1: adiciona cronogramas e planejamento de tarefas, a fim de monitorar o progresso do projeto;
- PSP2 - Gerenciamento de qualidade pessoal: neste nível, o desenvolvedor promove a revisão do design e do código, tornando como objetivo encontrar possíveis erros o quanto antes;
- PSP2.1: o objetivo é reduzir o número de defeitos no processo de desenvolvimento com base em *templates* e guias que o desenvolvedor elabora;
- PSP3 - Processo pessoal cíclico: o objetivo é dividir o projeto em pequenas partes, e seguindo o princípio do modelo espiral, já explicado anteriormente, realizar iterações até chegar no produto final.

Após esse breve histórico, apresenta-se como surgiram as metodologias denominadas ágeis e, também, os detalhes das abordagens ágeis que são utilizadas nessa pesquisa.

2.3.2 Metodologias ágeis

Em meados de 2001, um grupo de dezessete especialistas se reuniu para discutir aspectos importantes sobre desenvolvimento de software. A partir daí, um conjunto de valores e princípios foram estabelecidos para nortear o processo de desenvolvimento ágil de software, criando assim o chamado manifesto ágil (BECK et al., 2001).

Para a criação do manifesto, a ideia principal foi valorizar mais: indivíduos e interações do que processos e ferramentas; software em funcionamento do que documentação em excesso; colaboração com o cliente do que negociação de contratos; responder a mudanças do que seguir um plano (BECK et al., 2001).

Para melhor compreensão dessa nova filosofia de desenvolvimento, o grupo também criou uma lista com doze princípios que devem ser seguidos. São eles (BECK et al., 2001):

- Satisfação do cliente através da entrega adiantada e contínua de software de valor;
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento;
- Entregar software funcionando com frequência;
- Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente;
- Construir projetos com indivíduos motivados, dando ambiente e apoio necessário, e confiar que farão seu trabalho;
- Conversa face a face é o método mais eficiente e eficaz de transmitir informações;
- Software funcional é a principal medida do progresso;
- Desenvolvimento sustentável. Patrocinadores, usuários e desenvolvedores devem manter um ritmo constante indefinidamente;
- Atenção contínua à excelência técnica e bom design;
- Simplicidade - a arte de maximizar a quantidade de trabalho que não precisa ser feito;
- As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizáveis;
- Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficiente.

Seguindo nessa linha de pensamento, várias metodologias descritas como ágeis surgiram, sendo algumas delas: *Agile Unified Process* (AUP), *Test-Driven Development* (TDD), *Feature-Driven Development* (FDD) e *Dynamic Systems Development Method* (DSDM). No entanto, duas delas ganharam muitos adeptos no mercado: *Scrum* e *Extreme Programming* (XP). Essas abordagens são detalhadas nas próximas seções, juntamente com a *Agile Unified Process*.

2.3.2.1 Scrum

Segundo Schwaber e Sutherland (2016), *Scrum* é um *framework* difícil de dominar, simples de entender e leve, no qual as pessoas resolvem problemas adaptativos e complexos. O objetivo é entregar produtos com maior valor possível para o cliente, de forma criativa e produtiva.

Scrum é definido também como um modelo ágil de processo que está de acordo com o manifesto ágil proposto em 2001 (SOMMERVILLE, 2010). Em suma, essa metodologia para gerenciamento de projetos tem como característica entregar resultados de maneira incremental, efetiva e com menor custo, implementando requisitos realmente necessários. A flexibilidade e facilidade de adaptação do projeto diante das inevitáveis mudanças é um outro princípio dessa metodologia (SCHWABER, 1997).

A inspeção e a verificação contínua do produto em desenvolvimento têm um papel importante no *Scrum*. Como resultado, possíveis problemas ou correções podem ser encontrados e corrigidos o quanto antes. Outro princípio importante é manter uma equipe pequena, organizada e auto gerenciável, maximizando a comunicação e minimizando a supervisão entre os membros, ou seja, cria-se uma confiabilidade entre os membros da equipe (PRESSMAN, 2005).

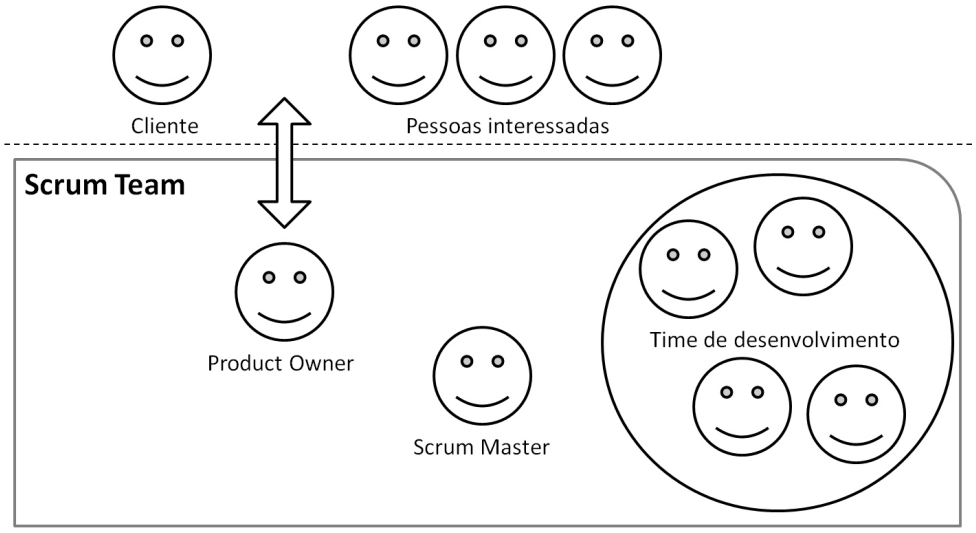
Na metodologia *Scrum* existem papéis fundamentais com o objetivo de definir a atuação de todos os participantes envolvidos em um determinado projeto. Uma esquematização de tais papéis pode ser vista na Figura 2.6. Esses papéis juntos formam o *Scrum Team* ou o time *Scrum*. Duas características importantes desse time é de serem multifuncionais e auto-organizáveis, por isso, tal modelo estimula a criatividade, flexibilidade e produtividade (SCHWABER; SUTHERLAND, 2016). Os papéis do *Scrum Team* são:

- *Product Owner* (P.O.): é quem conhece o produto, sendo responsável por fornecer a visão mais abrangente do sistema e priorizar quais funcionalidades são mais importantes. Ele é o responsável por manter o Backlog do produto (*Product Backlog*) sempre atualizado e priorizado. É ele quem dita o que deve ser executado pelo time de desenvolvimento;
- *Scrum Master*: possui um papel gerencial dentro do projeto, garantindo que a equipe siga

adequadamente as práticas do *Scrum*. Além disso, esse profissional atua como mediador de possíveis conflitos e como facilitador para eliminar os obstáculos;

- *Dev Team* (Time de desenvolvimento): é o time de profissionais multidisciplinares, responsável por desenvolver o produto. Geralmente é uma equipe pequena que, além de ser transfuncional (um membro pode possuir várias funções), é também auto gerenciável, na qual cada membro da equipe tem o poder de organizar seu próprio trabalho.

Figura 2.6: Papéis *Scrum* - Interpretado de Schwaber e Sutherland (2016)



Depois dos papéis envolvidos, existem alguns artefatos do *Scrum* que servem principalmente como transparência para todos os envolvidos no projeto, gerando uma flexibilidade para que todos possam inspecionar e sugerir adaptações de acordo com a dinâmica gerada no andamento do projeto. Os artefatos são:

- *Product Backlog*: é uma lista ordenada de funcionalidades do sistema, ou seja, todas as características, funcionalidades, melhorias, requisitos, correções e regras de negócios devem estar nessa lista. O responsável por manter essa lista priorizada e completa é do *Product Owner*. Uma característica do *Product Backlog* é ser uma lista dinâmica, pois alterações podem aparecer constantemente para deixar a aplicação sempre útil, funcional e objetiva;

- *Sprint Backlog*: enquanto o *Product Backlog* contém a lista de todas as funcionalidades do produto, o *Sprint Backlog* contém apenas uma parte dessas funcionalidades que deverão ser entregues ao final da *Sprint*. O time de desenvolvimento é responsável por manter essa lista sempre atualizada, mostrando quais tarefas já foram executadas, quais estão

com algum impedimento e quais ainda estão na fila. Essa lista também é dinâmica, já que algumas tarefas podem demandar outras tarefas não previstas;

- Incremento: é o resultado da *Sprint* como uma versão potencialmente utilizável. É a soma dos itens do *Product Backlog* que já foram executados, ou seja, o resultado da *Sprint* atual somado com as *Sprints* anteriores.

De acordo com Schwaber e Sutherland (2016), dentro da metodologia *Scrum* existem alguns eventos definidos para nortear a execução do projeto, permitindo uma melhor transparência e inspeção por parte de todos os participantes. Os eventos do *Scrum* são:

- *Sprint*: em um projeto de desenvolvimento utilizando *Scrum*, uma versão potencialmente utilizável é entregue no final de cada *Sprint*, dando início a *Sprint* seguinte. Aqui, as funcionalidades separadas no *Sprint Backlog* são desenvolvidas, por isso, a *Sprint* é conhecida como o coração do *Scrum*. Nela, os demais eventos acontecem e geralmente tem duração de duas a quatro semanas;

- *Sprint Planning*: é uma reunião que acontece sempre ao início de cada *Sprint*. Geralmente é dividida em duas etapas: na primeira, o *Product Owner* define quais itens do *Product Backlog* deverão ser implementados, definindo o objetivo principal da *Sprint*. Tendo a lista dos itens que serão entregues na *Sprint* (*Sprint Backlog*), tem-se início a segunda etapa, no qual time começa a definir como irá construir tais funcionalidades, dividindo cada item em tarefas e dando um prazo para cada uma, transformando-os em incremento do produto no final da *Sprint*;

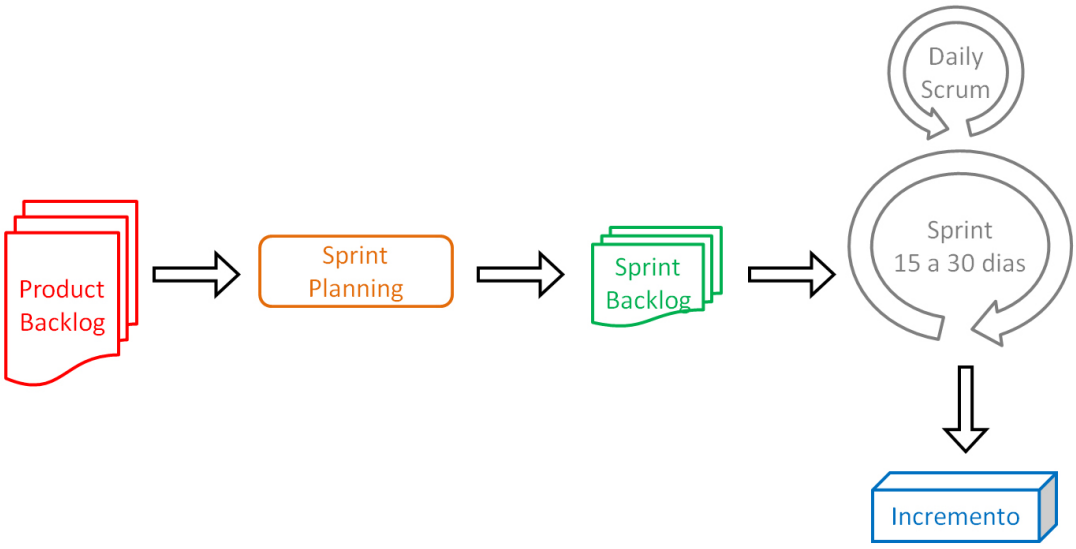
- *Daily Scrum*: todos os dias o time *Scrum* se reúne para trocar informações sobre o andamento da *Sprint* e sobre o que se pretende fazer até a próxima reunião. Essa é uma reunião rápida, geralmente de 15 minutos, na qual todos respondem a três perguntas: o que fiz da última reunião até agora que ajudou na meta da *Sprint*? O que farei hoje para atender a meta? Existe algum impedimento que precisa ser resolvido para atender a meta? Esses pontos servem para melhorar o sincronismo e a comunicação do time de desenvolvimento;

- *Sprint Review*: é uma revisão realizada ao final de cada *Sprint*. Geralmente as partes interessadas se reúnem junto com o time de desenvolvimento para inspecionar e colaborar com o incremento realizado. Se necessário, o *Product Owner* aproveita essa reunião para ajustar o *Product Backlog*;

- Retrospective: após a revisão da *Sprint*, o time de desenvolvimento realizada uma reunião de retrospectiva, levantando os principais pontos positivos e negativos do time. Esses pontos estão ligados ao relacionamento com os integrantes do time, às ferramentas e aos processos, e servem para criar um plano de melhorias a serem aplicadas na *Sprint* seguinte.

Após ter apresentado os principais pontos da metodologia, a seguir é descrito como é a aplicação da mesma em um projeto. A Figura 2.7 mostra uma abordagem geral do funcionamento do *Scrum*.

Figura 2.7: Funcionamento do *Scrum* adaptado de ScrumAlliance (2018)



O primeiro artefato a ser criado é o *Product Backlog*, sendo o *Product Owner* o responsável por gerenciar tal documento. Após ter conhecimento de tal documento é realizado então a *Sprint Planning*, onde o time *Scrum* se reúne para gerar a lista de funcionalidades que serão desenvolvidas pelo time de desenvolvimento na *Sprint*. Essa lista de funcionalidades é denominada *Sprint Backlog*.

Após essa reunião de planejamento, a *Sprint* em si é iniciada. O time de desenvolvimento se organiza e executa o que foi definido para as próximas semanas. Dentro de cada dia de execução da *Sprint*, ocorre a *Daily Scrum* entre os desenvolvedores para a integração do time com o andamento do projeto.

Ao término da *Sprint*, as funcionalidades previstas são concluídas e um incremento é realizado no produto final, tornando o produto cada vez mais completo e atrativo para o cliente.

Ocorre também a *Sprint Review* para inspecionar como foi realizado os trabalhos durante o período trabalhado.

Antes de começar uma nova reunião de planejamento para a próxima *Sprint*, o time de desenvolvimento realiza a reunião de retrospectiva. Todos os tópicos levantados nessa retrospectiva são melhor trabalhados para que o resultado da próxima *Sprint* seja melhor que a anterior.

Na próxima seção será abordado detalhes de outra metodologia ágil, denominada *Extreme Programming* (XP).

2.3.2.2 Extreme Programming - XP

Extreme Programming ou XP, segundo Beck (2000) é um estilo de desenvolvimento de software com foco na comunicação clara e objetiva, trabalho em equipe e aplicação de boas técnicas de programação. Tal abordagem baseia-se em cinco princípios ou valores básicos:

- Comunicação: todos são parte do time, e a comunicação diária é de extrema importância para o sucesso do projeto. A comunicação deve ser entre os membros do time de desenvolvimento e também da equipe com o cliente. Assim, todos os pontos importantes podem ser tratados com a agilidade e atenção que merecem;
- *Feedback*: a atuação do cliente é essencial para que possíveis alterações ou melhorias possam ser efetuadas o quanto antes. Quanto mais o cliente aprende com o produto, maior deve ser seu *feedback* ao time de desenvolvimento, pois assim a equipe pode focar em algo que realmente agregue valor ao cliente;
- Simplicidade: o foco é entregar somente o funcional e o mais simples possível. Isso agregará valor ao cliente, uma vez que ele vê o investimento dele em algo palpável;
- Coragem: dizer sempre a verdade sobre estimativas e andamento do projeto, encarar as mudanças assim que elas surgirem, e apoiar toda equipe, pois ninguém trabalha sozinho;
- Respeito: todos são responsáveis pelo projeto, então cabe a cada um respeitar a diversidade, o nível de conhecimento e as limitações do outro.

De acordo com Teles (2014), XP é um processo ágil de desenvolvimento, cujo principal objetivo é apresentar algo novo a cada dia para o cliente que, por sua vez, aprende mais sobre suas próprias necessidades conforme o produto vai sendo desenvolvido.

Um ponto importante da metodologia é que a mesma segue um conjunto de práticas que auxiliam todos os envolvidos no projeto a manterem sempre em mente os valores mencionados anteriormente. Algumas dessas práticas são (TELES, 2014):

- Cliente presente: a participação ativa do cliente é essencial para obter o máximo de valor do projeto. Com essa participação ativa, o *feedback* entre as partes torna-se mais efetivo, viabilizando maior simplicidade no processo de desenvolvimento;
- Jogo do planejamento: no início de cada *release* (módulos do sistema que serão entregues por etapas) e cada iteração (período de tempo que a equipe desenvolve um conjunto de funcionalidades) ocorre o jogo do planejamento, no qual o cliente avalia as funcionalidades que serão implementadas e a equipe faz uma estimativa do custo da implementação;
- Stand up meeting: traduzindo tem-se reunião em pé. É uma reunião rápida que acontece todos os dias pela manhã com o time de desenvolvimento. Nela, a equipe avalia o trabalho executado no dia anterior e prioriza o que será implementado no decorrer do dia;
- Programação em par: a ideia é colocar dois programadores em um mesmo computador para produzir o mesmo código. Nesta prática, o código produzido já é revisado em sua construção. Isso permite também que o código seja mais simples e eficaz, pois há uma troca de experiências a todo momento entre os desenvolvedores;
- Refactoring: refatorar é o ato de refazer um código sem alterar sua funcionalidade. É utilizado para que o código gerado fique sempre claro e objetivo, facilitando assim a sua manutenção;
- Código coletivo: diferentemente de outras metodologias, em que cada desenvolvedor é responsável por uma parte do sistema, no XP todos tem acesso a todas as partes do sistema, e podem alterar aquilo que julgarem importante. Cria-se assim mais uma opção de revisão de código, além de fornecer uma maior agilidade no processo;
- Design simples: a equipe é focada em desenvolver códigos simples para atender às necessidades da funcionalidade que está implementando, sem criar algoritimos complexos. Eles contam com a prática de refatoração quando precisarem incorporar qualquer necessidade futura;
- Integração contínua: Para que o sistema esteja sempre funcionando ao se incorporar uma nova funcionalidade, a equipe realiza a prática de integração contínua, adequando os códigos gerados com o restante do sistema;

- *releases* curtos: um conjunto de funcionalidades é colocado em produção em um curto espaço de tempo, para que o cliente já possa utilizar e gerar seu *feedback*. Com isso, o sistema é colocado em produção diversas vezes ao longo do projeto, gerando mais valor e incorporando mais funcionalidades a cada *release*.

Segundo Teles (2014) existem alguns papéis que representam as pessoas que compõe uma equipe que irá utilizar o XP, são elas:

- Gerente de projeto: responsável por resolver quaisquer problemas não relacionados ao dia-a-dia dos desenvolvedores. Além de resolver os assuntos administrativos e manter o contato com o cliente para tê-lo ativo no projeto;
- Coach: geralmente é um programador com maior conhecimento técnico, pois precisa orientar a equipe a seguir o projeto sem nenhum impedimento no processo;
- Analista de teste: tem como objetivo procurar eventuais defeitos do sistema o quanto antes, dando *feedback* para que a equipe possa realizar as correções;
- Redator técnico: com a intenção de deixar a equipe de desenvolvimento totalmente focada na programação do software, o redator geralmente fica responsável por toda a documentação envolvida no projeto;
- Desenvolvedor: normalmente é uma pessoa multidisciplinar, pois precisa saber não só programar, mas também realizar o papel de analista, projetista, e outros, quando for necessário.

Por fim, Teles (2014) descreve que esse processo é recomendado para projetos cujos requisitos são flexíveis ou vagos, para desenvolvimento orientado a objetos e de forma incremental, em que o projeto vai ganhando novas funcionalidades ao longo do processo. Inicialmente foi recomendado para times pequenos e médios, porém com a aplicação da metodologia em vários tipos e tamanhos de projetos, o resultado obtido foi o mesmo independente do tamanho da equipe, mostrando a eficácia do mesmo.

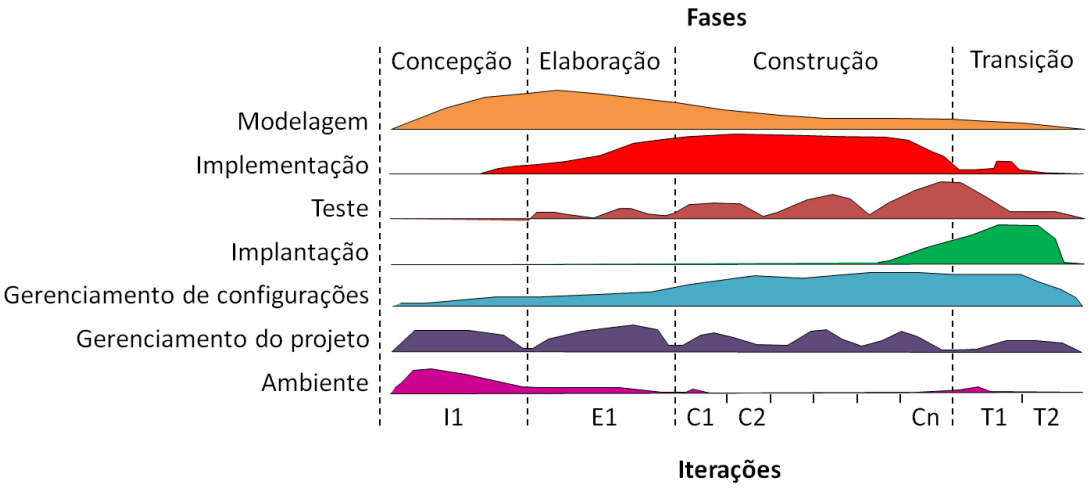
Após apresentados detalhes sobre *Scrum* e XP, na próxima seção são apresentados detalhes sobre a metodologia *Agile Unified Process* (AUP).

2.3.2.3 Agile Unified Process - AUP

Segundo Ambler (2002), *Agile Unified Process* (AUP) é uma versão simplificada da metodologia criada pela IBM[1] denominada RUP (*Rational Unified Process*). O mesmo descreve tal metodologia como uma abordagem simples e fácil de entender para o desenvolvimento de software, usando conceitos e técnicas ágeis, sem sair da proposta inicial do RUP.

O modelo é dividido em duas perspectivas, dinâmica e estática, como mostra a Figura 2.8. A perspectiva dinâmica é apresentada na horizontal e é composta por quatro fases: concepção, elaboração, construção e transição. A perspectiva estática é apresentada na vertical é composta por sete disciplinas: modelagem, implementação, teste, implantação, gerenciamento de configurações, gerenciamento de projeto e ambiente. Uma parte desse processo que o autor considera importante, é a disciplina de modelagem. Embora siga conceitos ágeis, a documentação elaborada precisa ser simples e objetiva, e ela tende a percorrer por todas as fases do processo.

Figura 2.8: Modelo AUP adaptado de Ambler (2018)



Ainda analisando a Figura 2.8, a perspectiva dinâmica é dividida nas seguintes fases:

- Concepção: tem como objetivos a identificação da possível arquitetura de sistema, o escopo e investimentos iniciais do projeto, e a aprovação por parte dos interessados pelo projeto;
- Elaboração: se resume em confirmar se a arquitetura adotada é consistente;
- Construção: criar, de maneira incremental, o software em si, com base nos requisitos e interesses definidos pelos interessados do projeto;

[1] International Business Machines - https://www.ibm.com

- Transição: o objetivo é validar o software desenvolvido e colocá-lo em execução em um ambiente de produção.

Por ser um modelo iterativo, as disciplinas da perspectiva estática são processadas de acordo com o andamento do projeto, no qual o time de desenvolvimento constrói, realiza a validação e entrega os resultados de acordo com o que foi definido pelas partes interessadas (AMBLER, 2018). As disciplinas são:

- Modelagem: comparando com o modelo RUP, essa disciplina incorpora as disciplinas modelagem de negócio, requisitos e também a análise e design. O objetivo é entender o negócio da empresa, os problemas que o projeto irá abordar, além de propor e documentar as possíveis soluções para tais problemas;
- Implementação: consiste em gerar algo executável a partir dos modelos criados, em implementar os componentes do software e integrar os resultados das iterações no produto final;
- Teste: os testes podem ocorrer de forma iterativa conforme o andamento do projeto, isso permite encontrar e consertar erros o quanto antes. O objetivo é avaliar a qualidade do que está sendo desenvolvido, e verificar se o produto está de acordo com os requisitos;
- Implantação: consiste em planejar a entrega do produto, disponibilizando e instalando uma versão para o cliente utilizar;
- Gerenciamento de configurações: esta disciplina tem como objetivo gerenciar as mudanças no que ocorrem no projeto, isso inclui o controle das versões dos artefatos do mesmo;
- Gerenciamento do projeto: está relacionada à gestão das atividades do projeto. O objetivo é direcionar as ações e as pessoas que estão envolvidas além de gerenciar os riscos. Outro ponto é garantir que preços e prazos sejam cumpridos, fazendo a interface com as pessoas e sistemas que estão fora do escopo de atuação do projeto;
- Ambiente: está relacionado com o todo ambiente necessário para que a equipe de desenvolvimento possa realizar seu trabalho, fornecendo artefatos e ferramentas adequadas conforme a necessidade.

