INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ
IFCE CAMPUS ARACATI
COORDENADORIA DE CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO DE MOURA ROCHA

ESTUDO EMPÍRICO DAS METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE EM AMBIENTES ACADÊMICOS

ARACATI-CE
2017

GUSTAVO DE MOURA ROCHA

ESTUDO EMPÍRICO DAS METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE EM AMBIENTES ACADÊMICOS

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador (a): Prof. Dr. Reinaldo Bezerra Braga

Aracati-CE
2017

GUSTAVO DE MOURA ROCHA

ESTUDO EMPÍRICO DAS METODOLOGIAS DE DESENVOLVIMENTO DESOFTWARE EM AMBIENTES ACADÊMICOS

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

10 DE OUTUBRO DE 2017

BANCA EXAMINADORA

Prof. Dr. Reinaldo Bezerra Braga (Orientador)
Instituto Federal de Educação, Ciência e Tecnologia doCeará - IFCE - Campus Aracati

---

Profa. Dra. Carina Teixeira de Oliveira
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE - Campus Aracati

---

Prof. MSc. Fábio José Gomes de Sousa
Instituto Federal de Educação, Ciência e Tecnologia doCeará - IFCE - Campus Aracati

DEDICATÓRIA

Dedico este trabalho aos meus pais que sempre me deram apoio, e me mostraram que o segredo da vida está em nunca desistir. Dedico também este trabalho a minha amada noiva Carmen, por acreditar em mim e estar sempre ao meu lado. A minha família que de alguma forma me ajudou a chegar ao final de mais um ciclo da minha vida. Dedico também aos meus amigos que me ajudaram até o último momento. E finalmente dedico este trabalho aos meus professores a quem sou muito grato por todo o conhecimento adquirido.

AGRADECIMENTOS

Ao professor Dr. Reinaldo Bezerra Braga pela orientação, confiança e ensinamentos transmitidos.

Ao professor MSc. Fábio José Gomes de Souza e a professora Dra. Carina Teixeira de Oliveira por aceitarem fazer parte da banca examinadora.

Ao Instituto Federal de Educação, Ciência e Tecnologia doCeará - IFCE - Campus Aracati pelo apoio acadêmico e infraestrutura.

A minha família e noiva pelo apoio, companheirismo e compreensão.

A todos que contribuíram para a completude deste trabalho.

RESUMO

Atualmente, em projetos de desenvolvimento de sistemas, têm-se buscado diversas alternativas de construção e gerenciamento de sistemas que tornem o processo de desenvolvimento cada vez mais simples, rápido e eficaz. Em virtude disso, metodologias de desenvolvimento tradicionais e ágeis são aplicadas com o intuito de aprimorar esses processos. Dentro desse contexto, o presente trabalho busca identificar alguns desafios e dificuldades no processo de implantação de metodologias em ambientes acadêmicos. Esses problemas incluem o atraso na entrega de projetos, a falta de profissionais qualificados, a dificuldade de comunicação com *stakeholders*, a sobrecarga de responsabilidades do professor pesquisador, a desmotivação e o principal desafio entre todos os projetos analisados, o aluno. Capaz de colocar em risco o sucesso dos projetos, os estudantes de graduação são os recursos considerados mais desafiadores. Para a validação e apresentação dos resultados dessa pesquisa, aplicou-se um questionário nas equipes envolvidas, de forma a obter a percepção do grupo em relação a metodologia utilizada em uma instituição pública do governo federal do estado do Ceará. O estudo empírico foi realizado com quatro estudos de casos. Este trabalho tem como principal contribuição evidenciar a importância da dedicação dos envolvidos, alunos e professores em um projeto de pesquisa de um ambiente acadêmico.

Palavras-chaves: Metodologias. Dificuldades. Desenvolvimento de Software. Ambientes Acadêmicos.

ABSTRACT

Currently, in system development projects, we have been looking for a variety of system construction and management alternatives that make the development process simpler, faster and more efficient. As a result, traditional and agile development methodologies are applied with the purpose of improving these processes. Within this context, the present work seeks to identify some challenges and difficulties in the process of implanting methodologies in academic environments. These problems include the delayed delivery of projects, the lack of qualified professionals, the difficulty of communication with stakeholders, the overloading of responsibilities of the researcher teacher, the demotivation and the main challenge among all the projects analyzed, the student. Capable of jeopardizing the success of projects, undergraduates are the most challenging resources. For the validation and presentation of the results of this research, a questionnaire was applied to the teams involved, in order to obtain the group's perception regarding the methodology used in a public institution of the federal government of the state of Ceará. The empirical study was carried out with four case studies. This work has as main contribution evidence the importance of the dedication of the involved, students and professors in a research project of an academic environment.

Keywords: Methodologies. Difficulties. Software Development. Academic Environments.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IFCE	Instituto Federal de Educação, Ciência e Tecnologia do Ceará
TCC	Trabalho de Conclusão de Curso
XP	*Extreme Programming*
LAR-A	Laboratório de Redes deAracati
RUP	*Rational Unified Process*
KIS	*Keep it simple*
CRC	*Class-Responsibility-Colaborator*
PO	*Product Owner*
SM	*Scrum Master*
ROI	*Return of investiment*
PDCA	*Plan - Do - Check - Act*
DECOM	Departamento de Computação
UFOP	Universidade Federal de Ouro Preto
APP	Associação Peter Pan
CNPQ	Conselho Nacional de Desenvolvimento Científico e Tecnológico

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo, é feita a contextualização do problema abordado neste trabalho, dentro da perspectiva da área de Engenharia de Software. Além disso, são apresentadas a motivação e os objetivos gerais para o seu desenvolvimento.

1.1 Contextualização

Atualmente, o quadro econômico está competitivo, por isso a disputa pelos clientes é cada vez mais acirrada. Desta forma exige-se que empresas tomem uma atitude ativa e efetiva em relação aos clientes. As constantes mudanças no contexto financeiro, político, social e tecnológico atribuem mais desafios para as empresas que buscam uma posição de destaque. Surge, nesse sentido, a grande importância de uma organização ser capaz de se adaptar à forças externas que podem atuar de forma contrária as suas ações planejadas.

Sabendo disso, a tecnologia da informação é apontada como uma ferramenta poderosa que impulsiona um processo de transformação nas empresas, com foco no desenvolvimento de software. Portanto, o desenvolvimento de novos softwares participa de forma cada vez mais relevante nas atividades dessas empresas que desenvolvem softwares. Com o intuito de melhorar a qualidade dos softwares em geral, aumentar a produtividade de tais produtos e reduzir gastos, surgiu a Engenharia de Software.

A Engenharia de Software propõe métodos sistemáticos com o uso adequado de ferramentas e técnicas, que levam em consideração o problema a ser resolvido, as necessidades dos clientes e os recursos disponíveis. O engenheiro de software deverá ter em mente, no entanto, que o valor de um sistema depende da qualidade de cada um de seus componentes. Um sistema pode ter algoritmos codificados em seu software, e ser de péssimo desempenho por defeito de seu hardware, rede ou banco de dados. Como mostrado na Figura 1, cada um destes elementos pode pôr a perder a confiabilidade e a usabilidade do sistema (FILHO, 2010).

Sendo assim, gerenciar um projeto de software envolve, dentre outros fatores, o planejamento e o acompanhamento das pessoas envolvidas no projeto, principalmente a metodologia que está sendo seguida para evoluir o software de um conceito preliminar para uma implementação concreta e operacional (PRESSMAN, 2001). Na intensa competitividade do ambiente de desenvolvimento de software existem as metodologias utilizadas para elaboração e entrega dos softwares. (LIBARDI, 2010)

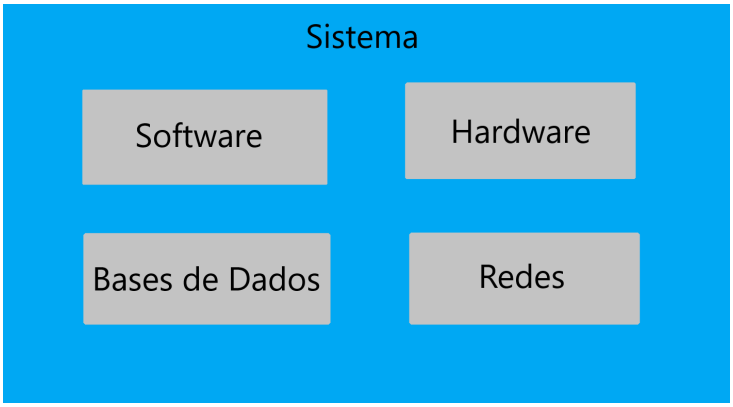

Figura 1 – Sistemas de Informática e suas partes

A Engenharia de Software divide as metodologias em duas grandes áreas: as metodologias tradicionais e as ágeis. As metodologias consideradas tradicionais, também chamadas de “pesadas”, têm como característica marcante ser dividida em etapas e/ou fases. Essas fases são muito bem definidas e englobam atividades como Análise, Modelagem, Desenvolvimento e Testes. Cada fase concluída gera um marco, que geralmente é algum documento, protótipo do software ou mesmo uma versão do sistema. Muitas metodologias pesadas são desenvolvidas em um modelo chamado de cascata, o que dificulta o controle do projeto, pois a cada alteração em determinado ponto do projeto, como os requisitos, será necessário uma volta ao início do mesmo para alteração de documentação ou de outro marco (PRESSMAN, 2001).

O foco principal das metodologias tradicionais é a previsibilidade dos requisitos do sistema, que traz a grande vantagem de tornar os projetos completamente planejados, facilitando a gerência do mesmo, mantendo sempre uma linha, caracterizando o processo como bastante rigoroso (OLIVEIRA S. R. B, 2004).

Em contrapartida, os métodos ágeis caracterizam-se por sua característica adaptativa e orientada para pessoas. Várias metodologias são classificadas como ágeis, por exemplo: *Extreme Programming* (XP) e Scrum. As metodologias ágeis trabalham com equipes motivadas e possuem suporte a mudanças durante o processo de desenvolvimento. Essas metodologias trabalham com a complexidade da construção de um software através da adaptação, controle de inspeção e visibilidade de requisitos. O projeto é dividido em ciclos chamados *Sprints*. Esses ciclos representam um conjunto de atividades que devem ser executado em uma *TimeBox* [1]. Diferentemente do modelo em cascata, esse modelo emprega uma estrutura iterativa e incremental. A cada iteração, a equipe de desenvolvimento analisa o que deve ser feito e seleciona tarefas para serem realizadas em uma *Sprints*, e que podem se tornar um incremento de valor ao produto no final da iteração. A equipe analisa a cada ciclo os requisitos,

[1] Tradução literal significa "caixa de tempo". Ou seja, é saber o tempo para realizar um trabalho, limitado, executando da melhor forma que puder nessa janela de tempo.

a tecnologia e suas habilidades. Desta forma, entrega o melhor software possível, adaptando-se diariamente conforme as dificuldades e surpresas que eventualmente ocorram.

1.2 Motivação

Uma empresa pode até ter outros objetivos, todos secundários, porém o objetivo principal é o lucro. Uma vez que os objetivos dessas empresas são quase sempre ambiciosos e a maioria deles de curto prazo. Uma universidade, por sua vez, trabalha com o conhecimento e tem como principal intuito preparar profissionais para a sociedade.

Se ciência e tecnologia, entendidas no sentido amplo, constituem a base de qualquer projeto de nação, como elemento básico para o desenvolvimento sustentável, é claro que a Universidade ocupa um papel fundamental neste contexto, pois ela é o centro de geração e difusão de conhecimento. A pesquisa aplicada e o desenvolvimento que levam à inovação tecnológica e ao aumento da competitividade precisam ocorrer dentro das universidades em parceria com empresas e empreendedores, construindo assim uma estratégia nacional de inovação (CHIMENDES, 2013).

Grande parte das metodologias de desenvolvimento de software foi projetada para satisfazer as necessidades das empresas desenvolvedoras de software atuantes no mercado. No entanto, laboratórios de desenvolvimento de software e sistemas de computação hospedados em universidades públicas, são ambientes diferentes das empresas que desenvolvem softwares (PEREIRA, 2014). Um produto com alta qualidade e menor tempo de entrega é uma necessidade que está cada vez mais concorrendo com essas empresas. Uma vez que os projetos de pesquisa em ambientes acadêmicos, apresentam as mesmas responsabilidades acerca do tempo de entrega e o valor do produto, porém com características diferentes. Nos laboratórios de pesquisas, as pessoas envolvidas nas equipes são geralmente, estudantes de graduação e pós-graduação como também os professores que são os responsáveis pelos laboratórios.

	Empresas de Desenv. de Software	Laboratórios de Pesquisa
Equipe	Profissionais da área	Estudantes e Professores
Dedicação	40hs/semana	12 a 20hs/semana
Comprometimento	Total	Parcial
Liderança	Gerente de Projeto	Professores/Líder Técnico

Tabela 1 – Tabela comparativa entre Empresas de Desenv. de Software vs Laboratórios de Pesquisas

As diferenças entre Empresas Desenvolvedoras de Software e os Laboratórios

de Pequisas que estão apresentadas na tabela 1 mostram as principais diferenças. As equipes das empresas são formadas por profissionais da área, ou seja, pessoas com capacitação e maturidade para desenvolver sua atividade. Por outro lado, os laboratórios possuem em sua equipe de desenvolvimento estudantes de graduação e pós-graduação, pessoas em fase de aprendizagem/formação. Outra diferença é a dedicação a respeito da quantidade de horas trabalhadas. Nas empresas os profissionais trabalham oito (8) horas diárias, gerando assim quarenta (40) horas semanais, enquanto nos laboratórios, os alunos trabalham por volta de quatro (4) horas por dia constituindo vinte (20) horas semanais. Outro ponto é o comprometimento, isto é, realizar aquilo que se propôs a alcançar. Os alunos possuem geralmente um comprometimento parcial, pois têm como prioridade suas disciplinas, períodos de provas e outros fatores que comprometem este ponto. E por fim, temos a grande diferença, que é a liderança dos projetos. As empresas possuem, como descrito, um profissional específico para desenvolver esse papel de importância, chamado Gerente de Projeto. Este profissional é responsável por planejar, controlar e assegurar a realização do projeto. Enquanto nos laboratórios, geralmente não existe um gerente, e sim um líder técnico, que na maioria das vezes é o professor/pesquisador.

Existem alguns desafios encontrados nos projetos de pesquisa em Universidades e Instituições Públicas. O aluno é o maior desafio para os projetos de iniciação científica. Pois estudantes de graduação, possuem limitações, não são profissionais, estão em processo de formação e assim podendo ter diferentes níveis de conhecimento. A quantidade de tempo disponibilizada por um estudante é completamente diferente de um profissional, pois o estudante dedica apenas uma parte do seu tempo e um projeto de pesquisa não é sua prioridade. Além disto, eles podem abandonar o projeto em qualquer momento, sem que cause complicações para sua caminhada acadêmica e profissional.

Com relação aos professores, são os responsáveis pelos laboratórios, necessitam além de desempenhar suas obrigações na universidade, realizar suas pesquisas, assegurar a qualidade dos produtos gerados, obter recursos financeiros para seus projetos de pesquisa e fazer o acompanhamento local dos alunos que estão participando dos projetos. Isso envolve o convívio de perto e contínuo com as equipes. Os professores pesquisadores muitas vezes transferem algumas atividades dos projetos para suas equipes, devido ao volume de suas ocupações.

Essas comparações de ambientes acadêmicos com empresas, nos remetem a alguns questionamentos com relação a metodologia de software utilizadas. As metodologias utilizadas em empresas são úteis para ambientes acadêmicos? Até que ponto se pode modificar uma metodologia? Existem metodologias específicas para universidades/instituições? A equipe tem maturidade para a utilização de uma meto-

dologia?

Desse modo, com a finalidade de analisar os ambientes acadêmicos e responder as perguntas anteriormente formuladas, foi elaborado um questionário com perguntas que verificam os projetos de pesquisa do Laboratório de Redes de Aracati(Lar-A). Este trabalho tem como uma de suas principais contribuições, evidenciar a importância da dedicação dos envolvidos, alunos e professores, em um projeto de pesquisa em um ambiente acadêmico para que se obtenha uma melhor adaptação das metodologias utilizadas e assim realizar uma entrega de alta qualidade.

