UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO LESSA RIBEIRO

Projeto de um ambiente de desenvolvimento para métodos ágeis

Maringá
2016

MARCELO LESSA RIBEIRO

Projeto de um ambiente de desenvolvimento para métodos ágeis

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Profª. Drª. Itana Maria de Souza Gimenes

Maringá
2016

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR., Brasil)

R484p

Ribeiro, Marcelo Lessa
Projeto de um ambiente de desenvolvimento para métodos ágeis / Marcelo Lessa Ribeiro. -- Maringá, 2016.
101 f. : il. col., figs., tabs., gráficos + apêndice

Orientadora: Prof.a Dr.a Itana Maria de Souza Gimenes.
Dissertação (mestrado) - Universidade Estadual de Maringá, Departamento de Informática, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação, 2016

1. Ambiente de Desenvolvimento de Software. 2. Métodos Ágeis. 3. Práticas ágeis. I. Gimenes, Itana Maria de Souza, orient. II. Universidade Estadual de Maringá. Departamento de Informática. Centro de Tecnologia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 21.ed. 005.4

MN-003868

FOLHA DE APROVAÇÃO

MARCELO LESSA RIBEIRO

Projeto de um ambiente de desenvolvimento para métodos ágeis

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Profa. Dra. Itana Maria de Souza Gimenes
Universidade Estadual de Maringá – DIN/UEM

Profa. Dra. Thelma Elita Colanzi Lopes
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Marcos Antonio Quináia
Universidade Estadual do Centro-Oeste – DECOMP/UNICENTRO

Aprovada em: 21 de dezembro de 2016.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

DEDICATÓRIA

A Jesus Cristo seja a honra a glória e o louvor. Para todo o sempre, amém.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me abençoado nesta jornada, que nos momentos difíceis me ajudou a supera-los e esteve comigo em todos os momentos. Sou grato a Deus pela sua oportunidade de desenvolver este trabalho e por todos os momentos vivenciados durante este período.

A minha família em especial a minha mãe Debora Maria Lessa pelo apoio durante todo a minha vida. Também agradeço a minha esposa Karyta Muniz de Paiva Lessa pela ajuda incondicional durante cada momento nesta jornada.

A minha orientadora Dr. Itana Maria de Souza Gimenes por sempre me guiar nas dificuldades, além de me motivar e sugerir a busca por novos horizontes no desenvolvimento desta pesquisa.

Aos professores do curso de Mestrado em Ciência da Computação do DIN da UEM, agradeço pelos ensinamentos e formação de qualidade que recebi. Também agradeço a todos os amigos que fiz durante meus estudos de mestrado, em especial para minha turma de 2014.

EPÍGRAFE

A tua palavra é lâmpada
que ilumina os meus passos
e luz que clareia o meu caminho.
Salmos 119:105

Projeto de um ambiente de desenvolvimento para métodos ágeis

RESUMO

Por volta dos anos 70 surgiram os conceitos de ambiente de desenvolvimento de software cujo objetivo é apoiar as atividades de desenvolvimento, manutenção e gerenciamento de projetos de software, utilizando diversas ferramentas de forma integrada. Um conceito importante, neste contexto, é o *Application Life-Cycle Management* (ALM) que apresenta um modelo de gerenciamento do ciclo de vida do software, desde sua concepção até sua manutenção. O surgimento dos Métodos Ágeis (MA), tem mudado como o processo de software é visto, no qual os valores humanos e a maneira como as pessoas trabalham em conjunto constituem os fatores principais enquanto que as práticas, as ferramentas e os processos usados são de segunda ordem. Em essência, eles fornecem entrega rápida e incremental de software, promovendo a cooperação entre fornecedor e cliente durante a construção do software. Assim é imprescindível a utilização de ferramentas apropriadas para se obter um gerenciamento ágil de projetos e impô-las aos colaboradores. Portanto, pesquisas são necessárias para reconhecer a importância das práticas de desenvolvimento de software e estudos são necessários para superar a barreira do não uso de ferramentas ou o uso de ferramentas não-flexíveis e para melhorar à conservação do uso de MA no desenvolvimento de software. Além disso, a concepção de um ambiente propício para o contexto de MA, demanda pesquisas e estudos exploratórios que combinem as experiências na indústria e na academia. Neste contexto, esta dissertação apresenta uma proposta de um projeto de um Ambiente de desenvolvimento de Software (ADS) apoiado nos princípios de ALM para articular o uso do *Scrum* como método de gerenciamento e as técnicas de apoio ao desenvolvimento de software, denominado *Agile Development Environment* (ADE). Além disso, no ADE são descritos os elementos pertencentes ao ambiente, sua arquitetura, o processo de desenvolvimento e as integrações entre os elementos. A avaliação do ADE foi realizada em duas partes, a primeira uma comparação entre o ADE e os conceitos de ADS. Já a segunda parte da avaliação foi realizada por meio de um estudo empírico qualitativo, adotando procedimentos de *Grounded Theory*, do ponto de vista de especialistas com experiência na indústria em desenvolvimento de software e métodos ágeis. Os resultados obtidos de tal estudo forneceram indícios de viabilidade para sua utilização e serviram como base para melhorias na proposta do ADE.

Palavras-chave: Métodos Ágeis, Práticas ágeis, Ambiente de Desenvolvimento de Software.

Designing a development environment for agile methods

ABSTRACT

The demand for software engineering support environment was evident since the 70s. It was necessary to control the integration between processes, tools and developers in order to increase software quality and productivity. An important concept in this context was Application Life Cycle Management (ALM). It is a model that covers software activities from conception to maintenance. The emergence of agile methods has changed how the software process is viewed, which human values and the way people work together are the key factors while the practices, tools and processes used are second-order. In essence, they provide fast and incremental software delivery, fostering co-operation between supplier and customer during software building. Thus, it is imperative to use appropriate tools to achieve agile project management and impose them on employees. Therefore, research is needed to recognize the importance of software development practices and studies are needed to overcome the barrier of non-use of tools or the use of non-flexible tools and to improve the conservation of the use of MA in software development. In addition, the design of an environment conducive to the context of MA requires research and exploratory studies that combine experiences in industry and academy. In this context, this paper presents a proposal for an ADS project based on ALM principles to articulate the use of Scrum as a management method and software development support techniques, named Agile Development Environment (ADE). In addition, the ADE describes the elements belonging to the environment, its architecture, the development process and the integrations between the elements. The evaluation of the ADE was carried out in two parts, the first a comparison a comparison with previous development environment is also presented. The second part of the evaluation was carried out through a qualitative empirical study, adopting Grounded Theory procedures, from the point of view of experts with experience in the software industry and agile methods. The results obtained from this study provided indications of feasibility of its use and served as a basis for improvements in the ADE proposal.

***Keywords*:** *Agile methods, Agile practices, Software Engineering Environments.*

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADE	*Agile Development Environment*
ADS	*Ambiente de Desenvolvimento de Software*
ALM	*Application Lifecicle Management*
BDD	*Behavior Drive Development*
CMMI	*Capability Maturity Model Integration*
IC	*Integração Contínua*
IDE	*Integrated Development Environment*
MA	*Métodos Ágeis*
OODBMS	*Object-Oriented Database Management System*
PO	*Product Owner*
PML	*Process Modeling Language*
PSEE	*Process-centered Software Engineering Environments*
RUP	*Rational Unified Process*
SDLC	*Software Development Life Cyle*
SWEBOK	*Software Engineering Body of Knowledge*
TCLE	*Termo de Consentimento Livre e Esclarecido*
TDD	*Test Drive Development*
UML	*Unified Modeling Language*
XP	*Extreme Programming*

SUMÁRIO

1

Introdução

1.1 Contextualização

O uso de técnicas, ferramentas e processos de apoio ao desenvolvimento de software surgiu no final dos anos 60 quando foi reconhecida a necessidade de estratégias rigorosas que pudessem ser replicadas. Neste contexto, surgiram vários modelos de processo, como por exemplo: Cascata, Espiral, Incremental e Prototipação (Bassi, 2008). Esses modelos definem etapas para a geração de artefatos desde o início até o fim de um projeto que produz um software executável. Exemplos de artefatos são: o modelo de casos de uso, o documento de requisitos, a arquitetura de software, os artefatos de teste, os planos de trabalho, as especificações, o manual de usuário e os relatórios. Esses processos, em geral, eram complexos e rígidos, portanto, sua introdução em situações reais se tornou difícil. Observou-se que, mesmo com o uso de técnicas de engenharia de software, a entrega do produto final aos clientes falhava. Essas falhas geravam um impacto econômico e humano negativo, portanto, tornou-se necessário encontrar uma nova maneira de desenvolver *software* (Beck, 2004).