Segundo Ambler (2018), o princípio da metodologia é dividir o produto em pequenas porções, que conforme o andamento do projeto, essas pequenas partes são implementadas e

entregues de modo iterativo. Uma característica do processo é que as primeiras iterações geralmente demoram mais para serem concluídas, pois alguns aspectos precisam ser definidos, e também a equipe vai aprimorando sua capacidade de colaboração com o passar do tempo.

2.4 Considerações finais

O uso de metodologias de desenvolvimento é algo essencial para o sucesso de qualquer projeto, desde o mais simples ao mais complexo. Porém, a escolha da melhor metodologia depende de cada empresa, de cada projeto e de cada equipe. O objetivo deste capítulo foi apresentar algumas metodologias que podem ser utilizadas para o desenvolvimento de um produto qualquer. Neste contexto, foram apresentadas, inicialmente, informações sobre a realidade do mercado de TI. Em seguida, foram apresentados alguns aspectos das metodologias de desenvolvimento e um pouco do seu histórico. Depois foi apresentado detalhes sobre o que são metodologias ágeis, e por fim três metodologias ágeis foram descritas: *Scrum*, XP e *Agile Unified Process*.

No próximo capítulo serão descritas metodologias que são utilizadas por um único indivíduo, conhecidas como metodologias solo. Além disso, tais metodologias seguem alguns princípios ágeis, juntando algumas propriedades das metodologias citadas neste capítulo.

Capítulo 3

ESTADO DA ARTE

3.1 Considerações iniciais

Após ter apresentado uma revisão geral de algumas metodologias de desenvolvimento de software, este capítulo apresenta um conjunto de abordagens propostas para que uma única pessoa fosse capaz de gerenciar e executar um projeto de desenvolvimento de software. Além de serem metodologias para um único desenvolvedor, todas seguem princípios ágeis.

3.2 Scrum Solo

A primeira que será apresentada é denominada Scrum Solo, que, segundo Pagotto et al. (2016) é uma adaptação da metodologia *Scrum* com ênfase no desenvolvimento individual de software. A Figura 3.1 apresenta o processo da metodologia, que por sua vez, tem uma certa semelhança com a metodologia *Scrum*, com algumas diferenças que serão apresentadas a seguir.

A metodologia caracteriza-se como um processo iterativo e incremental que une boas práticas do PSP e do *Scrum*. Por unir tais processos, um certo volume de documentação é necessário para que o PSP possa ser colocado em prática. A seguir, serão apresentadas as atividades, artefatos e atores da metodologia Scrum Solo (PAGOTTO et al., 2016).

O processo está dividido em quatro atividades, e todos os documentos gerados nestas atividades são armazenados em um repositório na nuvem, pois assim, todos os envolvidos no projeto podem acessar os documentos de qualquer lugar. As atividades são:

- *Requirement*: são gerados três artefatos nessa etapa: escopo, *Product Backlog* e protótipo do software. O objetivo é definir o escopo do produto e coletar o máximo de informações

Figura 3.1: Fluxo do processo Scrum Solo (PAGOTTO et al., 2016)

relevantes junto ao cliente;

- *Sprint*: são gerados outros quatro artefatos nessa etapa: *Sprint Backlog*, produto, ata e planta de desenvolvimento. O objetivo é implementar o conjunto de tarefas relacionadas no *Sprint Backlog* em um prazo de uma semana;
- *Deployment*: são gerados dois artefatos nessa etapa: produto e ata de validação. O objetivo é colocar o produto em produção para que o cliente possa testar;
- *Management*: outros quatro artefatos são gerados nessa etapa: estrutura analítica do projeto (EAP), planilha de custo, planilha de controle e cronograma. O objetivo é monitorar e planejar o desenvolvimento do produto.

Os atores definidos no processo são:

- *Product Owner* (PO): é quem conhece o produto que será construído, ele deve fornecer as informações necessárias para que o produto seja desenvolvido com as funcionalidades corretas;
- Desenvolvedor individual: diferentemente do *Scrum* que possui um time de desenvolvimento, aqui só existe uma pessoa responsável pela construção do produto;
- Orientador: é a pessoa que conhece o processo, a tecnologia utilizada e o escopo do projeto. Seu papel é atuar como um consultor;

- Grupo de validação: responsáveis por validar o produto produzido.

Como descrito nas atividades acima, existem alguns artefatos que são criados para auxiliar o processo de desenvolvimento. São eles:

- Escopo: além da lista de funcionalidades do produto (*Product Backlog*), esse artefato é composto pela descrição dos principais pontos do software, pelo perfil do cliente e pelo escopo do processo;
- Protótipo de software: imagem que apresenta a tela correspondente a um item do *Product Backlog*, ou seja, um esboço de como será a funcionalidade implementada, que, ao longo do tempo, irá conter várias imagens das telas do produto;
- *Product Backlog*: lista contendo as funcionalidades do produto que serão codificadas ao decorrer do projeto;
- Repositório do processo: é um espaço na nuvem utilizado para armazenar os artefatos gerados durante todo o processo;
- *Sprint Backlog*: contém a lista de funcionalidades que serão desenvolvidas na *Sprint*. Aqui o autor especifica que tais funcionalidades deverão conter alguns diagramas como de caso de uso, de sequência, de classes e de entidade e relacionamento;
- Produto ou parte dele em funcionamento: ao final de cada *Sprint*, o cliente recebe uma versão mais atualizada do produto;
- Ata: tem como objetivo registrar a validação do que foi entregue. Na entrega do produto, o próprio cliente faz a validação, enquanto nas *Sprints* o orientador realiza tal validação;
- Planta de desenvolvimento: como explicado no *Sprint Backlog*, alguns diagramas são criados para auxiliar na especificação das funcionalidades. O objetivo dessa planta é unir tais diagramas de forma customizada para cada projeto, não limitando somente aos diagramas citados;
- Estrutura analítica do Projeto (EAP): é um organograma que contém o nome do projeto, as atividades e todo o trabalho que será realizado. O objetivo é apresentar o escopo do projeto;
- Cronograma: cruza os dados dos trabalhos que precisam ser realizados em um certo espaço temporal, onde cada tarefa pode ser atribuída a um responsável por sua execução;

- Planilha de custo: tem como objetivo analisar e comparar o orçamento inicial com os custos reais utilizados no projeto.

O autor apresenta como resultado da pesquisa, a aplicação dessa metodologia por vários alunos da Universidade Federal do Paraná dentre os anos de 2012 a 2014, que tiveram sucesso na aplicação no desenvolvimento de seus produtos. Além de oito alunos e ex-alunos que aplicaram no desenvolvimento de software e também tiveram sucesso, e afirmaram que o modelo atende a necessidade tanto do cliente quanto do desenvolvedor.

Após analisar a metodologia apresentada, tem-se os seguintes pontos positivos encontrados no processo como um todo: a utilização da metodologia *Scrum*, que dá agilidade e transparência no projeto; a utilização de boas práticas do PSP, gerando assim uma melhoria contínua na qualidade e no gerenciamento individual de desenvolvimento; e também a divisão bem estruturada das etapas e artefatos utilizados.

Um ponto que poderia ser melhorado é com relação ao número de documentos gerados na documentação do projeto. Por utilizar abordagens ágeis, alguns documentos não fazem muito sentido se aplicados de maneira excessiva, pois o autor não deixa claro quais diagramas são esperados na especificação das funcionalidades.

Por se tratar de uma metodologia solo, outro ponto que poderia ser revisado, seria a existência do ator "orientador" em que não fica claro qual sua relevância. Em alguns casos, pode ser que não seja possível encontrar uma pessoa com o perfil indicado. Pensando nisso, talvez o cliente ou o próprio desenvolvedor poderia assumir as tarefas atribuídas a este autor.

3.3 Agile Solo

Outra metodologia encontrada na literatura é a denominada Agile Solo, que é um modelo de processo de desenvolvimento ágil de software projetado para desenvolvedores solo (ANNA, 2011). É o resultado da análise de processos ágeis, juntamente com a adaptação de práticas para desenvolvedores solo. Em resumo, as práticas propostas na metodologia são:

- Apresentações semanais e atualização de prioridades: como todas as metodologias ágeis, o principal ponto da metodologia é obter um *feedback* rápido. Toda semana o software é apresentado do jeito que se encontra, e as prioridades das tarefas podem ser alteradas de acordo com o desejo do cliente, para então entregar as funcionalidades mais importantes primeiro;

- Entregas mensais e teste do cliente: ao final de cada mês, parte do software em funcionamento é entregue para que o cliente possa realizar testes e apresentar o *feedback* do uso da ferramenta;

- Planejamento e iteração: na metodologia há dois tipos de iterações, uma curta no qual é apresentado o sistema, e uma longa em que o software é entregue para testes. O objetivo em ambas iterações é priorizar o que é mais importante para o cliente. Também deve ser realizada uma estimativa de entrega para as próximas iterações, e conferir se o que foi planejado realmente será entregue;

- Desenvolvimento guiado por testes: é considerada pelo autor a prática mais importante da metodologia. A ideia é visualizar o resultado final de uma tarefa, para que o desenvolvedor entenda o problema antes de começar a implementação da solução;

- A técnica Pomodoro: essa técnica tem como objetivo aumentar a concentração e reduzir o número de interrupções e distrações em um certo período de tempo (CIRILLO, 2009). A inclusão dessa técnica na metodologia Agile Solo foi para suprir a falta dos benefícios que a programação em pares trás, uma vez que o programador desenvolve sozinho e tende a perder o foco mais facilmente;

- Revisão semanal do código: o objetivo é encontrar erros ou códigos mal estruturados o quanto antes, garantindo a qualidade do que se está produzindo. A ideia é que o código seja revisado por outra pessoa que não o próprio programador;

- Revisão do próprio código: a ideia é revisar o próprio código o quanto antes, o objetivo é suprir novamente a ausência da programação em par, no qual o código já é revisado por outra pessoa no momento em que se está codificando;

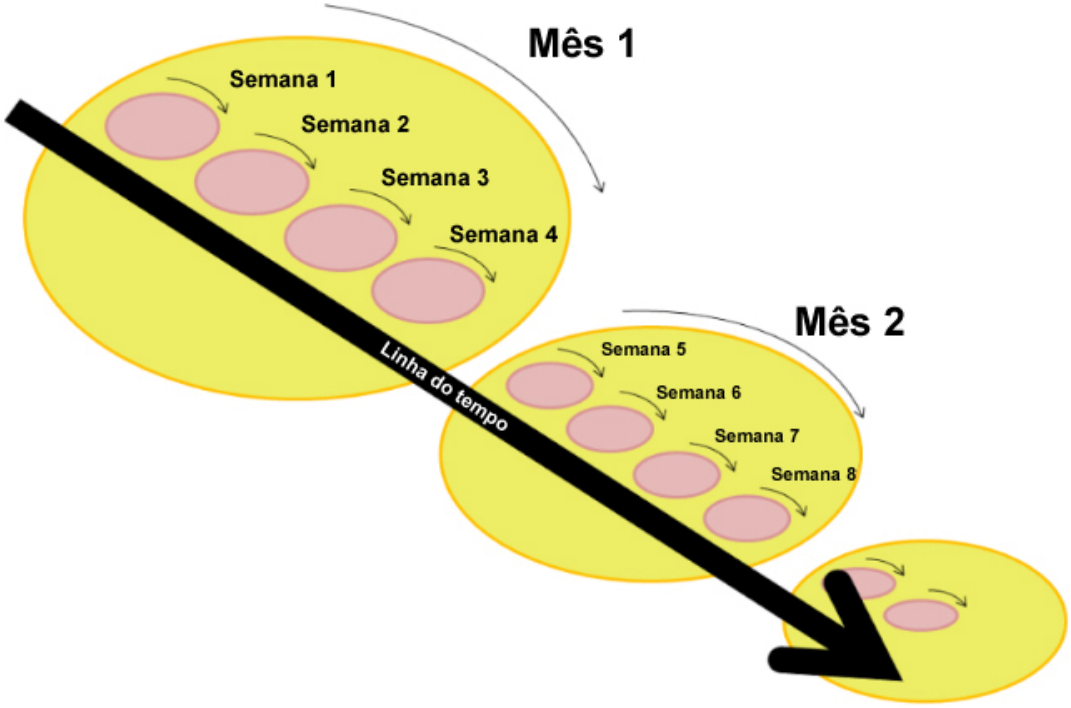
- Controle visual: tem como objetivo manter um rastreamento do trabalho de uma maneira visual. É importante tanto para o desenvolvedor quanto para o cliente, pois uma visão do que já foi realizado e do que ainda precisa ser feito, estará sempre à disposição;

- Modelagem: criar modelos simples e concisos pode melhorar a qualidade da comunicação entre o desenvolvedor e cliente;

- Gerenciamento das tarefas de iteração: a decisão de como gerenciar as tarefas deve estar em comum acordo entre desenvolvedor e cliente. A ideia é que seja algo simples, e que tenha como objetivo saber como está o andamento do projeto e se estão indo para a direção certa. Ao fim da iteração, uma revisão é realizada comparando o tempo gasto com o estimado, além de outros pontos a serem melhorados para a próxima iteração.

A Figura 3.2 apresenta as iterações da metodologia ao longo do tempo, no qual os círculos amarelos representam as iterações longas, em que o software é entregue para realização de testes. Já os círculos menores dentro dos amarelos representam as iterações curtas, em que o software é apresentado para o cliente no intuito de se obter *feedback* do que está sendo desenvolvido.

Figura 3.2: Iterações em um projeto utilizando Agile Solo (ANNA, 2011)



Não necessariamente todas as práticas acima precisam ser executadas em um único projeto. No mesmo trabalho o autor realiza um estudo de caso aplicando tal abordagem com o intuito de avaliar o modelo, porém ele utiliza apenas as seguintes práticas: modelagem, a técnica Pomodoro, revisão semanal do código, revisão diária do próprio código, testes mensais, gerenciamento de tarefas e desenvolvimento guiado por testes.

Algumas práticas tiveram uma avaliação positiva, sendo que uma delas é revisão semanal do código, que segundo Anna (2011), por ter outra pessoa revisando o que foi desenvolvido, apresenta duas melhorias: uma é ter o código revisado por uma pessoa que não está no contexto de desenvolvimento, geralmente um supervisor, outra melhoria é que durante a implementação, o desenvolvedor tenta codificar da maneira mais precisa, pois sabe que seu código será auditado.

Os testes mensais ajudaram a obter um *feedback* positivo, além de melhorar alguns pontos da aplicação. O gerenciamento de tarefas, por mais simples que seja, auxilia no entendimento do processo e permite uma percepção de quanto trabalho foi executado e quanto falta. E o

desenvolvimento guiado por testes ajuda a ter uma visão melhor do sistema antes de começar a codificação. Todos esses pontos foram descritos como positivos pela autora.

Um ponto a ser melhorado é com relação à modelagem, uma vez que a proposta dos documentos gerados é para se ter uma melhor comunicação entre o cliente e o programador. Tais documentos geralmente não são tão claros para o cliente, pois pode ser necessário algum conhecimento técnico para compreendê-los. Nesses casos, a comunicação seria melhor em uma conversa do que por meio de modelos (ANNA, 2011).

Outro ponto levantado é com relação à técnica Pomodoro, pois o tempo definido para o uso da técnica não foi adequado no estudo de caso apresentado. Segundo Anna (2011), é necessário um tempo maior de concentração quando se está programando, e às vezes tal interrupção pode atrapalhar ao invés de ajudar. Se for interrompido em um estado de concentração profunda, pode ser que tenha que retomar todo o raciocínio ao voltar a programar.

Após esses pontos acima, a autora apresenta como resultado a definição de um processo de desenvolvimento de software ágil para um desenvolvimento solo, no qual poderia ser utilizado em qualquer projeto. Ao avaliar a aplicação do processo, Anna (2011) conclui que não existe um processo perfeito para todos os tipos de projetos, pois algumas práticas propostas foram extremamente úteis, enquanto outras não foram utilizadas por não serem apropriadas ao projeto que se estava executando.

Após a conclusão, a autora recomenda o uso da metodologia para quem queira começar no desenvolvimento solo, porém, que ela seja adaptada para melhor se adequar aos projetos e às necessidades de cada um. O mesmo se aplica nas práticas da metodologia, que sejam adaptadas de acordo com o programador, o cliente e o projeto.

3.4 Personal Extreme Programming

Personal Extreme Programming (PXP) é a combinação da metodologia XP (*Extreme Programming*) com a PSP (*Personal Software Process*). O PSP auxilia o profissional a melhorar seu desempenho individual e a entender e planejar melhor seu processo de desenvolvimento. Enquanto os valores e práticas do XP auxiliam no gerenciamento e codificação do projeto (AGARWAL; UMPHRESS, 2008).

O objetivo principal é utilizar as práticas do XP de maneira individual, para isso, o autor adapta e descreve como utilizar cada uma dessas práticas por um desenvolvedor individual. Dentre elas, pode-se citar as seguintes (AGARWAL; UMPHRESS, 2008):

- Jogo do planejamento: embora essa prática seja realizada entre o cliente e o time de desenvolvimento, uma opção seria o próprio desenvolvedor incorporar os papéis durante o planejamento. Assim, ele poderia descrever as estórias, estimar prazos e priorizar as tarefas;

- Programação em par: esta prática permite que o código gerado por um desenvolvedor seja revisado por outro programador no momento em que se está sendo escrito, garantindo uma melhor qualidade do código. Porém, quando se está sozinho, não é possível obter todos os benefícios dessa prática, entretanto, é possível pedir para algum colega analisar o código que foi produzido;

- Código coletivo: por não ter outras pessoas codificando no mesmo projeto, o próprio desenvolvedor acaba se tornando dono do código. Desse modo, os benefícios que outras pessoas podem trazer para um projeto, podem não existir.

O processo segue um roteiro criado para guiar o desenvolvedor na aplicação da metodologia, e está dividido em três etapas: planejamento, desenvolvimento e pós-conclusão. Na primeira etapa, obtêm-se os requisitos do sistema, gerando uma lista de estórias de usuários, que por sua vez é dividida em funcionalidades. Após isso, as funcionalidades são agrupadas e priorizadas para que se possa começar a codificação.

Na etapa de desenvolvimento, o processo é dividido em iterações, e cada iteração possui um dos grupos de funcionalidades anteriormente criados, que deverão ser divididos em tarefas, que por sua vez, também são priorizadas. Após isso, cada tarefa deverá ter um plano de testes, que, após a codificação, deverá ser testada e aprovada. Com isso, a nova funcionalidade é integrada ao sistema. Por fim, na etapa de pós conclusão, após realizar mais testes, o sistema é colocado em base de produção.

A metodologia foi aplicada no desenvolvimento de uma ferramenta de gerenciamento. Tal ferramenta tem como finalidade guiar o desenvolvedor a utilizar a própria metodologia PXP. O processo de desenvolvimento foi realizado por uma única pessoa, e o autor descreve que a finalização do projeto no tempo estimado foi realizado de maneira eficiente e fácil (AGARWAL; UMPHRESS, 2008).

O autor não deixa claro quais os resultados esperados da aplicação da metodologia em outros ambientes, bem como os pontos fortes e fracos da mesma. Porém, devido às adaptações apresentadas para que as práticas da metodologia XP fossem utilizadas por um único indivíduo, tal proposta mostra que é possível utilizar o XP mesmo sem uma equipe de desenvolvimento.

3.5 Cowboy

A metodologia denominada Cowboy é um processo iterativo criado para auxiliar programadores solo a gerenciar o desenvolvimento de software. De modo a ser uma metodologia leve, segue algumas práticas ágeis de metodologias já conhecidas. O processo está dividido em quatro seções: práticas gerais, requisitos e interação com o cliente, gerenciamento iterativo e implantação (HOLLAR, 2006).

Uma das práticas gerais é organizar a listagem de tarefas e funcionalidades baseando-se no *Scrum Backlog*, tanto na lista do projeto inteiro, quanto na listagem de cada iteração. Outra prática sugerida é de manter os artefatos utilizados de maneira simples e clara, como definido na metodologia AUP.

Além dessas práticas, o desenvolvedor deve ter em mente que para criar software solo basta simplesmente seguir boas práticas e regras já bem sucedidas nos métodos ágeis, como criar de maneira iterativa, e a cada iteração adicionar novas funcionalidades, assim como resolver problemas das iterações anteriores.

Na seção de requisitos e interação com cliente, as práticas ágeis sugerem que não se deve gastar muito tempo com a documentação no início do projeto, e sim, que o cliente esteja sempre à disposição para discutir as funcionalidade e problemas durante o desenvolvimento do projeto.

Para o desenvolvimento das iterações, algumas diretrizes o desenvolvedor pode seguir, como por exemplo, não exceder o prazo máximo de duas semanas em uma iteração, refatorar o código das iterações anteriores e também documentar bem o código, utilizando algum padrão que consiga gerar uma documentação automática após o desenvolvimento.

Por fim, na seção de implantação, a codificação precisa estar organizada, se possível em um repositório no qual pode ser vinculado anotações e informações importantes como sugerido na disciplina de gerenciamento de configurações da metodologia AUP. Além disso, para garantir a qualidade do código implementado, testes unitários podem ser realizados conforme o andamento do projeto.

A seguir é apresentado um esboço dos componentes básicos da metodologia, que podem ser modificados de acordo com a necessidade de cada um e de cada projeto. Os componentes são (HOLLAR, 2006):

- Relação cliente/desenvolvedor: como toda metodologia ágil, o principal ponto é a comunicação entre o cliente e o desenvolvedor. Se o cliente não tiver disponibilidade para reuniões, ou mesmo para trocar e-mails, mensagens, telefonemas, etc., então o mesmo

deve designar alguém para tais tarefas;

- Reuniões: primeiramente tem-se uma reunião inicial para definição de alguns pontos básicos como os objetivos da aplicação, lista inicial de tarefas priorizadas para o desenvolvedor, um glossário para resolução de possíveis termos ambíguos e até mesmo desenvolver um protótipo de interface. As demais reuniões são de entrega e revisão nas quais o desenvolvedor apresenta as novas funcionalidades e o cliente apresenta seu parecer. Além disso, os objetivos são revistos e a lista de tarefas são reordenadas, caso necessário;

- Artefatos:

  - Lista de objetivos: os objetivos devem ser descritos de forma mais clara possível, identificando o que o sistema deve suprir, além dos desejos dos usuários finais. Tais objetivos podem ser categorizados por tipo de usuário, e geralmente são divididos em uma ou mais tarefas;

  - Lista de tarefas: uma tarefa pode seguir a seguinte sugestão "O (tipo de usuário) deve ser capaz de (ação)...". Em suma, as tarefas são ações que um usuário pode executar no sistema final;

  - Glossário: termos importantes ou termos que podem ser interpretados incorretamente deverão estar nesse glossário. Ambos, cliente e desenvolvedor, devem estar de acordo que tais termos estejam claros, concisos e corretos no contexto do projeto;

  - Código: algumas sugestões de boas práticas devem ser analisadas com relação à codificação do projeto, de modo que o que será produzido seja claro, consistente e legível. O uso de comentários para descrever classes e métodos, realizar testes unitários e seguir convenções de formatação de código podem ser de grande importância para a qualidade do código.

Após apresentar a metodologia, o autor aplica-a em um projeto para avaliar sua eficiência. O autor descreve o resultado da aplicação da metodologia como sendo satisfatória, pois o relacionamento entre o cliente e o desenvolvedor ocorre de maneira positiva, as reuniões ocorriam de forma satisfatória, e as entregas realizadas nas iterações estavam de acordo com o planejado.

Além dos pontos positivos acima, o autor apresenta algumas considerações que podem ser seguidas para um melhor aproveitamento da metodologia, são elas: garantir que a interação com o cliente ocorra sempre que necessário, garantir que as iterações realmente sejam curtas e realizar o desenvolvimento orientado por testes.

3.6 Análise das metodologias

Após apresentar as metodologias acima, é possível realizar uma breve comparação das suas estruturas, quais atividades elas englobam e quais atores estão presentes em cada uma delas. Como visto na Tabela 3.1 todas as metodologias utilizam uma combinação de duas ou mais metodologias. Essa semelhança mostra que as metodologias podem ser adaptadas de modo a atender melhor as necessidades de cada indivíduo, projeto ou cliente.

Tabela 3.1: Tabela comparativa das metodologias

	Scrum Solo	**Agile Solo**	**PXP**	**Cowboy**
Metodologias utilizadas	Scrum e PSP	Scrum, XP, Kanban e PSP	XP e PSP	Scrum, XP e Agile Unified Process
Atividades	Requirement, Sprint, Deployment e Management	Não especifica um processo, mas une um conjunto de práticas ágeis	Planejamento, Desenvolvimento e Pós conclusão	Requisitos e iteração com o cliente, Gerenciamento iterativo e Implantação
Atores	Product Owner, Desenvolvedor, Orientador e Grupo de validação	Cliente, Desenvolvedor e Supervisor	Cliente e Desenvolvedor	Cliente e Desenvolvedor

Outro ponto relevante é que três das metodologias apresentam um processo definido, no qual é possível perceber que pelo menos três etapas aparecem em comum, são elas: Requisitos/Planejamento, Desenvolvimento/Iteração e Implantação/Pós conclusão. Isso mostra que o processo de desenvolvimento de software individual tende a seguir uma ordem natural, que começa com o planejamento, passando pelo desenvolvimento até chegar no produto final.