1.3 Objetivo

O objetivo principal deste trabalho é analisar os desafios encontrados na utilização de metodologias de desenvolvimento de software dentro de um ambiente acadêmico levando em conta o grau de envolvimento dos alunos e professores. Bem como as constantes mudanças que ocorrem em diversos sentidos no decorrer do projeto, mudança da equipe de desenvolvimento, por exemplo, para que se possa melhorar ou inclusive criar uma metodologia específica para universidades/instituições.

Para validação deste trabalho foi realizado um estudo de caso com quatro projetos de desenvolvimento de software em atuação no Laboratório de Redes de Aracati (Lar-A).

1.4 Organização do Trabalho

Este trabalho é dividido em quatro capítulos. O Capítulo 2 é divido em três subseções: a primeira aborda os conceitos básicos sobre o entendimento geral de metodologias tradicionais de desenvolvimento; a segunda apresenta os conceitos básicos de metodologias ágeis; e a terceira mostra os trabalhos correlatos a esta pesquisa.

O Capítulo 3 contém a proposta adotada, apresentando os artefatos de entrada para a pesquisa, a metodologia e coleta dos dados.

O Capítulo 4 reporta a análise dos dados coletados com seus resultados.

Finalmente, o Capítulo 5 apresenta as limitações avaliadas, as lições aprendidas desta pesquisa e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Um conjunto de métodos, técnicas e ferramentas que determinam como um processo deve ser elaborado é chamado de metodologia. Método é uma maneira organizada de realizar uma tarefa enquanto técnica é uma forma, não organizada, de realizar uma tarefa. Desse modo, entende-se por metodologia a organização que se utiliza de técnicas e aplica métodos bem definidos. A equipe de desenvolvimento pode realizar adaptações das metodologias, adequando assim a realidade ou necessidade do projeto. Neste sentido, este capítulo aborda as metodologias de desenvolvimento de software desde a metodologia tradicional até as metodologias ágeis.

2.1 Metodologias Tradicionais

As metodologias tradicionais definem um conjunto distinto de atividades, ações tarefas, marcos e produtos de trabalho que são necessários para fazer engenharia de software com alta qualidade. Essas metodologias não são perfeitas mas efetivamente fornecem um roteiro útil para o trabalho da engenharia de software (PRESSMAN, 2001).

2.1.1 Modelo Cascata

Uma das primeiras metodologias propostas foi o modelo cascata. Ilustrado na Figura 2, na qual os estágios são apresentados em sequência, como em uma cascata (ROYCE, 1970). Como mostra a Figura 2 o desenvolvimento de um estágio deve terminar antes do próximo começar. Desse modo, só quando todos os requisitos forem enunciados pelo cliente, tiverem sua completeza e consistência analisadas e tiverem sido documentados em um documento de especificação, é que a equipe de desenvolvimento pode realizar as atividades do projeto do sistema. O modelo em cascata apresenta uma visão de muito alto nível do que acontece durante o desenvolvimento e sugere aos desenvolvedores a sequência de eventos que eles devem esperar encontrar (PLFEEGER, 2004).

O modelo cascata tem sido utilizado para indicar as atividades de desenvolvimento em uma variedade de contexto. Este modelo pode ser muito útil para ajudar os desenvolvedores a descrever o que eles precisam fazer. Sua simplicidade o torna fácil de explicar aos clientes não familiarizados com o desenvolvimento de software. Este método explicita os produtos intermediários para começar o próximo estágio de

desenvolvimento. Outros modelos mais complexos são apenas variações da metodologia cascata, incorporando *loops* de *feedback* e atividades extras.

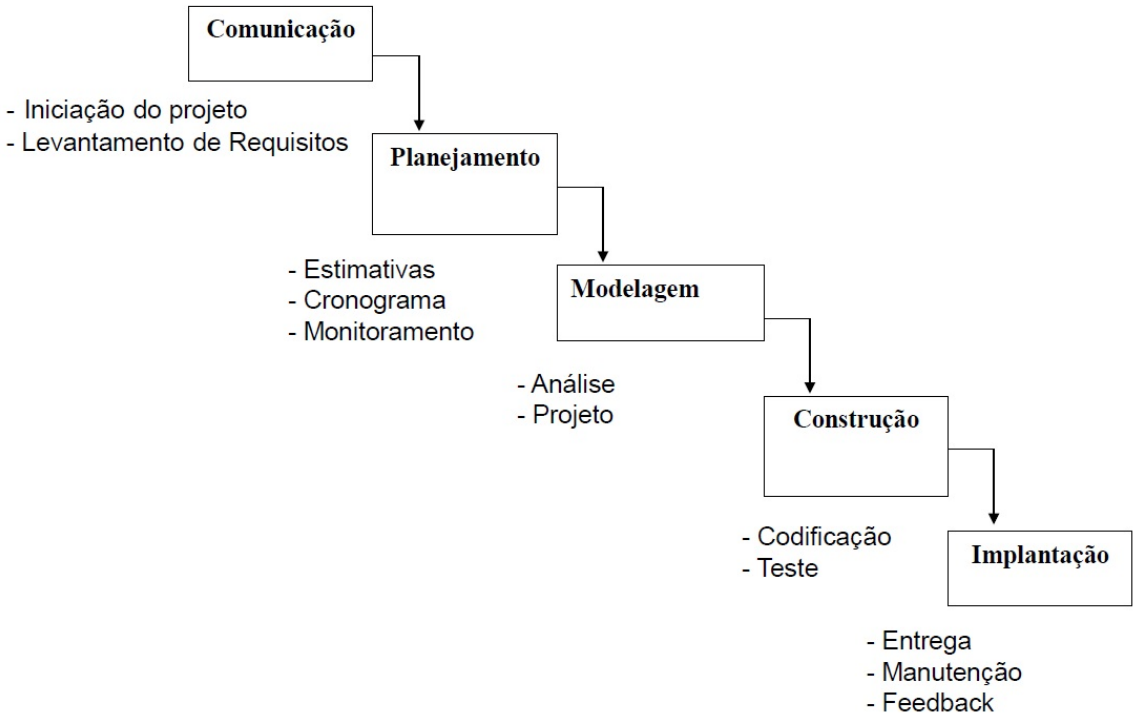

Figura 2 – Modelo Cascata

Fonte: Pressman (2001)

Entretanto, desde que o modelo cascata foi introduzido, ele tem sido muito criticado. Segundo McCracken D. D.; Jackson (1981) o modelo impõe uma estrutura de gerenciamento de projeto ao desenvolvimento de sistema. "Argumentar que qualquer esquema de ciclo de vida, até mesmo com variações, pode ser aplicado a todo desenvolvimento de sistema é fugir da realidade ou assumir um ciclo de vida tão rudimentar que chega a ser vazio."

Observa-se que o este modelo mostra que cada uma das principais fases de desenvolvimento acaba na produção de algum artefato (tais como requisitos, projetos ou códigos). Não há detalhes de como cada fase transforma um artefato em outro, por exemplo, os requisitos em projeto. O modelo não fornece orientação para os gerentes e desenvolvedores de como tratar as mudanças nos produtos e atividades que provavelmente ocorrerão durante o desenvolvimento. Durante a atividade de programação os requisitos sofrerão alterações, as mudanças subsequentes no projeto e no código não são indicados no modelo em cascata (PLFEEGER, 2004).

Curtis Bill; Krasner (1988) observaram que esta metodologia é derivada do mundo do hardware apresentando o desenvolvimento de software do ponto de vista do fabricante de hardware. A fabricação produz um item específico, que é, então,

reproduzido várias vezes. O software não é desenvolvido desta maneira, em vez disso, ele evolui à medida que o problema é melhor compreendido e as alternativas avaliadas. Logo software é resultado de um processo de criação e não de fabricação. O modelo cascata não informa nada sobre as atividades "de ida e volta"que levam à criação do produto final. Especificamente, a criação envolve tentar um pouco de várias alternativas, desenvolver e avaliar projetos, avaliar a viabilidade dos requisitos, contrastando diversos projetos, aprendendo com as falhas, e, finalmente, chegando a uma solução satisfatória para o problema (PLFEEGER, 2004).

2.1.2 Rational Unified Process (RUP)

Esta sessão examina as melhores práticas para o desenvolvimento de software e estabelece o propósito do Rational Unified Process.

Diferentes projetos de desenvolvimento de software falham de diferentes formas. Porém, é possível identificar alguns sintomas que caracterizam esses tipos de projetos (KRUCHTEN, 2003).

- Incompreensão das necessidades do usuário final;
- Inabilidade para lidar com requisitos variáveis;
- Módulos que não se ajustam;
- Software difícil de manter e estender;
- Descoberta tardia de sérias imperfeições do projeto;
- Baixa qualidade de software;
- Desempenho inaceitável de software;
- Os membros da equipe um no caminho do outro, tornando impossível reconstruir quem mudou o quê, quando, onde e por quê;
- Um processo de construção e lançamento indigno de confiança.

Infelizmente, considerar esses sintomas não trata o mal. Por exemplo, a descoberta tardia de sérias imperfeições do projeto é apenas um sintoma de problemas maiores, isto é, avaliação subjetiva de status do projeto e inconsistências não detectadas de requisitos, construções e implementações do projeto (KRUCHTEN, 2003).

Embora projetos diferentes falhem de formas diferentes, parece que a maioria deles falha por causa de uma combinação das seguintes causas de origem:

- Gerenciamento especial de requisitos;
- Comunicação ambígua e imprecisa;
- Arquiteturas frágeis;
- Complexidade subjugada;
- Inconsistências não detectadas em requisitos, construções e implementações;
- Teste insuficientes;
- Avaliação subjetiva de status do projeto;
- Deficiência para risco de ataque;
- Propagação de mudança incontrolada;
- Automação insuficiente.

Tratar essas causas de origem, não somente eliminará os sintomas, como estará em situação melhor para desenvolver e manter o software de qualidade de maneira repetitiva e previsível.

Isto é sobre o que são as melhores práticas de software: as abordagens experimentadas comercialmente para o desenvolvimento de software que, usadas em combinação, atacam as origens dos problemas de desenvolvimento de software [1]. São "melhores práticas"não tanto pelo motivo de se poder quantificar precisamente seu valor, mas porque essas práticas são geralmente usadas na indústria por organizações bem-sucedidas (KRUCHTEN, 2003).

Segundo Kruchten (2003) são melhores práticas:

1. Desenvolver o software iterativamente;
2. Gerenciar os requisitos;
3. Usar arquiteturas baseadas em componente;
4. Modelar visualmente o software;
5. Verificar continuamente a qualidade do software;
6. Controlar suas mudanças.

[1] Ver o trabalho de melhores práticas *Software Program Manager's Network em http://www.spmm.com*

Desenvolver o sistema iterativamente

O problema básico desta abordagem é que adia o risco de forma que torna caro desfazer erros de fases anteriores. Uma construção inicial provavelmente será imperfeita com respeito aos seus requisitos-chaves e, além disso, a descoberta tardia de defeitos de construção tende a resultar em custos excedentes ou cancelamento do projeto. Como Gilb (1988) habilmente disse: "Se você não ataca os riscos ativamente em seu projeto, ele só atacarão ativamente". A abordagem em cascata tende a mascarar os riscos reais para um projeto até que seja tarde para fazer qualquer coisa significante neles (KRUCHTEN, 2003).

Uma alternativa para a abordagem em cascata é o processo iterativo e incremental, mostrado na Figura 3. Nesta abordagem, construída no trabalho de modelo espiral [2], a identificação de riscos para um projeto é afetada cedo no ciclo de vida, quando é possível atacar e reagir de maneira pontual e eficiente. Esta abordagem é uma contínua descoberta, invenção e implementação, com cada iteração forçando a equipe de desenvolvimento a conduzir os artefatos do projeto a uma conclusão previsível e repetitiva (KRUCHTEN, 2003).

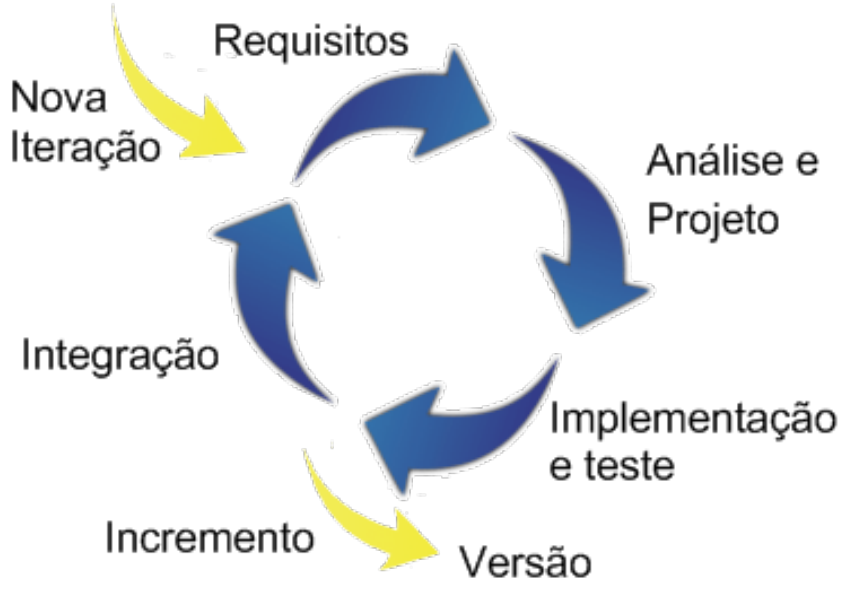

Figura 3 – Processo Iterativo e Incremental

Fonte: Kruchten (2003)

Gerenciar Requisitos

O desafio do gerenciamento de exigências de um sistema de software intensivo é que ele é dinâmico: deve-se esperar que estes requisitos se modifiquem durante

[2] Barry W. Boehm, "A Spiral of Software Development and Enhancement", IEEE Computer, maio de 1988, p.61-72

a execução do projeto. Além disso, identificar os verdadeiros requisitos de um sistema é um processo contínuo. Com excessão do sistema mais simples, é impossível expor completa e exaustivamente os requisitos de um sistema antes do início do desenvolvimento. De fato, a compreensão do usuário sobre os requisitos do sistema também muda, seja ele novo ou uma evolução de mudanças do sistema (LARMAN, 2004).

O gerenciamento ativo de requisitos abrange três atividades: dedução, organização e documentação da funcionalidade e embaraços exigidos do sistema; avaliação de mudanças para essas exigências e avaliação do seu impacto; localização e documentação de comercializações e decisões (KRUCHTEN, 2003).

Gerenciar os requisitos de seu projeto oferece várias soluções para as causas de origem de problemas do desenvolvimento de software:

1. Uma abordagem disciplinada é construída no gerenciamento de requisitos;
2. As comunicações são baseadas nos requisitos definidos;
3. Os requisitos podem ser priorizados, filtrados e localizados;
4. É possível uma avaliação objetiva da funcionalidade e do desempenho;
5. Inconsistências são detectadas mais cedo;
6. Com suporte de ferramentas satisfatório, é possível fornecer um repositório para requisitos, atributos e localizações de um sistema, com links automáticos para documentos externos.

Arquiteturas baseadas em componentes

A arquitetura de software está relacionada não somente à estrutura e ao comportamento, mas também ao uso, a funcionalidade, o desempenho, a capacidade rápida de recuperação, a reutilização, a compreensão, a economia e a limitação tecnológica e comercialização e o interesse estético (KRUCHTEN, 2003).

O desenvolvimento baseado em componente é uma abordagem importante para a arquitetura de software, porque capacita a reutilização ou personalização de componentes de milhares de fontes disponíveis comercialmente.

Junto com a prática de desenvolver iterativamente o software, usar arquitetura baseada em componente envolve a evolução contínua da arquitetura de um sistema. Cada iteração produz uma arquitetura executável que pode ser medida, testada e avaliada contra os requisitos dos sistema. Esta abordagem permite que a equipe ataque continuamente os riscos mais importantes para o projeto (KRUCHTEN, 2003).