Neste cenário, surgiram as primeiras iniciativas de Métodos Ágeis (MA), as quais resultaram no Manifesto Ágil (Williams e Cockburn, 2003). Neste manifesto, argumenta-se que os valores humanos e a maneira como as pessoas trabalham em conjunto constituem os fatores principais, enquanto que as práticas, as ferramentas e os processos usados deveriam ser tratados como elementos de segunda ordem (Fowler, 2005). A maneira como o processo de software é visto tem mudado com os MA. Em essência, eles fornecem entrega rápida e

incremental de software e, portanto, possibilitam a entrega mais imediata de valor de negócio aos clientes (Beck et al., 2001).

A necessidade de ambientes que fornecessem suporte para o desenvolvimento de software, possibilitando a coordenação entre os desenvolvedores e todas as ferramentas utilizadas no desenvolvimento de software e manutenção, de modo a permitir a eficiente produção de software de qualidade, foi percebida nos anos 70 (Brown, 1993). Assim, foram criados ambientes de desenvolvimento de software para apoiar cada atividade da construção do software, utilizando diversas ferramentas e facilidades integradas (Betemps, 2003). Decorrente das pesquisas realizadas, surgiram vários Ambientes de Desenvolvimento de Software (ADS) que introduziram conceitos importantes como repositório central de artefatos, processo de software bem definido e acompanhado e ferramentas de apoio, como gerenciamento de versão e configuração (Finkelstein et al., 1994).

Neste contexto, também surgiu o conceito do *Application Life-Cycle Management* (ALM), que apresenta um modelo de gerenciamento do ciclo de vida do software, desde sua concepção até sua manutenção (Lacheiner e Ramler, 2011). Seu foco está em promover a integração entre requisitos, desenvolvimento, implantação e manutenção. O ALM fornece princípios para conectar tarefas de gerenciamento com desenvolvimento de software, integrando as ferramentas usadas no gerenciamento do software, desenvolvimento, bem como operações (Fuggetta e Di Nitto, 2014). Assim, esta perspectiva de ALM auxilia a pensar como as pesquisas desenvolvidas em ADS, como é o caso dos ambientes descritos em Finkelstein et al. (1994), podem se alinhar às práticas de métodos ágeis.

1.2 Motivação

Empresas de software precisam atingir uma alta agilidade e qualidade no desenvolvimento de projetos de software. No entanto, aplicar apenas os princípios de MA não é o suficiente para que os projetos possam ser desenvolvidos com agilidade (Beck, 2004). Também é imprescindível a utilização de ferramentas apropriadas para se obter um gerenciamento ágil de projetos e impô-las aos colaboradores (Franky, 2011). Com base na literatura existente, Gandomani et al. (2013) dividiram os desafios na aplicação dos MA em quatro categorias: 1) organização e gestão; 2) pessoas; 3) processo; 4) ferramentas. O último desafio reflete que o não uso de ferramentas ou o uso de ferramentas não-flexíveis é uma barreira para adoção de MA. As empresas devem buscar ferramentas para apoiar a evolução incremental do software para ajudá-las a superar estes desafios e para melhorar a manutenção dos MA (Gandomani et

al., 2013). Gregory et al. (2015) chegaram à mesma conclusão de Gandomani et al. (2013) e além disso, acrescentam que o amadurecimento dos MA implica na ausência de obstáculos em relação a adoção de MA no desenvolvimento de software, em relação a manter o uso de MA no desenvolvimento de software. Dessa forma, são necessários estudos que contribuam para a conservação do uso de MA no desenvolvimento de software (Gregory et al., 2015).

O amadurecimento das práticas e a introdução de tecnologias de apoio ao desenvolvimento de software demandaram uma mudança significativa nas abordagens de engenharia de software (Fuggetta e Di Nitto, 2014). Fuggetta e Di Nitto (2014) afirmam que é difícil seguir regras ou um passo a passo em todos os cenários de desenvolvimento de software. Para cada problema, pode existir uma melhor solução; foi isso que chamaram de convergência inteligente. Assim, é mais importante uma abordagem flexível do que um processo que segue regras, procedimentos, restrições e padrões pré-definidos muito rígidos. Com o intuito de possibilitar a implementação da convergência inteligente, com agilidade, foi proposto o estudo de novas práticas de gestão, métodos de *design* e também soluções técnicas.

Acrescenta-se ainda que a cooperação entre desenvolvedores e clientes, durante a construção de software, vem alterando drasticamente as técnicas para conceber, projetar, desenvolver, testar, implantar e evoluir os produtos de software (Fuggetta e Di Nitto, 2014). Dessa forma, a indústria de software desenvolveu novas abordagens e ferramentas, inclusive aproveitando-se de técnicas desenvolvidas em ADS. A concepção de um ambiente propício, para o novo contexto de MA, demanda pesquisas e estudos exploratórios que combinem as experiências na indústria e na academia, de modo a dar mais atenção ao que foi desenvolvido e realmente aplicado pela indústria (Fuggetta e Di Nitto, 2014). Ainda vale acrescentar que, no contexto ágil, a relevância dos dados relatados pela indústria e o impacto prático são amplamente reconhecidos (Gregory et al., 2015).

Assim, pesquisas são necessárias para reconhecer a importância das práticas de desenvolvimento de software (Fuggetta e Di Nitto,2014) e atender os desafios mencionados por Gandomani et al. (2013), Fuggetta e Di Nitto (2014) e Gregory et al. (2015). Vários trabalhos relatam o uso de práticas ágeis e as várias ferramentas existentes, porém não definem como estruturar um Ambiente de Desenvolvimento de Software (ADS) que integre o gerenciamento com as práticas (Humble e Farley, 2010; Abrantes e Travassos, 2011; Bass, 2012; Silva, 2013; Collins et al., 2014).

Além disso, tendo em vista que a configuração de um ADS, para projetos ágeis pelas organizações, não é trivial, é importante investigar como as organizações podem escolher a

melhor combinação das práticas para auxilia-los no desenvolvimento de software (Abbas et al., 2010).

1.3 Objetivos

Diante do contexto e da motivação apresentados, esta pesquisa de mestrado tem como objetivo: propor o projeto de um ADS apoiado nos princípios de ALM para articular o uso do *Scrum* como método de gerenciamento e as técnicas e ferramentas de apoio ao desenvolvimento de software. O uso de *Scrum* deve-se ao fato de estudos demostrarem vantagens em sua aplicação, como é o caso de Striebeck (2006). Além disso, o *Scrum* é o método mais utilizado pela indústria (VersionOne, 2015; Melo et al., 2012). O uso dos princípios do ALM deve-se ao fato de seus conceitos oferecerem a gestão do ciclo de vida do software no âmbito de governança, desenvolvimento e manutenção (Chappel, 2014; Rossman, 2010). Ademais, o ALM possibilita o gerenciamento ágil, com a integração das ferramentas de gestão e desenvolvimento (Franky, 2011).

Para alcançar o objetivo geral desta pesquisa, propôs-se os seguintes objetivos específicos:

- Investigar a relação entre as práticas e técnicas aplicadas ao processo de desenvolvimento de software;
- Conceber um ADS que facilite a utilização sistemática de práticas e ferramentas ágeis;
- Fornecer evidências sobre o ADS proposto.

1.4 Metodologia de Desenvolvimento

Para o desenvolvimento desta pesquisa, foram utilizadas as abordagens teórica e empírica. A abordagem teórica está fundamentada em uma revisão bibliográfica realizada para contextualizar o cenário atual referente ao ambiente de desenvolvimento de software em projetos ágeis, o que é apresentado no Capítulo 2 desta dissertação. O projeto do Ambiente Ágil de Desenvolvimento, denominado *Agile Development Environment* (ADE), concentrou-se em definir, com base na revisão bibliográfica, fundamentado no *Scrum* e ALM, e os elementos que compõem o ADE, a arquitetura lógica do ADE, o processo de desenvolvimento, como ele articula os elementos do ADE e, por fim, foram descritas as integrações existentes no ADE. A avaliação levou em consideração uma análise empírica

qualitativa, por meio de inspeção por procedimentos de *Grounded Theory,* com base no conhecimento de especialistas em desenvolvimento de software com metodologias ágeis.