Quando se fala nos papéis ou atores existentes nessas metodologias, tem-se que o papel do cliente e do desenvolvedor estão presente em todas, porém na Scrum Solo, o cliente é representado pelo Product Owner. Ainda na Scrum Solo, existe um grupo de validação que avalia o produto entregue e também há o papel de um Orientador, que age como um consultor dentro do processo. Outra metodologia que se diferencia é a Agile Solo, pois apresenta um papel de um supervisor que atua como um auditor da codificação do projeto.

Conclui-se, portanto, que para a definição de uma metodologia a se seguir, não basta simplesmente escolher uma e colocar em prática. Tal definição precisa ser elaborada de acordo com que o desenvolvedor está familiarizado, com o projeto que será realizado e também com a necessidade e perfil do cliente.

Na intenção de realizar um projeto piloto para demonstrar a aplicação de uma metodologia ágil voltada para o desenvolvimento solo, o autor deste trabalho procurou adotar uma metodologia que melhor se encaixasse em seu ambiente de trabalho, já que a mesma seria aplicada em

um ambiente real de desenvolvimento.

Por ser a metodologia mais recente encontrada, pelos tipos de clientes que solicitam desenvolvimento de aplicações e pelos tipos de projetos que são desenvolvidos no ambiente de trabalho do autor, além da curiosidade científica em aplicar o Scrum de maneira solo, a metodologia escolhida para o projeto piloto seria a Scrum Solo. No entanto, algumas observações foram levantadas para a utilização em seu ambiente de trabalho. As observações são:

- A existência de um orientador e de um grupo de validação: a priori esses papéis definidos pela metodologia Scrum Solo não são possíveis no ambiente de trabalho em que o autor se encontra, a abordagem precisa seguir com um único desenvolvedor e o mesmo ter contato direto com o cliente;
- O uso do PSP: pelo fato do autor deste trabalho já desenvolver software no instituto onde trabalha desde 2013, o nível de maturidade em que se encontra já está em um nível cíclico, inclusive com regras e um manual de desenvolvimento estabelecidos, sendo assim, não há necessidade de seguir o processo PSP e todos os artefatos propostos pela metodologia Scrum Solo.

Pensando nos pontos acima, uma nova abordagem é apresentada no próximo capítulo, para atender as necessidades do dia-a-dia de trabalho do autor desta dissertação. Além disso, dois projetos piloto são apresentados como resultado da aplicação da abordagem proposta. Tal abordagem seguirá os pontos em comum das metodologias citadas neste capítulo, juntamente com os pontos positivos apresentados durante a análise das mesmas.

3.7 Considerações finais

Este capítulo apresentou quatro metodologias de desenvolvimento de software que seguem princípios ágeis e podem ser utilizadas por uma única pessoa, apresentando seus conceitos, pontos positivos e negativos. Logo após foi apresentado uma breve análise dos pontos em comum entre elas.

No próximo capítulo será apresentado uma proposta de metodologia de desenvolvimento solo utilizando abordagens ágeis, e dois projetos piloto para a utilização da mesma serão vistos.

Capítulo 4

RESULTADOS DA ABORDAGEM DE DESENVOLVIMENTO

4.1 Considerações iniciais

Não é de hoje que muitas pessoas desejam ter seu próprio negócio, abrir uma empresa e começar a ganhar dinheiro com seu próprio empreendimento. Sabe-se também que na área de tecnologia da informação (TI), mais especificamente no âmbito de desenvolvimento de software, não é difícil começar um projeto sem muito recurso, ou até mesmo sem uma equipe. Basta um computador, criatividade e conhecimento em programação.

Muitas vezes essa pessoa que gostaria de começar a trabalhar nesse mercado, já tendo uma ideia de um possível produto, ou até mesmo com uma ideia de um possível cliente, geralmente não possui recursos ou uma equipe para ajudar a construir tal produto. Pensando nessa situação, seria mais produtivo e interessante seguir uma abordagem que auxilie a gestão em todo o processo de desenvolvimento do produto.

Da mesma maneira, uma empresa que está pensando em contratar alguém da área de TI para automatizar alguns processos, ou desenvolver aplicações para o uso interno, precisa contratar um funcionário que tenha conhecimento em alguma abordagem consolidada que o auxilie no desenvolvimento do projeto, de modo a não comprometer a qualidade do mesmo.

Como visto no Capítulo 2, sabe-se que seguir uma metodologia é fundamental para o sucesso de um processo de desenvolvimento de software. Viu-se também no Capítulo 3, que existem algumas opções para o desenvolvimento ágil e individual de software. Sendo assim, neste capítulo é apresentado uma proposta de uma abordagem de desenvolvimento de software que tem como base os princípios ágeis, e pode ser utilizada por uma única pessoa. O obje-

tivo é adaptar e organizar práticas e filosofias que são utilizadas por equipes na construção de aplicações de modo geral para cenários de desenvolvimento individual.

O capítulo segue com uma visão geral da abordagem proposta, com a descrição dos artefatos utilizados e os papéis que constituem a mesma. Logo após, é apresentado como essa abordagem é aplicada na prática no desenvolvimento de aplicações web dentro do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC-USP), e também no desenvolvimento de software em um cenário comercial. Também são apresentados os resultados da aplicação da abordagem, e por fim, são apresentadas as considerações finais do capítulo.

4.2 Visão geral da proposta

Este trabalho visa apresentar uma alternativa para o desenvolvimento solo que utiliza como base a metodologia ágil *Scrum*. Essa alternativa proposta é adequada para os casos em que não se tem um time de desenvolvimento, e sim apenas um único desenvolvedor, e a interação com o solicitante da aplicação pode ser realizada de uma maneira bem acessível.

Tal proposta pode ser utilizada em instituições ou empresas que demandam automatizações de processos, ou qualquer outro serviço que possa ser informatizado por um único indivíduo. Essa proposta também é adequada para ser utilizada por alguém que queira começar sozinho algum projeto de desenvolvimento de uma aplicação.

Fazendo uma analogia a metodologia *Scrum*, e já adicionado os artefatos sugeridos, tem-se a seguir os descritivos do que é proposto na abordagem de desenvolvimento de aplicações:

- Análise de requisitos: nesta etapa, o próprio desenvolvedor realiza uma ou mais reuniões juntamente com o(s) solicitante(s), nas quais se define as funcionalidades da aplicação e gera-se uma lista priorizada de tais funcionalidades. Em casos em que não existe um cliente, o próprio desenvolvedor gera a lista das funcionalidades da aplicação que se deseja construir;

- *Product Backlog*: contém todas as funcionalidades que deverão estar contempladas na aplicação após a finalização do desenvolvimento, este artefato é gerado na análise de requisitos, e pode ser atualizado conforme o andamento do projeto;

- *Sprint Planning*: apesar de não ter um time atuando no desenvolvimento do software, é feito um planejamento ao início de cada *Sprint* para determinar quais funcionalidades

serão implementadas. Este planejamento pode ser feito juntamente com o solicitante, na própria apresentação da *Sprint* anterior, ou pode ser feito conforme o entendimento do desenvolvedor;

- *Sprint Backlog*: contém as tarefas que deverão ser desenvolvidas na *Sprint* e apresentadas na próxima reunião com o solicitante;
- *Sprint*: geralmente dura cerca de 15 a 30 dias, e nesse período são desenvolvidas as tarefas do *Sprint Backlog*. Ao final da *Sprint*, uma ou mais funcionalidades novas podem ser testadas pelo(s) solicitante(s) em um ambiente de testes;
- *Daily Scrum*: a reunião diária não é realizada como na descrição literária do *Scrum*, porém, todo dia, o próprio desenvolvedor pode separar um pequeno tempo para fazer os seguintes questionamentos: o que fiz de ontem para hoje ajudou na meta da *Sprint*? O que farei hoje? Existe algum impedimento que preciso resolver? Tais perguntas precisam estar claras para que a meta seja alcançada de forma mais objetiva e rápida;
- Incremento do sistema: como resultado da *Sprint*, é gerado um incremento no sistema, que pode ser uma ou mais funcionalidades novas que é apresentado ao solicitante;
- Revisão da *Sprint*: realiza-se uma retrospectiva da *Sprint*, para saber o que deu certo e o que deu errado na aplicação do processo, para assim melhorar a eficiência na próxima *Sprint*;
- Testes e integração: implementa-se a nova funcionalidade na aplicação em um ambiente de testes. Com isso, o(s) solicitante(s) pode(m) realizar os testes necessários para verificar se está ou não de acordo com os requisitos iniciais;
- Produto final: aplicação final, testada e homologada pelo solicitante, com todas as tarefas do *Product Backlog* implementadas;
- Documentação: a documentação tende a ser a menor possível, porém o suficiente para que se tenha registro do que é a aplicação. Recomenda-se que seja elaborado um modelo relacional do banco de dados, pois consegue-se ter uma visão geral da aplicação, e que o mesmo seja uma das primeiras tarefas a ser realizada.

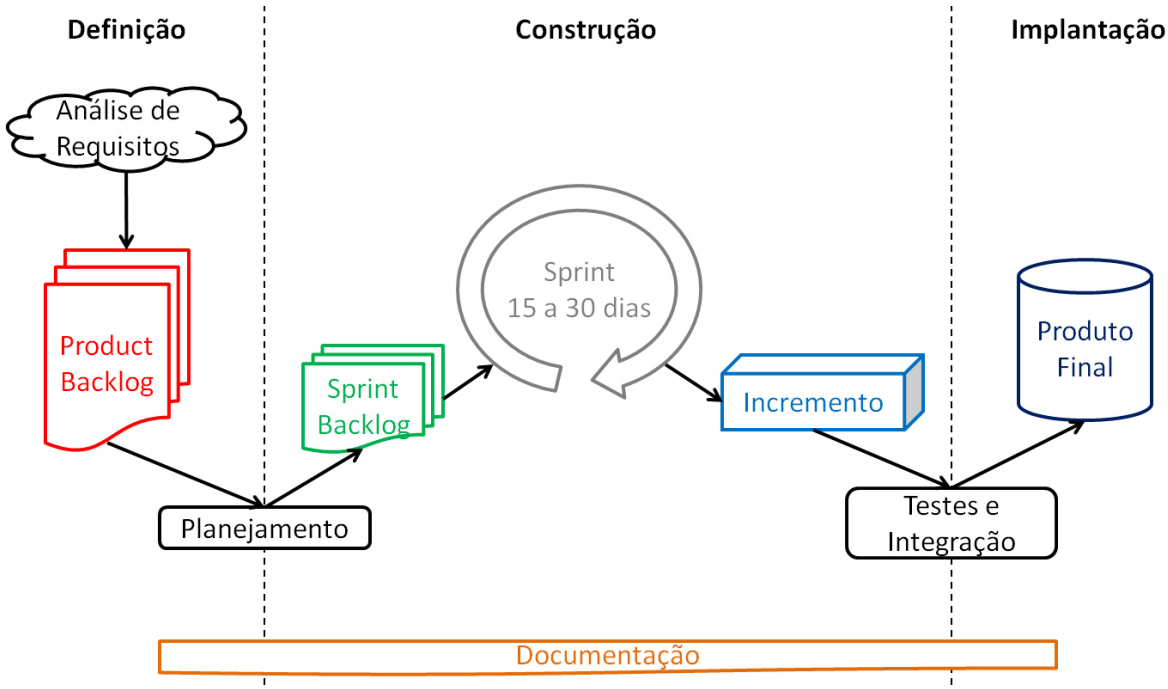
A Figura 4.1 apresenta uma visão geral da abordagem dessa proposta, que encontra-se estruturada em três partes: definição, construção e implantação. Na parte de definição é realizada a análise de requisitos e também é criado o *Product Backlog*. Caso a execução dessa primeira

parte seja uma reunião com o cliente, pode-se também realizar um planejamento da *Sprint* inicial, separando as primeiras funcionalidades a serem entregues.

Na etapa de construção tem-se a codificação do projeto em si, na qual as *Sprints* do projeto são executadas. Esta etapa começa com o planejamento da *Sprint*, gerando o *Sprint Backlog*. Após isso, tem-se a criação do que se é esperado dentro da *Sprint*, e então é gerado um incremento no produto, no qual o cliente já pode realizar testes para verificar se está de acordo com o que foi proposto.

Por fim, tem-se a etapa de implantação, que é realizada a entrega do produto final, no qual todas as funcionalidades previstas devem estar implementadas na aplicação. Porém, são realizados os últimos testes antes da entrega definitiva, validando assim todo o *Product Backlog*. Tendo a aprovação do cliente, a aplicação pode ser colocada em produção para a utilização definitiva.

Figura 4.1: Visão geral da abordagem proposta



Para essa abordagem de desenvolvimento, alguns papéis são redefinidos, levando-se em conta que apenas um desenvolvedor será responsável pela aplicação, são eles:

- Pessoas interessadas: alguém que possui uma ideia de um possível produto, ou qualquer um que realize uma tarefa e percebe que tal tarefa poderia ser automatizada ou melhorada, pode se tornar um solicitante. Pode ser um único indivíduo, ou até mesmo um grupo de pessoas com interesse comum na possível aplicação;

- Desenvolvedor: diferente do *Scrum* que possui um time de desenvolvimento (*Scrum Team*), a ideia desta proposta é que a aplicação seja desenvolvida por somente um programador. E o mesmo também seja responsável por gerenciar os requisitos e funcionalidades da aplicação diretamente com as pessoas interessadas.

Pelo fato da abordagem aqui proposta ser uma adaptação da metodologia *Scrum*, alguns dos papéis não foram anteriormente definidos, porém estão incorporados em outros papéis como descrito abaixo:

- *Product Owner*: por ser uma única pessoa no time de desenvolvimento, geralmente o próprio solicitante é considerado o *Product Owner*, pois ele tem conhecimento do produto que está sendo desenvolvido, e portanto, pode passar informações essenciais para o desenvolvimento do software. Vale reforçar que o solicitante precisa estar sempre disponível para tratar qualquer informação importante sobre a aplicação;
- *Scrum Master*: na teoria este papel também não existe nesta abordagem, porém o próprio desenvolvedor assume as características do *Scrum Master*, pois ele deve resolver quaisquer impedimentos que apareçam, e também garantir que está seguindo a metodologia proposta.

Em casos em que não se tem um cliente, o próprio desenvolvedor é responsável tanto pela ideia do produto, quanto pelo seu desenvolvimento, o que torna ainda mais complexo o processo, pois ele tem que confiar plenamente na sua ideia e no seu potencial.

É importante frisar algumas características pessoais e profissionais que o desenvolvedor pode aprimorar para que tenha mais sucesso na aplicação desta metodologia. Uma delas é ter uma boa comunicação e saber se relacionar com as pessoas, pois ele é quem terá que conversar com quem for preciso para solucionar um problema, ou até mesmo entender o que o cliente quer na sua aplicação.

Outro ponto importante para o desenvolvedor é que tenha uma boa visão gerencial do negócio, além de ter bons conhecimentos técnicos, tanto do ponto de vista de programação, quanto da análise de dados. Uma vez que terá que realizar todas etapas no decorrer do projeto, desde o levantamento de requisitos, até chegar no produto final, testado e implantado.

Além desses artefatos, características e papéis descritos, a abordagem proposta precisa seguir os cinco princípios definidos na metodologia XP e descritos na Seção 2.3.2.2. São eles:

- Comunicação: para obter sucesso no projeto e para que todos os pontos importantes do projeto possam ser tratados com agilidade e atenção, a comunicação entre desenvolvedor e solicitante precisa acontecer sempre que necessário;
- *Feedback*: a proatividade do solicitante em fornecer o máximo de *feedback* ao desenvolvedor é de extrema importância para que seja construído realmente o que é necessário no projeto;
- Simplicidade: concentrar forças para entregar o que de fato será útil para o solicitante, de forma mais simples e clara possível;
- Coragem: aceitar as mudanças, sugestões e críticas recebidas pelo solicitante, e informar a real situação do desenvolvimento, sem medo de dizer quando algo estiver atrasado ou com algum impedimento grave. O mais importante é assumir que existe algo que não está bom, e tentar resolver;
- Respeito: os pensamentos das pessoas não são iguais, por isso é muito importante respeitar as limitações, opiniões e argumentos de todos os envolvidos no projeto.

Cabe enfatizar que esses princípios são recomendados nessa abordagem, pois são de extrema importância para que se tenha uma equidade de tratamento dentre todos os possíveis solicitantes, tenha um ambiente agradável e respeitoso para trabalhar, além da transparência de todo o processo.

Além dos princípios citados, algumas práticas da metodologia XP podem ser utilizadas para a melhora da qualidade do produto desenvolvido, uma delas é a prática de refatoração. Refatorar um código, ou simplesmente revisar o que foi construído ao final de cada *Sprint*, pode ser uma tarefa que traga benefícios para o projeto, tanto por poder encontrar possíveis erros na programação, quanto por poder simplificar algo que não estava claro.

Outra prática é a do *design* simples, pois assim garante que todo o tempo investido no projeto seja com algo que realmente terá valor no produto final, evitando assim que o desenvolvedor não gaste tempo com funcionalidades que não serão utilizadas, ou criando documentos que não são essenciais para o entendimento da aplicação.

Após essa apresentação inicial da proposta, a seguir será descrito um projeto piloto da abordagem proposta em um ambiente real de desenvolvimento, que no caso foi realizado no setor de desenvolvimento da Seção Técnica de Informática do ICMC-USP, e logo depois outro projeto será apresentado, sendo este realizado de maneira *freelancer*.

4.3 Projeto piloto da metodologia ágil em cenário solo no ICMC-USP

Nesta seção é apresentado um projeto piloto de como a abordagem de desenvolvimento de software anteriormente proposta é utilizada na Seção Técnica de Informática do ICMC-USP. Tal abordagem tem como base a metodologia *Scrum*, e segue algumas práticas e valores básicos do XP, porém, como ambas metodologias foram criadas para serem utilizadas por equipes, são apresentadas algumas particularidades e adaptações utilizadas no desenvolvimento solo de aplicações web no ICMC-USP.

A equipe de desenvolvimento do ICMC-USP atualmente é composta por dois analistas e três técnicos, e todos desenvolvem seguindo o Manual de Desenvolvimento do ICMC[1], o que garante a padronização e compatibilidade dos sistemas desenvolvidos no instituto. Apesar da existência dessa equipe, na maioria dos casos, cada um desenvolve um sistema isolado, pois assim consegue-se atender um maior número de solicitantes ao mesmo tempo. Devido a essa distribuição de sistemas, cada desenvolvedor segue uma abordagem para gerenciar suas tarefas.

O autor deste trabalho é um dos membros da equipe de desenvolvimento, e se propôs a aplicar a abordagem aqui proposta nos trabalhos por ele executados, gerando assim o projeto piloto para validar a mesma. Sendo assim, tal abordagem passou a ser utilizada no desenvolvimento de aplicações web no ICMC-USP, que está descrita na seção anterior, e foi apresentada na Figura 4.1.

Nota-se que a estrutura dessa abordagem é bem parecida com a metodologia *Scrum*. Na parte da esquerda tem-se a etapa de definição do projeto, que é a análise de requisitos e a criação do *Product Backlog*. Na parte central tem-se a etapa de construção, que engloba desde o planejamento das *Sprints* até o teste das funcionalidades prontas. E na parte de implantação o produto final é entregue com todas as funcionalidades testadas.

Antes de entrar na aplicação da abordagem, algumas características pessoais e técnicas o desenvolvedor precisa explorar em si mesmo para que possa realizar todas as tarefas de uma maneira clara e objetiva. Uma vez que irá se relacionar com as pessoas interessadas nas aplicações que serão desenvolvidas, terá que informar prazos, resolver pendências, dentre outras tarefas, é de extrema importância que o desenvolvedor tenha habilidade de se relacionar com outras pessoas, saber gerenciar projetos, além de saber lidar com o inesperado.

Ao iniciar a aplicação da metodologia, na etapa de definição, o primeiro passo é realizar a

[1] http://www.icmc.usp.br/e/335a7

análise de requisitos, para isso, uma reunião juntamente com o solicitante precisa ser realizada, no qual se define as funcionalidades do sistema e é gerado o *Product Backlog*. Geralmente esse artefato já é gerado em uma ordenação priorizada, e nessa mesma reunião também é executado o planejamento para a *Sprint* inicial.

Recomenda-se, como primeira tarefa da primeira *Sprint*, a criação da modelagem do banco. A ideia é fazer a modelagem mais próxima do que será o banco de dados final, porém, sabe-se que em cada *Sprint*, ou até mesmo no andamento de cada tarefa, pode-se ter alterações para suprir necessidades não previstas. Tais alterações são realizadas no momento em que aparecem e a documentação é atualizada sempre que houver modificações.

Outros documentos podem ser criados para a compreensão de uma ou mais funcionalidades, porém a ideia é manter o projeto o mais simples possível, evitando excesso de documentação. Só criar um documento se realmente agregar valor ao projeto.

Na parte de construção da abordagem concentra-se a maior parte do projeto. Começa com o planejamento da *Sprint*, que geralmente é feito em uma reunião com o solicitante, em que se define quais tarefas serão realizadas nas próximas semanas e então é gerado o *Sprint Backlog*, ou seja, as tarefas que serão executadas na mesma.

Após isso, tem-se a *Sprint* propriamente dita, que geralmente tem duração de 15 a 30 dias, e no final da *Sprint*, é apresentado um incremento do sistema, ou parte do sistema que já pode ser testado pelo solicitante. Durante esta etapa, todos os dias o desenvolvedor realiza as seguintes perguntas: o que fiz de ontem para hoje ajudou na meta da *Sprint*? O que farei hoje? Existe algum impedimento que preciso resolver?

Caso exista algum impedimento, ele precisa ser eliminado o quanto antes para que o projeto continue no caminho certo. E se para resolver tal problema for necessário uma conversa com o solicitante, ele precisa estar sempre disponível para esclarecer possíveis dúvidas.

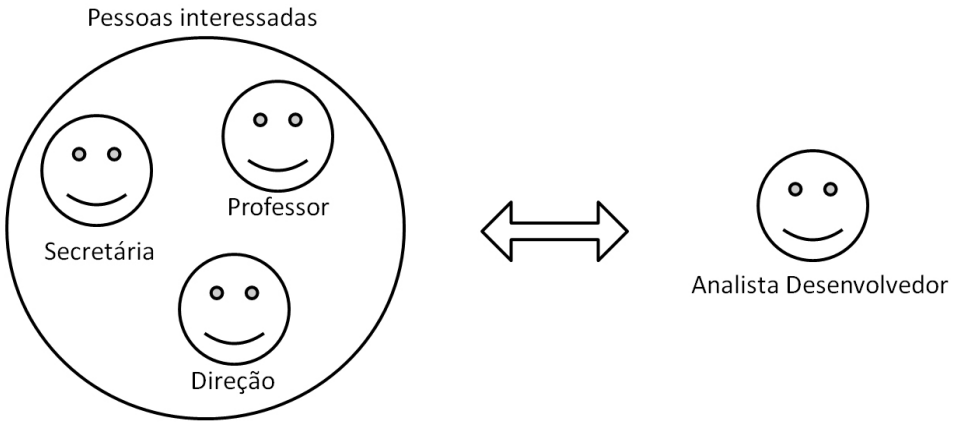
Com a *Sprint* finalizada, ocorre também uma revisão dela, para anotar os pontos positivos e negativos do processo, a fim de manter o que funcionou, e evitar que aconteça novamente aquilo que não deu certo. Por fim, vem a integração de tal incremento no que será o produto final, aumentando as funcionalidades da aplicação a cada etapa.

A parte de implantação da Figura 4.1 mostra a parte final do processo, na qual são realizados testes em ambiente de homologação, e, após validação do solicitante, o produto é colocado em produção para uso definitivo. Para o gerenciamento do projeto no ICMC-USP é utilizado a

ferramenta Redmine[2], e a mesma encontra-se integrada com o controlador de versão Git[3], para versionar toda documentação e códigos gerados no decorrer do projeto.

Diferentemente das metodologias ágeis citadas no Capítulo 2, que possuem vários papéis, a adaptação aqui proposta tem um número bem reduzido de papéis, já que possui apenas um desenvolvedor, e nem todos os artefatos são utilizados da forma como apresentado na literatura. A Figura 4.2 apresenta os papéis existentes nessa metodologia, que são apenas dois, e esses dois papéis precisam estar em constante comunicação para que a abordagem seja utilizada com eficiência.

Figura 4.2: Papéis envolvidos na abordagem proposta



Os papéis utilizados na abordagem proposta são:

- Pessoas interessadas: dentro do ICMC-USP, qualquer um que realize uma tarefa e percebe que tal tarefa poderia ser automatizada ou melhorada, pode se tornar um solicitante, desde uma secretária de departamento, um técnico que cuida da parte financeira, um analista que trata informações acadêmicas, um docente que precisa informatizar algum procedimento administrativo, ou até mesmo várias pessoas juntas que precisam de uma interação com diferentes setores;
- Analista desenvolvedor: embora exista cinco desenvolvedores no ICMC, cada um desenvolve um sistema isolado, e cada um trabalha os requisitos do sistema diretamente com o solicitante, o que exige um conhecimento técnico mais abrangente.