Software visualmente modelado

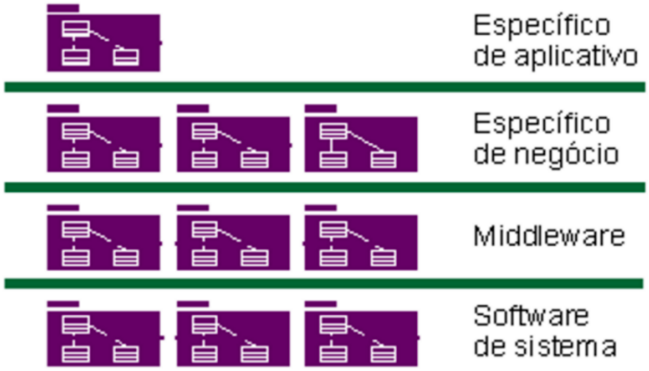

Figura 4 – Arquitetura de software baseada em componentes

Fonte: Rational Software Corporation em `http://www.funpar.ufpr.br:8080/rup/manuals/intro/im_bp3.htm`

As ferramentas de modelagem visual facilitam o gerenciamento desses modelos deixando-o ocultar ou expor detalhes quando necessário. A modelagem visual também ajuda a manter a consistência entre os artefatos do sistema: seus requisitos, construções e implementações. Resumindo, a modelagem visual ajuda a melhorar a capacidade da equipe em gerenciar a complexidade do software (KRUCHTEN, 2003).

Unido à pratica do desenvolvimento iterativo de software, a modelagem visual ajuda-o a expor e avaliar as mudanças arquiteturais e comunicar essas mudanças para a equipe inteira de desenvolvimento. Com o tipo correto de ferramentas, pode-se então sincronizar seus modelos e código-fonte durante cada iteração.

Verificar continuamente a qualidade do software

Verificar a qualidade de software oferece soluções para as causas das origem de problemas o desenvolvimento de software. A avaliação de status do projeto é feita de forma objetiva e não subjetiva, porque são avaliados os resultados de testes, não documentos em papel. Essa avaliação objetiva expõe inconsistências de requisitos, construções e implementações. Teste e verificação são concentrados nas áreas de risco mais alto, aumentando assim a qualidade e efetividades dessas áreas. Deficiências são identificadas cedo, reduzindo radicalmente o custo para fixa-los. Ferramentas de teste automatizadas fornecem teste para funcionalidade, confiabilidade e desempenho (KRUCHTEN, 2003).

Controlar mudanças do software

Coordenar as atividades, os artefatos desenvolvidos e equipes envolve estabelecer fluxos repetíveis para gerenciar mudanças em software e artefatos de desenvolvimento. Esta coordenação permite uma melhor alocação de recursos, baseando-se nas prioridades e riscos do projeto, e gerencia o trabalho ativamente nessas mu-

danças em iterações. Junto com o desenvolvimento iterativo de seu software, esta prática lhe permite monitorar continuamente as mudanças, de forma que pode-se descobrir ativamente e depois reagir aos problemas (KRUCHTEN, 2003).

Segundo Kruchten (2003) controlar as mudanças do software oferece soluções para os problemas no desenvolvimento:

- O fluxo de mudança de requisitos é definido e repetível;
- Solicitações de mudança facilitam comunicações claras;
- Estações de trabalho isoladas reduzem a interferência entre membros da equipe trabalhando em paralelo;
- As estações de trabalho contêm todos os artefatos, o que facilita a consistência;
- a propagação de mudança é avaliável e controlada;
- As mudanças podem ser mantidas num sistema robusto e personalizável.

2.2 Metodologias Ágeis

Em fevereiro de 2001, um grupo com 17 membros da comunidade de software se reuniram e definiram oficialmente o Desenvolvimento Ágil de Software, criando o *manifesto ágil*, com o objetivo de propor melhores maneiras de desenvolver softwares. O manifesto ágil afirma valorizar "os indivíduos e interações sobre processos e ferramentas, software funcionando mais que documentação abrangente, colaboração do cliente sobre negociação de contratos, e responder as mudanças seguindo um plano" (SCHWABER, 2001).

Usuários das metodologias ágeis devem seguir um conjunto de 12(doze) princípios destacado por esse manifesto (FOWLER, 2001). São eles:

1. A maior prioridade é satisfazer o cliente através da entrega contínua e precoce de software funcionais;
2. Aceitar as mudanças nos requisitos, mesmo no fim do desenvolvimento, pois processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas;
3. Entregar software funcionando com frequência, em uma escala de semanas a poucos meses, com preferência aos períodos mais curtos;
4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente por todo o projeto;

5. Construir projetos em torno de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para, e entre a equipe de desenvolvimento é através de uma conversa face a face;
7. Software funcional é a medida primária de progresso;
8. Os processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter um ritmo constante indefinidamente;
9. Constante atenção à excelência técnica e bom design, aumenta a agilidade;
10. Simplicidade é a arte de maximizar a quantidade de trabalho que não precisou ser feito;
11. As melhores arquiteturas, requisitos e designs surgem de times auto organizáveis;
12. Em intervalos regulares, a equipe reflete como se tornar mais eficaz e então se então refina e ajusta seu comportamento de acordo.

Sendo uma de suas principais características, a habilidade de reduzir os custos das mudanças ao longo do processo de desenvolvimento do software, os métodos ágeis surgiram a partir de um interesse em sanar as fraquezas existentes na engenharia de software convencional (PRESSMAN, 2001).

2.2.1 Manifesto Ágil

Devido ao grande número de referências a esses processos leves, que emergiam como resposta aos constantes fracassos de projetos utilizando abordagens tradicionais, em fevereiro de 2001 um grupo de profissionais extraordinários do desenvolvimento de software reuniu-se em um Resort de Ski em Wasatch Range para discutir melhores maneiras de desenvolver software. Esse encontro deu origem ao manifesto ágil, uma declaração com os princípios que regem o desenvolvimento ágil (BECK et al., 2001).

De acordo com Beck et al. (2001):

"Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar:"

- **Indivíduos e a interação** entre eles mais que processos e ferra- mentas;
- **Software em funcionamento** mais que documentação abrangente;
- **Colaboração com o cliente** mais que negociação contratual;
- **Responder a mudanças** mais que seguir um plano.

"Mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda."

Os profissionais que deram origem ao manifesto ágil foram Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland e Dave Thomas.

O Manifesto ágil é o embasamento filosófico de todos os métodos ágeis e diversos métodos de desenvolvimento de software estão alinhados a ele. A maioria deles se utiliza de ciclos curtos, que são chamados de iterações e normalmente têm duração de poucas semanas, dessa forma garantindo *feedback* frequente e respostas rápidas às mudanças (GOMES, 2010).

Cada iteração contém todas as etapas necessárias para que se realize um incremento no produto, ou seja, no software: planejamento, análise, design, codificação, testes e documentação. Em métodos não ágeis, também conhecidos como métodos tradicionais, geralmente se encontra um processo em cascata em que todas as etapas citadas são executadas uma única vez e em sequência (ainda que idealmente, prevendo-se revisões incrementais de cada etapa, se necessário) (GOMES, 2010).

Métodos ágeis assumem imprevisibilidade natural do desenvolvimento de software, por isso, considera-se que o cliente também está aprendendo sobre o que precisa e, que a cada *feedback* pode mudar de idéia e alterar o escopo do projeto. Assume-se também que estimativas de esforço e tempo de desenvolvimento são, de fato, estimativas, e não devem ser tratadas como algo certo e sem margem de erro. Por assumir a imprevisibilidade envolvida no desenvolvimento de software, métodos ágeis se baseiam nos ciclos de *feedback* constante permitindo que o cliente e a equipe de desenvolvimento possam adaptar-se às mudanças, aumentando assim as chances de sucesso do projeto (GOMES, 2010).

Um fato interessante, que muitas vezes é deixado de lado nas discussões sobre métodos ágeis, é que o manifesto ágil não foi uma reação apenas aos métodos tradicionais e burocráticos, mas também foi uma reação aos métodos caóticos que resultavam em produtos de baixa qualidade. Os métodos ágeis representam, justamente, um meio termo entre métodos estruturados demais e métodos sem estrutura, são o meio termo entre a ordem e o caos (APPELO, 2011).

2.2.2 Extreme Programming (XP)

Proposto por Kent Beck, Ward Cunningham em 1996, a XP é resultado da união de princípios e boas práticas de programação. Ao juntar técnicas de desenvolvimento comprovadas por décadas em uma única abordagem, a XP, tem por inovação garantir que as mesmas sejam executadas ao extremo, assim como, assegurar que elas apoiem ao máximo umas às outras (BECK et al., 2001).

Extreme Programming é definido como sendo uma metodologia leve, eficiente e de baixo risco, com forma flexível, previsível, científica e apropriada para pequenas e médias equipes de desenvolvimento de software, onde os requisitos são vagos ou mudam rapidamente, segundo Beck et al. (2001).

Para Beck et al. (2001) existem um conjunto de cinco valores que estabelecem as bases para todo trabalho realizado como parte do XP. Onde cada um desses valores é usado como um direcionador das atividades, ações e tarefas específicas do XP.

1. **Comunicação:** deve ocorrer de maneira contínua entre os próprios desenvolvedores. Problemas em projetos pode acontecer porque alguém deixou de comunicar algo a outra pessoa. XP tem por objetivo manter a comunicação fluindo de forma rápida e eficaz entre a equipe de desenvolvimento e o cliente para criar um senso de cooperação eficiente, sendo que o XP emprega muitas práticas que não podem ser feitas sem comunicação.

2. **Simplicidade:** XP aposta no conceito de simplicidade onde fazer o mais simples e futuramente gastar um pouco mais para fazer mudanças é mais vantajoso que fazer algo mais complexo, mas que pode nunca ser utilizado.

3. **Feedback**: contribui para o aprendizado no sentido de aprender e melhorar. Quanto mais rápido for o feedback do cliente, mais rápido será identificado se o projeto está seguindo na direção correta e, caso não esteja, o esforço para efetuar correções será menor.

4. **Coragem:** Combinada com os três valores anteriores, a coragem se torna extremamente valiosa. É preciso coragem para inovar, para propor mudanças, para pedir ajuda quando necessário, para comunicar problemas ao cliente e encarar mudanças no projeto. Porém, em contrapartida a coragem utilizada de forma isolada, sem contrabalancear os demais valores é perigosa. Fazer algo sem levar em conta as suas consequências não é um exemplo de equipe de trabalho eficiente.

5. **Respeito:** é a parte fundamental para que os demais valores colaborem entre si. Respeitar os outros membros da equipe é essencial para o sucesso de um projeto de software, e somente com o respeito os outros valores poderão ser amplamente seguidos.

O XP usa uma abordagem orientada a objetos como seu paradigma de desenvolvimento. O XP inclui um conjunto de regras e práticas que ocorrem no contexto de quatro atividades de arcabouço: planejamento, projeto, codificação e teste. A Figura 5 ilustra o processo XP e mostra algumas das ideias-chave e tarefas que estão associadas a cada atividade da estrutura.

Figura 5 – O processo de *Extreme Programming*

Fonte: Adaptado de (PRESSMAN, 2001)

As atividades-chave do XP são resumidas nos parágrafos seguintes:

- ***PLANEJAMENTO:*** A atividade de planejamento começa com a criação de um conjunto de *histórias* (também chamado de *histórias de usuários*) que descrevem as características e funcionalidades requeridas para o software a ser construído. Cada história é escrita pelo cliente e é colocada em um cartão de indexação. O cliente atribui um *valor*(isto é, uma prioridade) para a história, com base no valor de negócio global da característica u da função [3]. Membros da equipe

[3] O valor de uma história pode também depender da presença de outra história.

XP avaliam então cada história e lhe atribuem um *custo*, medido em semanas de desenvolvimento. Se a história precisar mais do que 3(três) semanas de desenvolvimento, pede-se ao cliente para dividir a história em histórias menores e a atribuição de valor e custo ocorre novamente. É importante notar que novas histórias podem acontecer a qualquer momento.

Os cliente e a equipe XP trabalham juntos para decidir como agrupar histórias na versão seguinte (o próximo incremento do software) a ser desenvolvida peça equipe XP. Uma vez feito o *compromisso* básico para uma versão (um acordo quanto às histórias a ser incluídas, data de entrega e outros assuntos de projeto), a equipe XP determina as histórias que serão desenvolvidas em um dos 3(três) modos a seguir:

1. Todas as histórias serão implementadas imediatamente(dentro de poucas semanas);
2. As histórias com valor mais alto serão antecipadas no cronograma e implementadas primeiro.
3. As histórias de maior risco serão antecipadas no cronograma e implementadas primeiro.

Depois que a primeira versão do projeto (também chamada de incremento de software) tiver sido entregue, a equipe XP calcula a velocidade do projeto. Dito simplesmente, *velocidade do projeto*, é a quantidade de histórias do cliente implementadas durante a primeira versão. A velocidade do projeto pode ser usada para ajudar a estimar as datas de entrega e o cronograma para as versões subsequentes. E determinar se um comprometimento excessivo foi feito para todas as histórias ao longo de todo projeto de desenvolvimento. Se um comprometimento excessivo ocorrer, o conteúdo das versões será modificado ou as datas de entrega finais serão alteradas (PRESSMAN, 2001).

À medida que o trabalho de desenvolvimento prossegue, o cliente pode adicionar histórias, mudar o valor de uma história existente, subdividir histórias e eliminá-las. A equipe então reconsidera todas as versões remanescentes e modificas os seus planos adequadamente (PRESSMAN, 2001).

- ***PROJETO:*** O projeto XP segue rigorosamente o princípio KIS (*keep it simple* - mantenha a simplicidade). Um projeto simples é sempre preferível em relação a uma representação mais complexa. Além disso, o projeto fornece diretrizes de implementação para uma história exatamente como ela está escrita. O projeto de funcionalidade extra é desencorajado [4] (PRESSMAN, 2001).

[4] Essas diretrizes de projeto devem ser seguidas em todo método de engenharia de software, ainda que haja ocasiões em que notação e terminologia sofisticadas de projeto podem ficar no caminho da simplicidade

O XP encoraja o uso de cartões CRC como um mecanismo efetivo para raciocinar sobre o software no contexto orientado a objeto. Os cartões CRC (*Class-Responsibility-Colaborator*) identificam e organizam as classes orientadas a objetos que são relevantes para o incremento de software atual. Os cartões CRC são o único produto de trabalho do projeto que é realizado como parte do processo XP (PRESSMAN, 2001).

Se um problema de projeto difícil é encontrado como parte do projeto de uma história, o XP recomenda a criação imediata de um protótipo operacional daquela parte do projeto. Denominado *solução de ponta*, o protótipo de projeto é implementado e avaliado. A intenção é diminuir o risco quando a implementação verdadeira começa e validar as estimativas originais correspondem à história que contém o problema de projeto (PRESSMAN, 2001).

O XP encoraja a *refabricação*, uma técnica de construção que também é uma técnica de projeto. Segundo Fowler (2001) a refabricação é o processo de modificar um sistema de software de tal modo que ele não altere o comportamento externo do código, mas aperfeiçoe a estrutura interna. É um modo disciplinado de limpar o código (e modificar/simplificar o projeto interno) que minimiza as chances de introdução de defeitos. Em essência, quando você refabrica, está aperfeiçoando o projeto do código depois que ele foi escrito.

Como o projeto XP não utiliza praticamente nenhuma notação e produz poucos, ou quase nenhum, produto de trabalho que não sejam os cartões CRC e as soluções de ponta, o projeto é visto como um artefato provisório que pode e deve ser continuamente modificado à medida que a construção prossegue. A intenção da refabricação é controlar essas modificações sugerindo pequenas alterações de projetos que "podem aperfeiçoar radicalmente o projeto" (FOWLER, 2001). No entanto, o esforço necessário à refabricação pode crescer sensivelmente a medida que o tamanho de uma aplicação cresce (PRESSMAN, 2001).

Uma notação central do XP é de que o projeto ocorre tanto antes quanto *depois* que a codificação começa. Refabricação significa que o projeto ocorre continuamente à medida que o sistema é construído. De fato, a atividade de construção em si vai fornecer à equipe XP diretrizes de como aperfeiçoar o projeto.

- ***CODIFICAÇÃO:*** O XP recomenda que depois que as histórias forem desenvolvidas e o trabalho preliminar de projeto for feito, a equipe não avance para o código mas, em vez disso, desenvolva uma série de teste unitários que exercitarão cada uma das histórias que cada uma devem ser incluída na versão atual (incremento de software [5].Uma vez criada os testes unitários, o desenvolvedor

[5] essa abordagem é análoga a saber as questões do exame antes de começar a estudar. Ela torna o estudo muito mais fácil, focalizando a atenção apenas nas questões que irão ser formuladas.

está melhor preparado para focalizar o que precisa ser implementado para passar no teste unitário. Uma vez completado o código, ele pode ser submetido imediatamente ao teste unitário, fornecendo assim, *feedback* instantâneo para os desenvolvedores.