1.5 Organização

Este capítulo apresentou a contextualização desta dissertação, a motivação, objetivos e metodologia de desenvolvimento. O restante deste documento está estruturado da seguinte forma: no Capítulo 2, é apresentada uma revisão da literatura sobre Construção de Software, Métodos Ágeis, o estado dos Métodos Ágeis nas organizações, as práticas de desenvolvimento de software em Métodos Ágeis, Ambiente de Desenvolvimento de software e Gerenciamento do Ciclo de Vida da Aplicação. O Capítulo 3 apresenta a especificação do projeto do ADE. A avaliação empírica do ADE por especialistas é apresentada no Capítulo 4, bem como a comparação com os conceitos de ADS. No Capítulo 5, é apresentada a conclusão acerca desta dissertação, bem como as suas contribuições, limitações e trabalhos futuros.

2

Revisão Bibliográfica

2.1. Considerações Iniciais

Este capítulo contém conceitos importantes sobre métodos de desenvolvimento de software que servem de base para esta proposta de mestrado. Na Seção 2.2, é apresentada a área de conhecimento Construção de Software. Na Seção 2.3, são abordados os MA; enquanto na Seção 2.4, é apresentado como eles são avaliados pelas organizações e sua aceitação. Por sua vez, a Seção 2.5 apresenta práticas de desenvolvimento de software em MA; enquanto na Seção 2.6, são apresentados os conceitos de ambiente de desenvolvimento de software. Por fim, a Seção 2.7 aborda gerenciamento do ciclo de vida da aplicação.

2.2. Construção de *Software*

Um processo de software genérico compreende cinco atividades: 1) Comunicação com o cliente; 2) Planejamento; 3) Modelagem; 4) Construção; 5) Entrega. A atividade de construção combina a produção de código e testes necessários para revelar erros na codificação (Pressman, 2011).

A atividade de construção de software também está presente no *Software Engineering Body of Knowledge* (Swebok, 2014). Neste, ela constitui uma área do conhecimento chave para o desenvolvimento de software, que está presente em todos os processos de software (Swebok, 2014) tradicionais ou ágeis.

A construção de Software refere-se à sua codificação em uma linguagem de programação, sua verificação, testes de unidade, testes de integração e depuração. Esta área encontra-se interligada com todas as demais áreas de conhecimento, porém é fortemente ligada à de projeto e teste de Software (Swebok, 2014). A Construção de Software é dividida em cinco subáreas principais (Swebok, 2014):

1. Fundamentos do Desenvolvimento de Software: compreende em minimizar a complexidade das funcionalidades, antecipar-se a mudanças dos requisitos, possibilitar que as falhas sejam facilmente encontradas pelos desenvolvedores e testadores, promover o reuso no desenvolvimento e aplicar padrões de desenvolvimento.
2. Gerenciamento do Desenvolvimento: refere-se a seguir processo de desenvolvimento que permite realizar o planejamento da construção do software e extrair métricas de desenvolvimento.
3. Considerações Práticas: remete-se a decisões que devem ser tomadas no desenvolvimento do software, por exemplo, paradigmas e linguagens de programação a serem usados, tipos de testes a serem implementados e organização do código.
4. Tecnologias de Desenvolvimento: refere-se a decisões de tecnologias a serem usadas na construção do software como qual *framework* e bibliotecas usar e tecnologias a serem usadas para o desenvolvimento distribuído.
5. Ferramentas de Desenvolvimento de Software: remete-se a decisões sobre quais ferramentas devem ser usadas para o desenvolvimento do software, como é o caso dos testes automatizados, para a construção das telas do software e qual IDE será usada para a implementação.

2.3. Métodos Ágeis

O manifesto ágil (Beck et al., 2001) lança a fundamentação conceitual e filosófica na qual o desenvolvimento ágil deve se pautar. Ele é representado por quatro premissas:

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

Em relação às premissas do manifesto, Kalermo e Rissanen (2002) afirmam que são mais valorizados os fatores humanos e a maneira como as pessoas trabalham em vez de processos, ferramentas e negociação de contratos. Após a criação do manifesto, surgiram vários MA, alguns voltados ao gerenciamento de projeto e outros mais direcionados a questões técnicas, como por exemplo, Scrum, Programação Extrema (XP), Crystal e Kanban.

Os MA fundaram suas bases na necessidade de incorporar as mudanças do ambiente, em vez de tentar eliminá-las por meio de esforços de planejamento antecipado — como é feito tradicionalmente —, e na satisfação dos clientes por meio de entregas rápidas e frequentes. Argumenta-se que o desenvolvimento deve se concentrar no código em funcionamento, pois este é o único resultado real, e também deveria ser orientado à interação humana — intensa colaboração e equipes auto-organizáveis — em detrimento a ostensivos processos e documentação. Os MA concentram-se nas características do software, mais do que nas tarefas, e realizam priorização dinâmica (Highsmith e Cockburn, 2001).

A maneira como o processo de software é visto tem mudado com os MA. Em essência, eles fornecem entrega rápida e incremental de software e, portanto, possibilitam a entrega mais imediata de valor de negócio aos clientes (Beck et al., 2001). Não somente, mas também os MA proporcionam processos estruturados de planejamento e controle e, por meio de ciclos iterativos e rápidos, permitem aprender a se adaptar eficientemente às condições de mudança (Batra, et al., 2010).

Na última pesquisa realizada pela VersionOne (2015), em comparação com a pesquisa realizada em 2009, houve um crescimento de 67% para 95% dos respondentes que usam ou possuem a intenção de usar MA. É fato que houve um crescimento da utilização dos MA, porém ainda existem barreiras que dificultam sua adoção nas organizações. Como pode ser observado no Gráfico 2.1, as maiores resistências para adoção de agilidade são a habilidade de mudar a cultura organizacional e a resistência à mudança (VersionOne, 2015). Também foram levantados os principais motivos de falhas nos projetos que fazem uso de MA. São eles: divergência entre a cultura da organização e os princípios ágeis, falta de experiência com MA, falta de apoio ao gerenciamento de projeto, falta de apoio para a transição de metodologia, processos e práticas ágeis inconsistentes e pressão externa para seguir o fluxo do processo cascata (VersionOne, 2015).

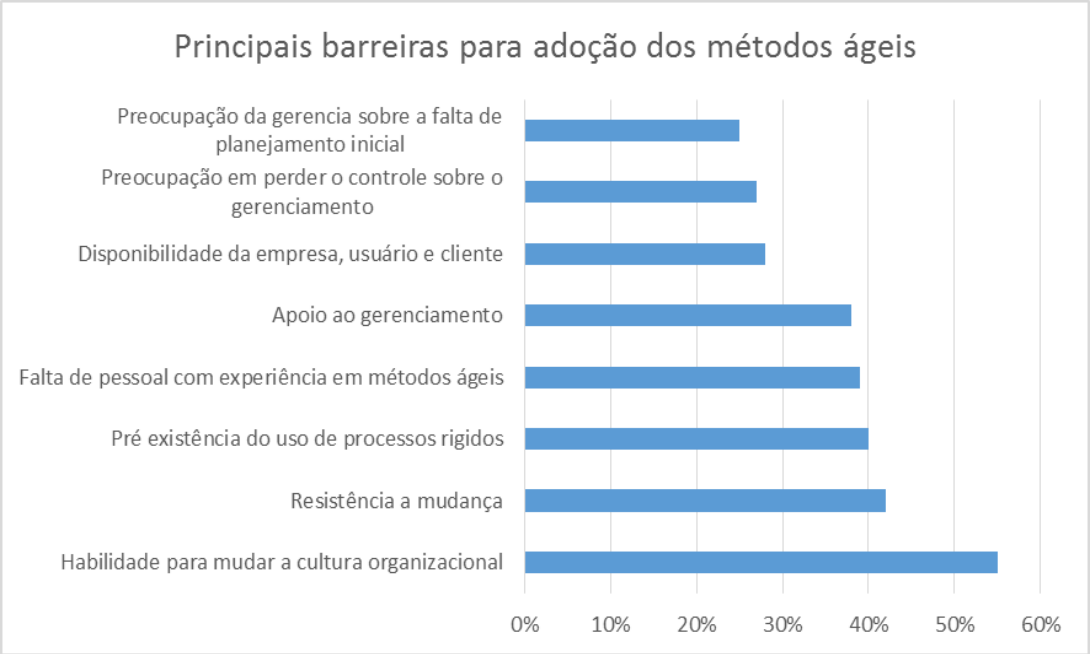


Gráfico 2.1: Principais barreiras para adoção dos MA (VersionOne, 2015)

2.3.1. Scrum

O *Scrum* (Schwaber e Sutherland, 2013) é um MA com foco no gerenciamento de projetos. Os autores não o consideram como uma técnica ou processo para construir produtos e sim um *framework*, em que é possível empregar vários processos e práticas para tratar e resolver problemas complexos, adaptativos e, dessa forma, entregar produtos com maior valor aos clientes.