Como existe só uma pessoa no time de desenvolvimento, os papéis do *Scrum Master* e do *Product Owner* estão incorporados no desenvolvedor e no solicitante, respectivamente. O

[2]https://www.redmine.org
[3]https://git-scm.com

próprio desenvolvedor é responsável por apresentar as evoluções da aplicação para o solicitante. Mais uma vez reforçando a comunicação entre esses papéis.

Dentro deste projeto piloto, algumas aplicações foram desenvolvidas utilizando a abordagem aqui apresentada, e uma breve descrição desses produtos encontram-se exemplificadas no Anexo A.

4.3.1 Resultado da aplicação da metodologia no primeiro projeto

O autor deste trabalho tem atuado no desenvolvimento de software de maneira solo no ICMC-USP desde 2013, porém, no início não se utilizava uma abordagem ágil, mas um processo *ad hoc*. Em meados de 2016 a abordagem apresentada neste trabalho começou a ser aplicada pelo autor no desenvolvimento de novas aplicações web, o qual vem sido utilizada até então.

Ao aplicar a metodologia, algumas melhorias já foram observadas nos primeiros passos, uma vez que o solicitante estava sempre presente para esclarecer o que fosse preciso, o processo estava melhor definido, e com as *Sprints*, o *feedback* começou a ser fundamental para que defeitos e imprevistos fossem solucionados o quanto antes.

Com a evolução do processo, as aplicações desenvolvidas também tiveram sua complexidade aumentada, o que permitiu uma abrangência de atuação maior do que vinha sendo desenvolvido. Prova dessa evolução pode-se utilizar um exemplo de uma aplicação desenvolvida para o gerenciamento de solicitações de treinamento e desenvolvimento dos funcionários do ICMC-USP (sistema de T&D). Tal sistema surgiu com uma primeira versão antes do uso dessa metodologia, e uma segunda versão foi criada utilizando a metodologia, no qual praticamente toda a aplicação foi reescrita.

Para contextualizar, o principal objetivo do sistema de T&D é armazenar dados e gerenciar o fluxo das solicitações de treinamento do quadro de funcionários do ICMC-USP. Possui um formulário de entrada em que o funcionário solicita o treinamento desejado, inserindo informações sobre o treinamento em si, sobre o apoio solicitado e também sobre a proposta pós treinamento.

O sistema possui um fluxo de aprovação, que, após o funcionário enviar os dados do formulário, é enviado um e-mail para a chefia avaliar a solicitação. A chefia então entra no sistema e faz a avaliação do pedido, podendo aprovar ou negar, sendo que ao negar, precisa-se justificar sua posição.

Quando a chefia avalia a solicitação, um e-mail é enviado para os membros da comissão de

T&D do ICMC-USP, que também avaliam a solicitação, observando, além dos dados cadastrados, os comentários apresentados pela chefia. Depois que a comissão de T&D avalia, é enviado um e-mail para o presidente da comissão de qualidade e produtividade do instituto (CQP), que dá o parecer final sobre a solicitação.

Ao final desse processo, um e-mail é enviado para o solicitante dizendo que sua solicitação foi avaliada, no qual pode entrar no sistema e verificar os pareceres dos avaliadores. Um e-mail também é enviado para o responsável do setor financeiro para tomar as providências com relação ao pagamento do treinamento.

Seguindo essa contextualização, uma primeira versão do sistema foi desenvolvida utilizando uma metodologia *ad hoc*, tendo como base o método cascata. Foi realizado o levantamento de requisitos, juntamente com a equipe de T&D, e criado um modelo relacional do banco de dados conforme mostra a Figura 4.3. Tal modelagem foi realizada para atender basicamente a quatro funcionalidades definidas como principais da aplicação:

- Solicitar pedido: formulário com os dados da solicitação e do usuário;
- Avaliar pedido (chefia, T&D, CQP): avaliação pelos responsáveis de cada etapa, controlando o fluxo de aprovações;
- Exibir detalhes do pedido: apresenta uma tela com os detalhes da solicitação, e também os pareceres dos avaliadores;
- Apresentar páginas estáticas: apresenta uma página para divulgar a política de T&D[4], outra para documentos importantes e outra para orientações de preenchimento do formulário.

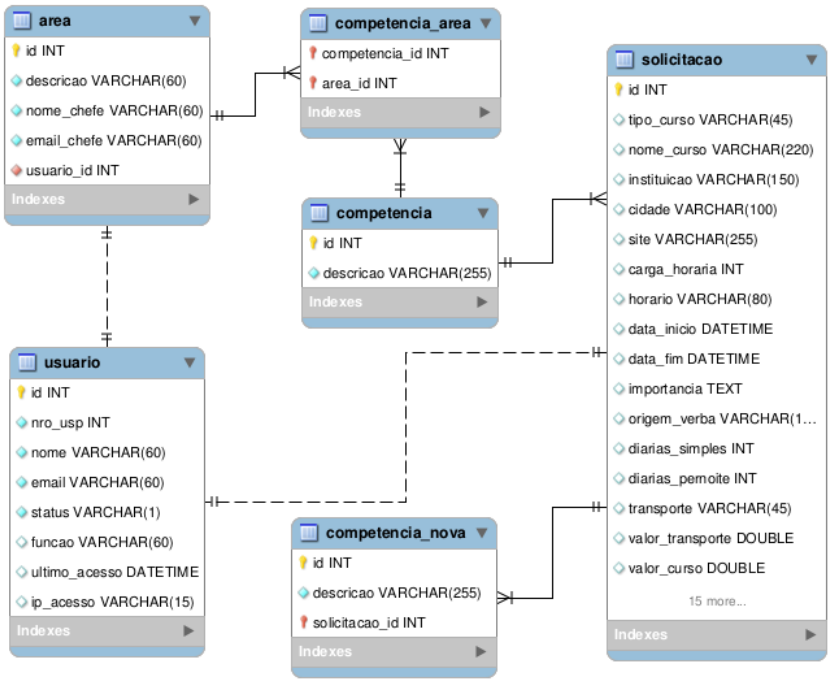
Após realizar a modelagem da aplicação, o desenvolvimento foi realizado e o sistema foi apresentado com todas as funcionalidades prontas. Entre a reunião de requisitos, e a reunião de entrega, não teve outra comunicação com o grupo solicitante, e o processo todo teve duração de quatro meses, de novembro de 2014 à fevereiro de 2015. Resultando em uma média de uma funcionalidade desenvolvida por mês.

Mais informações sobre essa versão, assim como registro das telas da aplicação, encontram-se no Anexo B deste trabalho.

Para o desenvolvimento da nova versão do sistema de T&D, a metodologia proposta neste trabalho foi utilizada. O processo começou com uma reunião de planejamento no começo de

[4]http://ted.icmc.usp.br/politica/

Figura 4.3: Modelo relacional da primeira versão do sistema de T&D



maio de 2017 com a equipe responsável pelo sistema para **analisar e definir os requisitos**. O ***Product Backlog*** foi primeiro artefato gerado e contém as seguintes funcionalidades:

- Gerenciar usuários e permissões: módulo para gerenciar os usuários e as permissões de acesso ao sistema;
- Gerenciar tipo de cursos: módulo para gerenciar os tipos de cursos que o usuário pode solicitar;
- Cadastro de setores: gerenciamento dos setores do instituto, no qual se define a área em que o setor está alocado, e também o respectivo chefe;
- Gerenciar mapa de competências do setor: módulo para gerenciar o mapa de competências de cada setor;
- Solicitar pedido: formulário com os dados da solicitação e do usuário;
- Avaliar pedido (chefia imediata, chefia de área, T&D e CQP): avaliação pelos responsáveis de cada etapa, controlando o fluxo de aprovações;
- Exibir detalhes do pedido: apresenta uma tela com os detalhes da solicitação, e também os pareceres dos avaliadores;

- Devolver pedido: devolução para o avaliador anterior, até chegar no solicitante para realizar as alterações necessárias no formulário de solicitação;
- Histórico do fluxo: exibe o histórico de todas as ações realizadas para um treinamento, apresentando quem e quando realizou a avaliação e quais observações foram apresentadas;
- Gerenciar FAQ: módulo para gerenciar o FAQ do T&D, que contém um conjunto de perguntas e respostas sobre possíveis dúvidas de capacitação;
- Mural de oportunidades: utilizado para gerenciar as oportunidades de T&D que são divulgadas para os funcionários do instituto;
- Módulo financeiro: responsável por controlar as despesas gastas com diárias, transporte, inscrição e demais custos das solicitações, fazendo um balanço com o saldo reservado para cada ano;
- Páginas dinâmicas: transformação das páginas estáticas em páginas dinâmicas, para que o próprio grupo de T&D possa gerenciar o conteúdo;
- Resumo anual: apresenta o número de treinamentos realizados pelo funcionário no ano, além do número de horas que ele treinou, do número de treinamentos do seu setor, e do total de treinamentos no instituto inteiro;
- Prestação de contas: após a realização do treinamento, o funcionário precisa realizar uma prestação de contas no sistema para que o mesmo possa fazer outra solicitação de treinamento;
- E-mails automáticos: automatizar o envio de alguns e-mails do sistema, como por exemplo, uma cobrança da prestação de contas após um período da finalização do treinamento, um lembrete para um avaliador caso exista um pedido pendente e também um aviso para o solicitante enviar uma solicitação que está salva mas ainda não foi enviada para a chefia avaliar.

Ainda na primeira reunião, foi realizado um planejamento da primeira *Sprint* (***Sprint Planning***) e elaborado o ***Sprint Backlog*** inicial. A primeira *Sprint* está representada na Tabela 4.1 e contém um conjunto de três funcionalidades, divididas em dez tarefas, na qual foram desenvolvidas em um período de oito semanas.

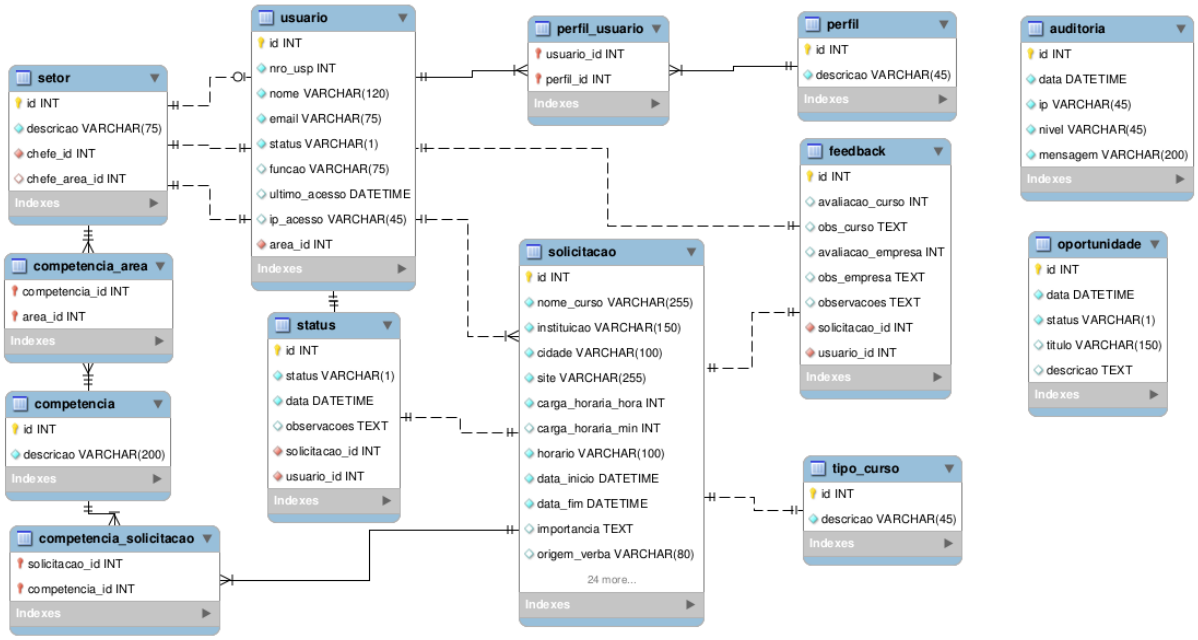
Seguindo a proposta da metodologia, a primeira tarefa a ser elaborada foi a criação do modelo relacional do banco de dados. A Figura 4.4 apresenta a nova modelagem do sistema

Tabela 4.1: Tabela com as funcionalidades da Sprint inicial do projeto

	***Sprint* inicial**
Tarefa inicial:	Criar modelo do banco
Funcionalidade:	Gerenciar usuários e permissões
Tarefas:	Fazer lógica de login
	Buscar dados na base da USP/SP
	Definir permissões do sistem
	Gerenciar permissões dos usuários
Funcionalidade:	Solicitar pedido
Tarefas:	Fazer view do formulário
	Lógica para salvar uma solicitação
	Lógica para enviar solicitação
Funcionalidade:	Gerenciar tipos de cursos
Tarefas:	CRUD dos tipos
	Listagem dos tipos no formulário de solicitação
Total de funcionalidades:	3
Total de tarefas:	10
Tempo de desenvolvimento:	8 semanas

completo, no qual se pode ver que contém mais do que o dobro de tabelas da versão anterior, aumentando assim o tamanho e a complexidade do sistema.

Figura 4.4: Modelo relacional da segunda versão do sistema de T&D



Durante o processo de desenvolvimento da Sprint, no começo de cada dia, o desenvolvedor separava uns minutos para verificar se existia algum impedimento, e se tudo estava andando conforme o esperado (referente ao ***Daily Scrum***). Todas as dúvidas ou imprevistos que surgiam eram tratados diretamente com um responsável do grupo solicitante o quanto antes.

Ao final da *Sprint*, uma reunião foi realizada com a equipe solicitante para apresentar as novas funcionalidades implementadas. Um **ambiente de testes** foi liberado para a equipe utilizar o sistema e deixar suas observações. Após a realização dos testes, as novas funcionalidades foram **incrementadas no sistema**. Por fim, o desenvolvedor realizou uma **retrospectiva** para ver quais pontos da metodologia poderiam ser melhorados e quais poderiam ser mantidos.

Esse processo se repetiu por mais sete vezes, passando pela segunda até a oitava *Sprint* do projeto. As funcionalidades e tarefas executadas nestas *Sprints* estão representadas nas tabelas de 4.2 à 4.8.

Tabela 4.2: Tabela com as funcionalidades da segunda *Sprint* do projeto

	***Sprint* 2**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Cadastro de setores
Tarefas:	CRUD dos setores
	Atualização usando dados de SP
Funcionalidade:	Avaliar pedido
Tarefas:	Criar form de avaliação
	Lógica do controle de fluxo
Tarefa final:	Atualizar documentação
Total de funcionalidades:	2
Total de tarefas:	6
Tempo de desenvolvimento:	3 semanas

Tabela 4.3: Tabela com as funcionalidades da terceira *Sprint* do projeto

	***Sprint* 3**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Gerenciar mapa de competências do setor
Tarefas:	CRUD das competências
	Vincular competências ao setor
Funcionalidade:	Vincular competências à solicitação
Tarefas:	Atualizar form de solicitação
Tarefa final:	Atualizar documentação
Total de funcionalidades:	1
Total de tarefas:	6
Tempo de desenvolvimento:	2 semanas

Após todas as funcionalidades desenvolvidas e testadas o sistema foi liberado para uso em produção. Entre o início e o fim do projeto foram 8 meses de comunicação e desenvolvimento, de maio de 2017 à dezembro de 2017. Implementou-se um total de 16 funcionalidades divididas em 57 tarefas. A Imagem 4.5 apresenta um gráfico da evolução do desenvolvimento das funcionalidades no decorrer dos meses do projeto. Mais informações sobre essa versão da aplicação, assim como registro das telas, encontram-se no Anexo C deste trabalho.

Tabela 4.4: Tabela com as funcionalidades da quarta *Sprint* do projeto

	***Sprint* 4**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Exibir detalhes do pedido
Tarefas:	Criar view dos detalhes
	Controle das requisições
Funcionalidade:	Devolver pedido
Tarefas:	Criar form de devolução
	Lógica do controle de fluxo
Funcionalidade:	Histórico do fluxo
Tarefas:	Criar view do histórico
	Controle das requisições
	Atualizar página de detalhes
Tarefa final:	Atualizar documentação
Total de funcionalidades:	3
Total de tarefas:	9
Tempo de desenvolvimento:	5 semanas

Tabela 4.5: Tabela com as funcionalidades da quinta *Sprint* do projeto

	***Sprint* 5**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Gerenciar FAQ
Tarefa:	CRUD do FAQ
Funcionalidade:	Mural de oportunidades
Tarefas:	CRUD das oportunidades
	Criar view específica para o mural
Tarefa final:	Atualizar documentação
Total de funcionalidades:	2
Total de tarefas:	5
Tempo de desenvolvimento:	3 semanas

Tabela 4.6: Tabela com as funcionalidades da sexta *Sprint* do projeto

	***Sprint* 6**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Módulo financeiro
Tarefas:	Listagem dos pedidos que tenham custos
	Gerenciar origem de verbas
	Controlar orçamentos anuais
Funcionalidade:	Prestação de contas
Tarefas:	Criar form de prestação de contas
	Atualizar módulo financeiro
Tarefa final:	Atualizar documentação
Total de funcionalidades:	2
Total de tarefas:	7
Tempo de desenvolvimento:	4 semanas

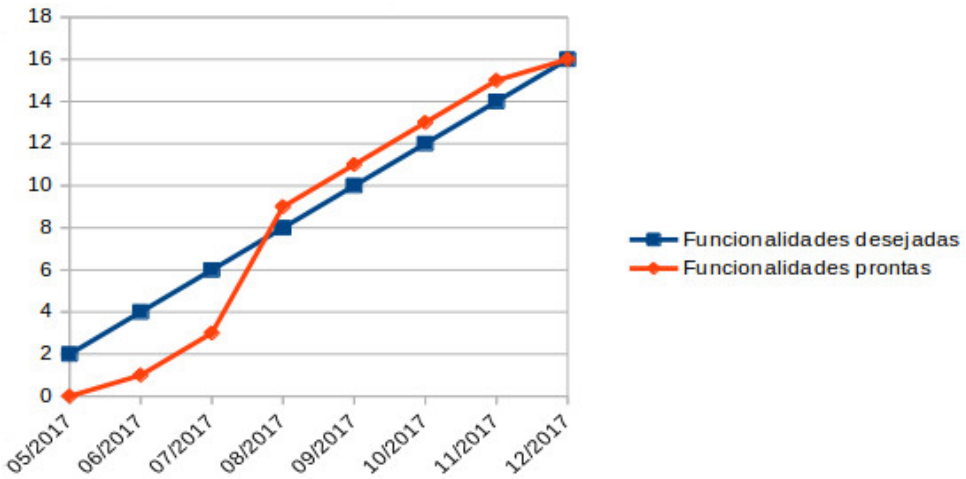
Tabela 4.7: Tabela com as funcionalidades da sétima *Sprint* do projeto

	***Sprint* 7**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Páginas dinâmicas
Tarefas:	CRUD das páginas
	View da política de T&D
	View dos documentos
Funcionalidade:	Resumo anual
Tarefas:	Total de solicitações do funcionário
	Total de horas treinadas no ano
	Total de solicitações do setor do funcionário
	Total de solicitações do instituto
Tarefa final:	Atualizar documentação
Total de funcionalidades:	2
Total de tarefas:	9
Tempo de desenvolvimento:	4 semanas

Tabela 4.8: Tabela com as funcionalidades da oitava *Sprint* do projeto

	***Sprint* 8**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	E-mails automáticos
Tarefas:	Cobrança da prestação de contas
	Lembrete para o avaliador sobre pedido pendente
	Aviso ao solicitante de solicitação não enviada para avaliação
Tarefa final:	Atualizar documentação
Total de funcionalidades:	1
Total de tarefas:	5
Tempo de desenvolvimento:	2 semanas

Figura 4.5: Evolução das funcionalidades durante o projeto



Com a finalização dessa segunda versão, pode-se comparar alguns pontos com o processo da primeira versão. O primeiro ponto é a comunicação entre desenvolvedor e solicitante que permitiu que fosse criada uma ferramenta complexa e objetiva, além de apresentar várias mudanças positivas e úteis ao longo do desenvolvimento, devido ao *feedback* rápido por parte dos solicitantes.

Mesmo sendo uma aplicação mais complexa, o número de funcionalidades desenvolvidas por mês subiu de 1 para 2, ou seja, o dobro de funcionalidades por mês, mostrando assim que essa abordagem de desenvolvimento de software pode ser uma boa proposta para se utilizar, tanto que que continuará sendo aplicada e melhorada pelo autor no desenvolvimento de novas aplicações no ICMC-USP.

Após apresentar a comparação acima, serão explorados a seguir alguns pontos positivos e outros negativos sobre a utilização dessa metodologia proposta. Primeiramente, os pontos positivos:

- Melhor interação com o solicitante: o contato direto com o solicitante realmente é um ponto muito forte dessa abordagem, por mais que o papel do *Product Owner* e do *Scrum Master* seja de grande importância na metodologia *Scrum*, a interação direta entre desenvolvedor e solicitante superou as expectativas, agilizando a resolução dos problemas encontrados no projeto;

- *Feedback* rápido: seguindo na linha da interação com o solicitante, o *feedback* rápido permitiu que as modificações acontecessem antes de o projeto crescer com as funcionalidades erradas ou mal interpretadas;

- Redução do número de falhas: um ponto positivo muito interessante foi a redução do número de falhas, pois, após a implantação da aplicação, ainda não foi realizado nenhuma manutenção, seja corretiva, seja evolutiva;

- Autonomia para tomar decisões: como o desenvolvedor conhece todas as funcionalidades e etapas do projeto, já que ele é quem elaborou todo o código, a confiança entre ele e o solicitante aumenta, e assim, o mesmo passa a tomar decisões em quase todos os problemas ou dúvidas que aparecem, e quase sempre o solicitante concorda com tais propostas;

- Melhor gerenciamento do tempo: por ser um processo bem estruturado, o desenvolvedor tem um controle mais efetivo do seu tempo de trabalho, podendo apresentar prazos mais realísticos ao solicitante.

Apesar dos pontos positivos anteriormente apresentados, alguns pontos ainda precisam ser revisados ou melhorados, são eles:

- Documentação: pela abordagem acreditar na documentação mínima, e não definir outro modelo senão a modelagem relacional, pode ser que essa omissão de documentação atrapalhe em algum momento no futuro;
- O cliente precisa estar sempre disponível: por esse ser o ponto principal da abordagem, se o cliente não estiver sempre disponível, pode ocorrer um estado de bloqueio ao se deparar com algum impedimento que depende de seu olhar;
- Muitas mudanças: apesar de a metodologia ser flexível, e o desenvolvimento se adaptar às mudanças que aparecem, muitas vezes esse número de mudanças pode atrapalhar se não tiver uma documentação mínima das decisões tomadas nas reuniões.

Após analisado os pontos positivos e negativos, conclui-se que a abordagem aplicada supriu as necessidades de gerenciamento e desenvolvimento solo de aplicações web desenvolvidas pelo autor em seu ambiente de trabalho.

4.3.2 Resultado da aplicação da metodologia por outro desenvolvedor

Após os resultados obtidos com o uso da metodologia proposta, outro desenvolvedor do ICMC-USP também a utilizou em um projeto que já se encontrava em andamento, porém não se seguia uma metodologia. O desenvolvedor precisava concluir algumas funcionalidades que estavam faltando em um período de 5 semanas, porém não se tinha certeza se iria conseguir. A metodologia deste trabalho foi apresentada e colocada em prática em seguida.

Contextualizando, a aplicação em questão é o sistema de inscrição da pós-graduação do ICMC-USP. Tal aplicação começou a ser desenvolvida em meados de 2015 e o prazo final para entrar em ambiente de produção era para as inscrições do segundo semestre de 2017. No começo de junho de 2017 o desenvolvedor começou a utilizar a metodologia para concluir as seguintes funcionalidades:

- Controle de usuários e perfis: gerenciamento de usuários e suas permissões no sistema, tanto dos usuários administrativos do sistema, quanto os candidatos que se inscrevem nos programas;
- Gerenciamento de localização: esta funcionalidade inclui os dados de país, estado e cidade que são apresentados no formulário de inscrição;

- Gerenciamento de recomendadores: controle dos recomendadores do sistema, e vinculação às inscrições;
- Controle de vagas: gerenciamento das vagas e editais dos programas de pós-graduação;
- Inscrição: gerenciamento de todo o processo de inscrição de um candidato em um programa.

Tais funcionalidades foram divididas em cinco *Sprints*, e cada funcionalidade foi dividida em algumas tarefas. Como o usuário já possuía um modelo do banco de dados da aplicação, a primeira tarefa da primeira *Sprint* foi utilizada para apresentar a metodologia para o desenvolvedor. Mais detalhes das *Sprints*, assim como o modelo do banco de dados da aplicação, e detalhes da interface encontram-se no Anexo D.