Um conceito chave durante a atividade de codificação(e um dos aspectos mais comentados do XP) é a *programação em pares*. O XP recomenda que duas pessoas trabalhem juntas em uma estação de trabalho de computador para criar o código correspondente a uma história. Isso fornece um mecanismo de solução de problemas em tempo real e de garantia de qualidade em tempo real. Também mantém os desenvolvedores focados no problema em mãos. Na prática, cada pessoa assume um papel ligeiramente diferente. Por exemplo, uma pessoa poderia pensar nos detalhes de código de uma parte específica do projeto, enquanto a outra garante que as normas de codificação estão sendo seguidas e que o código gerado vai se encaixar no projeto mais amplo da história (PRESSMAN, 2001).

A medida que os pares de programadores completam seu trabalho, o código que eles desenvolvem é integrado ao trabalho de outros. Em outros casos, os pares de programadores têm responsabilidade de integração. Essa estratégia de "integração contínua"ajuda a evitar problemas de compatibilidade e interface e fornece um ambiente de teste que ajuda a descobrir rapidamente os erros (PRESSMAN, 2001).

- ***TESTE:*** Já mencionamos que a criação de um teste unitário [6] antes da codificação começar é um elemento-chave da abordagem XP. Os testes unitários que são criados devem ser implementados usando uma estrutura que lhe permita ser automatizados(consequentemente, podem ser executados fácil e repetidamente). Isso encoraja uma estratégia de este de regressão sempre que o código é modificado(considerando a filosofia de refabricação) (PRESSMAN, 2001).

  À medida que os testes unitário individuais são organizados em uma "sequência universal de testes" (WELL, 2003), o teste de integração e validação do sistema pode ocorrer diariamente. Isso fornece a equipe XP uma indicação contínua de progresso e também pode levantar sinais de alerta se as coisas estiverem se deteriorando. Well (2003) afirma que "resolver pequenos problemas a cada intervalo de umas poucas horas, leva menos tempo do que resolver grandes problemas perto da data de entrega".

[6] O teste unitário focaliza um componente de software, exercitando a interface, as estruturas de dados e a funcionalidade do componente, um esforço para descobrir erros que são locais para o componente.

Os *testes de aceitação XP* também chamados de *testes do cliente*, são especificados pelo cliente e focam nas características e funcionalidades do sistema global que são visíveis e passíveis de revisão pelo cliente. Testes de aceitação são derivados das histórias do usuário que foram implementadas como parte de uma versão do software.

2.2.3 Scrum

Os métodos de desenvolvimento ágeis são fundamentais para o futuro dos sistemas de software flexíveis. Scrum é uma das avançadas maneira de gerenciar o desenvolvimento de software quando as condições do negócio estão mudando. Concebido por Jeff Sutherland e sua equipe de desenvolvimento no início dos anos 1990, o Scrum é um método de desenvolvimento ágil de software. (SCHWABER, 2001)

O Scrum contém um conjunto de princípios, estes usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as atividades estruturais de requisitos, análise, projeto, evolução e entrega. Em cada atividade deste método, ocorrem tarefas a realizar dentro de um padrão de processo chamado de Sprint (PRESSMAN, 2001).

É um processo para projetos que realizam desenvolvimento de software que seja focado nas pessoas, no qual o ambiente tenha uma mudança constante nos requisitos, podendo ser também considerado uma técnica utilizada para realizar o gerenciamento de processo para o desenvolvimento de software, além de propor entregar produtos no prazo correto, visando satisfação do cliente e com alto grau de qualidade. (SCHWABER, 2015)

O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos. Fundamentado nas teorias empíricas de controle de processo, ou em- pirismo. O empirismo afirma que o conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido. (SCHWABER, 2015)

A metodologia ágil Scrum baseia-se em averiguar continuamente se os requisitos do cliente (usuário/ *Stakeholders*) estão realmente sendo feitos, isto é, se os resultados são os esperados. Fundamentalmente, a verificação dos resultados é realizada a cada iteração, também chamada de *Sprint*, que ocorre dentro de um *TimeBox*. A cada *Sprint* é entregue um incremento do produto com um valor significativo, para avaliar se o objetivo dos requisitos estão sendo implementados de maneira correta.

Figura 6 – Ciclo da Metodologia Scrum

2.2.3.1 Papéis do Scrum

O Scrum possui três papéis principais, são eles, o *Scrum Master*(SM), o *Product Owner*(PO) e a equipe de desenvolvimento, formando o *time Scrum*. Esses papéis são ocupados por pessoas que trabalham em coletivo, numa base diária, para garantir o bom fluxo das informações e a resolução rápida de mudanças. (LIBARDI, 2010)

Equipe de Desenvolvimento

A Equipe de Desenvolvimento é auto-organizável, escolhe qual a melhor forma para completar seu trabalho, em vez de ser dirigido por outros de fora do time. Também é multifuncional, possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe. (SCHWABER, 2015)

Scrum Master

O *Scrum Master*(SM) tem o papel de remover os impedimentos para que uma iteração não perca seu prazo e nem se atrase, conforme o *TimeBox*. Retirando esses empecilhos, o trabalho da equipe é facilitado, e assim, melhora a produtividade da equipe. Outra função do SM é ensinar ao *Product Owner* a maximizar o retorno do investimento(*return of investment* ROI) e cumprir os objetivos por meio do Scrum. O SM é responsável ainda por manter as informações sobre o progresso da equipe visível a todos de maneira clara e organizada. De uma forma geral, sua finalidade principal é incentivar e facilitar a capacidade de tomada de decisão e resolução de problemas relacionados ao desenvolvimento, de modo que a equipe e o PO possam trabalhar com maior eficiência sem a necessidade de supervisão, isto é, ter uma equipe auto-organizável.

Product Owner

Product Owner é o dono do produto. Ele trabalha em conjunto com a equipe especificando as necessidades dos usuários, os requisitos técnicos e define a sequência da sua execução. Ele é o responsável por gerenciar o *Product Backlog* (repositório de todas as informações) cuidando do nível de detalhe e da qualidade que a equipe precisa. O PO é o único responsável por manter atualizado esse repositório com os requisitos e correções/adaptações que porventura venham a acontecer. É também o encarregado de esclarecer as dúvidas da equipe sobre os requisitos do produto. Em suma, o PO é a interface entre a empresa e os clientes. (NASCIMENTO CSM, 2013)

2.2.3.2 Releases

As *releases* são partes essenciais para o Scrum. *Release* é a entrega de um ou mais incremento do produto pronto, gerados pela equipe de desenvolvimento em uma ou mais *Sprints* sucessivas. (SCHWABER, 2015) Cada *Sprint* gera um incremento que possui valor suficiente para ser utilizado. O PO necessita saber qual a melhor estratégia para o desenvolvimento e para a entrega do produto. Essa estratégia vai permitir ao PO tomar decisões como: em que data sairão as primeiras versões, que funcionalidades essas versões devem conter.

Ao invés de realizar a entrega de um produto "perfeito", o mais importante é realizar pequenas entregas, com funcionalidades relevantes em curtos períodos de tempo, para que assim se tenha um *feedback* [7]. Esse ponto é importante no desenvolvimento ágil, visto que a análise feita pelos usuários pode guiar os desenvolvedores a um eficiente produto. O *feedback* é uma prioridade. Desse modo, tenta-se reduzir ao máximo o tempo de cada *release*. Uma tática muito utilizada é encontrar o menor conjunto possível de funcionalidades, e assim, buscar uma versão do produto que seja a menor possível para essa entrega. A estratégia de entrega depende de cada produto e da visão de negócio. Ao possuir esse método de desenvolvimento definido, com as entregas previstas e as metas de cada um, pode-se pensar mais detalhadamente na primeira entrega. O planejamento da entrega é conhecido como *release planning*. Trata-se de uma reunião de planejamento de alto nível que abrange a iteração de *Sprints* futuros.

A *release planning* é o momento de organizar as *sprints* para alcançar da melhor forma possível a meta definida para a entrega. Cada *release* contém *sprints* a serem executadas, as quantidades são definidas pelo PO e pelo equipe de desenvolvimento. As *sprints* são formadas por histórias de usuários.

[7] Tradução literal, dar resposta. Um trabalho é apresentado para os clientes com o intuito de avaliar o seu desempenho e espera-se um retorno positivo ou negativo

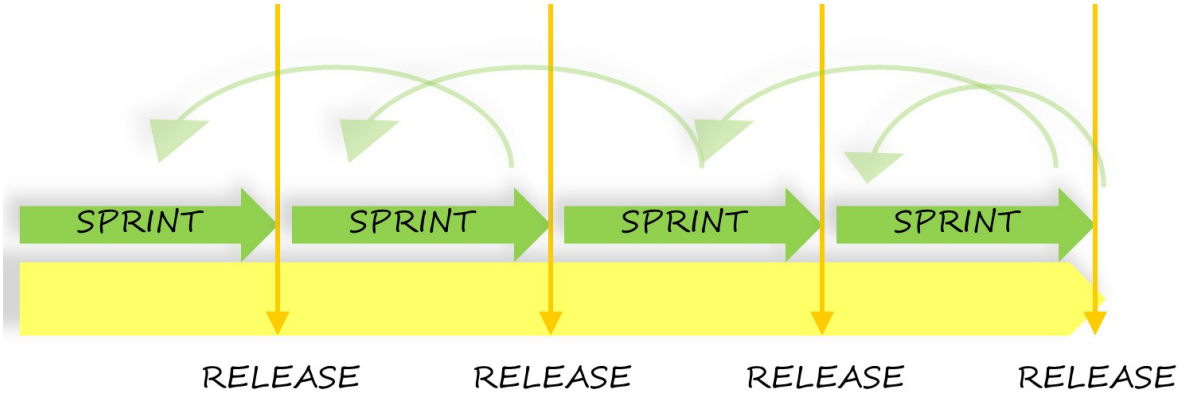

Figura 7 – Release

Fonte: Marmol (2015)

2.2.3.3 Product Backlog

O *Product Backlog* é uma lista que contém todas as funcionalidades desejadas para um produto. O conteúdo dessa lista é formada pelas histórias de usuários, definido pelo *Product Owner* (SCHWABER, 2001). A análise dos requisitos transforma-se em histórias. Estas são classificadas por ordem de prioridade; quanto mais ao topo do *backlog* estiver a história mais prioritária será, considerando a criticidade do alcance da meta. Essa priorização é responsável por guiar a ordem em que as histórias serão inseridas nas *sprints*.

A equipe então determina quais itens do *backlog* serão capazes de completar durante a Sprint que está por começar. Tais itens são transferidos do *Product Backlog* para o *Sprint Backlog*. A equipe divide cada item do *Product Backlog* em uma ou mais tarefas do *Sprint Backlog*. Isso ajuda a dividir o trabalho entre os membros da equipe. Podem fazer parte do *Product Backlog* tarefas técnicas ou atividades diretamente relacionadas às funcionalidades solicitadas.

2.2.3.4 Sprint Backlog

A *Sprint Backlog* é uma especificação do *Product Backlog*. São classificadas para esta *sprint* exclusivamente as histórias mais relevantes, ou seja, as histórias que possuem maior prioridade, aquelas que estão mais ao topo do *Product Backlog*. As histórias consideradas com maior criticidade são as que possuem maior nível de detalhamento. Esse aspecto facilita a decomposição das histórias em tarefas. De maneira geral, essas tarefas devem ser independentes, realizáveis e testáveis.

É importante destacar que o *Sprint Backlog* não deve ser mudado porque a

meta da *Sprint* não pode ser alterada. Incluir um novo item ao *Sprint Backlog* poderia ocasionar a não conclusão de um outro item. Por isso, no ciclo de desenvolvimento apenas itens novos que estiverem relacionados à meta podem entrar. Uma maneira eficiente de controlar esse ciclo é a utilização de um *Task Board*.O Task Board é uma lista de objetivos que devem ser completados em uma iteração, onde é possível gerenciar e visualizar o andamento dessas tarefas. Esse quadro é utilizado diariamente para manter a equipe de desenvolvimento ciente das tarefas realizadas por todos. Um exemplo de Task Board virtual é o *Trello*. Esta ferramenta auxilia na elaboração do quadro de tarefas. Por via de regra, esse quadro se divide, basicamente, em 4 colunas: histórias do usuário, a fazer, em processo e feito, podendo possuir mais colunas como na Figura.

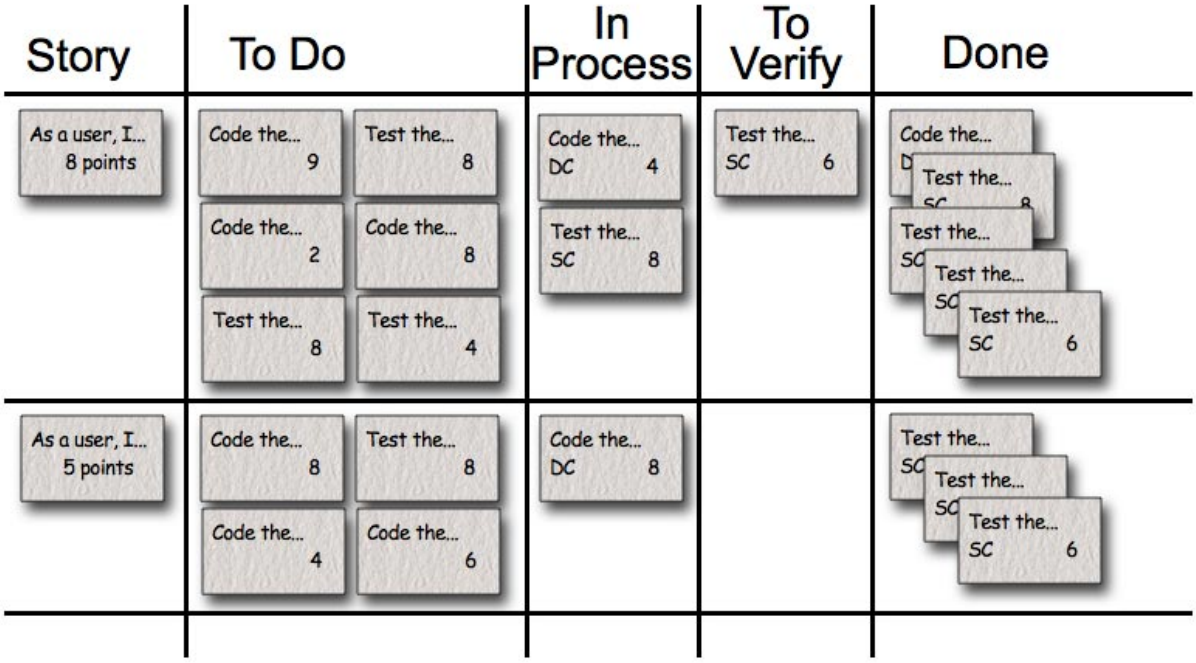

Figura 8 – Quadro de Tarefas

Fonte: Mountain Goat Software

2.2.3.5 Gráfico Burndown

Para se medir o progresso do trabalho em desenvolvimento, as histórias são medidas por pontos e as tarefas são medidas por horas. O Gráfico *Burndown* é uma forma de avaliar diariamente a evolução. Após cada dia de trabalho o gráfico apresenta a porção de trabalho finalizada em comparação com o trabalho total planejado. A análise do acompanhamento dos trabalhos é feito pelo gráfico da seguinte maneira:

- Eixo horizontal são os dias da *Sprint*;
- Eixo vertical são as horas(tarefas), ou pontos (histórias);
- Linha vermelha é o tempo estimado para a conclusão da *Sprint*;
- Linha azul é o andamento diário da equipe;

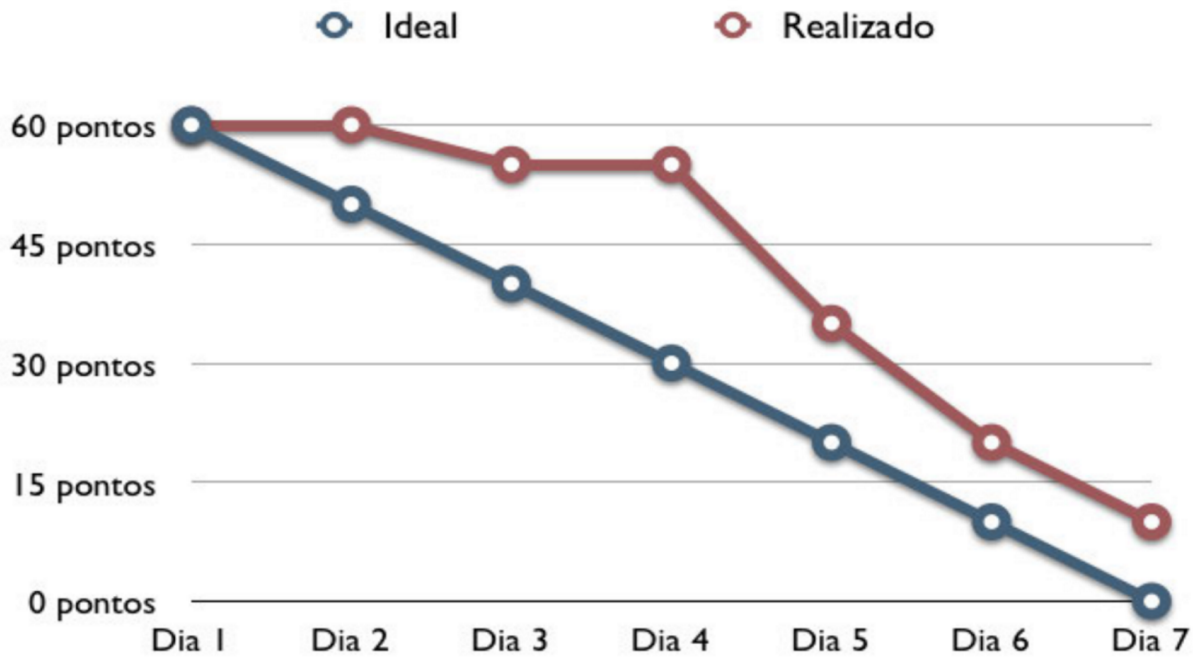

Figura 9 – Gráfico Burndown

Fonte: Gomes (2010)

A linha azul é traçada no início, determinando o andamento ideal da Sprint, enquanto que a linha vermelha é traçada dia a dia e representa o andamento real da Sprint. A linha vermelha estando acima da linha vermelha, significa um possível atraso em relação à estimativa. Abaixo da linha azul significa uma possível antecipação em relação à estimativa. O gráfico acima representa o final da Sprint dentro da estimativa inicial. Dessa forma, tem-se, resumidamente, o andamento da equipe de desenvolvimento.