Os times *Scrum* devem ser auto-organizáveis e multifuncionais. Dessa forma, o time *Scrum* deve ter as competências necessárias para não depender de pessoal externo. Além do mais, o próprio time define a melhor maneira para realizarem o trabalho, não sendo dirigido ou gerenciado por pessoas externas (Schwaber e Sutherland, 2013). Segundo Schwaber e Sutherland (2013), o time *Scrum* possui três papéis característicos:

- ***Product Owner*:** é responsável por gerenciar o *Backlog* do Produto, que é a lista de requisitos que comporão as entregas das *Sprints*, bem como deve priorizá-lo.
- **Time de Desenvolvimento**: formado por um grupo de profissionais multifuncionais e auto-organizáveis, os quais realizam o trabalho de entregar uma versão usável do software que é potencialmente incrementável até que se chegue ao produto final de cada *Sprint*.

- ***Scrum Master***: é responsável por garantir que o *Scrum* seja entendido e aplicado; assegura que o time *Scrum* ligue as teorias, práticas e regras do *framework.*

O *Scrum* possui eventos pré-definidos para minimizar a necessidade de reuniões não definidas e apoiar na inspeção e adaptação. Os eventos possuem tempo máximo de duração (*time-boxed*). Uma *Sprint* é uma iteração de desenvolvimento com objetivo de entregar incrementos funcionais. Um evento cíclico *Sprint* tem duração pré-fixada de duas ou quatro semanas normalmente. Segundo Schwaber e Sutherland (2013), a realização de uma *Sprint* inclui os seguintes eventos:

- **Reunião de planejamento**: tem por objetivo entender os itens prioritários do *Product Backlog* para selecionar os que serão entregues nesta iteração e traçar um plano de trabalho definindo as tarefas para compor o *Backlog* da *Sprint*.
- **Reunião Diária**: tem por objetivo sincronizar o andamento da *Sprint* entre os participantes do time de desenvolvimento. Este evento é usado para inspecionar o progresso da *Sprint*. A reunião diária melhora a comunicação entre o time, elimina outras reuniões, identifica e remove impedimentos para o desenvolvimento.
- **Revisão da *Sprint***: ocorre ao final da *Sprint,* com objetivo de inspecionar as funcionalidades que foram entregues na iteração. Nesta reunião, o time colabora para priorizar funcionalidades que podem otimizar a entrega de valor do produto.
- **Retrospectiva**: tem por objetivo encorajar o time a revisar o processo de desenvolvimento e descobrir pontos fracos e fortes. Como resultado, são identificadas melhorias que devem ser implementadas na próxima *Sprint*. Este evento ocorre após a Revisão da *Sprint* e antes da reunião de planejamento da próxima *Sprint*.

Os artefatos do *Scrum* têm o propósito de representar o trabalho realizado nos eventos. Além disso, os artefatos gerados possuem o objetivo de proporcionar transparência do que será realizado. São eles (Schwaber e Sutherland, 2013):

- ***Backlog* do Produto**: é uma lista ordenada dos requisitos do sistema, a qual contém todas as funcionalidades necessárias para produzir o software para o cliente. Normalmente o *Backlog* do Produto nunca está completo e evolui tanto

quanto o produto pode sofrer mudanças que possibilitem aprimorar e torná-lo mais competitivo e útil.

- ***Backlog* da *Sprint*:** é uma sub lista de *Backlog* do Produto. Este artefato constitui a previsão das funcionalidades que comporão um novo incremento do produto ao final da *Sprint*.

Striebeck (2006) realizou um estudo aplicando o *Scrum* no desenvolvimento de um projeto e obteve os seguintes resultados: permitiu alcançar controle sobre o desenvolvimento; priorização mais cuidadosa dos recursos; e prazos mais realistas para uma empresa que partiu de uma situação caótica e imatura.

Para o desenvolvimento deste Projeto de Mestrado, foi escolhido o *Scrum* para gerenciamento do ciclo de vida. A escolha foi feita com base nas pesquisas VersionOne (2015) e Melo et al. (2012), entre as quais o *Scrum* é o método mais usado pela indústria de software.

2.4. O estado dos métodos ágeis nas organizações

Para evidenciar quais são as práticas ágeis usadas pela indústria, a VersionOne (2015) inseriu em sua pesquisa a pergunta “Quais as principais práticas ágeis adotadas?”. As respostas para esta questão destacam práticas de gerenciamento e de desenvolvimento. Em relação às técnicas de desenvolvimento, pode-se observar no Gráfico 2.2 que a maioria das organizações usam teste unitário, padrão de código, integração contínua e *build* automatizado. Porém, mesmo havendo um número expressivo de organizações que utilizam teste unitário, o uso de Desenvolvimento Dirigido por Testes (TDD – *Test Drive Development*) não é tão expressivo.

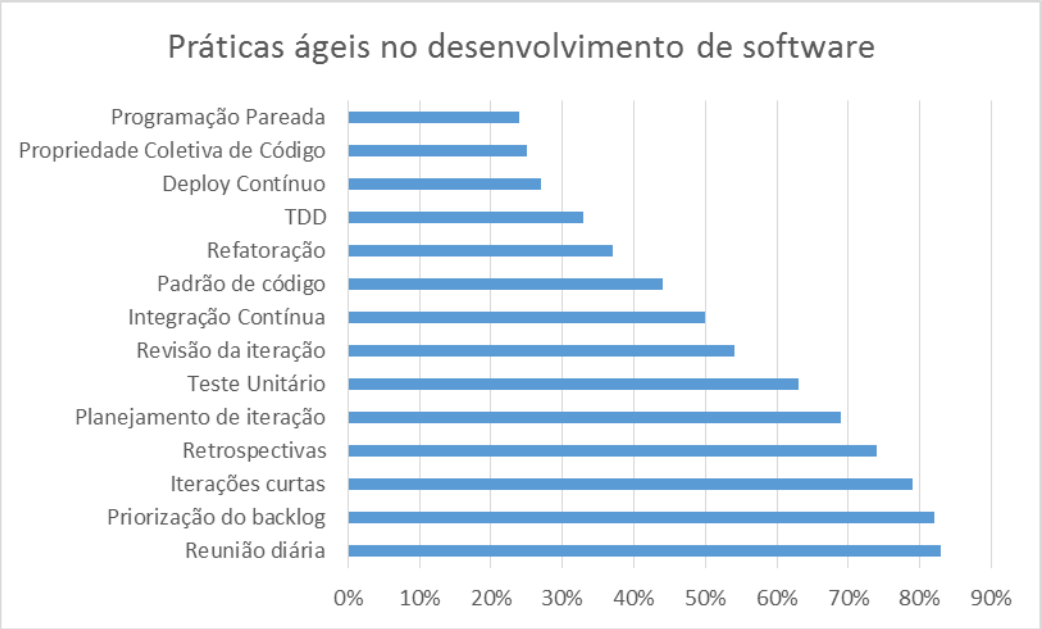


Gráfico 2.2: Uso de práticas ágeis (VersionOne, 2015)

Uma pesquisa semelhante à VersionOne, realizada no Brasil em 2011, teve o objetivo de levantar o estágio de adoção e adaptação dos MA no país (Melo et al. 2012). Como pode ser observado no Gráfico 2.3, os números são muito semelhantes, embora os questionários tenham sido respondidos com a diferença de quatro anos. Um destaque é em relação à prática de refatoração que, na pesquisa mundial, fica em 37%, enquanto na pesquisa de âmbito nacional, este percentual cresce em mais de 20%, chegando a 59,2%.