Durante as *Sprints* o processo de desenvolvimento seguiu conforme a proposta, e todos os impedimentos eram tratados diretamente com o solicitante o quanto antes. A duração de cada *Sprint* foi de uma semana, e em cada *Sprint* era entregue uma das funcionalidades acima apresentadas, e o projeto foi concluído dentro do prazo.

Após a utilização da metodologia, o desenvolvedor apresentou alguns pontos positivos:

- Gerenciamento do tempo: o divisão das funcionalidades em tarefas permitiu que o desenvolvedor realizasse um melhor gerenciamento do tempo gasto em cada tarefa, permitindo até um melhor planejamento do tempo gasto em cada uma delas;
- Interação com o solicitante: toda semana era entregue parte do sistema para o cliente, no qual já conseguia avaliar e realizar as considerações necessárias. Essa interação foi considerada muito importante pelo desenvolvedor;
- *Feedback* rápido: dentro do ambiente do ICMC-USP a comunicação entre o desenvolvedor e solicitante ocorreu de maneira eficiente, independente do desenvolvedor. Com isso, os problemas foram resolvidos o quanto antes.

Um ponto que o desenvolvedor sugere como melhoria é que a *Sprint* seja realizada em um tempo menor. A proposta da metodologia é que dure que duas a quatro semanas, porém o desenvolvedor realizou em uma semana. Com isso, o mesmo apresenta que o resultado é muito positivo quando se apresenta algo novo para o solicitante a cada semana.

Após analisado os resultados apresentados, além da qualidade das aplicações geradas por ambos desenvolvedores com o uso dessa abordagem, conclui-se que a abordagem proposta

neste trabalho supriu as necessidades de gerenciamento e desenvolvimento solo de aplicações web no âmbito do ICMC-USP. Na sequência outro projeto piloto é apresentado em um cenário diferente do apresentado até o momento.

4.4 Projeto piloto da metodologia ágil em cenário solo em uma aplicação comercial

O autor deste trabalho, além de ter um emprego fixo no ICMC-USP, trabalha como *free-lancer* em desenvolvimento de sites e aplicações para vários clientes. Pelo fato de obter vários resultados positivos no primeiro projeto piloto, decidiu-se adotar a abordagem apresentada neste trabalho em seus projetos particulares a partir de Dezembro de 2017.

Este segundo projeto piloto foi criado para atender a necessidade de uma escola de futebol da cidade de São Carlos denominada Multi Sport [5], que além de realizar aulas de futebol para crianças, possui vários campos para locação.

Para contextualizar, a Multi Sport possui três unidades, e cada uma delas possui um certo número de campos que estão disponíveis para locação. No cenário atual, quando uma pessoa liga em uma das unidades para realizar uma reserva, ela não consegue saber quais campos das outras unidades estão liberados, e precisa ligar em cada unidade para conseguir encontrar um campo no horário que ele precisa.

O principal objetivo do sistema a ser desenvolvido é gerenciar as locações de todos os campos de forma centralizada, além de permitir a realização de reservas *online*. A ideia é criar uma aplicação web para gerenciar a parte de locação de campos, e também criar um site e um aplicativo mobile para interagir com a aplicação principal.

Assim como foi descrito na seção 4.3, que apresentou os papéis envolvidos no primeiro projeto piloto, que era composto de pessoas interessadas e o analista desenvolvedor, neste segundo projeto também são descrito dois papéis: o cliente, que no caso é um dos proprietários da escola, e o analista desenvolvedor.

Na aplicação da metodologia, na etapa de definição, o primeiro passo foi realizar a análise de requisitos em uma reunião com o cliente e assim elaborar o *Product Backlog*, além disso, foi escolhido algumas funcionalidades para a realização da primeira *Sprint*.

Após tais definições, o projeto entrou na parte de construção e ainda se encontra em desenvolvimento, porém alguns resultados já puderam ser coletados. Tais resultados são descritos na

[5] http://www.multsportescoladefutebol.com.br

próxima seção.

Mais detalhes deste projeto piloto, como uma descrição da aplicação, a modelagem do banco e a sugestão de interface, estão no Anexo E.

4.4.1 Resultado da aplicação da metodologia no segundo projeto

No começo da aplicação da metodologia neste segundo projeto piloto algumas diferenças já foram surgindo em relação ao primeiro projeto. A primeira delas é que o cliente, que neste caso é um dos proprietários da escola, nem sempre estava disponível, o que dificultou o agendamento logo da primeira reunião, e viu-se que a troca de mensagens não era tão efetiva quanto uma conversa pessoalmente.

Com a primeira reunião realizada no final de Dezembro de 2017, e após apresentados os desejos do cliente, o *Product Backlog* foi gerado contendo as seguintes funcionalidades:

- Gerenciar unidades: apresenta uma página para controlar todas as unidades que a escola possui;
- Gerenciar campos: faz o gerenciamento de todos os campos que existe em cada unidade da escola;
- Controle de horários: gerenciar os horários que são disponíveis para locação dos campos;
- Gerenciamento de usuários: controle de usuários do sistema, tanto os usuários administrativos, quanto os usuários que podem realizar reservas;
- Cadastro de reservas: formulário no qual os funcionários da escola podem realizar as reservas dos campos;
- Agenda de horários: apresentar um calendário com as reservas de todos os campos de cada unidade;
- Apresentar site institucional: elaborar um site para apresentar dados institucionais, a agenda de horários e também o formulário de reserva para os usuários;
- Criar aplicativo mobile: elaborar um aplicativo mobile para apresentar os mesmos dados do site institucional.

Ainda na primeira reunião, assim como no primeiro projeto piloto, a primeira *Sprint* foi definida e foi elaborado o *Sprint Backlog* inicial. A Tabela 4.9 apresenta a primeira *Sprint*

que contém um conjunto de quatro funcionalidades, divididas em onze tarefas, no qual foram desenvolvidas em um período de duas semanas. Seguindo a recomendação da metodologia, a primeira tarefa dessa *Sprint* foi a elaboração do modelo relacional do banco de dados. Tal modelagem encontra-se no Anexo E.

Tabela 4.9: Tabela com as funcionalidades da *Sprint* inicial do segundo projeto

	***Sprint* inicial**
Tarefas iniciais:	Criar modelo do banco de dados
	Elaborar arquivo de configuração
	Gerar classes do modelo do sistema
Funcionalidade:	Gerenciamento de usuários
Tarefas:	Crud dos usuários
	Definir permissões
	Gerenciar permissões dos usuários
Funcionalidade:	Gerenciar unidades
Tarefa:	Crud das unidades
Funcionalidade:	Gerenciar campos
Tarefas:	Crud dos campos
	Vincular campos nas unidades
Funcionalidade:	Controle de horários
Tarefas:	Criar tipos de horários
	Crud de horários
Total de funcionalidades:	4
Total de tarefas:	11
Tempo de desenvolvimento:	2 semanas

Todos os dias de desenvolvimento, o programador realizava o *Daily Scrum* para verificar se existia algum impedimento no projeto. As dúvidas e imprevistos que surgiram foram tratadas com o cliente, porém ele não estava sempre acessível para discutir tais problemas, o que ocasionou em um pequeno atraso em algumas tarefas.

Ao final da *Sprint*, outra reunião foi realizada com o cliente para a apresentação das funcionalidades desenvolvidas e para a definição da segunda *Sprint*, no qual está representada na Tabela 4.10. Após a reunião, o desenvolvedor realizou uma retrospectiva para avaliar os pontos positivos e negativos do processo, além de realizar uma refatoração no código.

Este segundo projeto piloto ainda está em andamento, mas pode-se citar alguns pontos se comparado ao primeiro projeto apresentado na Seção 4.3:

- O fator que gerou maior impacto no projeto foi a comunicação com o cliente. Enquanto no primeiro projeto o *feedback* do solicitante era muito rápido e objetivo, no segundo projeto o cliente nem sempre estava disponível, o que gerou um certo atraso no desenvolvimento de algumas tarefas. Isso mostra que um dos pontos principais para se desenvolver de maneira solo e ágil é sem dúvida a interação entre cliente e desenvolvedor;

Tabela 4.10: Tabela com as funcionalidades da segunda *Sprint* do segundo projeto

	***Sprint* inicial**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Cadastro de reservas
Tarefas:	Listagem dos horários
	Listagem das unidades e campos
	Crud das reservas
Funcionalidade:	Agenda de horários
Tarefa:	Criar calendário
Tarefa final:	Atualizar documentação
Total de funcionalidades:	2
Total de tarefas:	6
Tempo de desenvolvimento:	1 semanas

- O gerenciamento do tempo foi um ponto positivo em ambos os projetos, pois o desenvolvedor teve um controle mais preciso dos prazos que precisava informar;
- Embora a documentação nos dois projetos tenha sido somente a modelagem relacional do banco de dados, não foi preciso elaborar outros documentos para a compreensão por parte do cliente. O *Product Backlog* apresentou uma visão de todas as funcionalidades e o modelo do banco apresentou a estrutura geral do sistema.

Após apresentado os dois projetos piloto, e analisado alguns pontos positivos e negativos de ambas, conclui-se que a abordagem proposta neste trabalho supriu as necessidades de gerenciamento e desenvolvimento solo de aplicações, tanto no âmbito do ICMC-USP quanto no desenvolvimento particular.

Embora os cenários apresentados sejam diferentes, no qual o primeiro tem-se uma instituição pública do estado de São Paulo sem fins lucrativos, e no outro uma empresa que tem objetivo de realizar locações de campos, a abordagem aqui apresentada supriu as necessidades em ambos os casos.

Outra diferença é com relação ao tipo de cliente. Enquanto no primeiro projeto o cliente simplesmente solicitava um serviço a um outro setor dentro da mesma instituição, no segundo projeto o cliente contratou o serviço de um desenvolvedor *freelancer*. Com isso, tem-se que independente do cliente ou do projeto, a abordagem aqui proposta pode ser utilizada.

Além disso, como resultado do trabalho, tem-se que é possível adaptar metodologias ágeis no cenário solo, como apresentado neste capítulo.

4.5 Considerações finais

Existem algumas metodologias que foram criadas para o desenvolvimento solo de software, como apresentado no capítulo anterior. Porém o objetivo deste capítulo foi apresentar uma abordagem mais simples, com os conceitos diretamente ligados com a metodologia *Scrum*, para que fosse familiar para quem já conhece a metodologia e gostaria de aplicar de maneira solo.

Nesse contexto, foi apresentado uma visão geral da abordagem proposta, com seus artefatos, papéis e valores. Em seguida foi apresentado como pode-se aplicá-la em um ambiente real de desenvolvimento. Essa proposta também pode ser utilizada por alguém que queira começar uma empresa de desenvolvimento de software e ainda não tem uma equipe de desenvolvimento, pois se aproxima do estudo de caso apresentado na Seção 4.3, e pode-se seguir os passos descritos.

Na Seção 4.3.1 foi apresentado o resultado da aplicação da abordagem em um estudo de caso de um sistema para gerenciar as solicitações de treinamento e desenvolvimento dos funcionários do ICMC-USP. Tal sistema foi desenvolvido em duas versões, sendo que na primeira não se utilizou uma metodologia consolidada, e na segunda foi utilizada a metodologia aqui apresentada. Por fim, alguns pontos positivos e negativos foi discutido.

Logo depois foi apresentado o resultado da aplicação da abordagem por outro desenvolvedor em um sistema de inscrição para a pós-graduação do ICMC-USP. Tal sistema começou a ser desenvolvido sem a utilização da abordagem, e apenas no último mês de desenvolvimento ela foi utilizada para concluir algumas funcionalidades. O *feedback* da utilização foi positivo e o desenvolvedor sugeriu como melhoria, a redução do tempo das *Sprints* para uma semana.

Em seguida, outro projeto piloto foi apresentado no qual tem-se um sistema para gerenciamento de locação de campos de futebol. Com isso, viu-se que a abordagem proposta pode ser utilizada em projetos e clientes diferentes, não se limitando a um único escopo de atuação.

Na sequência, será apresentada a conclusão do trabalho, com base no que foi apresentado até aqui. E por fim, os trabalhos futuros são descritos.

Capítulo 5

CONCLUSÃO

5.1 Considerações iniciais

Neste trabalho foi apresentado alguns dados que representam a realidade atual do mercado de TI no Brasil, e como as metodologias de desenvolvimento consolidadas na área de engenharia de software podem auxiliar as pessoas a criarem um software com maior qualidade e agilidade. Descreveu-se também um breve histórico de algumas metodologias de desenvolvimento de software, começando pelo modelo tradicional, conhecido também como modelo cascata, e chegando até as metodologias ágeis. Dentre elas, foram descritos detalhes sobre as metodologias *Scrum*, *Extreme Programming* (XP) e *Agile Unified Process* (AUP).

Um levantamento de quais metodologias foram criadas ou adaptadas para o uso individual também foi apresentado, mostrando quatro metodologias diferentes, sendo elas: Scrum solo, Agile solo, *Extreme Programming* para equipe de uma pessoa e também uma metodologia denominada Cowboy. Todas essas metodologias seguem os princípios ágeis de desenvolvimento, mostrando que é possível utilizar metodologias ágeis sem necessariamente ter uma equipe de desenvolvimento.

O objetivo principal deste trabalho foi apresentar uma nova proposta de abordagem de desenvolvimento solo utilizando princípios ágeis, e executar dois projetos piloto para demonstrar a aplicação da mesma em ambientes reais de desenvolvimento.

5.2 Resultados e contribuições

Com os resultados obtidos neste trabalho, é possível responder às questões de pesquisa descritas no Capítulo 1:

- É possível utilizar metodologias ágeis para desenvolver sistemas com somente uma pessoa no time de desenvolvimento? Para responder essa pergunta primeiramente foi feito um levantamento histórico sobre algumas metodologias de desenvolvimento, entrando também nas metodologias ágeis, no qual viu-se com mais detalhes sobre *Scrum*, XP e *Agile Unified Process*. Então foi feito um levantamento de algumas metodologias com foco em desenvolvimento solo, que tinham características de algumas das metodologias ágeis apresentadas anteriormente. Por fim, uma proposta de uma abordagem de desenvolvimento solo com princípios ágeis foi apresentada, e dois projetos piloto foram conduzidos para aplicar tal metodologia. Após análise de tais informações, tem-se que é possível adaptar metodologias ágeis para desenvolvimento solo;

- Quais opções existem para quem quer começar no mercado de desenvolvimento sem ter uma equipe? Apesar deste trabalho apresentar quatro metodologias diferentes, e ainda propor uma outra abordagem, existem outras que não foram aqui apresentadas. O objetivo foi apresentar algumas opções que podem ser usadas de exemplo e guia para quem deseja se aprofundar no tema, ou mesmo começar uma empresa de desenvolvimento de software com pouco recurso;

- Existe metodologia certa para esse tipo de situação? Como apresentado neste trabalho, existem diferentes metodologias para desenvolver de maneira solo, porém não existe uma metodologia correta, pois a escolha de uma abordagem depende não só do desenvolvedor, mas também do projeto em que será aplicada, do cliente e de fatores como tempo e recursos financeiros.

Tais resultados contribuíram para uma melhor definição no processo de desenvolvimento de software no seção técnica de informática do ICMC-USP. Além disso, mostrou que é possível adaptar uma metodologia de acordo com a necessidade de cada um, pois as pessoas, os projetos e os cliente são diferentes. Por isso, seguir uma abordagem que o auxilie todo o processo de desenvolvimento de software é de extrema importância.

5.3 Limitações do trabalho

Devido ao grande número de metodologias existentes, e também às várias combinações que podem ser realizadas, foram apresentadas apenas quatro adaptações de metodologias para o uso individual. O foco ficou na proposta de uma abordagem ágil para desenvolvimento solo, e também na aplicação da mesma em projetos reais de desenvolvimento. Tal decisão, limitou um estudo mais detalhado e comparativo das metodologias existentes para um único desenvolvedor.

Outra limitação foi que apenas o autor deste trabalho realizou a aplicação da abordagem proposta. Apesar de outro funcionário do ICMC-USP ter começado a utilizar a abordagem proposta em um projeto já em andamento, ainda não tem-se resultados consolidados da aplicação por uma outra pessoa do começo ao fim de uma aplicação.

5.4 Trabalhos futuros

Com base no que foi apresentado, e sabendo-se que os projetos piloto continuarão sendo aplicados pelo autor, tem-se os seguintes pontos a serem explorados:

- Ampliar a aplicação da proposta para os demais desenvolvedores do ICMC-USP, com o objetivo de coletar mais detalhes com diferentes pessoas utilizando a mesma metodologia;
- Melhorar os pontos negativos da proposta apresentados no Capítulo 4, na intenção de evoluir o processo, e propor melhorias para o mesmo;
- Realizar um estudo comparativo das metodologias solo, fazendo uma análise mais aprofundada das principais características de cada metodologia, analisar os pontos positivos e negativos de cada uma e até mesmo utilizar métricas para realizar tal comparação.

REFERÊNCIAS

AGARWAL, R.; UMPHRESS, D. Extreme programming for a single person team. *Proceedings of the 46th Annual Southeast Regional Conference on XX - ACM-SE 46*, n. March, p. 82, 2008.

AMBLER, S. W. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York, NY, USA: John Wiley & Sons, Inc., 2002. 400 p. ISBN 0471202827.

AMBLER, S. W. *The Agile Unified Process (AUP) Home Page*. 2018. Disponível em: <http://www.ambysoft.com/unifiedprocess/agileUP.html>.

ANNA, N. Agile Solo: Defining and Evaluating an Agile Software Development Process for a Single Software Developer. *Agile Solo : Defining and Evaluating an Agile Software Development Process for a Single Software Developer Master*, 2011.

AVESON, D.; FITZGERALD, G. Methodologies for developing information systems: A historical perspective. In: ____. *The Past and Future of Information Systems: 1976–2006 and Beyond: IFIP 19th World Computer Congress, TC-8, Information System Stream, August 21–23, 2006, Santiago, Chile*. Boston, MA: Springer US, 2006. p. 27–38.

BECK, K. *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <http://agilemanifesto.org/>.

BOEHM, B. W. et al. A Spiral Model of Software Development and Enhancement. *Computer*, v. 21, n. May, p. 61–72, 1987. ISSN 0018-9162.

CIRILLO, F. *The Pomodoro Technique*. Lulu Enterprises Incorporated, 2009. ISBN 9781445219943. Disponível em: <https://books.google.com.br/books?id=ThkbQwAACAAJ>.

COSTA, A. D. *8 opções para gerenciar finanças no PC*. 4 2012. Acesso em: 30 out. 2016. Disponível em: <http://exame.abril.com.br/blogs/aplicativos/windows/8-opcoes-para-gerenciar-financas-no-pc/>.

DEFENCE, U. D. of. Military Standard 105D. n. May, 1989.

FLORêNCIO, P. *Desenvolvimento de Software: Mercado de Trabalho*. ago. 2016. Disponível em: <https://www.targetso.com/2016/08/15/mercado-de-trabalho-desenvolvimento-de-software/>.

FRANCO, E. F. *Um modelo de gerenciamento de projetos baseado nas metodologias ágeis de desenvolvimento de software e nos princípios da produção enxuta*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2007.

HASTIE, S.; WOJEWODA, S. *Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch*. out. 2015. Disponível em: <https://www.infoq.com/articles/standish-chaos-2015>.

HEWLETT PACKARD ENTERPRISE. *Software de Big Data*. 2016. Acesso em: 30 out. 2016. Disponível em: <http://www8.hp.com/br/pt/software-solutions/big-data-analytics-software.html>.

HOLLAR, A. B. Cowboy: An Agile Programming Methodology for a Solo Programmer. n. December, p. 1–73, 2006.

HUMPHREY, W. S. Introducing the personal software process. *Annals of Software Engineering*, v. 1, n. 1, p. 311–325, 1995. ISSN 15737489.

HUMPHREY, W. S. *The Personal Software Process (PSP)*. [S.l.], 2000. v. 22, n. 1, 99–131 p. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0733861903000963>.

LEITE, P. R. *Metodologias de Desenvolvimento de Software na atualidade*. abr. 2016. Disponível em: <https://paulorobertoleite.com.br/2016/04/13/metodologias-de-desenvolvimento-de-software-na-atualidade/>.

NETO, J. S. Mercado Brasileiro de Software: Panoramas e Tendências. *Abes*, 2015. Disponível em: <http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados 2011/mercado-brasileiro-de-software-2015.pdf>.

PAGOTTO, T. et al. Scrum solo: Software process for individual development. In: *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.]: IEEE, 2016. p. 1–6. ISBN 978-9-8998-4346-2.

PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*. 6th. ed. [S.l.]: McGraw-Hill Higher Education, 2005.

RIBEIRO, C. *TOP 10 softwares para engenharia*. 8 2016. Acesso em: 30 out. 2016. Disponível em: <http://blogdaengenharia.com/top-10-softwares-para-engenharia/>.

ROBERTS, R.; SIKES, J. *A rising role for IT: McKinsey Global Survey results*. dez. 2011. Disponível em: <http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/a-rising-role-for-it-mckinsey-global-survey-results>.

ROYCE, W. Managing the development of large software systems: Concepts and techniques. 1970.

SCHWABER, K. Scrum development process. In: ____. *Business Object Design and Implementation: OOPSLA '95 Workshop Proceedings 16 October 1995, Austin, Texas*. London: Springer London, 1997. p. 117–134. ISBN 978-1-4471-0947-1. Disponível em: <http://dx.doi.org/10.1007/978-1-4471-0947-1_11>.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide*. 7 2016. Acesso em: 20 out. 2016. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf>.

SCRUMALLIANCE. *What is Scrum? An Agile Framework for Completing Complex Projects - Scrum Alliance*. 2018. Disponível em: <https://www.scrumalliance.org/why-scrum>.

SOMMERVILLE, I. *Software Engineering*. 9th. ed. USA: Addison-Wesley Publishing Company, 2010.

TELES, V. M. *Extreme Programming*. 2nd. ed. [S.l.]: Novatec, 2014.

Anexo A

APLICAÇÕES DESENVOLVIDAS UTILIZANDO A ABORDAGEM PROPOSTA NO TRABALHO

1) Relatório de monitoria

Descrição da Aplicação:

A aplicação foi criada para auxiliar o controle das monitorias dos departamentos. Todo semestre a secretaria de cada departamento cadastra os monitores e os respectivos orientadores. Todo mês o monitor preenche um relatório com as atividades executadas por ele no mês, e o orientador preenche um formulário avaliando a atuação do monitor. Ao final de cada semestre um relatório final é preenchido por ambos, para então finalizar o processo da monitoria. Um lembrete de preenchimento é enviado todo mês para os monitores.