Realizados em intervalos regulares, o Scrum possui eventos de duração fixa. Cada um desses eventos é uma oportunidade para inspeção e adaptação (PRIKLADNICKI R.; WILLI, 2014).

A Sprint é uma das partes fundamentais do Scrum, um time-boxed de um mês ou menos, no qual um incremento potencialmente utilizável do produto, é criado. Ela é composta por uma Reunião de Planejamento da Sprint, Reuniões Diárias, uma Revisão da Sprint e a Retrospectiva da Sprint (SCHWABER, 2015).

A Reunião de Planejamento da Sprint é feita no início de cada Sprint, o Time Scrum se reúne para planejar o que será feito naquela Sprint. Uma vez selecionados os itens, o Time Scrum define uma meta da Sprint. Essa meta serve como um guia sobre o que será desenvolvido durante a Sprint e representa o compromisso firmado entre o Time de Desenvolvimento e Dono do Produto (PRIKLADNICKI R.; WILLI, 2014).

As Reuniões Diárias são curtas tipicamente com duração de 15 minutos, realizadas diariamente pelo Time Scrum. Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária, e prever o trabalho que deverá ser feito antes da próxima Reunião Diária (SCHWABER, 2015). Onde devem ser respondidas 3(três) perguntas por todos os integrantes da equipe.

1. O que você executou desde a última reunião do time?
2. O que você fará em seguida?
3. Existe algo o impedindo de progredir?

Ao final de cada Sprint é realizada a Revisão da Sprint. O Time Scrum e as partes interessadas colaboram sobre o que foi feito na Sprint, como objetivo de inspecionar o que o Time de Desenvolvimento produziu e colher opiniões e impressões dos presentes para, caso seja necessário, adaptar o plano para a Sprint seguinte com o foco principal no aprimoramento do produto (PRIKLADNICKI R.; WILLI, 2014)

O último evento realizado no Scrum é a Retrospectiva da Sprint. Realizada logo em seguida a Revisão da Sprint. Participam dessa reunião todos os membros do Time Scrum, e seu foco é a interação entre os membros do Time de Desenvolvimento, as práticas e ferramentas utilizadas, o que funcionou e o que precisa ser melhorado na próxima Sprint (PRIKLADNICKI R.; WILLI, 2014).

Após descrito detalhadamente cada metodologia, é importante ter uma visão geral das metodologias descritas anteriormente, É mostrado na tabela 2 algumas de suas principais características, onde serão comparadas no Capítulo 3 na subseção 3.1.1 com os estudos de casos deste trabalho. Entretanto, como as metodologias XP e Scrum são processos ágeis, elas possuem características similares. Entregas rápidas, reuniões frequentes, pouca documentação, colaboração com cliente, resposta a mudanças, são algumas características ágeis em comum.

RUP	XP	SCRUM
focado nos processos	padrões de codificação	escalável
baseado em componentes	programação em pares	papéis bem definidos
utilização de UML[8]	interação com o usuário final	garantia de funcionalidade
processos formais	refatoração do código	remove obstáculos
artefatos padronizados	simplicidade de código	alta comunicação

Tabela 2 – Principais características das metodologias

2.3 Trabalhos Relacionados

Realizada uma abordagem dos conceitos dos tipos de metodologias, passando por seus papéis, eventos, artefatos e regras, o entendimento deste estudo se torna mais compreensível. Por tanto, nesta parte do Capítulo 2, será apresentado trabalhos que estão relacionados com a ideia desta pesquisa.

Para localizar estes trabalhos correlatos, foram realizadas buscas com o seguinte tópico: Metodologias de desenvolvimento de software em Universidades. Para estas buscas utilizou-se as bases ACM DL Library , o IEEE Explorer, Google Scholar e no Peródicos CAPES.

Foi observado que diversos trabalhos se preocupam com a análise de metodologias de desenvolvimento de projeto para o setor privado. Pois esse setor já possuem um escopo bem definido, requisitos bem definidos como os trabalhos Melo e Ferreira (2011), Rocha et al. (2017), Corrêa (2016), Andrade et al. (2014), MALLMANN (2011). No entanto foram selecionados três trabalhos voltados para ambientes acadêmicos como trabalhos correlatos.

O primeiro trabalho relacionado realizou uma pesquisa sobre a análise dos desafios envolvidos no desenvolvimento de software *inovador* em universidades públicas brasileiras. O trabalho foi realizado ao longo de 2(dois) anos, para a aplicação de um processo ágil de desenvolvimento de software, que foi customizado e implantado em 3(três) laboratórios.

Segundo o autor Pereira (2014) a pesquisa foi dividida em duas etapas. Na primeira, o objetivo foi customizar um processo de desenvolvimento de software capaz de tornar produtivo um laboratório acadêmico cujos colaboradores são, na sua maioria, estudantes de graduação. Para isto, um laboratório voltado à pesquisa e à inovação em computação foi selecionado e uma primeira versão do processo de desenvolvimento de software foi implantada em seu ambiente. Então, o ciclo PDCA [9] utilizado para evoluir o processo considerando critérios de desempenho e qualidade.

[9] PDCA (do inglês: PLAN - DO - CHECK - ACT ou Adjust) é um método iterativo de gestão de quatro passos, utilizado para o controle e melhoria contínua de processos e produtos.

Após a seleção de alguns projetos de software para o estudo de caso, eles foram monitorados por meio de métricas selecionadas a partir da literatura. A cada liberação de versões desses projetos, as métricas coletadas foram utilizadas para identificação de falhas e de oportunidades de melhoria no processo. No decorrer do estudo, o processo foi alterado inúmeras vezes para melhorar essas métricas.

Na segunda etapa, o objetivo foi implantar o processo customizado em outros laboratórios e avaliar seu impacto. Para isto, as métricas de processo já selecionadas foram utilizadas para monitorar as atividades de desenvolvimento desses laboratórios antes, durante e depois implantação do processo de desenvolvimento customizado. Desta maneira, uma linha de base (*baseline*) foi construída para permitir análises comparativas entre o "antes"e o "depois". Finalmente, as equipes de todos os laboratórios responderam a questionários que avaliam a percepção acerca do processo de desenvolvimento de software implantado e acerca do ambiente de trabalho (C. SIM, 2005).

O Laboratório para Modelagem e Simulação do Sistema Terrestre - TerraLAB, do Departamento de Computação (DECOM), da Universidade Federal de Ouro Preto (UFOP), foi selecionado como ambiente propício ao desenvolvimento da primeira etapa desta pesquisa. Justificam esta escolha o fato dele possuir, desde 2006, projetos de software de longo prazo em andamento (cinco anos) e possuir produtos utilizados por usuários espalhados pelo mundo e por diversas instituições públicas brasileiras - o simulador TerraME (PEREIRA; CARNEIRO; PEREIRA, 2013). Os laboratórios HPC LAB e Imobilis, do DECOM/UFOP, foram selecionados para o desenvolvimento da segunda etapa desta pesquisa. O interesse dos professores coordenadores destes laboratórios e a maturidade das inovações desenvolvidas por seus times justificaram esta escolha.

Segundo Pereira (2014) atualmente, o TerraLAB conta com 22(vinte e dois) colaboradores, 1 bolsista de desenvolvimento tecnológico e inovação (CNPq), 4(quaro) pós-graduandos e 17(dezessete) estudantes de graduação. É acordado que os estudantes de graduação dediquem 20(vinte) horas semanais e os de pós-graduação 40(quarenta) horas.

Um processo de desenvolvimento de software chamado BOPE, ao longo de 3(três) anos, evoluiu cíclica e incrementalmente para assegurar que equipes formadas por estudantes de graduação pudessem ser bem avaliadas segundo as métricas de processos selecionadas. As próprias métricas fazem parte do processo e, desta forma, também evoluíram ao longo do tempo. A cada falha percebida no processo ou nas métricas, foram realizadas modificações para assegurar que o processo incluiria uma métrica capaz de capturar a falha. Além disto, os papéis, as atividades e os artefatos do processo eram modificados de forma a evitar ou impedir a ocorrência das falhas. Na ausência de falhas, os principais valores norteadores desta evolução foram

a boa qualidade dos produtos (ausência de bugs), o respeito aos prazos e aos custos estimados, a resiliência dos projetos à rotatividade das equipes e a pouca capacitação dos recursos humanos. Por isto, os papéis deveriam ser muito claros e fáceis de serem aprendidos por imitação, como uma atividade cotidiana cuja técnica pudesse ser ensinada de um colaborador para outro (PEREIRA, 2014).

De acordo com Pereira (2014), no BOPE, os projetos têm no mínimo a seguinte estrutura e informações:

- **Escopo do projeto:** Ele define o contexto em que o projeto está envolvido. Define a missão do projeto e os objetivos específicos. Descreve os benefícios pretendidos pelo cliente. Para gerenciar as expectativas do cliente, ele descreve aquilo que o software fará e suas limitações (escopo negativo).
- **Backlog do Projeto(entregáveis do projeto):** O product backlog é uma lista dos principais produtos esperados do projeto, por exemplo, software, modelos computacionais, cursos, monografias, publicações, etc. É ideal que ele contenha o prazo e custo que cliente estima para cada produto. Estas informações são essenciais para se construir o cronograma inicial do projeto e preparar uma proposta técnica e comercial. Quantas estórias de usuários podem ser feitas? Qual a complexidade que elas podem ter? Em qual período? Por qual valor? Quais os recursos serão necessários? O product backlog é aberto e pode ser detalhado e mudado ao longo do projeto para satisfazer as expectativas do cliente.
- **Estórias de Usuários:** As principais estórias de usuários (ou requisitos) são identificadas e priorizadas. Estórias de usuários são descrições leves de como o produto deve ser usado e como ele deve se comportar. Ao longo do projeto, elas devem ser detalhadas e tornarem-se específicas, mensuráveis, alcançáveis, relevantes e contidas (M., 2003). Por específicas e mensuráveis significa que as estórias de usuários devem ser testáveis e que os pares (entrada e saída esperada) devem ser conhecidos. Por alcançáveis significa que se os desenvolvedores não puderem entregar uma estória de usuário em uma iteração, elas devem ser decompostas em várias estórias de usuários. Por contidas entende-se que é importante saber quando desistir de uma estória de usuário quando o prazo estimado para sua conclusão é extrapolado diversas vezes. Por relevantes entende-se que as estórias de usuários fornecerem o máximo de valor possível para o cliente.
- **Cronograma do Projeto:** O cronograma do projeto é, em geral, estimado para o período total do projeto. No entanto, ao longo do projeto, o cronograma pode ser refinado para cada sprint. O cronograma pode ser detalhado em estrutura de

árvores. O nó raiz representa todo o projeto. O segundo nível de nós representa os produtos do projeto. Os próximos níveis de nós representam subprodutos. Os nós folha representam as estórias de usuários (ou requisitos) que devem ser implementados a fim de realizar as entregas do projeto. Considerando os recursos disponíveis, é possível estimar o tempo total necessário para implementar as estórias de usuários.

Durante os estudos de caso, algumas ferramentas para a automação dos processos e para coleta de métricas foram utilizadas no laboratório TerraLAB. O BOPE [10] foi projetado para ser independente de ferramentas específicas. Os laboratórios que participaram deste trabalho tiveram liberdade de escolher as ferramentas que lhes interessavam. Por exemplo, o HPC LAB utilizou o software Asana 8 para o controle de mudanças e o NetProject para coleta de horas. O laboratório iMobilis optou utilizar o software GroupCamp 9 para apoiar todo o processo.

No TerraLAB, a implantação de um processo de desenvolvimento de software teve início em 2010. No ano de 2011, o processo implantado era muito similar ao RUP, muito burocrático e contraproducente. Ao invés de concentrar em questões técnicas e de pesquisa envolvidas nos projetos de software, uma grande parte do esforço do coordenador do laboratório era dedicada ao treinamento dos estudantes no uso do processo e na construção dos artefatos. O processo de desenvolvimento não previa *sprints* e o cronograma anual era estimado no início do ano e então seguido.

Ao longo do tempo, o processo de desenvolvimento gradativamente incorporou práticas ágeis, combinando as disciplinas e papéis identificados no RUP, os ciclos curtos, os artefatos e reuniões do Scrum, os processos de teste do XP e a especificação de estória de usuário e de cenários de teste conforme o BDD (*Behavior Driven Development*).

A dedicação dos envolvidos no projeto foi um ponto de grande desafio. Pois como mostrado na Figura 10 é notável que os estudantes de graduação são recursos desafiadores para gestão dos projetos e que, o comportamento dos graduandos é diferente dos pós-graduandos.

Como a equipe do TerraLAB é majoritariamente formada por graduandos, cerca de 80%, a baixa dedicação deles acaba por reduzir drasticamente a dedicação média da equipe. Enquanto os alunos de pós-graduação retomam rapidamente o ritmo de trabalho após as férias (janeiro e setembro), os estudantes de graduação o fazem muito lentamente. Diferentes dos pós-graduandos, os graduandos desaceleram o ritmo de trabalho antes das provas escolares (abril e setembro) e demoram a retomar um bom ritmo. A dedicação média dos estudantes de graduação ao longo de 2013 foi

[10] Processo personalizado utilizado no desenvolvimento dos softwares deste trabalho relacionado

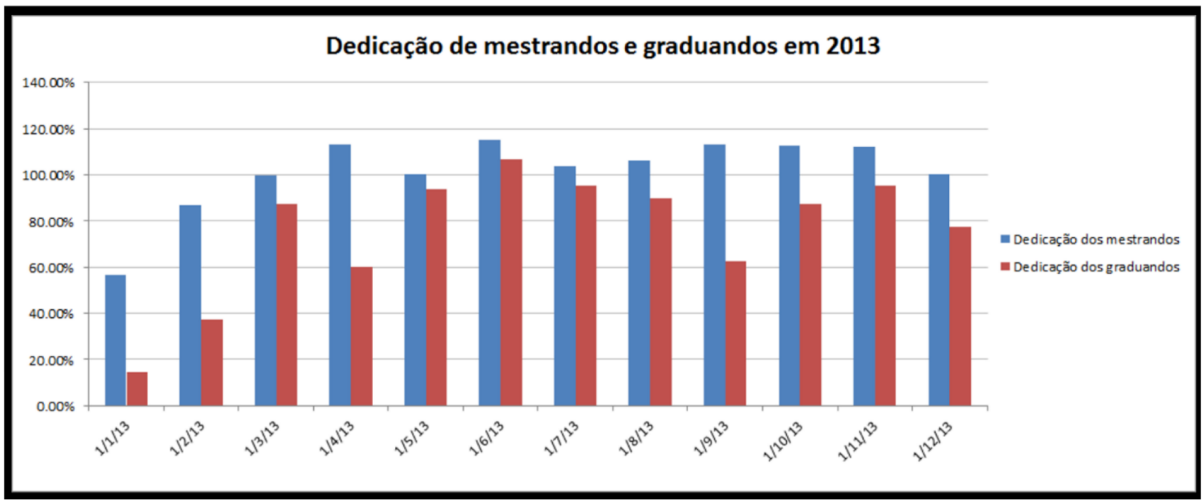

Figura 10 – Dedicação de mestrandos e graduandos em 2013

Fonte: Pereira (2014)

de 63,32% com desvio padrão de 27,63%, resultando em um coeficiente de variação de 43,80%. Enquanto que, a dedicação média dos estudantes de pós-graduação foi de 92,08%, com desvio padrão de 16,63%, resultando em um coeficiente de variação de 18,05%.