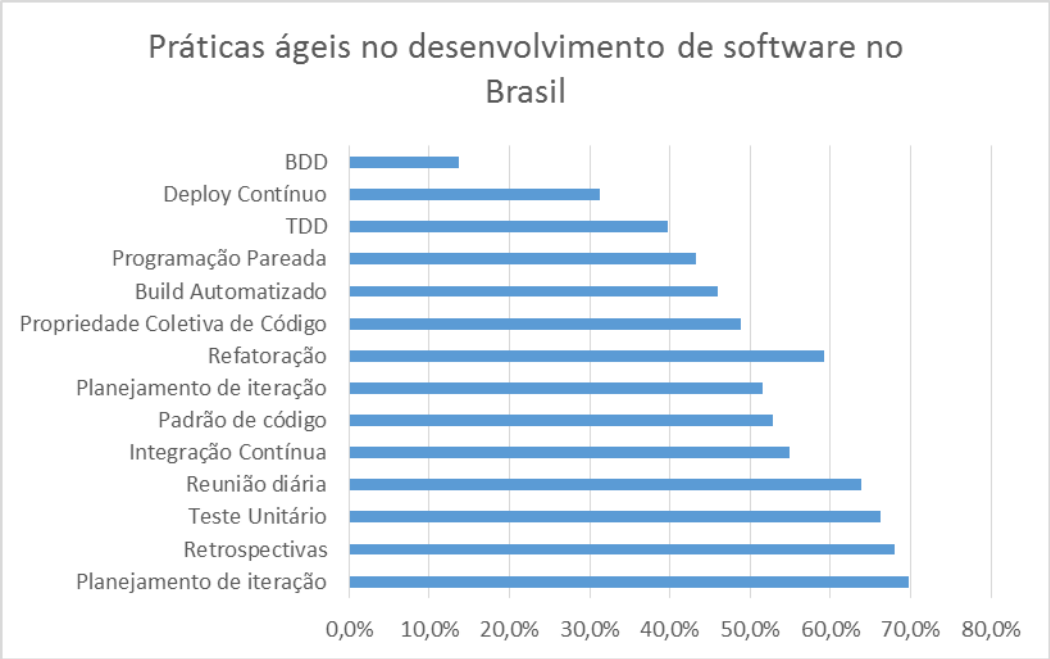


Gráfico 2.3: Uso de técnicas ágeis no Brasil (Melo et al., 2012)

Para entender como as grandes empresas adotam as práticas ágeis, Bass (2012) realizou uma pesquisa com sete companhias internacionais, e entrevistou dezenove praticantes de MA dessas organizações. O autor chegou à conclusão que não houve conflito entre as práticas do *Scrum*, mais voltadas ao gerenciamento, com as políticas e processos da organização. Entretanto, não houve o mesmo nível de adoção quanto às práticas do XP, que são mais técnicas. Os motivos encontrados como justificativa para a não adoção de tais práticas estão descritos na Tabela 2.1. O mesmo não se pode dizer quanto à padronização de código e propriedade coletiva do código que são amplamente usadas nas empresas participantes da pesquisa.

Tabela 2.1: Práticas Ágeis em grandes empresas (Bass, 2012)

Prática / Técnica	**Situação nas organizações**
TDD	Problemas com a garantia de qualidade da organização
Programação Pareada	Usada apenas em algumas circunstâncias, como em um código complexo
Integração Contínua	É um desafio devido ao tamanho e complexidade dos projetos das organizações. A prática é muito atraente, porém o tempo de realizar um *build* e rodar os testes de regressão chega a três dias em algumas empresas.
Padrão de código	Prática bem difundida nas organizações
Propriedade Coletiva do Código	Prática bem difundida nas organizações

Abrantes e Travassos (2011) apresentam uma revisão sistemática com o objetivo de investigar as práticas usadas no contexto de MA de desenvolvimento de software. Nesta revisão, foram encontrados 5.093 trabalhos relacionados, em que 24 deles, publicados entre 2001 e 2009, foram analisados e encontradas 236 práticas. Ao analisá-las identificando repetições, estas foram agrupadas em 51 práticas, das quais 17 delas são as que mais aparecem em trabalhos técnicos. Nelas estão práticas voltadas ao gerenciamento e ou ao desenvolvimento de software, a saber: 1) Desenvolvimento Dirigido por Testes; 2) Integração Continua; 3) Programação Pareada; 4) Propriedade Coletiva do Código; 5) Refatoração; 6) Padrão de Código.

Os dados obtidos nas pesquisas realizadas por Abrantes e Travassos (2011), VersionOne (2015) e Melo et al. (2012) serviram de referência para escolha das práticas e técnicas de desenvolvimento de software a serem utilizadas na concepção do ambiente proposto.

2.5. Práticas de desenvolvimento de software em Métodos Ágeis

Esta seção tem o objetivo de apresentar as práticas usadas na construção de software no contexto de MA, exemplificando as ferramentas que possibilitam a aplicabilidade de tais práticas. As práticas de implementação apresentadas, nesta seção, foram extraídas do método Programação Extrema (XP) e das pesquisas Abrantes e Travassos (2011), VersionOne (2015) e Melo et al. (2012).

2.5.1. Desenvolvimento Dirigido por Testes – TDD

O desenvolvimento dirigido por teste (*Test Driven Development* – TDD) é uma prática que se usa para desenvolver o software, na qual a escrita dos testes acontece antes de se escrever o código do software, dividindo a implementação em três fases: 1) codificar o teste; 2) implementar o código para corrigir o teste; 3) refatorar. Essas três etapas seguem um ciclo até que todo o software tenha sido desenvolvido (Agarwal e Deep, 2014; Beck, 2004).

Uma vantagem do TDD foi evidenciada por um experimento realizado por Maximilien e Williams (2003), no qual duas equipes desenvolveram um software similar, em uma aplicou TDD e na outra não. Ao final deste experimento a equipe que aplicou TDD produziu 50% menos defeitos em relação à outra equipe (Maximilien e Williams, 2003). Também foi

possível evidenciar maior qualidade de código com o uso de TDD (Maximilien e Williams, 2003; George e Williams, 2004).

Dessa forma, a literatura técnica destaca a importância de se documentar os artefatos de software com ferramentas de testes (Collins et al., 2014). Além do uso de ferramentas acelerarem o tempo de execução dos testes, também permite a redução do custo com a execução de testes manuais que diminuirá ou até mesmo não existirá (Razak e Fahrurazi, 2011).

2.5.2. Integração Contínua

A Integração Contínua (IC) é uma prática de desenvolvimento de software na qual os membros de uma equipe de desenvolvimento integram seu trabalho com frequência, geralmente, diária ou até mais de uma vez por dia. Cada integração é verificada por um *build* automatizado, incluindo testes automatizados, para detectar erros de integração o mais rápido possível (Fowler, 2006).

Para possibilitar a execução da IC e acompanhar via interface gráfica, obtendo métricas em relação ao status dos testes, é necessário o uso de ferramentas, como é o exemplo da Hudson (Collins et al., 2014). A Figura 2.1 ilustra a prática descrita anteriormente. Além desta, também é ilustrado o versionamento de código, que é possibilitar o armazenamento do código fonte do software em um local onde todos os desenvolvedores possuem acesso. Neste exemplo, após a fase de implementação, o desenvolvedor insere o código produzido no controlador de versão, neste caso, o *Subversion*. O servidor de IC, Hudson, monitora o controlador de versão para a cada nova atualização executar o serviço de compilação do código, execução dos testes unitários, empacotamento do software e execução dos testes de tela com Selenium (Collins et al., 2014).

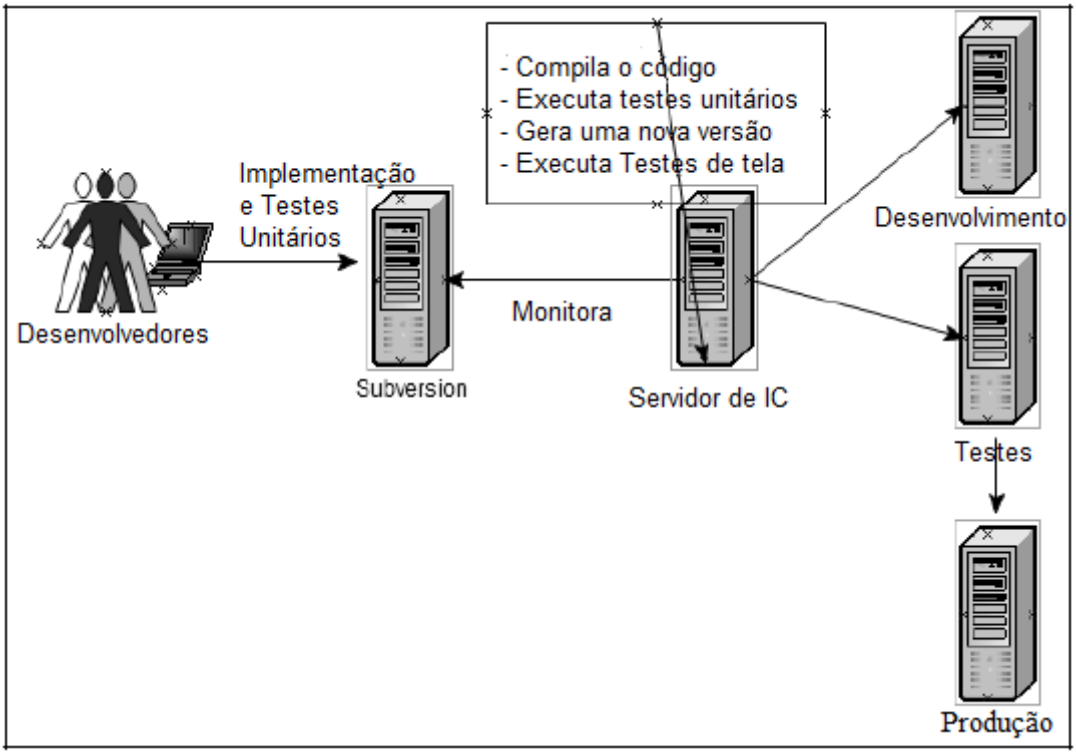


Figura 2.1: Estrutura da automação do *build* com integração contínua Adaptada de Collins et al. (2014)