Características Técnicas:

- Linguagem: PHP
- Possui Manual de Operação: Não
- Possui Documentação: Apenas modelo do banco de dados
- Recursos de Necessários: Segundo Manual de Desenvolvimento STI/ICMC
- Intuitivo: Sim

Screenshots:

Figura A.1: Página de gerenciamento das disciplinas

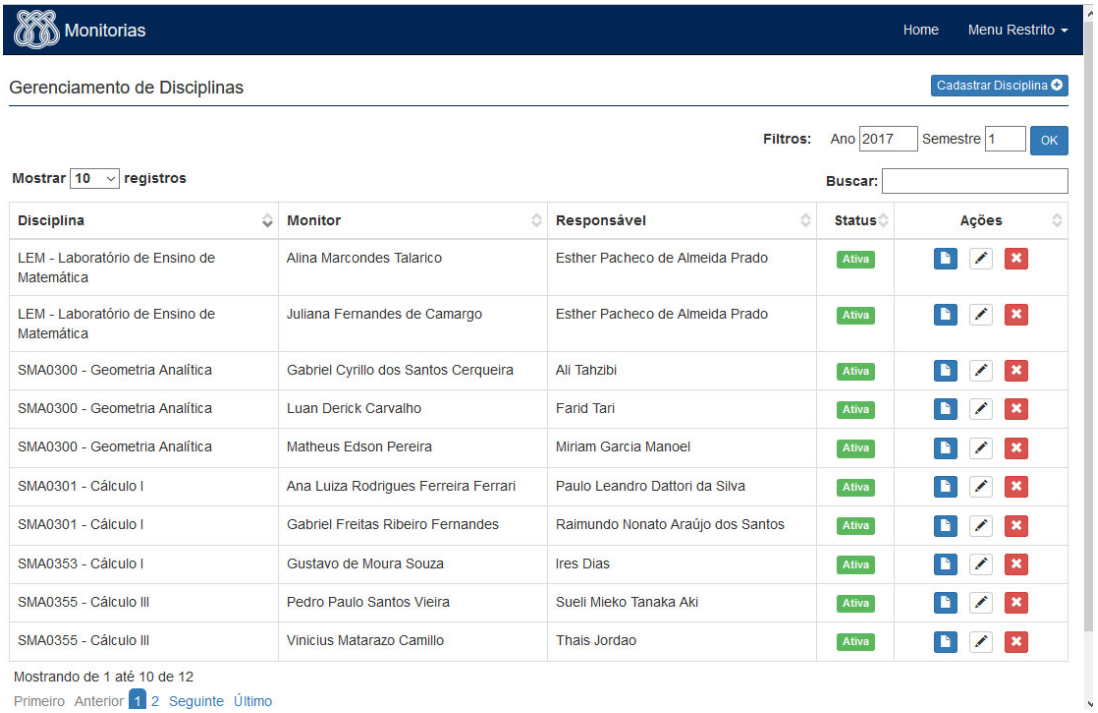


Figura A.2: Página de cadastro de uma disciplina

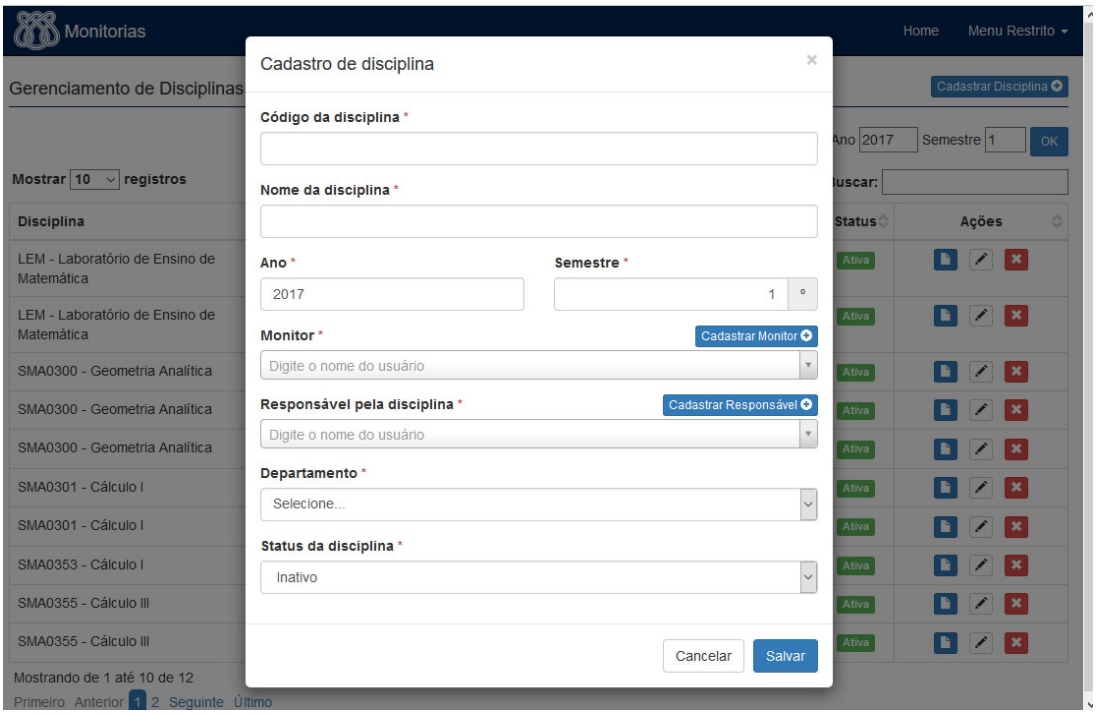


Figura A.3: Listagem das disciplinas do semestre

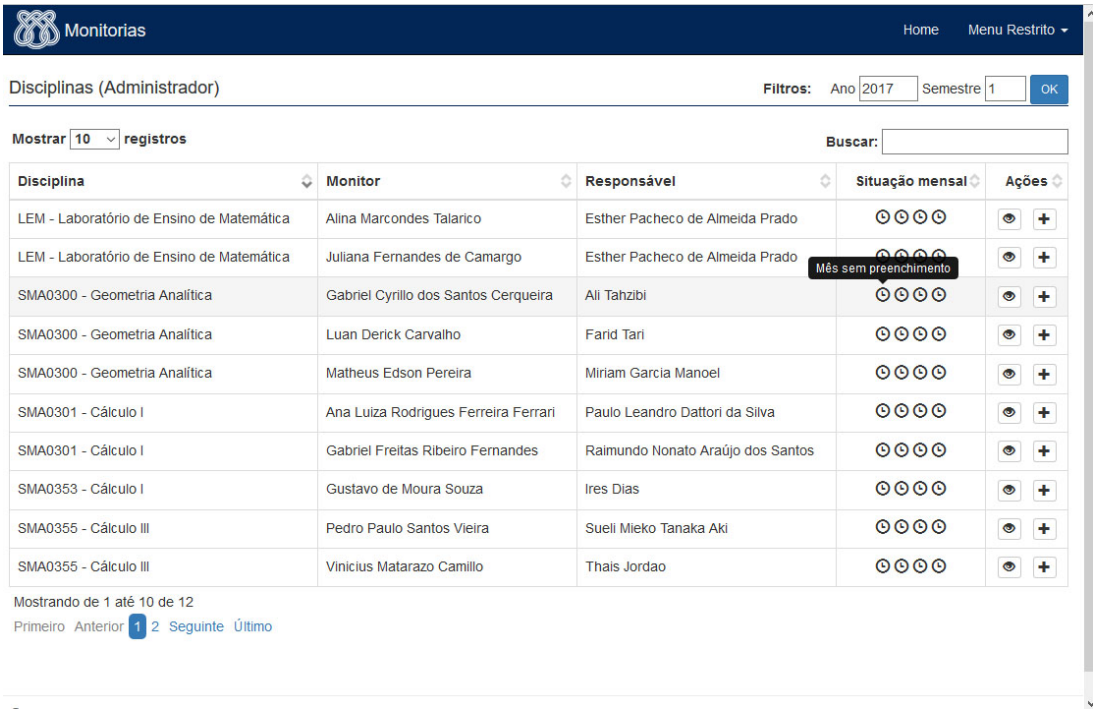


Figura A.4: Página de preenchimento do relatório mensal

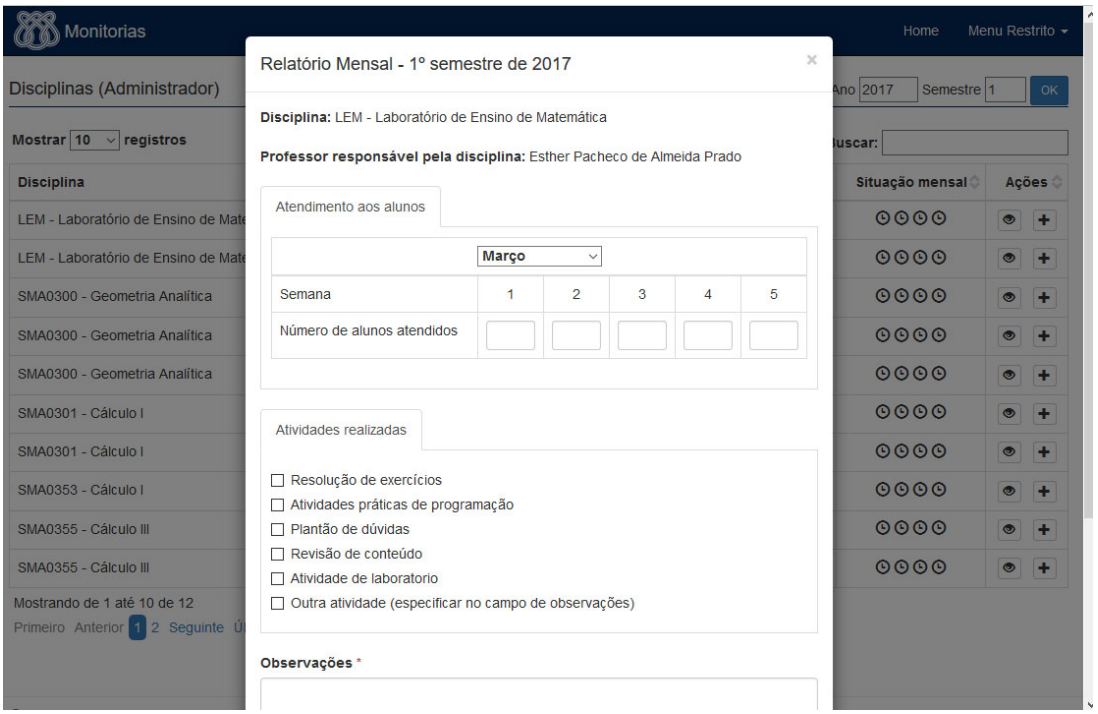


Figura A.5: Relatório final

Monitorias Home Menu Restrito

ICMC USP SÃO CARLOS SMA

UNIVERSIDADE DE SÃO PAULO
INTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Matemática

Relatório Semestral - 1º semestre de 2017

Monitor: Alina Marcondes Talarico **Nº USP:** 5744750

Disciplina: LEM - Laboratório de Ensino de Matemática

Professor responsável pela disciplina: Esther Pacheco de Almeida Prado

Tabela de atendimentos aos alunos:

Mês	Março					Abril					Maio					Junho				
Semana	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Número de alunos atendidos	0					0					0					0				

Atividades Realizadas:

	Março	Abril	Maio	Junho
Resolução de exercícios				
Atividades práticas de programação				
Plantão de dúvidas				
Revisão de conteúdo				
Atividade de laboratorio				
Outra atividade (especificar no campo de observações)				

2) Portal de serviços

Descrição da Aplicação:

Portal centralizado de aplicações/sistemas em produção no ICMC. A ideia é elaborar um portal como o http://uspdigital.usp.br para relacionar todos os sistemas do ICMC, e centralizar o login desses sistemas em um único lugar.

Características Técnicas:

- Linguagem: PHP
- Possui Manual de Operação: Não
- Possui Documentação: Não
- Recursos de Necessários: Segundo Manual de Desenvolvimento STI/ICMC
- Intuitivo: Sim

Screenshot:

Figura A.6: Tela com a listagem das aplicações em produção no ICMC

3) Sistema de visitantes

Descrição da Aplicação:

A aplicação foi criada para gerenciar os visitantes que chegam no Instituto. Sempre que um professor for receber um visitante, o mesmo precisa realizar o cadastro deste visitante no sistema, preenchendo os seguintes dados: nome, e-mail e instituição do visitante, data provável da visita e idioma principal do visitante. O sistema envia um e-mail para o visitante no idioma selecionado no cadastro, para que o mesmo preencha um cadastro completo no sistema. Após o cadastro completo, o professor anfitrião pode confirmar os serviços do ICMC que o visitante irá utilizar durante sua visita.

Características Técnicas:

- Linguagem: PHP
- Possui Manual de Operação: Não
- Possui Documentação: Apenas modelo do banco de dados
- Recursos de Necessários: Segundo Manual de Desenvolvimento STI/ICMC
- Intuitivo: Sim

Screenshots:

Figura A.7: Tela com a listagem dos convites de visitantes no ICMC

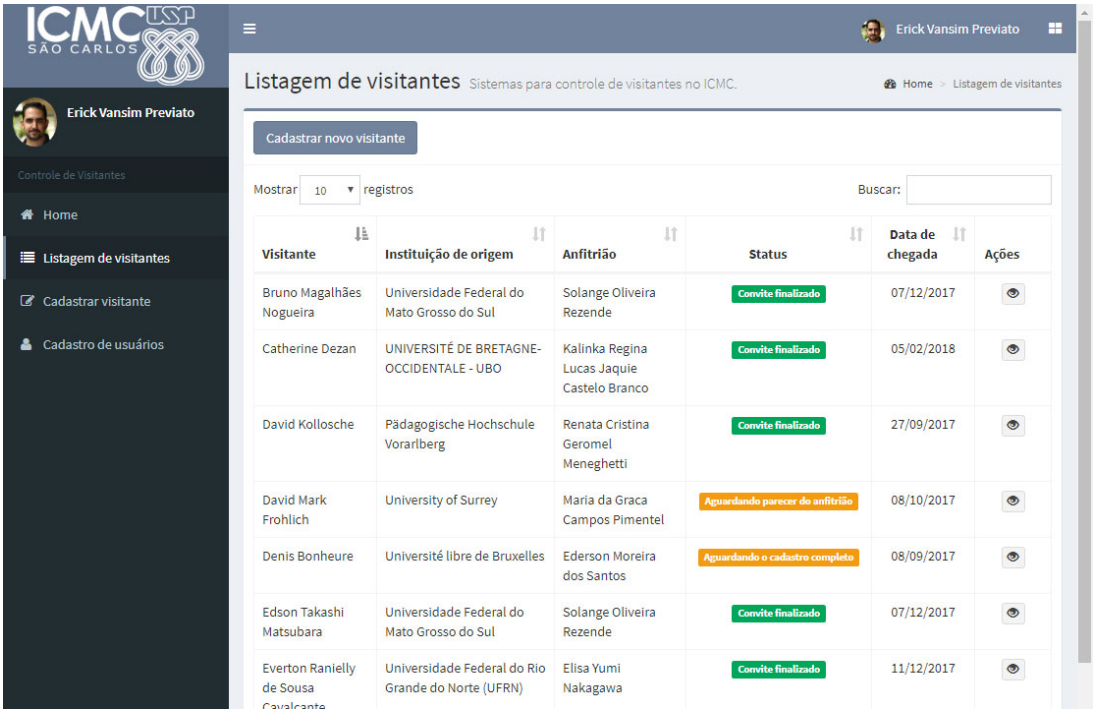
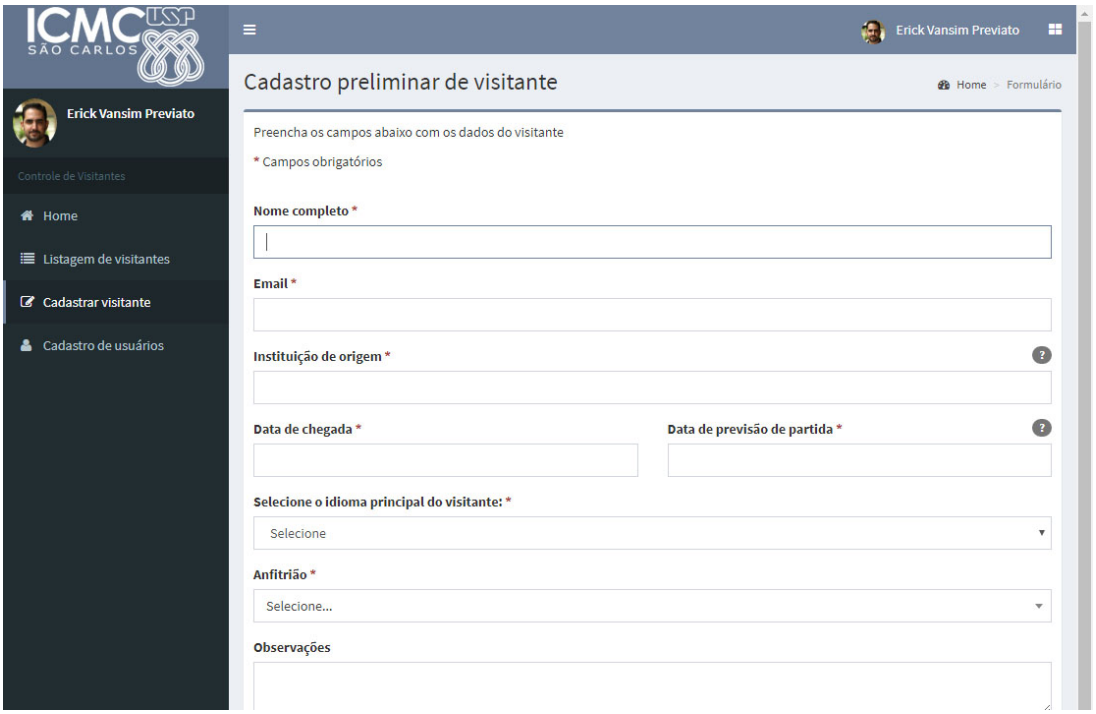


Figura A.8: Tela de cadastro de um novo visitante



Anexo B

PRIMEIRA VERSÃO DO SISTEMA DE T&D

Aplicação criada para gerenciar as solicitações de Treinamento & Desenvolvimento do quadro de funcionários do ICMC-USP. Possui um controle de fluxo para aprovação das solicitações conforme descrito abaixo:

- Solicitante preenche os dados do treinamento e envia para o chefe de área;
- Chefe de área aprova/reprova e envia para a comissão de T&D avaliar;
- Após avaliação do T&D, o presidente da Comissão de Qualidade dá o parecer final;
- Um e-mail é enviado para o solicitante, e todos os envolvidos com o processo, para aprovar as despesas;

Características Técnicas:

- Linguagem: PHP
- Possui Manual de Operação: Sim
- Possui Documentação: Apenas modelo do banco de dados
- Recursos de Necessários: Segundo Manual de Desenvolvimento STI/ICMC
- Intuitivo: Sim

Screenshots:

Figura B.1: Página com o formulário de solicitação de treinamento

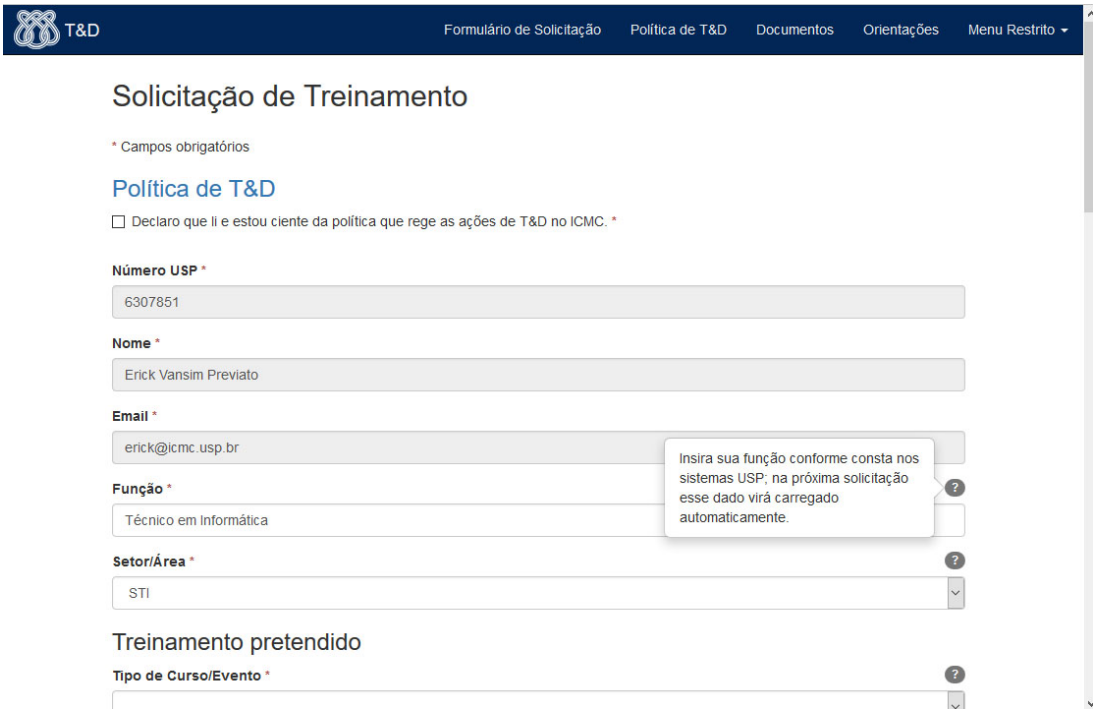


Figura B.2: Página com o política de T&D

Figura B.3: Listagem de solicitações do usuário

Figura B.4: Listagem das solicitações vista pela comissão de T&D com opções de visualizar detalhes e de avaliar as solicitações

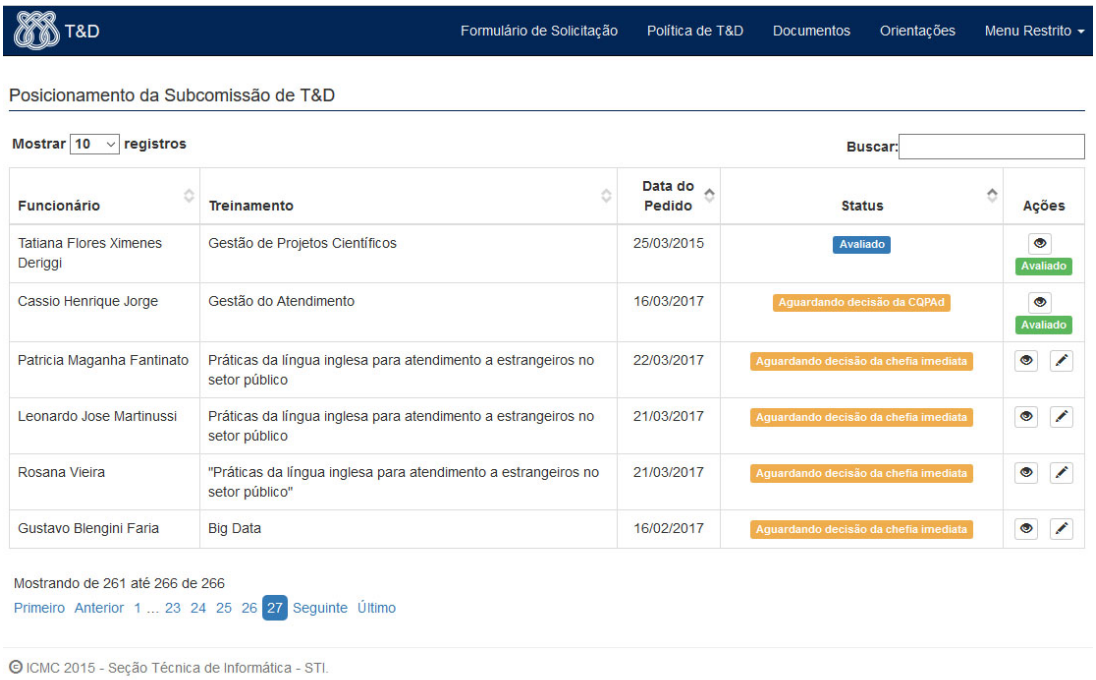


Figura B.5: Popup com os detalhes da solicitação

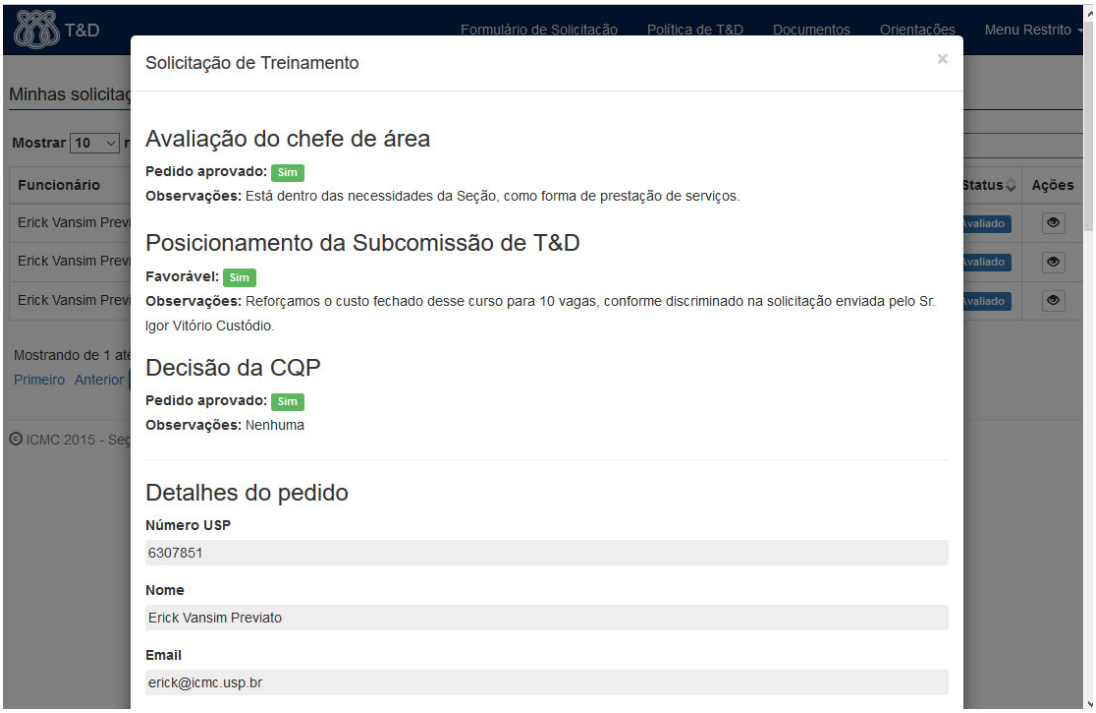
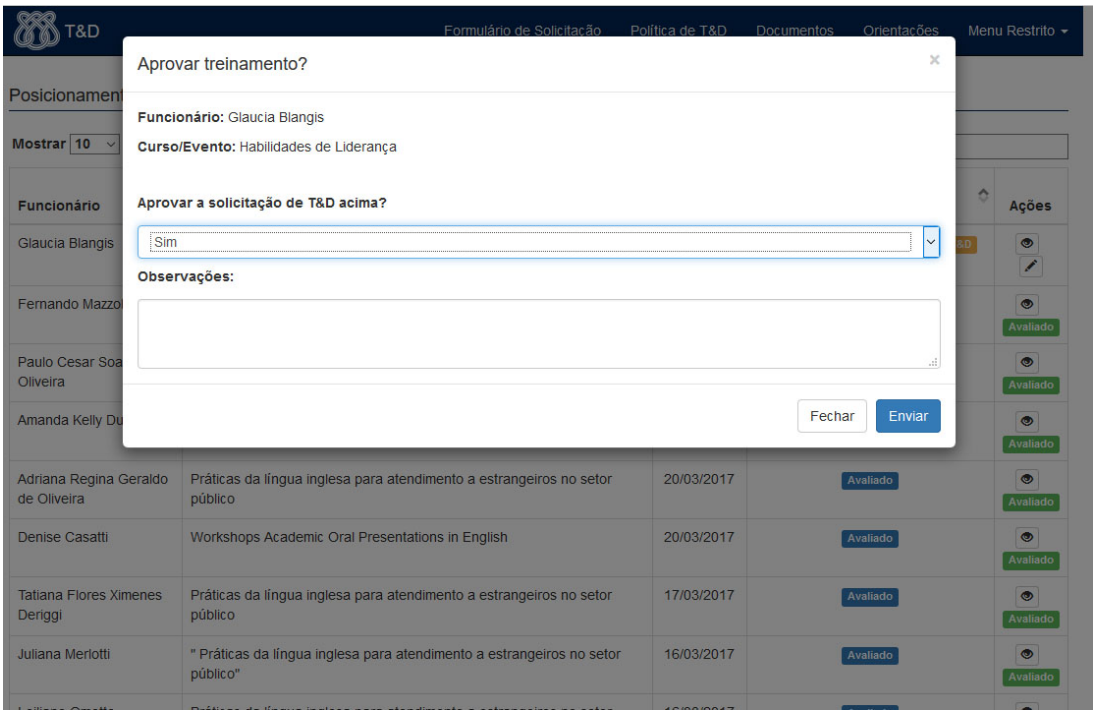