Outro ponto de bastante relevância é com relação ao número de falhas, pois o número de falhas de software detectadas em cada versão dos produtos é a principal métrica de qualidade utilizada para evoluir o processo BOPE. Para cada *sprint* ou cronograma, a Figura 11 apresenta o percentual de falhas encontradas por estórias de usuários realizadas. É notória a melhoria da qualidade dos produtos realizados ao longo do tempo. Esta melhoria pode ser justificada pela adoção de testes regressivos automatizados a partir de setembro de 2013 e pelo fato das equipes reduzirem a granularidade das estórias de usuário (PEREIRA, 2014).

Para validar o BOPE como um processo replicável em ambientes correlatos ao do TerraLAB, os laboratórios HPC Lab e Imobilis do DECOM/UFOP [11] foram utilizados como estudos de caso da implantação do processo BOPE. Antes da implantação do BOPE, estes laboratórios de pesquisa e inovação em computação não seguiam um processo de desenvolvimento bem definido ou metodologia de gestão de projetos.

O mesmo questionário utilizado no TerraLAB foi usado nestes dois laboratórios

[11] DECOM - Departamento de Computação / UFOP - Universidade Federal de Ouro Preto

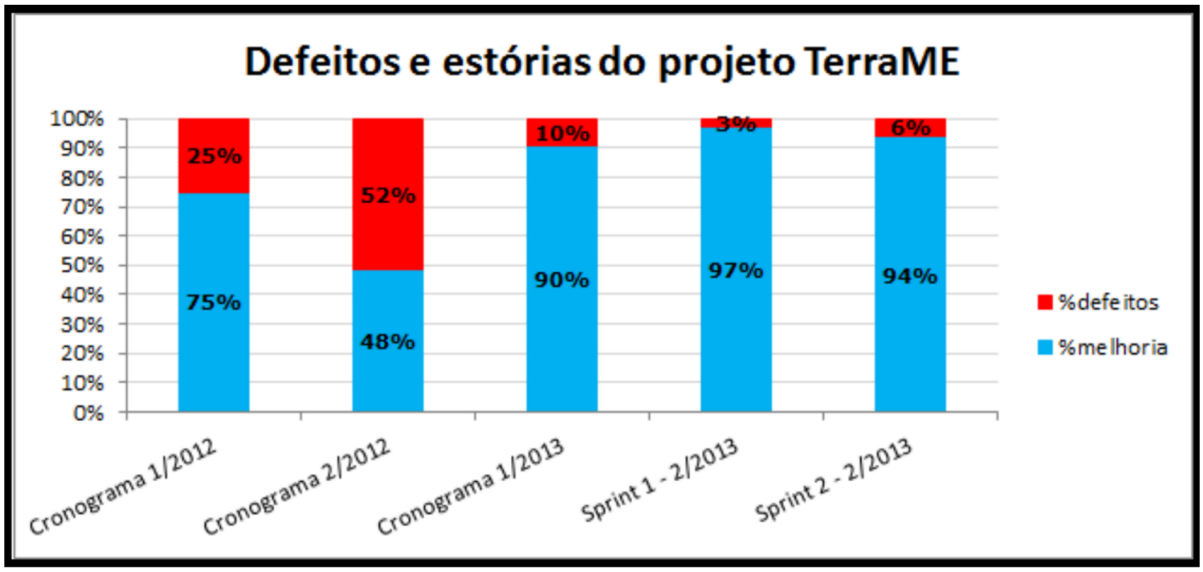

Figura 11 – Percentual de falhas por estórias de usuário no TerraME em 2012 e 2013

Fonte: Pereira (2014)

afim de conhecer de uma maneira geral os sentimentos dos membros de suas equipes e realizar ações para tentar modificar algum conceito ou percepção equivocados.

Precisamente nos laboratórios HPC Lab e Imobilis está a principal relação com o trabalho proposto. Os trabalhos se assemelham pela pouca quantidade de alunos envolvidos nos projetos. Imobilis, é formado 10 colaboradores, sendo dois pós-graduandos e oito estudantes de graduação. A dedicação esperada foi de 40 horas para pós-graduandos e 12 ou 20 horas para estudantes de graduação. O laboratório HPC LAB conta com 19 colaboradores, sendo todos estudantes de graduação. A dedicação esperada é de 12 ou 20 horas para os estudantes de graduação, dependendo do contrato.

O objetivo desse estudo foi avaliar que na utilização de estudantes de graduação para o desenvolvimento de software se faz necessária a customização de um processo à essa realidade. Para tanto Pereira (2014) faz implantação de um processo ágil denomidado BOPE, que mescla a utilização de eventos e artefatos do Scrum e XP com algumas orientações do PMBOK (INSTITUTE, 2013).

Em um segundo trabalho, Lisboa et al. (2008) destacam a importância do empenho dos especialistas para o sucesso do projeto. Também é descrio uma metodologia educacional para ensinar o nível mais alto do desenvolvimento de softwares reutilizáveis em cursos de pós-graduação, bem como os passos para o processo de

desenvolvimento baseado em linhas de produto de software. A deficiência para encontrar métricas para a análise de recursos e a dificuldade em definir a granularidade dos requisistos e casos de usos (LISBOA et al., 2008).

Goldman et al. (2004) mostram formas eficientes de instruir estudantes e profissionais, seguindo os princípios do processo XP, para desenvolver um software de alta qualidade. A customização, foi com o adicional de um líbero, que tem um novo e importante papel no processo. Ele é um desenvolvedor experiente que não tem a responsabilidade de desenvolver estórias de usuário. Sua tarefa é articular juntamente com outros desenvolvedores e ajudá-los na resolução das tarefas. Uma vez que os desenvolvedores e gerentes tiveram um contato real com um projeto bem executado em XP, as preocupações sobre as consequências e eficácia da metodologia aplicada se acabam consideravelmente.

3 PROPOSTA

Nas metodologias tradicionais, a análise de requisitos é uma preocupação contínua dos desenvolvedores de software, pois os problemas que acontecem na fase final do software são o resultado do mau uso de técnicas realizadas. Dado o avanço do processo de desenvolvimento torna assim a correção desses problemas mais complexa e um com custo financeiro elevado.

Por outro lado, o desenvolvimento ágil possui o envolvimento do cliente que tem um maior valor, sendo responsável por participar junto com o time de desenvolvimento, promovendo comunicação aumentando o nível do requisito e diminuindo as mudanças.

Nesse sentido, este trabalho apresenta uma análise das metodologias em 4 (quatro) estudos de casos no laboratório de redes e sistemas de Aracati (Lar-A). Estes estudos de caso são apresentados a seguir.

3.1 Estudos de Casos

3.1.0.1 SISAPP

Os centros especializados que oferecem tratamento para o câncer infanto-juvenil no Brasil, em sua maioria, não possuem uma solução computacional que colete, extraia e transforme dados em informação. Ainda há diversos centros que utilizam papel para registro dos dados coletados (e.g., prontuários de pacientes). Com a ajuda da informática o processo de diagnóstico pode ser bem mais ágil, no sentido em que ela permite um armazenamento eficaz de enormes quantidades de dados e pode oferecer informações precisas para a tomada de decisão de profissionais e gestores de saúde (KIM; GROENEVELD, 2017). Então, percebendo a importância do diagnóstico precoce na cura do câncer infanto-juvenil, como também a carência de sistemas computacionais especializados nesta área, o presente artigo tem o objetivo de apresentar um sistema Web desenvolvido para apoiar profissionais de saúde e gestores no processo de tomada de decisão, visando, principalmente, elevar o índice de cura e melhorar a qualidade de vida de crianças e adolescentes com câncer e de suas famílias (OLIVEIRA et al., 2017).

Desde 2016, o sistema está implantado na Associação de Combate ao Câncer Infanto-Juvenil, conhecida como Associação Peter Pan (APP) (PAN, 2017), que está localizada em Fortaleza-CE. A APP é um centro de excelência no combate à doença

no Ceará, em parte do Nordeste e em todo o Norte do país, tendo como missão elevar os índices de cura e a qualidade de vida das crianças e adolescentes portadores de câncer. Atualmente, atende cerca de 2.300 crianças e adolescentes. Os gestores e profissionais de saúde da APP enfrentavam inúmeros desafios no processo de tomada de decisão do hospital. Destaca-se que eles necessitavam de diversos tipos de informação sobre a APP para que pudessem visualizar de maneira rápida, otimizada e precisa as vantagens, os riscos e as prioridades em suas ações. Atualmente, através do sistema, denominado Sistema APP (SISAPP), os profissionais de saúde e os gestores da APP podem realizar o acompanhamento de múltiplos indicadores importantes para a tomada de decisão (OLIVEIRA et al., 2017).

3.1.0.2 DENGOSA

O Dengosa é um sistema de gestão e informação geográfica que gera indicadores socioeconômicos e ambientais, disponibilizando serviços de apoio à decisão para o controle de epidemias. A proposta possui uma nova metodologia que facilita e otimiza a comunicação entre os agentes de endemias, entre a população e os gestores do município, atores relevantes para o controle da epidemia de dengue. O sistema fornece informações e funcionalidades para que os profissionais e os gestores de saúde de um município acompanhem a proliferação de doenças em uma escala espaço-temporal.

A complexidade no tratamento das notificações de saúde no Brasil e a ausência de um controle satisfatório para a qualidade dos dados no nível municipal resulta na inconfiabilidade dos dados cadastrados no SINAN. Além disso a burocracia no processo de coleta de dados feita no papel à nível municipal, não é suficiente para resolver esse problema. Com o objetivo de mitigar essa problemática, o DENGOSA propõem uma solução que auxilia a coleta de dados para o SINAN, oferecendo ferramentas no apoio à decisão para os gestores municipais.

Assim, tem-se um novo cenário onde o DENGOSA complementa no nível municipal as ações do SINAN, que conduzem ações de caráter estratégico e complementar no nível nacional e estadual. O SINAN é um sistema oficial brasileiro. A metodologia do sistema proposto otimiza o apoio à decisão por meio da informatização do processo de notificação compulsória das doenças, a dengue por exemplo, no nível municipal e da integração dos dados do Levantamento de Índices Rápidos dos focos (LIRAa). Com essas ferramentas o sistema proposto pode atingir os seguintes objetivos.

- Detectar precocemente os casos para promover o tratamento adequado e oportuno e evitar o óbito do paciente;

- Detectar precocemente o aumento de ocorrência da doença para adoção de medidas de controle;
- Realizar a investigação da epidemia para identificar a área de transmissão e orientar ações controle epidêmico;
- Acompanhar a curva epidêmica, identificando a área de maior ocorrência de casos;
- Realizar investigação de óbitos suspeitos, visando identificar possíveis as possíveis causas.

3.1.0.3 PrESP

Uma Proposta baseada em informações Espaço-temporais para identificar Similaridade de interesses entre usuários em redes de Próxima geração. Esta proposta visa criar uma camada de serviços com o objetivo de gerar informações enriquecidas a partir de informações provenientes de diferentes fontes de dados. Consequentemente, o PrESP proverá meios para auxiliar a execução de sistemas de recomendação, bem como o desenvolvimento de um grande número de aplicações móveis e sensíveis ao contexto em redes de próxima geração.

Uma vez que é observada a necessidade de aumentar a inteligência e acurácia das soluções oferecidas, este projeto propõe uma camada de serviços para prover informações enriquecidas para aplicações em cenários de redes de próxima geração, tendo como principal parâmetro a rotina diária de usuários em centros urbanos. A ideia chave é oferecer uma camada capaz de capturar, armazenar e processar e identificar similaridades entre rotinas diárias de usuários móveis. Em primeiro lugar, serão utilizados smartphones e seus sensores para capturar rotinas diárias dos usuários e do contexto informação. Em segundo lugar, toda a informação será transferida e armazenada em um banco de dados relacional, localizado em um servidor. Finalmente, um algoritmo de identificação de similaridade é executado para identificar interesses semelhantes entre dois ou mais usuários.

O contexto deste projeto de pesquisa é voltado para a utilização de informações geradas por dispositivos móveis e seus sensores (ex.: GPS, acelerômetro, etc.) bem como informações existentes na rede (ex.: topologia lógica de uma rede sem fio, conteúdo de redes sociais, etc.) para oferecer serviços em cenários de redes de próxima geração.

3.1.0.4 SMARTBIKE

Os avanços tecnológicos aplicados ao contexto das Cidades Inteligentes resultam em aplicações para a melhorar a segurança, infraestrutura e eficiência dos ambientes urbanos. De acordo com o WorldWatch Institute, em 1965, a produção mundial de carros e motos era essencialmente a mesma, com aproximadamente 20 milhões de unidades cada. No entanto, nos últimos anos, a produção de bicicletas subiu para mais de 100 milhões de unidades por ano em comparação com 42 milhões de unidades de carros. No ano passado, a produção de bicicletas foi de 127 milhões de unidades em todo o mundo, um aumento de 18% em relação a 2004. Esta tendência de consumo é socialmente desejável, pois tem um impacto positivo na saúde da população e no tráfego urbano (D.; K., 2011).

O crescimento do número de bicicletas, consequentemente, levou ao aumento de rotas exclusivas para eles. No entanto, não garante a segurança total de seus usuários, seja pela falta de atenção dos motoristas, seja por buracos causados por degradação natural ou ações humanas. Assim, é útil ter maneiras colaborativas de informar os usuários sobre possíveis mudanças em suas rotas diárias ou permitir o uso de novas rotas para o mesmo destino. Neste sentido, as bicicletas devem evoluir e se adaptar a esta nova realidade (VIANA et al., 2017).

O objetivo deste trabalho é apresentar uma abordagem de visualização e análise de dados de ciclistas obtidos através de sensores acoplados a uma bicicleta. Além disso, dois exemplos de ferramentas são apresentados: o primeiro usa dados de um acelerômetro conectado a um Arduino [1] para identificar a superfície onde o ciclista está realizando o passeio e o segundo usa dados obtidos por um sensor de distância ultrassônica para sinalizar em uma aplicação Web geográfica Locais com alto risco de acidente (VIANA et al., 2017).

3.1.1 Metodologia de análise

O custo, o esforço envolvido na implementação, a falta de conhecimento sobre práticas de gestão, e o foco em atividades de pesquisa e inovação, são razões que dificultam a utilização de algumas metodologias em laboratórios de pesquisas de Universidades e Instituições. No entanto, é mais comum verificar a adoção de processos definidos para o desenvolvimento de software, como RUP, XP e Scrum.

A baixa utilização das metodologias tradicionais, em especial o RUP dentro do contexto analisado do estudo de caso deste trabalho, se dá pelo desafio de adotar

[1] É uma plataforma eletrônica de código aberto baseada em hardware e software fáceis de usar. É destinado a qualquer pessoa que faça projetos interativos.

uma pesada carga de burocracia, obrigando os estudantes a seguirem rigorosamente a planos definidos em um primeiro momento. Desta forma, os projetos evoluem em fases não sobrepostas, seguindo processos formais e usando artefatos padronizados, que os estudantes têm dificuldade de entender e utilizar.