2.5.3. Entrega Contínua

Com o intuito de simplificar a distribuição do software e permitir o ciclo de *feedbacks* mais curto dos clientes, a entrega contínua trata-se de um conjunto de práticas e princípios para atingir uma distribuição do software mais rápida e com mais frequência (Humble e Farley, 2010). A aplicabilidade desta prática depende da IC, pois a diferença entre estas se refere à disponibilização do software para a etapa subsequente do ciclo de desenvolvimento (testes, qualificação, operações).

2.5.4. Refatoração

A refatoração é definida como uma maneira disciplinada de melhorar o código e minimizar as chances de erros no software. Esta prática refere-se à melhoria da estrutura interna do software, sem, contudo, modificar o seu comportamento externo (Opdyke, 1992; Fowler et al., 1999). Como a aplicação da refatoração promove alterações no código, existe a possibilidade de gerar novos defeitos, portanto, é essencial a existência de testes

automatizados para garantir que novos erros não sejam introduzidos (Runeson, 2006; Vonken e Zaidman, 2012).

2.5.5. Revisão de Código

A revisão de código é uma prática em que o código produzido por um desenvolvedor é analisado por outro, para identificar possíveis erros e melhorias a serem realizadas. Devido à possibilidade de detecção de defeitos ocorrer durante a etapa de desenvolvimento, o custo para as correções é menor em relação a um defeito encontrado no software em produção (Sommerville, 2010; McConnell, 2004). Para estruturar as revisões e aumentar a rastreabilidade do código, é imprescindível a utilização de ferramentas (Rigby et al., 2012). A Tabela 2.2 lista algumas ferramentas e descreve as vantagens e desvantagens de cada uma.

Tabela 2.2: Comparação de algumas ferramentas de revisão de código (Rigby et al., 2012)

Ferramenta	**Vantagens**	**Desvantagens**
CodeCollaborator	Gera relatórios de métricas e integra com vários ambientes de desenvolvimento, como o Eclipse e Visual Studio	Possui taxa de licença comercial
Crucible	Integra com ferramentas da *Atlassian*	Possui taxa de licença comercial
ReviewBoard	Tem uma interface Web completa para revisão	Requer um servidor próprio para configurá-lo
Gerrit Rietveld	Apoia a revisão de código para projetos no *Subversion*. Por rodar na plataforma do *Google App Engine* é rápido e fácil de utilizá-lo	Requer hospedagem pública no *Google Code* ou configurá-lo em um servidor próprio
CodeStriker	Possui interface *Web* que suporta inspeção tradicional	Não possui suporte a técnicas de revisão mais leves

2.5.6. Análise de Qualidade de Código

O código fonte de um programa é geralmente a sua documentação mais atualizada. Ao mesmo tempo, o código fonte é o portador requintado do conhecimento, processos de negócios e metodologia acumulados ao longo de um período de tempo. A perda de qualidade de código acontece devido a muitas correções rápidas e pressão por não haver tempo para o desenvolvimento da melhor solução. Baixa qualidade de código resulta em aumento de custos de desenvolvimento e testes, e riscos operacionais (Bakota et al., 2012). Apesar disso, o

código fonte geralmente recebe tratamento hostil e é meramente considerado como uma ferramenta (Ferenc et al., 2014).

Ferenc et al. (2014) apresentaram uma ferramenta para gerenciar a qualidade de código, o SonarQube, o qual é um *software* de código aberto que permite gerar métricas de qualidade de código. Por exemplo: acoplamento, coesão, código duplicado, cobertura de testes unitários e padrões de desenvolvimento.

A diferença entre Revisão de Código e Análise de Qualidade de Código é que a Análise de Qualidade de Código é o processo automático de análise do código em relação à regras e padrões pré-estabelecidos e configurados em uma ferramenta de análise. Enquanto a Revisão de Código é a análise do código por uma outra pessoa. Deste modo, é possível que a Análise de Qualidade de Código auxilie na Revisão de código, uma vez que automaticamente encontre quais as regras de qualidade foram violadas.

2.5.7. Histórias de Usuários

Os métodos ágeis são baseados na rápida iteração em todo o processo de desenvolvimento de software a partir do levantamento dos requisitos até o lançamento de uma nova versão do software (Schwaber e Beedle, 2002; Beck, 2004; Schwaber e Sutherland, 2013). Assim, algumas técnicas para escrever os requisitos dos sistemas estão sendo usadas nos métodos ágeis.

Histórias de usuários são usadas para descrever uma característica do software e são compostas em três aspectos: descrição escrita, conversar com o cliente e testes (Cohn, 2004). A história do usuário representa uma característica sobre o que os clientes esperam encontrar no software (Beck, 2004). Cohn (2004) ainda salienta que a história de usuário é a unidade a partir da qual os recursos do software são estimados e desenvolvidos.

Cada história é formulada em uma ou duas frases e em linguagem apropriada ao cliente. Os detalhes são então clarificados nas conversas, e transportados e documentados em forma de testes (Haugen, 2006). Nas reuniões de planejamento, são escolhidas quais histórias de usuários serão implementas para a próxima versão do software; a escolha é feita pelo cliente. Nesta reunião, as histórias são estimativas individualmente e então é realizado o planejamento da próxima iteração (Beck e Fowler, 2001).

2.6. Ambiente de Desenvolvimento de Software

Por volta dos anos 70, pesquisadores e desenvolvedores de software perceberam a necessidade de processos e ambientes de apoio ao desenvolvimento de software para coordenar as atividades entre os desenvolvedores, bem como as ferramentas utilizadas na construção, manutenção e gerenciamento de projetos. (Brown, 1993). Devido a essa necessidade, foram criados os Ambientes de Desenvolvimento de Software (ADS) (em inglês, *Software Engineering Environment* - SEE) que visam proporcionar aos seus usuários um conjunto de funcionalidades e facilidades que tornam a tarefa de desenvolvimento de software menos árdua e mais organizada, integrando técnicas, processos, métodos e ferramentas (Oinas-Kukkonen e Rossi, 1999).

Um ADS auxilia aos desenvolvedores a correta utilização das técnicas de engenharia de software; para isso, fornece as ferramentas necessárias para apoiar o desenvolvimento de software, tais como: repositórios de dados, dispositivos de ativação das ferramentas do ambiente, controle de gerência das atividades a serem desenvolvidas e atribuição de atividades aos desenvolvedores (Oinas-Kukkonen e Rossi, 1999; Betemps, 2003).

Para fornecer suporte automatizado para desenvolvimento de software, uma das questões-chave é a integração de ferramentas (Bosua e Brinkkemper, 1995; Wasserman, 1990). Por ser um aspecto bastante amplo, a questão da integração em ADS é comumente dividida em dimensões elencadas a seguir (Thomas e Nejmeh, 1992; Wasserman, 1990):

- Integração de Dados: essa dimensão lida com o modo do ambiente e suas ferramentas compartilham dados;
- Integração de Controle: permite que o ambiente perceba quando e quais funcionalidades são executadas e também que seja capaz de realizá-las quando necessário;
- Integração de Processo: estabelece uma ligação explícita entre o processo de software definido e as ferramentas utilizadas no desenvolvimento. Permite que as ferramentas e os recursos definidos no processo se relacionem adequadamente;
- Integração de Apresentação: mantém as interfaces com o usuário do ambiente homogêneas e consistentes, possibilitando que os usuários alternem entre várias ferramentas sem mudanças substanciais de estilo e aparência.