Figura B.6: Tela de aprovação de uma solicitação



Anexo C

SEGUNDA VERSÃO DO SISTEMA DE T&D

Aplicação criada para gerenciar as solicitações de Treinamento & Desenvolvimento do quadro de funcionários do ICMC. Possui um controle de fluxo para aprovação das solicitações conforme descrito abaixo:

- Solicitante preenche os dados do treinamento e envia para o chefe imediato;
- Chefe imediato aprova/reprova e envia para a chefia de área ou para comissão de T&D avaliar;
- Chefe imediato tem a opção de devolder a solicitação caso encontre algum erro no preenchimento do formulário;
- Caso tenha chefia de área, segue os mesmos passos da chefia imediata, ou seja, o mesmo pode avaliar, ou devolver a solicitação;
- Avaliação do T&D, podendo enviar a solicitação para aprovação da Comissão de Qualidade, ou devolver para a chefia;
- Após avaliação do T&D, o presidente da Comissão de Qualidade dá o parecer final;
- Um e-mail é enviado para o solicitante, e todos os envolvidos com o processo, para aprovar as despesas;

Além de mais um passo no fluxo, que é a devolução do pedido, outras funcionalidades foram desenvolvidas nesta nova versão do sistema:

- Cadastro de oportunidades que ficam disponíveis para todos os funcionários visualizarem;

- Gerenciamento do FAQ, que é um conjunto de perguntas e respostas para tirar dúvidas dos funcionários;
- Gerenciamento de documentos importantes que ficam disponíveis para todos os funcionários;
- Formulário de prestação de contas após a realização de um treinamento;
- Gerenciamento do mapa de competências dos setores do Instituto;
- Gerenciamento de usuários do sistema;
- Gerenciamento dos tipos de treinamentos permitidos pelo sistema;
- Controle financeiro dos recursos de T&D

Características Técnicas:

- Linguagem: PHP
- Possui Manual de Operação: Sim
- Possui Documentação: Apenas modelo do banco de dados
- Recursos de Necessários: Segundo Manual de Desenvolvimento STI/ICMC
- Intuitivo: Sim

Screenshots:

Figura C.1: Página inicial do novo sistema

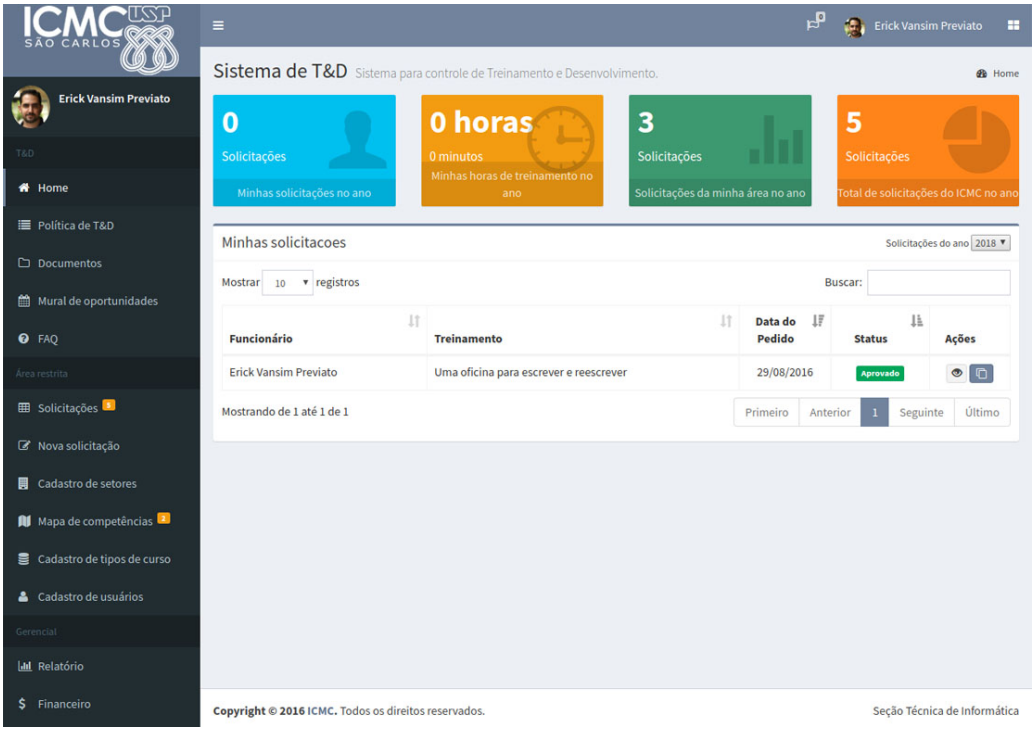


Figura C.2: Listagem das solicitações vista pela comissão de T&D com opções de visualizar detalhes, de avaliar as solicitações e de devolver as solicitações

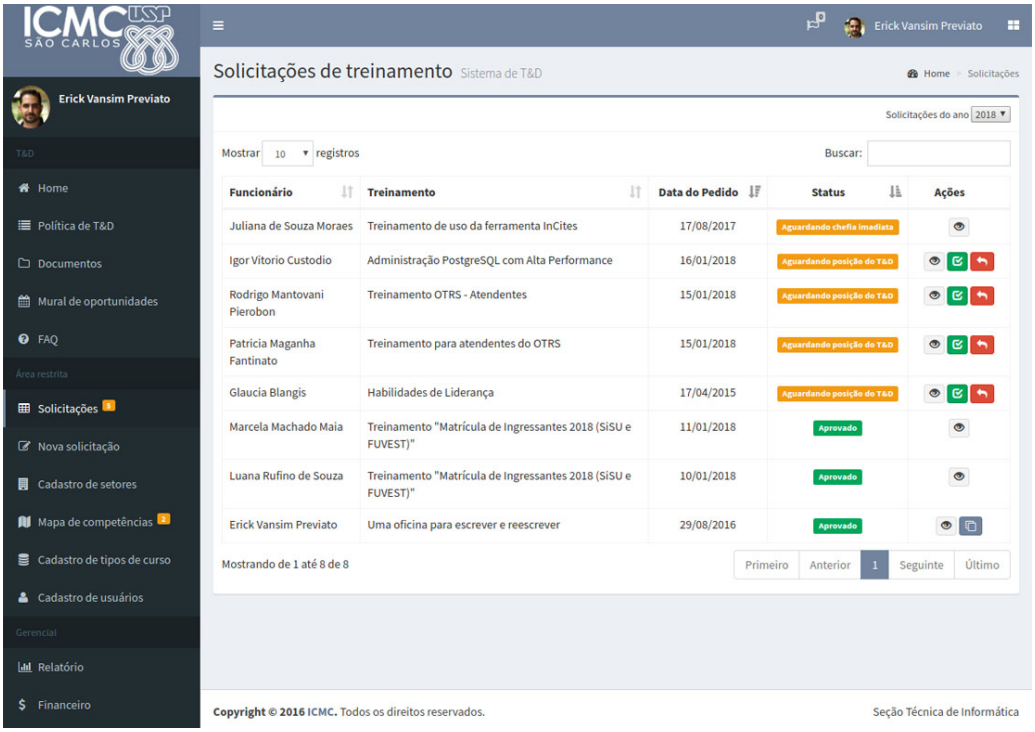


Figura C.3: Página de gerenciamento do FAQ

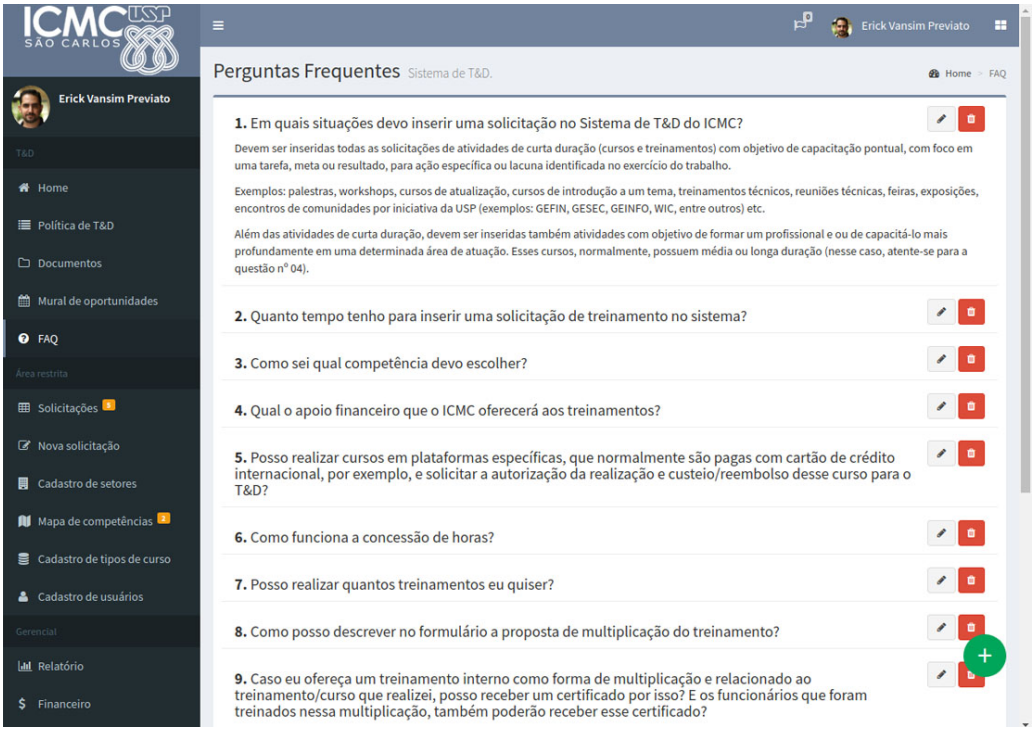


Figura C.4: Página com o formulário de prestação de contas de um treinamento

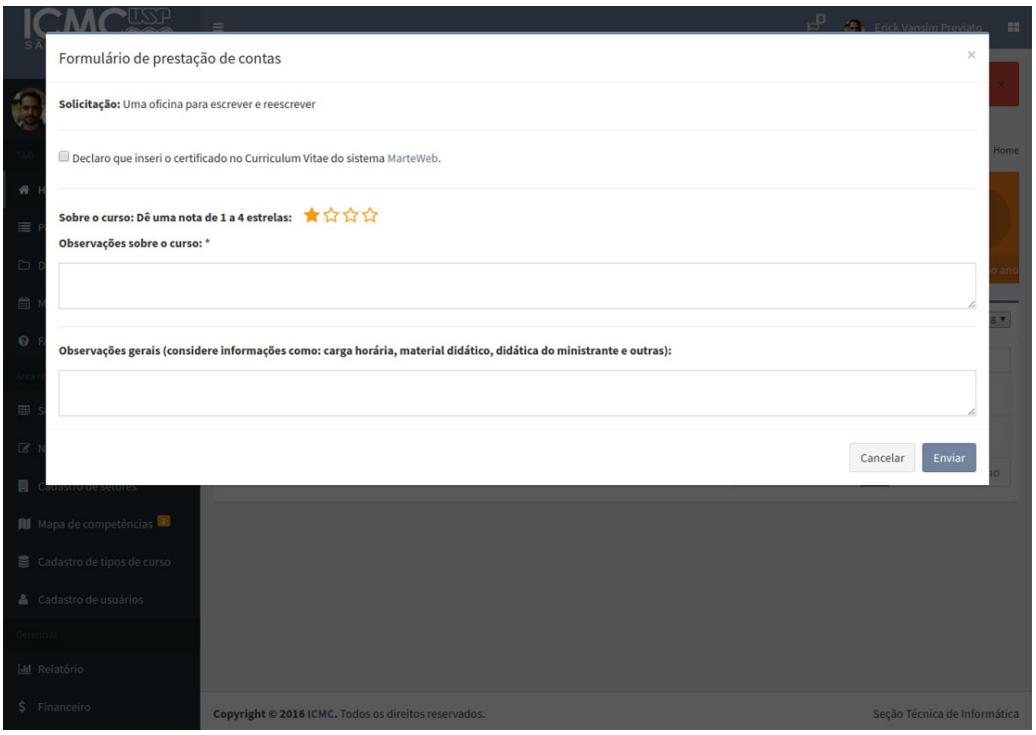


Figura C.5: Página com o controle financeiro do sistema

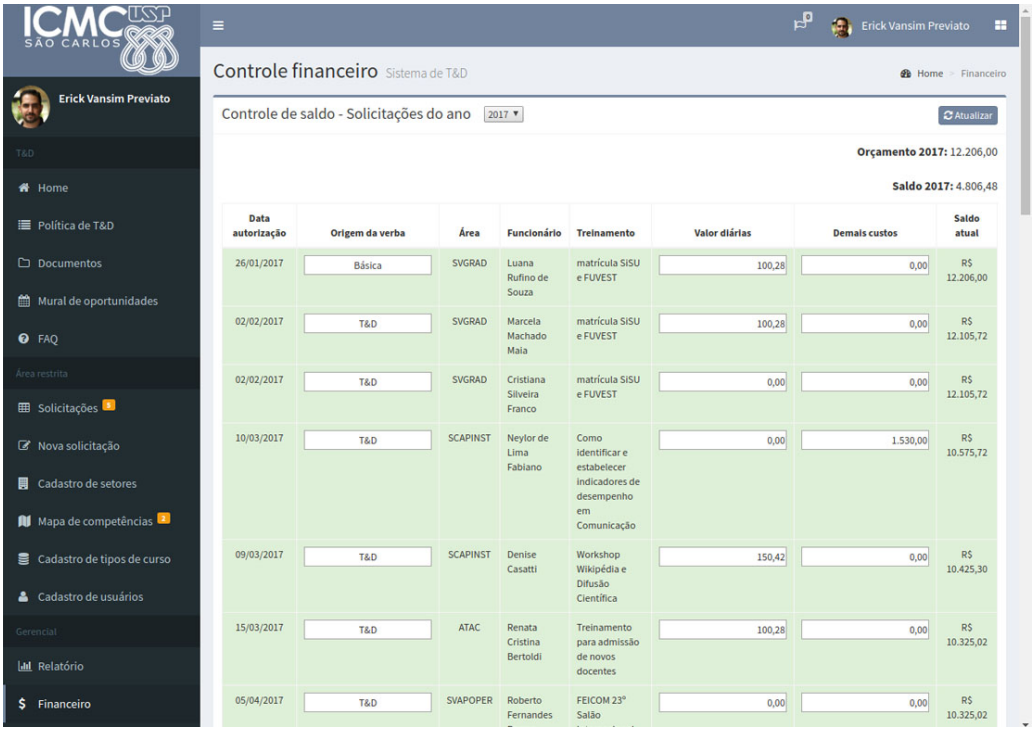


Figura C.6: Página com o formulário de solicitação de treinamento

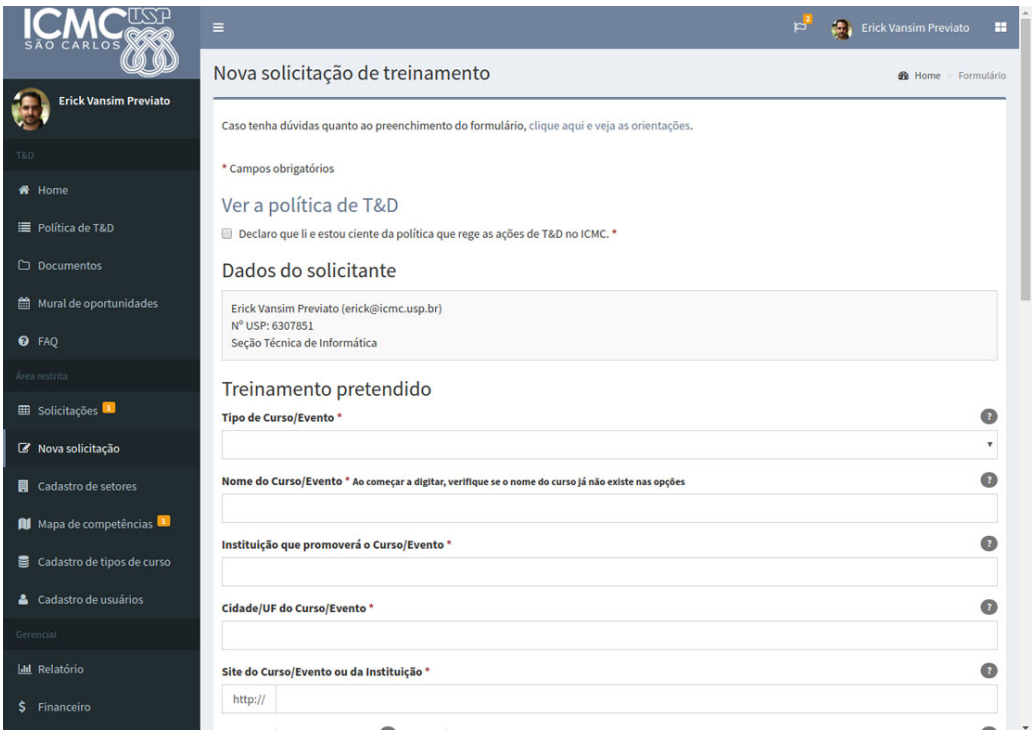
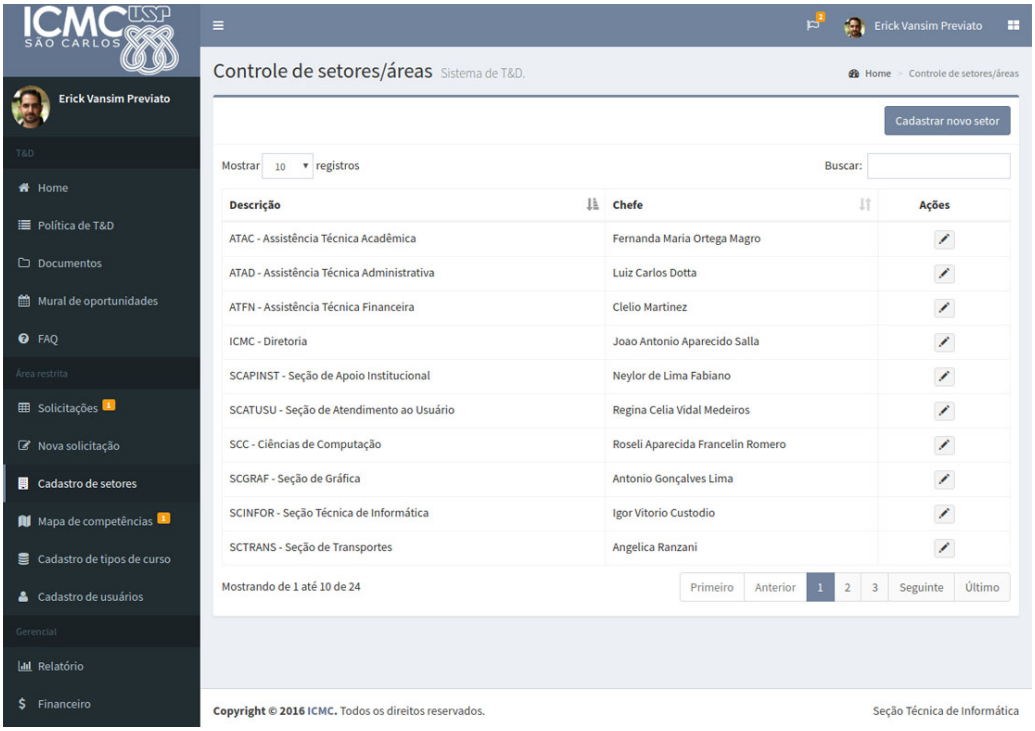


Figura C.7: Página de gerenciamento de setores e áreas do Instituto



Anexo D

SISTEMA DE INSCRIÇÃO DA PÓS-GRADUAÇÃO DO ICMC-USP

Descritivo do sistema

Características Técnicas:

- Linguagem: PHP
- Possui Documentação: Apenas modelo do banco de dados (Figura D.1)

Sprints

Tabela D.1: Tabela com as funcionalidades da primeira *Sprint* do projeto

	***Sprint* 1**
Tarefa inicial:	Apresentação da metodologia
Funcionalidade:	Controle de usuários e perfis
Tarefas:	Cadastro de permissões
	Cadastro de usuários (administrativo)
	Cadastro de candidatos
	Lógica de login
Tarefa final:	Atualizar documentação
Total de funcionalidades:	1
Total de tarefas:	6
Tempo de desenvolvimento:	1 semana

Figura D.1: Modelo do banco de dados do sistema de inscrição da pós-graduação

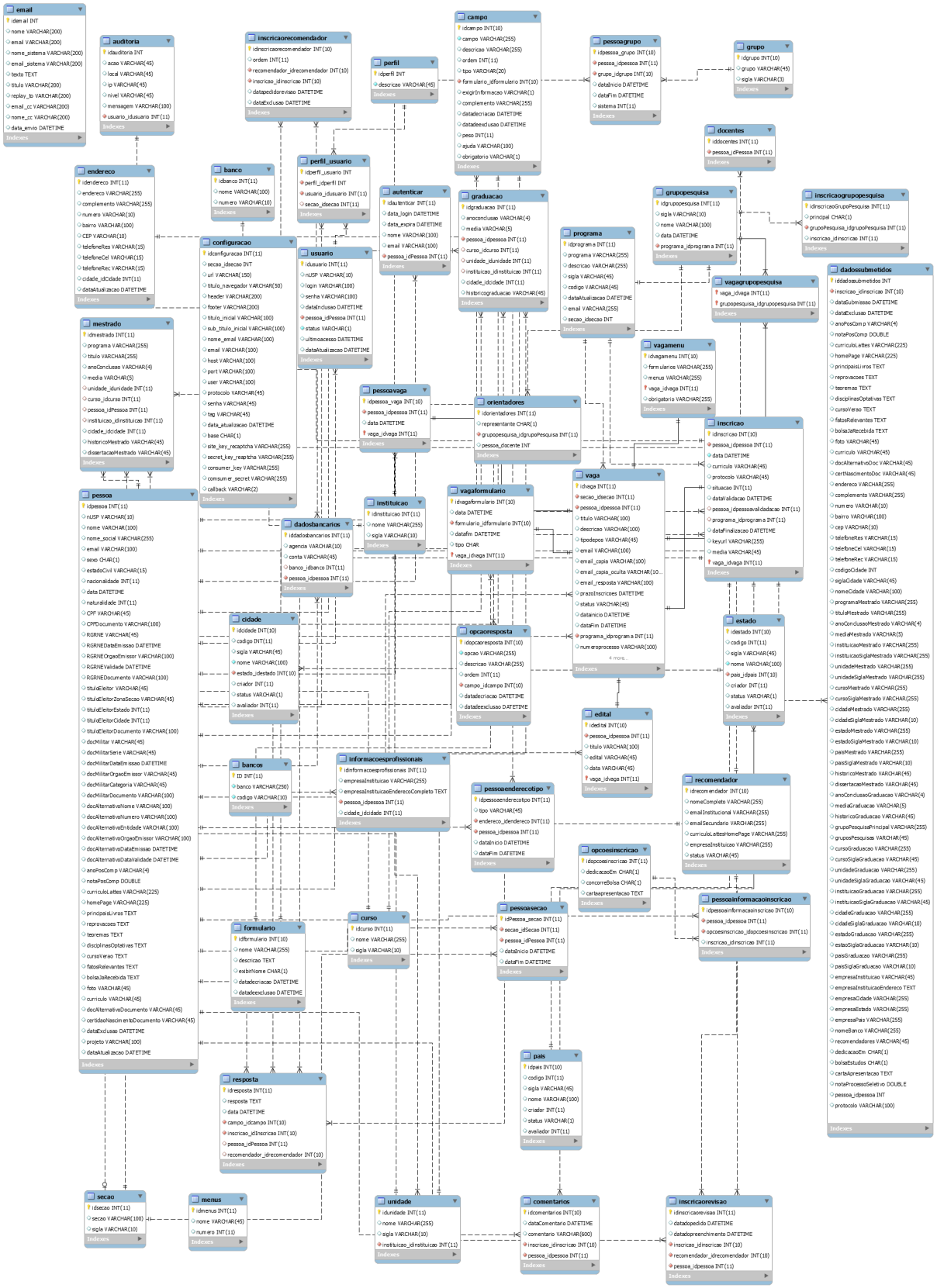


Tabela D.2: Tabela com as funcionalidades da segunda *Sprint* do projeto

	***Sprint* 2**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Gerenciamento de localização
Tarefas:	Recriação dos formulários de cadastro
	Reaproveitamento de tais formulários na inscrição
	Validação dos dados inseridos pelos candidatos
	Resolução de conflitos com dados duplicados
Tarefa final:	Atualizar documentação
Total de funcionalidades:	1
Total de tarefas:	6
Tempo de desenvolvimento:	1 semana