Outro problema da utilização desse processo tradicional é devido à alta rotatividade das equipes envolvidas nos projetos do laboratório, fazendo com que muito tempo seja consumido na implantação do RUP. Os requisitos de produtos inovadores ou de pesquisa estão propensos a mudar com alguma constância. Este fato desafia o RUP que busca evitar tais mudanças para não comprometer o cronograma e o custo estimados para um projeto. Ou seja, no RUP, mudanças de escopo quase sempre resultam em negociação de prazos e custos (KRUCHTEN, 2003). Em paralelo, o grande volume de artefatos (documentos, diagramas, etc) gerados durante o processo também é visto como um problema. Em geral, eles devem ser revistos pelo líder do laboratório e/ou pelo professor pesquisador responsável, que rapidamente irá se tornar o gargalo do processo. Por estas razões, o RUP pode não ser bem replicado em laboratórios de pesquisa em computação.

Em contrapartida, os métodos ágeis favorecem a interação entre os colaboradores do que o uso de processos formais e artefatos padronizados. Eles voltam-se para respostas rápidas às mudanças mais do apenas que seguir um plano. Os métodos ágeis, também são baseados na colaboração do cliente sobre a negociação de contratos, objetivando mais o funcionamento do software do que uma documentação abrangente (BECK et al., 2001). Os métodos ágeis são leves e fáceis de implementar. No entanto, podem haver alguns problemas com sua aplicação direta em laboratórios acadêmicos.

Nas metodologias ágeis, o valor dos produtos está diretamente ligado à habilidade e experiência dos envolvidos. Contudo, nem todos os estudantes são totalmente comprometidos com o sucesso do projeto e não são profissionais aplicados. Os estudantes da graduação estão em processo de formação e têm as atividades acadêmicas como prioridade. Por estas razões, o baixo nível de burocracia dos processos ágeis pode comprometer o cronograma do projeto e a qualidade do produto. Além disto, o líder do laboratório, que muitas vezes atua como cliente do projeto, não está frequentemente disponível para interagir com a equipe de desenvolvimento. Se existem muitos projetos em execução, ele vai se tornar o gargalo do processo. Mais uma vez, o processo poderá não escalar bem.

Apesar destas duas abordagens, tradicionais e ágeis, existirem a décadas no Brasil, são poucas as iniciativas de adoção de processos de desenvolvimento de software bem definidos no contexto de laboratórios acadêmicos.

Sabendo disso, o Laboratório de Redes do Aracati (LAR), um laboratório de

pesquisa e desenvolvimento em computação na cidade de Aracati – Ceará, foi selecionado para ser o estudo de caso deste trabalho. Um questionário foi enviado para os projetos de pesquisa desenvolvidos em seu ambiente, sendo este o método de coleta de dados. O questionário possui 19 questões, mas há algumas delas serão analisadas de forma qualitativa, sendo este um trabalho exploratório em um ambiente acadêmico. O questionário foi respondido por 16 (dezesseis) pessoas, dentre elas, 2 (dois) professores pesquisadores e 14 alunos de graduação em Ciência da Computação. O questionário foi respondido individualmente, entretanto 7 (sete) pessoas do projeto PrESP, 3(três) para o SISAPP, 3 (três) para o DENGOSA e 3 (três) para SMARTBIKE.

Questionário	Respostas
Respostas	16
Projetos de Pesquisa	4
Semestre de ingresso no projeto	maioria entre 3º, 4º e 5º
Horas dedicadas	20hs por semana
Tempo no projeto	maioria entre 12 e 24 meses

Tabela 3 – Tabela explicativa sobre o questionário

Os alunos que responderam o questionário dedicam por volta de 20(vinte) horas semanais, enquanto os professores dedicam aproximadamente 4 (quatro) horas semanais. Como mostrado na tabela 3, quatro alunos ingressaram no projeto de pesquisa no quarto semestre, 3 (três) entraram no quinto semestre, 2 (dois) no terceiro e no sexto semestre, os restante dos alunos entraram um no primeiro e outro no sétimo semestre. Com relação ao tempo que os alunos estão nos projetos é mostrado que, entre 12 meses e 23 meses temos um percentual de 37,6%, entre 24 meses e 35 meses temos 31% e mais de 36 meses temos 31,4%.

A metodologia de análise busca avaliar como os projetos de pesquisas atuam em relação ao emprego das metodologias e a dedicação dos alunos graduandos, realizando a análise de quais atividades são realizadas e as técnicas que são utilizadas em determinada metodologia.

Foi utilizada a ferramenta de pesquisa online gratuita Google Forms 1 [2]. Foram coletadas as respostas em um período de até no máximo 15 dias. Logo em seguida as respostas foram armezenadas para serem analisadas e apresentadas no capítulo seguinte.

[2] Ferramenta online Google Form: https://www.google.com/forms/about/

4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta o resultado da análise realizada no estudo de caso que avalia a maneira de como as metodologias de desenvolvimento de software e o comprometimento dos envolvidos em um projeto de pesquisa de um ambiente acadêmico.

Esse estudo não tem a intenção de proporcionar uma pesquisa extensiva de "implantação de metodologias para o ambiente acadêmico". A pesquisa empírica é desenvolvida apenas no Lar-A, o que sugere que os resultados aqui encontrados não podem ser generalizados.

Enquanto o professor têm as funções de coordenar o projeto, analisar requisitos, captar recursos financeiros, ele também tem o papel de gerente de projeto. Porém essa função não fica clara para 62,5% dos alunos envolvidos no projeto, como mostra a Figura 12. Os editais do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) não prevê a contratação de um especialista em gerência de projeto, nem a qualificação da equipe o que afeta diretamente no desempenho dos projetos. Assuntos discutidos mais adiante.

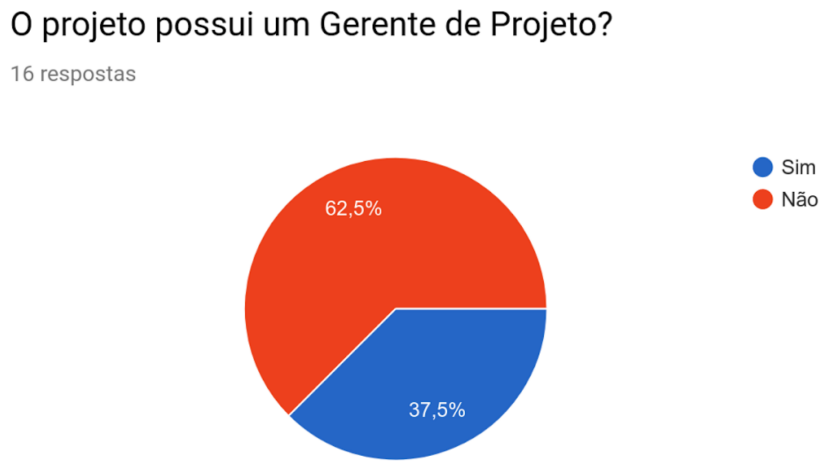

Figura 12 – Resposta questionário sobre Gerente de Projeto

De todos os projetos analisados apenas um (SISAPP) foi finalizado e concluído com sucesso em 100%, o restante dos projetos estão em andamento. O DENGOSA e o PrESP estão em torno de 75% da sua conclusão e o SMARTBIKE em 80%. Mesmo com a dificuldade de não aplicação rigorosa de metodologias de desenvolvimento de software para os projetos desenvolvidos pelo LAR, o único projeto concluído com sucesso foi entregue e implantado.

Dentre as metodologias apresentadas, o SISAPP fez o uso aproximado das metodologias ágeis, particularmente o SCRUM. Segundo uma entrevista pessoal com o professor, coordenador, gerente e responsável pelo projeto, o emprego dessa metodologia foi possível graças a utilização de um líder técnico, podendo ser comparado com o *Scrum Master*, já previsto na metodologia SCRUM, que ameniza a sobrecarga de responsabilidades.

Aplicar a Engenharia de Software para desenvolver um produto de qualidade, normalmente, exige conhecimento específico do domínio de problema. O envolvimento de especialistas destes domínios é essencial para o sucesso dos projetos de desenvolvimento de software. Os estudantes precisaram aprender e dominar conceitos, técnicas, métodos e ferramentas. Portanto a capacitação da equipe é fundamental para alcançar o sucesso. Quando o time está trabalhando adequadamente em conjunto, os resultados começam a surgir mais rapidamente, além de ter um desempenho superior.

Aproximadamente 50% dos alunos que responderam o questionário entraram nos projetos entre o terceiro e o quinto semestre. No curso de bacharelado de Ciências da Computação de Aracati, os alunos têm contato pela primeira vez com a disciplina de Engenharia de Software apenas no quinto semestre. Onde é possível ter uma visão geral sobre as metodologias, pois o conhecimento dos processos de desenvolvimento de software é importante na ajuda para a aplicação das metodologias de maneira eficiente. As respostas ficaram divididas em 50% entre os que conheciam as metodologias e os que não conheciam. A engenharia de software é um domínio altamente orientado ao conhecimento, no qual alguns fatores de sucesso estão relacionados com a experiência das pessoas envolvidas nas diversas fases e atividades do processo.

Por outro lado, houveram contrariedades acerca da metodologia que estaria sendo seguida. As opções sobre este questionamento foram abrangentes, porém direta, mesmo assim, os alunos de um mesmo projeto entraram em choque com a metodologia que supostamente estariam seguindo. Deve-se supor que para cada projeto apenas uma metodologia deve ser seguida. Entretanto em um mesmo projeto os alunos responderam que utilizavam as metodologias tradicionais e ágeis, mostrando assim dificuldade e ambiguidade na definição da metodologia. Apresentado na Figura 13.

Como dito anteriormente neste trabalho, estudantes de graduação são os recursos mais desafiadores para os projetos de desenvolvimento de software. Após uma entrevista com um dos professores doutores, responsável por um dos estudos de casos deste trabalho, foi constatado que os alunos durante o período de provas e entrega de trabalhos, a prioridade principal é o curso. Os estudantes são profissio-

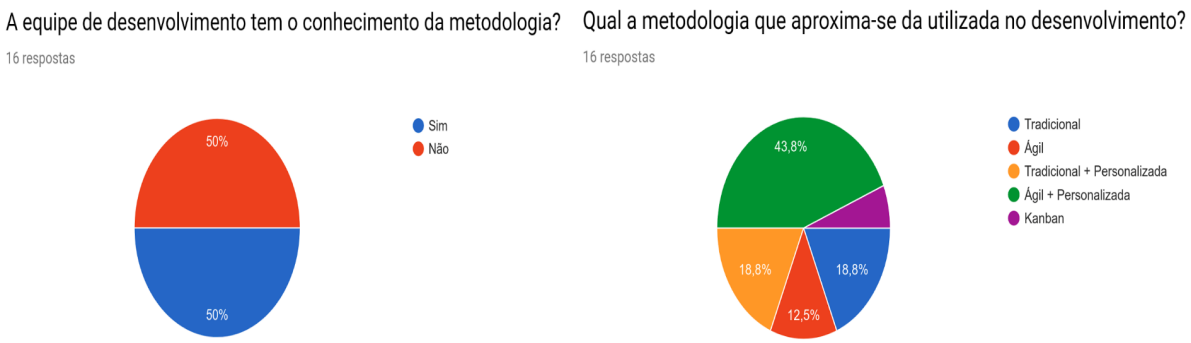

Figura 13 – Resposta questionário sobre conhecimento da metodologia e metodologia utilizada

nais em formação e podem ter diferentes níveis de conhecimento. Logo, não podem assumir todos os papéis no processo de desenvolvimento. A conclusão do curso desses alunos não depende do sucesso dos projetos. Diante disso, eles podem deixar a equipe em qualquer momento, sem causar grandes problemas nas suas carreiras.

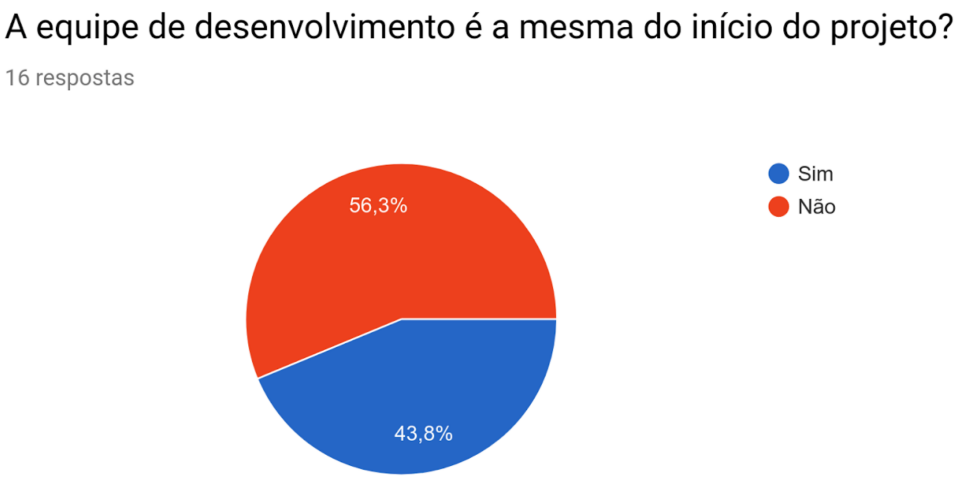

Figura 14 – Resposta questionário sobre Mudança na equipe de desenvolvimento

Em alguns momentos durante o desenvolvimento do projeto os alunos se sentiram desmotivados. A desmotivação ocorre devido a um conjunto de fatores que im-

pedem a realização de um ato. O acúmulo de tarefas, obrigações e as cobranças acabam gerando o desinteresse. A fim de recuperar a motivação, deve-se identificar os problemas e procurar meios para resolvê-los e superá-los. Os aspectos apontados pelos estudantes que lhes causam desmotivações são: falta de organização, atrasos do pagamento, "queda"de internet, complexidade do conteúdo do projeto, constante mudança de requisitos ao longo do projeto e falta de comunicação. Estes fatores geram cansaço físico e mental do aluno, ocasionando desgaste emocional e desmotivação na realização dos afazeres e na busca dos objetivos.

Através do questionário foram expostas diversas dificuldades enfrentadas pelas equipes. Existem alguns fatores que tornam difícil o processo de utilização das metodologias. Para que uma mudança organizacional ocorra de forma bem sucedida, é necessário ter apoio de todas as pessoas envolvidas no projeto. As pessoas que executarão as tarefas precisam colaborar para que elas realmente possam ocorrer. A seguir são feitos comentários sobre cada dificuldade apontada pelos alunos no questionário.

Dificuldades apontadas pelo questionário

- Requisitos não tiveram muito impacto no projeto.

  A análise de requisitos é vital para o desenvolvimento do sistema, ela vai determinar o sucesso ou o fracasso do projeto. Essas análises estabelecerão o acordo entre cliente e fornecedor sobre o que o software fará e consequentemente reduzirão os custos de desenvolvimento, pois requisitos mal definidos implicam num retrabalho. Existem técnicas de validação de requisitos, onde cliente e fornecedores, avaliam juntos se os requisitos solicitados realmente possuem valor para o desenvolvimento. Assim, fazendo com que os requisitos tenham impacto no projeto.

- Infraestrutura do Laboratório;

  O Laboratório de Redes de Aracati existe desde 2012. Hoje funciona com a participação de 4 (quatro) professores doutores, 3 (três) professores mestres, 2 alunos de pós-graduação e mais de 30 alunos graduandos. Todos dividindo o mesmo ambiente e máquinas disponíveis, que são insuficientes para o acesso de todos. Assim como a estrutura, os recursos e equipamentos também são limitados. Porém, o problema de captação de verba para os projetos é algo que não depende de nenhum professor ou diretor ou instituição, o problema é financeiro nacional.

- Cumprir atividades dentro do prazo;

O planejamento e a organização são as chaves para o cumprimento de prazos. Ter um maior controle de todas as suas responsabilidades é um ótimo caminho não só para ser pontual como também mais produtivo. Para tornar uma rotina mais leve e fácil de gerenciar, é de boa prática dividir as grandes tarefas em pequenas ações. Tarefas muito grandes costumam ser mais complicadas de resolver e, inconscientemente, essas tarefas acabam intimidando os responsáveis pela execução das mesmas. O uso de ferramentas tecnológicas facilitam a organização e execução das tarefas dentro do prazo.

- Conciliação entre os objetivos do projeto e as disciplinas do curso;

  Os estudantes de graduação são recursos desafiantes, capazes de colocar em risco o sucesso de projetos de pesquisa em computação. No entanto, também mostra que, por meio de processos de desenvolvimento bem definidos, os laboratórios formados majoritariamente por graduandos, são capazes de desenvolver e manter projetos de longo prazo. A falta de comprometimento dos estudantes com o sucesso dos projetos também foi uma grande dificuldade encontrada, pois os estudantes priorizam suas disciplinas do curso.