Neste contexto, foi percebida a possibilidade de agregar à estrutura dos ADS um conjunto de recursos de apoio, à definição e execução de processos de software (Finkelstein et

al., 1994; Gimenes et al., 1999). Isso trouxe um novo conceito aos ADS, os Ambientes de Engenharia de Software Centrados em Processo (em inglês, *Process-centered Software Engineering Environments* - PSEE) (Finkelstein et al., 1994).

Os PSEE são compostos por três partes principais: motor de processo, interface com o usuário e repositório de dados (Bandinelli et al., 1996; Fuggetta, 1996) conforme descritos a seguir:

1. Motor de processo: responsável por invocar ferramentas e recuperar, manipular e armazenar artefatos do processo. O motor de processo deve possibilitar a especificação e descrição de como as atividades de desenvolvimento serão executadas, os papéis, e a atribuição de atividades aos membros da equipe de desenvolvimento, bem como utilização e controle das ferramentas de apoio ao desenvolvimento (Ambriola et al., 1997).
2. Interface com o usuário: responsável em apoiar o trabalho do usuário e sua iteração com o ambiente de forma uniforme.
3. Repositório de dados: responsável por armazenar os artefatos produzidos e manipulados durante o desenvolvimento, tornando possível o seu acesso por uma variedade de ferramentas integradas (Betemps, 2003). O repositório pode ser um banco de dados dedicado, um sistema de arquivo ou dicionário de dados (Notari, 2000; Oinas-Kukkonen e Rossi, 1999; Jarke et al., 1994).

Betemps (2003) identificou os requisitos necessários para um ambiente com apoio ao processo de software através da análise dos trabalhos de Perin (1992), Finkelstein et al. (1994), Gimenes (1994) Ortigosa (1995), Callahan e Ramakrishnam (1996), Chan e Leung (1997), Chan (1999), Maurer et al. (1999), Barnes e Gray (2000), Notari (2000) e Manzoni (2001). Esses requisitos incluem:

- **Suporte ao Gerenciamento**: possibilitar a recuperação de informações do projeto como: os responsáveis por cada atividade, data de início e fim das atividades e o tempo gasto com elas. Além disso, deve dar visibilidade aos projetos em andamento, controle do projeto por parte dos gerentes, auxiliar o controle do acesso aos dados e atribuir tarefas para as pessoas;
- **Suporte aos Eventos do Processo**: apoiar os eventos do processo como: notificar os responsáveis sobre cada atividade para cada evento, possibilitar que as informações necessárias para execução das atividades sejam encontradas, propiciar a coleta das informações geradas como resultado da execução da atividade pelo usuário;

- **Flexibilidade**: possibilitar que o ambiente utilize diversas metodologias de desenvolvimento e se adeque à realidade da empresa. Ademais, deve apoiar alterações no processo de maneira dinâmica;
- **Extensibilidade**: o ambiente deve permitir a integração de novas ferramentas para possibilitar a inclusão de novas funcionalidades, à medida que novas ferramentas surgem;
- **Reusabilidade**: o ambiente deve possibilitar o reuso de artefatos durante o processo de desenvolvimento de software, uma vez que isso é relevante para questões de qualidade e redução de custos e esforços;
- **Colaboração**: um ADS deve apoiar a integração entre diversos usuários que podem não estar localizados em um mesmo local durante o desenvolvimento;
- **Fácil de Usar**: o ambiente deve possibilitar a sua utilização com necessidades mínimas de treinamentos;
- **Monitoramento**: deve possibilitar o monitoramento e análise dos dados gerados para permitir o uso efetivo dos recursos do ADS;
- **Consistência**: deve suportar a consistência dos dados para permitir o acesso e atividades por múltiplos usuários;
- **Verificação**: deve possibilitar a configuração para que diferentes ferramentas possam ser ativadas durante a execução do processo de desenvolvimento;
- **Integração de Controle**: o ambiente deve possibilitar a configuração para que diferentes ferramentas possam ser ativadas durante a execução do processo de desenvolvimento;
- **Integração de Dados**: deve permitir o compartilhamento das informações e artefatos gerados de diferentes fontes e ferramentas em um banco de dados comum com um pequeno esforço de conversão e um alto grau de consistência.

2.6.1. PSEE Existentes

Nesta seção, serão descritos os PSEE EPOS, SPADE e OIKOS, pois são considerados representantes significativos das pesquisas realizadas no contexto de ADS (Finkelstein et al., 1994).

2.6.1.1. EPOS

A ênfase do EPOS está na assistência de um processo flexível e em evolução para desenvolvimento e manutenção de software para vários desenvolvedores de software. Os modelos de processo do EPOS são expressos em SPELL, uma PML orientada a objetos, concorrente e reflexiva. Os processos e artefatos são armazenados no EPOS-DB, um banco de dados de engenharia de software. O EPOS possibilita a modelagem do processo por meio de uma ferramenta gráfica, além de propiciar modificações durante a sua execução.

A integração por controle no EPOS é realizada pela especificação de procedimentos que possibilitam as integrações de ferramentas que são realizadas no SPELL. Além disso, o EPOS utiliza a ferramenta BMS, auxiliar no controle das ferramentas e para integrar novas ferramentas ao ambiente.

O EPOS-DB armazena o modelo de processo e mantém um contexto de transação em versões. Acrescenta-se ainda que os artefatos gerados no EPOS são armazenados como arquivos sob controle total do banco de dados. Outro aspecto não menos relevante é que o banco de dados não sabe sobre o conteúdo interno dos arquivos, portanto, não existe integração semântica de dados.

2.6.1.2. SPADE

O projeto SPADE tem como conceito principal a adoção de redes de Petri como paradigma de PML. Além disso, inclui um banco de dados orientado a objetos comercial. A integração de ferramentas é realizada por meio de um mecanismo auxiliar, o DEC FUSE. O ambiente também oferece um utilitário, chamado EnCASE, para integrar novas ferramentas no ambiente.

A definição e evolução do processo no SPADE é realizada pela PML SLANG. O SLANG possibilita que os modelos de processo sejam tratados como dados de processo e, assim, a evolução do processo pode ser descrita como parte do processo. Salienta-se ainda que o SLANG possibilita a definição de papéis de cada usuário e a integração por controle, o qual permite especificar as ferramentas que devem ser ativadas para realização de cada atividade definida no processo.

Em relação à integração por dados, o SPADE define um repositório centralizado orientado a objetos, implementa e usa o OODBMS, para armazenamento de todos os modelos

e artefatos de processo. Logo, a integração de dados é conseguida compartilhando objetos O2. A recuperação de informações no SPADE é realizada por meio de repositório de dados.

2.6.1.3. OIKOS

O PSEE OIKOS usa Inteligência Artificial, o qual é realizado por meio das linguagens Limbo e Patè como PML, que são executadas por um interpretador totalmente integrado na arquitetura do OIKOS. Uma ferramenta gráfica possibilita a modelagem do processo e também alterações dinâmicas.

A arquitetura do OIKOS é baseada em uma máquina virtual, a Expo 2.0, que permite a execução de programas Patè, a integração de ferramentas externas e a interação com os usuários por uma interface gráfica. A máquina virtual do OIKOS permite que novos componentes possam ser conectados, na qual a interação com outros componentes é realizada através da passagem de mensagens. Ao contrário de outros PSEE, o OIKOS não explora uma ferramenta de integração de ferramentas comerciais e é baseado em um protocolo de comunicação personalizado.

O banco de dados OIKOS é centralizado. A camada de serviço abstrai essa visão, permitindo a criação de um conjunto de repositórios independentes. Cada repositório é controlado por um serviço que fornece operações para mover documentos do repositório para o espaço de trabalho, e vice-versa, para bloquear documentos e para inspecionar o estado do repositório.

2.7. Gerenciamento do Ciclo de Vida da Aplicação

Conforme Rossman (2010), Gerenciamento do Ciclo de Vida da Aplicação (do inglês, *Application Lifecycle Management* - ALM) é o processo contínuo de gestão do ciclo de vida da aplicação por meio da governança, do desenvolvimento e da manutenção desde sua concepção até sua manutenção. O ciclo inicia-se desde a definição dos requisitos, desenvolvimento, implantação até a manutenção (Lacheiner e Ramler, 2011; Lee, 2004). A manutenção visa ao desenvolvimento de novas funcionalidades, mediante a identificação de novos requisitos e a correção de erros não identificados anteriormente; o que leva à primeira fase (definição de requisitos) e cria um ciclo de atividades que deverão ser executadas durante todo o tempo de vida útil da aplicação na organização (Baessa, 2011). Como pode ser

observado na Figura 2.2, Chappel (2014) subdivide a ALM em três áreas distintas, conforme definidas a seguir.