Tabela D.3: Tabela com as funcionalidades da terceira *Sprint* do projeto

	***Sprint* 3**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Gerenciamento de recomendadores
Tarefas:	Recriação do formulário de cadastro
	Reaproveitar formulário na inscrição
	Validação dos dados inseridos pelos candidatos
	Solicitação de carta convite
Tarefa final:	Atualizar documentação
Total de funcionalidades:	1
Total de tarefas:	6
Tempo de desenvolvimento:	1 semana

Tabela D.4: Tabela com as funcionalidades da quarta *Sprint* do projeto

	***Sprint* 4**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Controle de vagas
Tarefas:	Cadastro de vagas e editais
	Listagem de vagas abertas para inscrição
Tarefa final:	Atualizar documentação
Total de funcionalidades:	1
Total de tarefas:	4
Tempo de desenvolvimento:	1 semana

Tabela D.5: Tabela com as funcionalidades da quinta *Sprint* do projeto

	***Sprint* 5**
Tarefa inicial:	Ajustar pendências da *Sprint* anterior
Funcionalidade:	Aprimoramento da inscrição
Tarefas:	Gerenciamento dos status da inscrição
	Controle de comentários de uma inscrição
Tarefa final:	Atualizar documentação
Total de funcionalidades:	1
Total de tarefas:	4
Tempo de desenvolvimento:	1 semana

Screenshots:

Figura D.2: Página de login e de listagem de vagas abertas

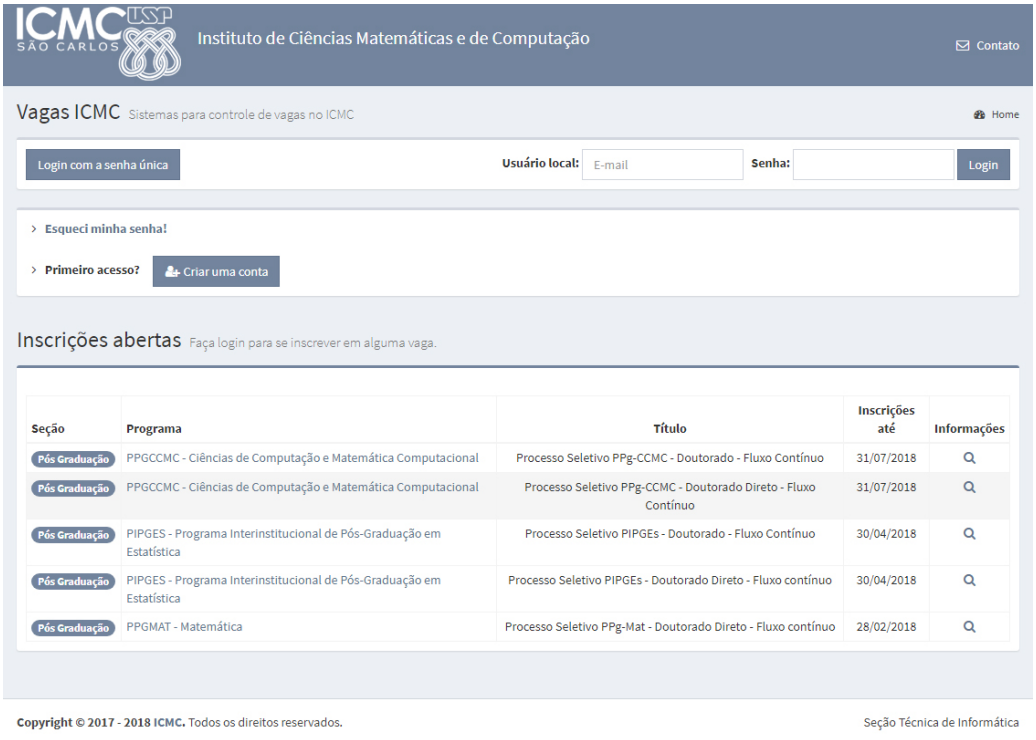


Figura D.3: Página de controle de usuários e permissões

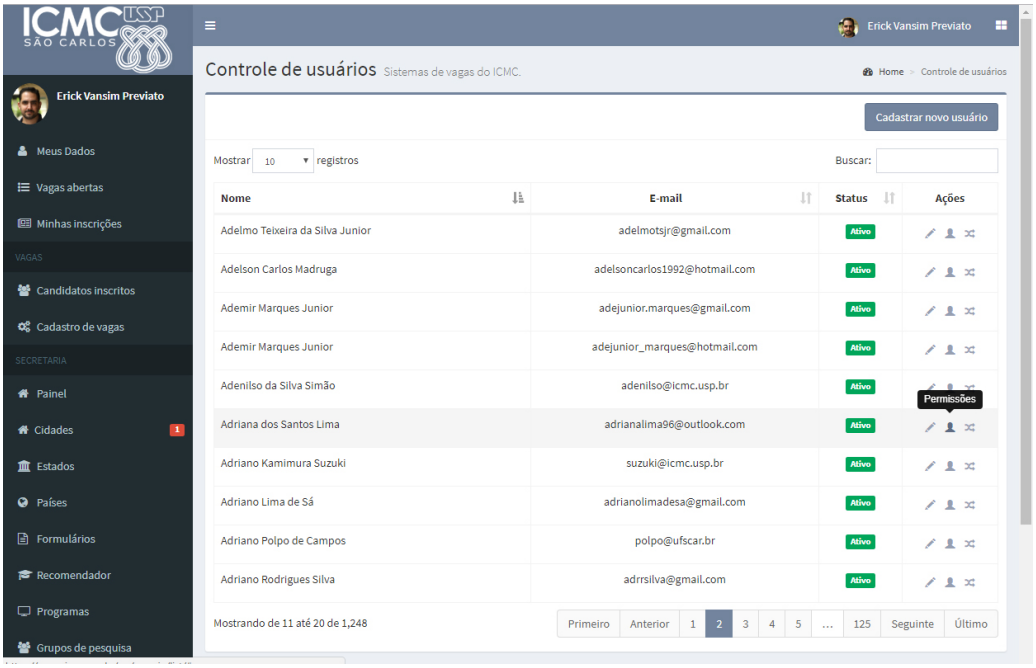


Figura D.4: Página de gerenciamento de cidades

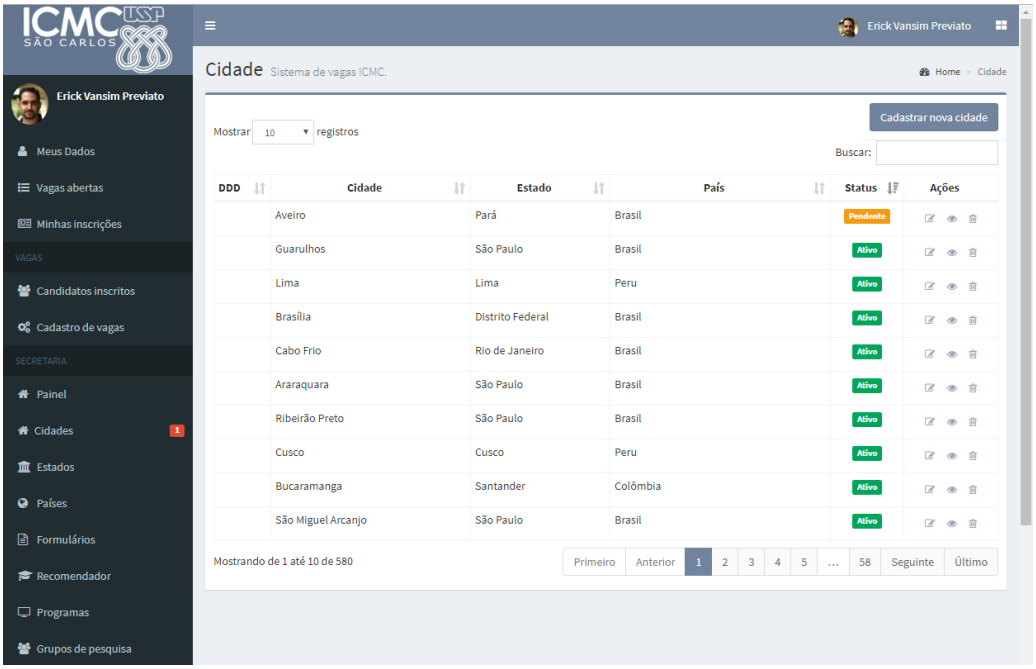


Figura D.5: Página de controle de vagas e editais

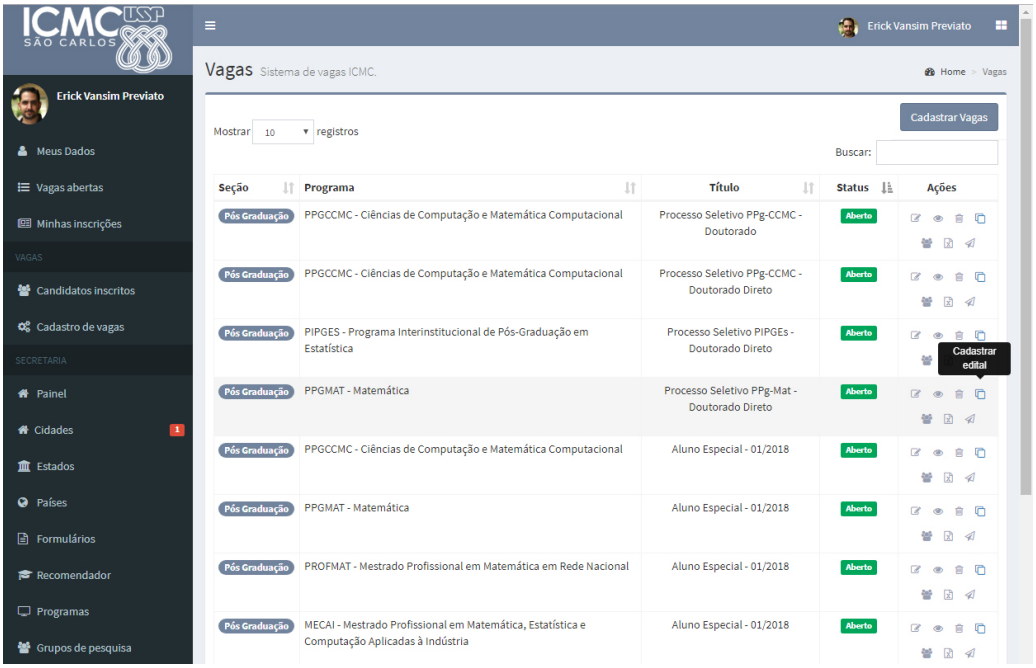
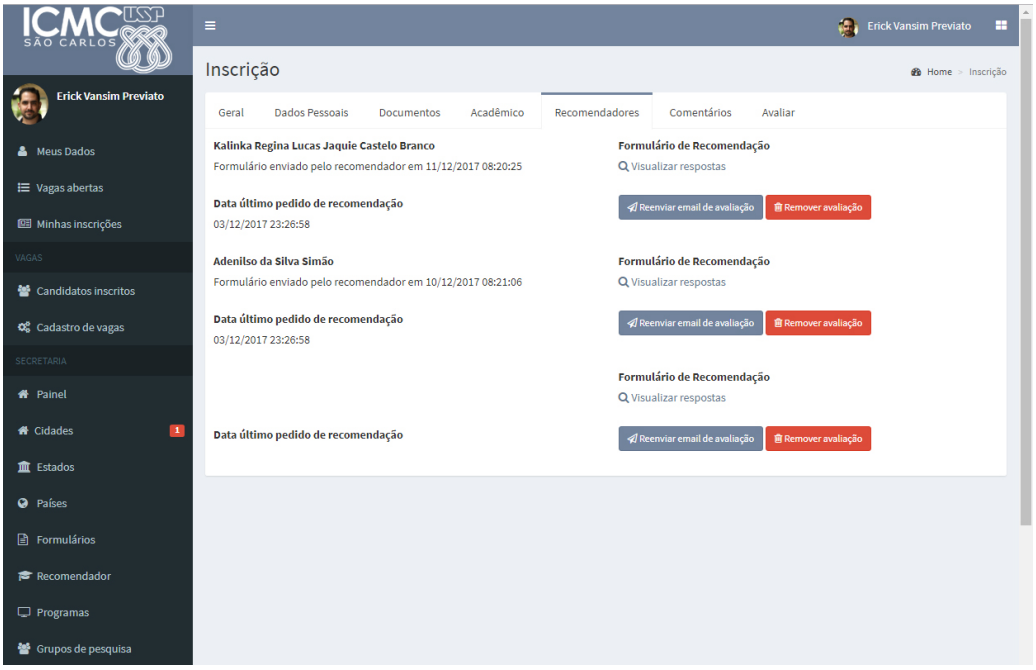


Figura D.6: Página dos detalhes de recomendadores de uma inscrição



Anexo E

SISTEMA DE RESERVA DE CAMPOS DA MULTI SPORT

Aplicação em desenvolvimento para gerenciar as reservas dos campos da escola Multi Sport. A aplicação final conterá as seguintes funcionalidades:

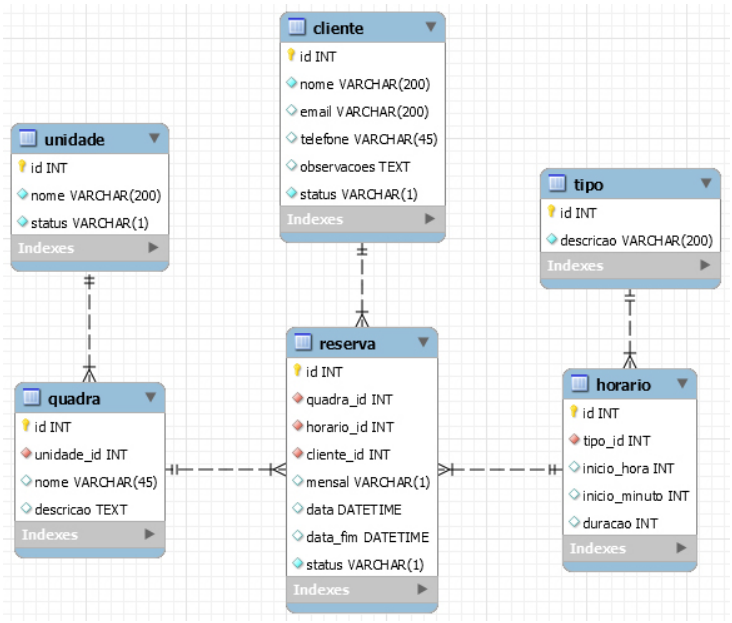
- Gerenciamento de unidades;
- Gerenciamento dos campos;
- Controle de horários;
- Gerenciamento de usuários;
- Cadastro de reservas;
- Agenda de horários.

Além das funcionalidades, a aplicação contará com um site institucional para apresentar dados institucionais, a agenda de horários e também o formulário de reserva para os usuários. Um diferencial será desenvolver um aplicativo mobile para apresentar os mesmos dados do site institucional.

Características Técnicas:

- Linguagem: PHP
- Possui Documentação: Apenas modelo do banco de dados (Figura E.1)

Figura E.1: Modelo do banco de dados da aplicação de reserva de campos



- Intuitivo: Sim

Screenshots:

Figura E.2: Página de gerenciamento de reservas

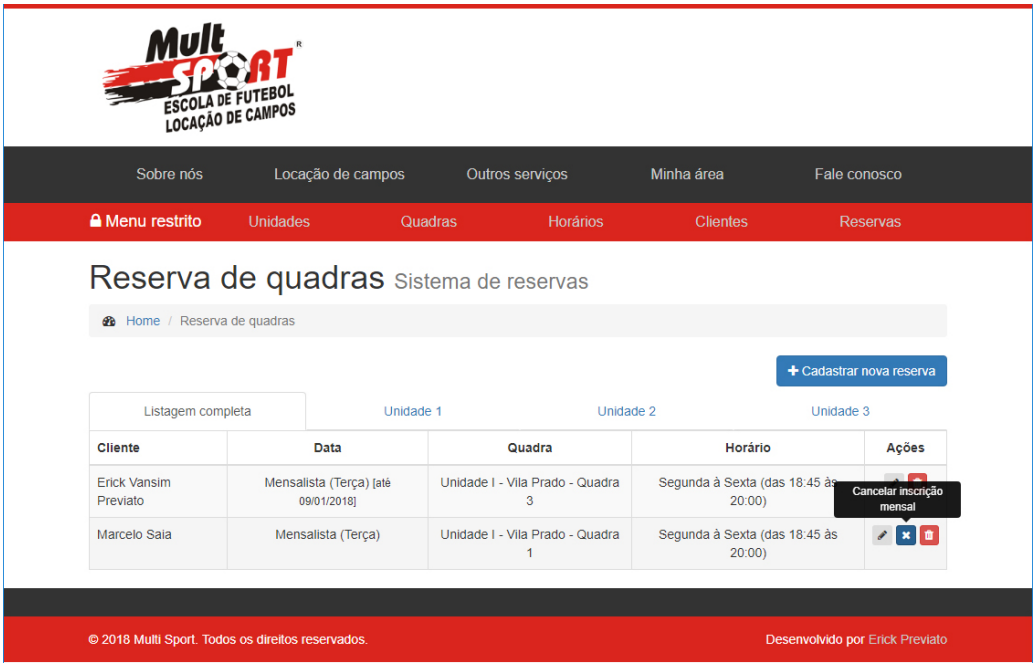


Figura E.3: Formulário de nova reserva

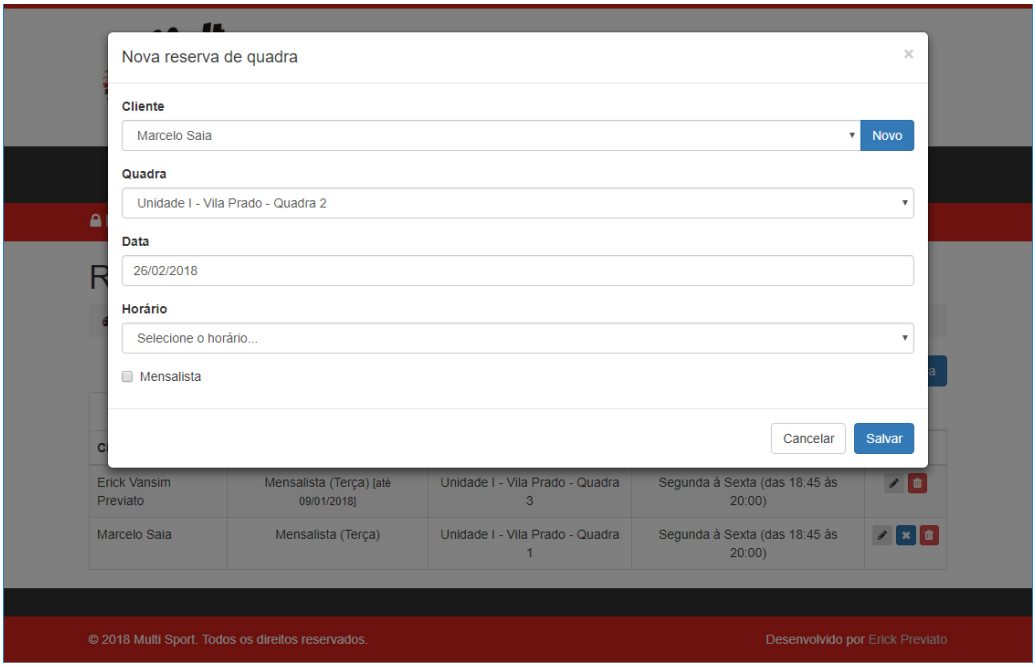


Figura E.4: Página de gerenciamento de clientes

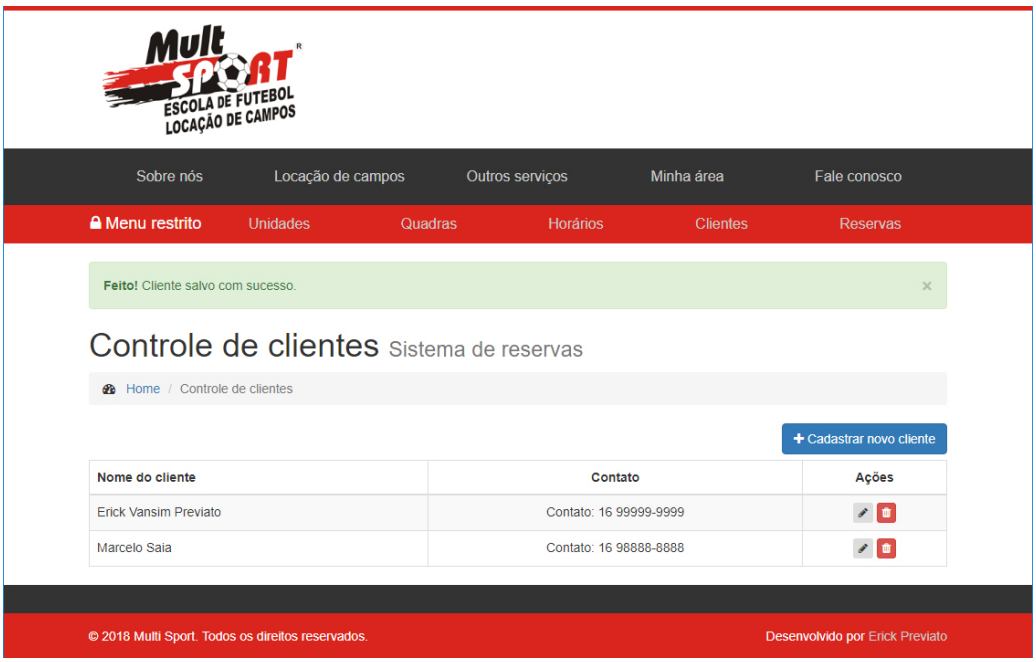


Figura E.5: Página de gerenciamento de quadras

Figura E.6: Página de gerenciamento das unidades da escola

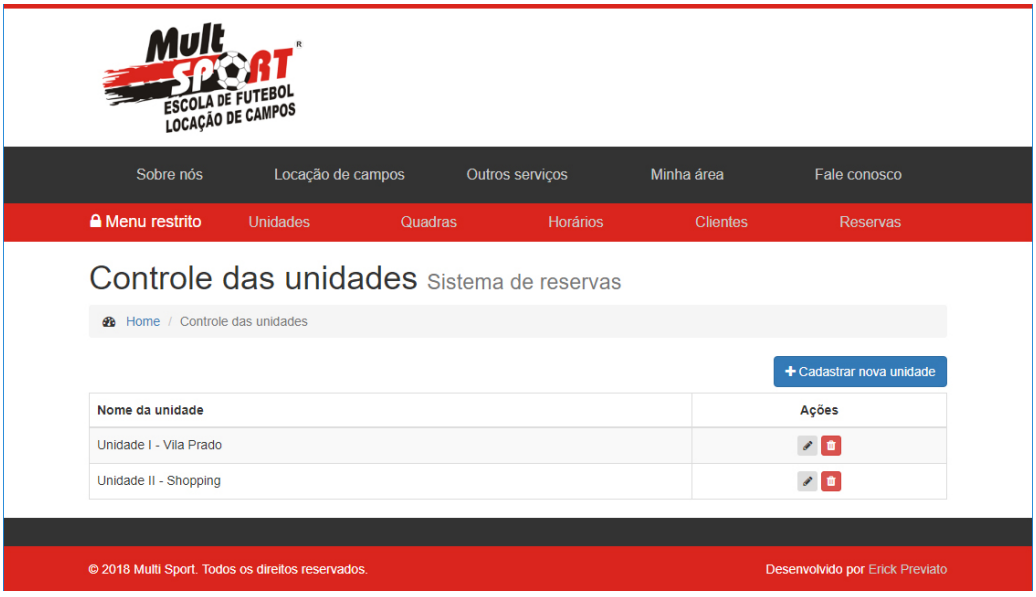


Figura E.7: Página da agenda de horários