- Comunicação com clientes/stakeholders;

  A comunicação com os clientes foi um dos maiores desafios encontrados pelo projeto DENGOSA. Por se tratar de um sistema que envolve dados confidenciais de saúde no Brasil.

- Documentação do software;

  A documentação de software é necessária e auxilia na redução de horas na correção de problemas. A documentação é parte integrante de qualquer sistema ou programa criado. Sem a devida documentação, *bug's* e pontos vulneráveis no sistema demoram a ser encontrados e corrigidos, permitindo assim que os ataques continuem levando à falência múltipla do sistema e, consequentemente, de seu usuário.

- Capacitação da equipe;

  Muitas vezes a escolha de alunos que possuam preparo em determinada área ou de certa forma, tenham um conhecimento específico, pode ser uma tarefa árdua. À vista disso, investir no treinamento e desenvolvimento de habilidades dos graduandos é uma ótima opção. Oferecer capacitação pode trazer produtividade dentro dos projetos de pesquisas. O investimento em qualificação também está relacionado com a valorização dos envolvidos, que por sua vez, se sentem reconhecidos e motivados, tendendo a ter um melhor desempenho.

- Falta de profissionais qualificados;

Há muitas áreas de conhecimento e muitas habilidades necessárias para se construir um software. Muitas pessoas acham que software se resume a codificação, mas construir software é uma atividade complexa e exige dedicação. Nos editais dos financiadores de projetos de pesquisa, como CnPQ, não está previsto a contratação de especialistas.

- Gerenciamento das atividades.

  Ordenar e priorizar as atividades, são tarefas básicas para ordenar a sequência de atividades, fazendo com que assim seja respeitado os prazos de entregas e é dado a atenção necessária àquilo que traz resultados. Um bom gestor deve conhecer sua equipe de trabalho para saber delegar as atividades. Assim permitindo a gerencia do tempo com tranquilidade. Um software de gerenciamento de projetos pode ajudar na delegação de tarefas para sua equipe e ainda controlar a execução e entrega de cada uma delas. Gerenciamento de atividades é uma das grandes preocupações de todo gestor de projetos, pois elas impactam diretamente no tempo e na qualidade do projeto.

A universidade se diferencia de uma empresa regular pelos objetivos sociais que persegue; por sua função de formadora de cidadãos; pelas características de sua natureza humanista e pelo modelo de administração que adota. Ao considerar a universidade apenas como empresa, ou compará-la com instituições a ela não assemelhadas, corre-se o risco de esquecer o mais importante: a missão de serviço à sociedade, às pessoas e aos próprios fins da Educação.

5 CONCLUSÕES E TRABALHOS FUTUROS

As metodologias de desenvolvimento de software são uma alternativa ao processo de desenvolvimento acessível, ágil e eficiente. Por meio de um referencial teórico, conceitos e práticas das metodologias ágeis e tradicionais, ambos temas abordados na execução dos processos de desenvolvimento de software. A partir dessa apresentação de conceitos e práticas, estabeleceram-se as bases que fundamentam a proposta deste trabalho.

Gerenciar um projeto de software envolve, dentre outros fatores,o planejamento e o acompanhamento das pessoas envolvidas no projeto, principalmente a metodologia que está sendo seguida para evoluir o software de um conceito preliminar para uma implementação concreta e operacional. Para adotar métodos de desenvolvimento de software em uma organização ou ambientes acadêmicos, são necessários diversos passos de planejamento e uma execução cuidadosa. Neste artigo foram apresentados os resultados empíricos obtidos no estudo de caso de adoção de métodos ágeis e tradicionais em uma instituição pública brasileira de grande porte.

A maioria das equipes do Lar-A é formada por graduandos que, ainda estão em formação, têm atividades acadêmicas como suas prioridades e são, parcialmente, dedicados e comprometidos com os projetos. Por meio desta pesquisa, foi possível avaliar como metodologias de gestão de projetos e características dos processos de desenvolvimento de software utilizados na indústria poderiam ser combinados para melhorar a produtividade desses laboratórios e a qualidade de seus produtos.

Foi possível constatar através do questionário aplicado, que os projetos possuem desmotivações e dificuldades em comum. Com um pouco mais de cinco anos o Laboratório de Redes de Aracati está em uma fase de amadurecimento, com relação aos envolvidos dos projetos analisados.

Apesar de todas as dificuldades mostradas nos resultados, é notório o poder de desenvolvimento dos alunos. Acredita-se que com o auxílio de um líder técnico ou gerente de projeto, os projetos devem melhorar com relação ao cumprimento das atividades dentro do prazo, na comunicação com *stakeholders* e no gerenciamento de uma maneira geral, fazendo com que assim seja possível uma melhor distribuição das tarefas e um "desafogamento"das responsabilidades concentradas nos professores pesquisadores.

Com a contratação de um gerente de projeto espera-se ter uma maior dedicação dos estudantes nos projetos, conseguir motivá-los para o sucesso dos projetos, promover o comprometimento de cada um com suas tarefas, aumentar a qualidade

dos produtos desenvolvidos, e tornar o processo replicável em ambientes correlatos.

Como trabalhos futuros, sugere-se aprofundar a avaliação dos estudos de casos para o gerenciamento dos projetos em metodologias tradicionais e ágeis diferentes das apresentadas aqui, a fim de identificar novas vantagens e desvantagens em utilizá-lo e encontrar o que há de incomum.

REFERÊNCIAS

ANDRADE, C. L. de et al. Identificando dificuldades na implementação e gerência de contratos em projetos ágeis de software em belo horizonte. *Abakós, Belo Horizonte, v. 3, n. 1, p. 18-37, nov. 2014 – ISSN: 2175-5841*, 2014. Citado na página 39.

APPELO, J. *Management 3.0: Leading Agile Developers, Developing Agile Leaders*. [S.l.]: Addison-Wesley Professional, 2011. Citado na página 26.

BECK, K. et al. *Manifesto for agile software development*. 2001. Disponível em: <http://agilemanifesto.org>. Acesso em: 10.8.2017. Citado 3 vezes nas páginas 25, 27 e 50.

C. SIM, S. E. S. J. L. T. *Studying Software Engineers: Data Collection Techniques for Software Field Studies [Artigo] // Empirical Softw. Engg.*. [S.l.]: MA, USA : Kluwer Academic Publishers, 2005. - 3 : Vol. 10., 2005. Citado na página 40.

CHIMENDES, V. C. G. *Relacionamento com universidades e institutos de pesquisa: a visão dos empresários*. [S.l.], 2013. Citado na página 14.

CORRêA, M. P. de O. Maturidade em gerenciamento de projetos: sistemática que gera ganhos para as organizações. *IPTEC – Revista Inovação, Projetos e Tecnologias*, 2016. Citado na página 39.

CURTIS BILL; KRASNER, H. I. N. *A field study of the software desing process for large systems*. [S.l.]: Communications of ACM, 1988. Citado na página 18.

D., F. S.; K., L. *"Cycling for transport and public health: a systematic review of the effect of the environment on cycling," The European Journal of Public Health, vol. 21, no. 6*. [S.l.]: Lecture Notes in Computer Science, 2011. Citado na página 49.

FILHO, W. de P. P. *Engenharia de Software: fundamentos, métodos e padrões*. [S.l.]: McGraw Hill Companies, 2010. Citado na página 12.

FOWLER, M. H. *The agile manifesto. Software Development, [San Francisco, CA: Miller Freeman, Inc., 1993-, v. 9, n. 8, p. 28–35*. [S.l.]: Communications of ACM, 2001. Citado 2 vezes nas páginas 24 e 30.

GILB, T. *Principles of Software Engeneering Management*. [S.l.]: UK: Addison-Wesley, 1988. Citado na página 21.

GOLDMAN, A. et al. Being extreme in the classroom: Experiences teaching xp. *Journal of the Brazilian Computer Society ISSN 0104-6500*, 2004. Citado na página 45.

GOMES, A. F. *Agile desenvolvimento de software com entregas frequentes e foco no valor do negócio*. [S.l.]: Casa do Código, 2010. Citado 2 vezes nas páginas 26 e 37.

INSTITUTE, P. M. *A Guide to the Project Management Body of Knowledge (PMBOK Guide) - Sixth Edition*. [S.l.]: Golbal Standard, 2013. Citado na página 44.

KIM, J.; GROENEVELD, P. Big data, health informatics, and the future of cardiovascular medicine. *American College of Cardiology Foundation*, 2017. Citado na página 46.

KRUCHTEN, P. *Introdução ao RUP - Rational Unified process*. [S.l.]: Editora Ciência Moderna LTDA, 2003. Citado 7 vezes nas páginas 19, 20, 21, 22, 23, 24 e 50.

LARMAN, C. *Agile and Iterative Development*. [S.l.]: Agile Developed Series, Alistair Cockburn and Jim Highsmith, Series Editors, 2004. Citado na página 22.

LIBARDI, V. B. P. L. O. Métodos Ágeis. 2010. Citado 2 vezes nas páginas 12 e 33.

LISBOA, L. B. et al. A case study in software product: Lines an educational experience. *Software Engineering Education and Training, 2008. CSEET '08. IEEE 21st Conference on*, 2008. Citado 2 vezes nas páginas 44 e 45.

M., W. *Adaptations for Teaching Software Development with Extreme Programming: An Experience Report [Artigo] - Vol. 2753*. [S.l.]: Lecture Notes in Computer Science, 2003. Citado na página 41.

MALLMANN, P. R. Um modelo abstrato de gerência de software para metodologias ágeis. 2011. Citado na página 39.

MARMOL, L. D. *Release Management*. 2015. Disponível em: <http://www.agile-scrum.be/blog/scrum-release-management/>. Acesso em: 10.8.2017. Citado na página 35.

MCCRACKEN D. D.; JACKSON, M. A. *A minority dissenting opinion. In: W.W. COTTERMAN ET AL. ETS. SYSTEMS ANALYSIS AND DESING*. [S.l.]: Elsevier, 1981. Citado na página 18.

MELO, C. de O.; FERREIRA, G. R. M. Adoção de métodos ágeis em uma instituição pública de grande porte - um estudo de caso. 2011. Citado na página 39.

NASCIMENTO CSM, S. M. G. L. P. *Primeiro passo de um projeto Scrum: Visão*. 2013. Wiki do abnTeX2. Disponível em: <http://blog.myscrumhalf.com/2012/07/qual-o-primeiro-passo-de-um-projeto-scrum-visao/>. Acesso em: 06.08.2017. Citado na página 34.

OLIVEIRA, R. et al. Sistema de apoio à tomada de decisão na gestão de atendimento a pacientes com câncer infanto-juvenil. *WebMedia'2017, Gramado, RS Brazil*, 2017. Citado 2 vezes nas páginas 46 e 47.

OLIVEIRA S. R. B, R. T. A. V. A. M. L. Adequação de processos para fábricas de software. *Anais do Simpósio Internacional de Melhoria de Processos de Software – SIMPROS, São Paulo*, 2004. Citado na página 13.

PAN, A. P. *Associação Peter Pan*. 2017. Disponível em: <http://www.app.org.br>. Acesso em: 25.8.2017. Citado na página 46.

PEREIRA, I.; CARNEIRO, T.; PEREIRA, R. Developing innovative software in brazilian public universities: Tailoring agile processes to the reality of research and development laboratories (artigo). *4a Annual International Conference on Software Engineering Applications. - Phuket : GSTF*, 2013. Citado na página 40.

PEREIRA, I. M. *Desenvolvendo software inovador em universidades públicas: Adaptando processos ágeis para a realidade de laboratórios de pesquisa e desenvolvimento*: uma perspectiva de arquitetura da informação da escola de brasília. Dissertação (Mestrado) — Universidade Federal de Ouro Preto, Ouro Preto -MG, Março 2014. Citado 6 vezes nas páginas 14, 39, 40, 41, 43 e 44.

PLFEEGER, S. L. *Engenharia de Software, Teoria e Prática, 2. Ed.* [S.l.]: Prentice Hall, 2004. Citado 3 vezes nas páginas 17, 18 e 19.

PRESSMAN, R. S. *Engenharia de Software, 6. Ed.* [S.l.]: McGraw Hill Companies, 2001. Citado 10 vezes nas páginas 12, 13, 17, 18, 25, 28, 29, 30, 31 e 32.

PRIKLADNICKI R.; WILLI, R. M. F. *Métodos ágeis para desenvolvimento de software. [S.l.]*. [S.l.]: Bookman Editora, 2014. Citado 2 vezes nas páginas 37 e 38.

ROCHA, A. R. et al. Dificuldades e fatores de sucesso na implementação de processos de software utilizando o mr-mps e o cmmi. 2017. Citado na página 39.

ROYCE, W. W. Anaging the development of large software systems. *IEEE Wescon*, 1970. Citado na página 17.

SCHWABER, K. B. *Agile Software Development with Scrum*. [S.l.]: 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001, ISBN 0130676349, 2001. Citado 3 vezes nas páginas 24, 32 e 35.

SCHWABER, K. S. *Guia do scrum. um guia definitivo para o scrum: As regras do jogo.* [S.l.]: Perigee Books, 2015. Citado 5 vezes nas páginas 32, 33, 34, 37 e 38.

VIANA, J. D. F. et al. A visualization and analysis approach of cyclist data obtained through sensors. *IEEE S3C*, 2017. Citado na página 49.

WELL, D. *Extreme Programming Perspectives*. [S.l.]: Alberto Sillitti, 2003. Citado na página 31.

Apêndice

APÊNDICE A - QUESTIONÁRIO DE AVALIAÇÃO

O questionário apresentado faz parte da pesquisa para extrair os dados apresentado no capítulo 4.

Questionário sobre metodologias no Laboratório de pesquisas e desenvolvimento de softwares. (Lar-A)

Olá, sou aluno do IFCE campus Aracati, faço o curso de Bacharelado em Ciência da Computação. Estou enviando um questionário, composto por perguntas referentes à Metodologias de desenvolvimento de software. Qualquer dúvida ou questionamento entrar em contato por email: gustavorocha.ifce@gmail.com

*Obrigatório

1. **Endereço de e-mail** *

2. **Qual o nome do seu projeto?** *
   Marcar apenas uma oval.
   - ◯ Sysapp
   - ◯ Dengoso
   - ◯ Redes Mesh

3. **Quantidade de pessoas envolvidas no projeto (Alunos e Professores)** *

4. **Quando entrou no projeto, estava em que semestre da graduação?** *

5. **O projeto chegou a ser concluído de acordo com o previsto?** *
   Marcar apenas uma oval.
   - ◯ Sim
   - ◯ Não
   - ◯ Em andamento

6. **O projeto possui um Gerente de Projeto?** *
   Marcar apenas uma oval.
   - ◯ Sim
   - ◯ Não

7. **Qual a metodologia que aproxima-se da utilizada no desenvolvimento?** *
   Marcar apenas uma oval.

   - ◯ Tradicional
   - ◯ Ágil
   - ◯ Tradicional + Personalizada
   - ◯ Ágil + Personalizada
   - ◯ Outro: _____

8. **A equipe de desenvolvimento tem o conhecimento da metodologia?** *
   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

9. **Quantas horas são dedicadas para o projeto por semana? (Aluno)** *

   _____

10. **Quantas horas são dedicadas para o projeto por semana? (Professor)** *

    _____

11. **O projeto está muito dependente do professor?**
    Marcar apenas uma oval.

    - ◯ Sim
    - ◯ Não
    - ◯ Muitas vezes

12. **A equipe de desenvolvimento é a mesma do início do projeto?** *
    Marcar apenas uma oval.

    - ◯ Sim
    - ◯ Não

13. **A equipe de desenvolvimento em algum momento se sentiu desmotivada por algum motivo? Qual?** *

    _____

    _____

    _____

    _____

    _____

14. **Quanto tempo você está no projeto?** *

    _____

15. **A equipe teve problemas de comunicação entre os envolvidos no projeto?** *
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Muitas vezes

16. **A equipe teve problema com mudanças de requisitos durante o projeto?** *
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Algumas vezes

17. **A equipe teve problemas com as entregas dentro do prazo?** *
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

18. **Houveram muitas mudanças nas prioridades ou falta de prioridade?** *
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

19. **Houveram mudanças nos requisitos durante o desenvolvimento do projeto?** *
Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Outro: _____

Análise geral

20. **Descreva as principais dificuldades encontradas na execução do projeto.** *

_____

_____

_____

_____

_____

☐ Envie para mim uma cópia das minhas respostas.