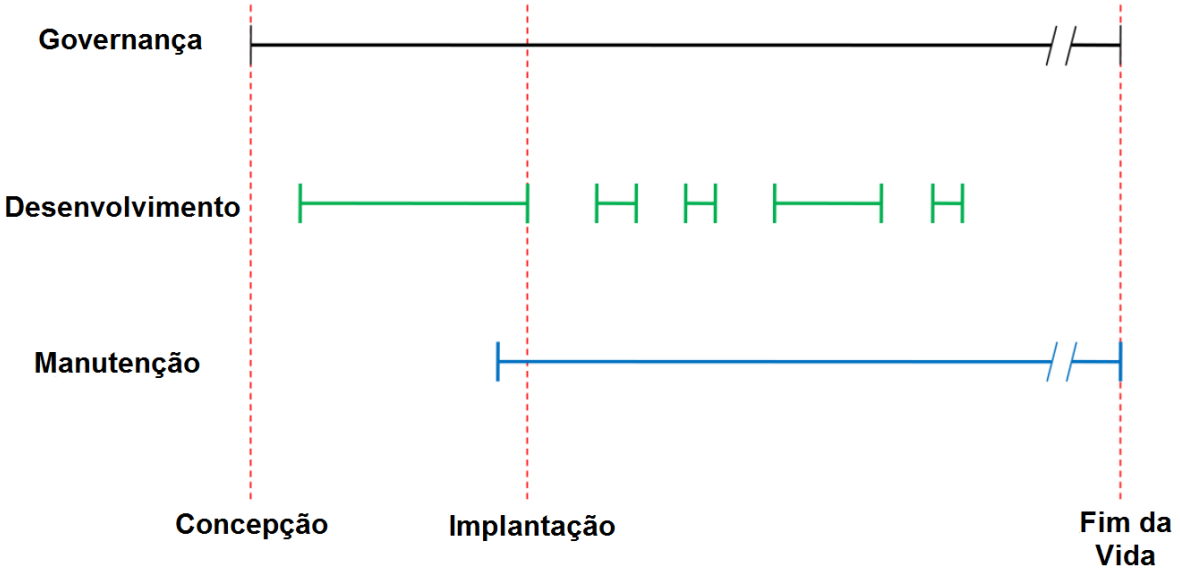


Figura 2.2: Divisão do ALM (Chappel, 2014)

1. Governança: abrange todo o processo de decisão e o gerenciamento da aplicação. Desde a concepção da ideia do software, passando pelo gerenciamento do desenvolvimento, até a gestão da aplicação em produção.
2. Desenvolvimento: consiste em efetivamente criar o software. Após o software ser implantado e estar na área de manutenção, podem ocorrer novos ciclos de desenvolvimento, uma vez que pequenas atualizações podem ser necessárias ou até mesmo a identificação de novas funcionalidades.
3. Manutenção: consiste em monitorar o software implantado após a etapa de desenvolvimento.

Kovair (2016) define a fase de desenvolvimento do software, de requisitos até a geração de uma nova versão do sistema, como *Software Development Life Cyle* (SDLC). Enquanto o ALM trata de uma perspectiva mais ampla do ciclo de vida do software, desde o planejamento inicial do sistema, desenvolvimento, manutenção do software em produção até a sua retirada de funcionamento, quando não possui mais valor para o negócio. Todavia, o SDLC é limitado apenas aos ciclos de desenvolvimento de software: requisitos, projeto, codificação, testes e gerenciamento do desenvolvimento do software (Kovair, 2016). O SDLC pode ocorrer uma ou mais vezes no ALM, pois novas funcionalidades podem surgir enquanto o software está na área de manutenção.

Com o uso de ALM, a qualquer momento, os envolvidos em projeto de software podem visualizar o estado do fluxo de trabalho. Os itens desde requisitos até manutenção estão visíveis para análise (Kim et al., 2011). Kääriäinen e Välimäki (2008) relatam que os principais elementos do ALM são: 1) criação e gestão dos artefatos gerados; 2) rastreabilidade dos artefatos durante o ciclo de vida; 3) relatórios de artefatos de ciclo de vida; 4) automação de processos; 5) integração das ferramentas; 6) comunicação.

O objetivo é oferecer previsibilidade, visibilidade, rastreabilidade, eficiência no desenvolvimento, automação de processos e entrega de software apoiados pela integração das atividades do ciclo de desenvolvimento (Rossberg, 2008; Kravchik, 2009). Logo, a integração entre as ferramentas é a forma mais eficaz de realização destas atividades.

Rossman (2010) identifica alguns benefícios do ALM:

- Aceleração do desenvolvimento através da integração simplificada;
- Maximização de investimentos em conhecimento e qualificações, processos e tecnologias;
- Aumento da flexibilidade pela redução do tempo necessário para construir e adaptar as aplicações que suportam novas iniciativas de negócios.

Os ADS surgiram por volta dos anos 70 (Brown, 1993) e posteriormente, o ALM, e os dois se assemelham na maioria dos seus objetivos. A Tabela 2.3 objetiva ressaltar as diferenças entre os objetivos dos dois conceitos, com base no que foi apresentado na Seção 2.6 e na Seção 2.7.

Tabela 2.3: Diferenças entre ADS e ALM

	ADS	ALM
Integração de processo	O processo guia o ADS realizando o controle entre as ferramentas. A definição e especificação do processo é realizada por meio do motor de processo.	Possibilita a automação de processos, porém ele não realiza o controle das ferramentas.
Integração de dados	Um repositório centralizado para armazenar e gerenciar todos os artefatos produzidos e manipulados durante o desenvolvimento de software.	Não possui um repositório centralizado para armazenamento dos artefatos produzidos. O compartilhamento dos dados é realizado por meio de comunicação entre as ferramentas.
Gestão dos artefatos gerados	Realizado por meio da integração de dados.	Cada ferramenta armazena os seus artefatos, possibilitando o compartilhamento entre elas quando necessário.

Nesse sentido, o ALM abrange um conceito mais amplo em relação ao gerenciamento do ciclo de vida do software e não apenas do desenvolvimento do software. O ALM não inicia apenas na fase de requisitos, mas desde o gerenciamento da ideia de negócio, risco e planejamento, até quando o software não possuir mais valor de negócio.

2.8. Considerações Finais

Este capítulo apresentou os fundamentos teóricos necessários para o desenvolvimento desta pesquisa. Neste sentido, acerca dos temas aqui apresentados e revisados, a compreensão sobre ADS, ALM, as práticas de desenvolvimento de software em MA e o estado dos MA nas organizações, auxiliam a compreensão da proposta desta pesquisa. Portanto, a viabilidade de propor o ADE torna-se interessante, pois poucos trabalhos sobre definição de ambientes de desenvolvimento de software em projetos ágeis foram identificados na literatura.

O próximo capítulo apresenta o ambiente ágil de desenvolvimento baseado em *Scrum*, que é desenvolvido nesta pesquisa de mestrado e, posteriormente, a avaliação desta proposta.

3

Ambiente Ágil de Desenvolvimento

3.1. Considerações Iniciais

Este capítulo apresenta o projeto de um Ambiente Ágil de Desenvolvimento (ADE). Ele foi concebido para facilitar a utilização sistemática de práticas e ferramentas ágeis em um ambiente de desenvolvimento integrado baseado em *Scrum*.

Dessa forma, este capítulo discorre inicialmente sobre os princípios de projeto do ADE, a sua arquitetura, a descrição do processo do ADE, e por fim, a integração dos elementos.

3.2. Princípios

O projeto do ADE adotou os seguintes princípios iniciais: o método de gerenciamento de processos ágeis Scrum (Schwaber e Sutherland, 2013); o ciclo de desenvolvimento ALM (Chappel, 2014) e as dimensões de integração de ferramentas de engenharia de software definidas por Thomas e Nejmeh (1992), conforme justifica-se a seguir:

- **Scrum**: foi adotado como método de gerenciamento de projetos por ser o método mais utilizado atualmente pela indústria de desenvolvimento de software (Melo et al., 2012; VersionOne, 2015). Este método tem apresentado vantagens em sua aplicação que possibilitam alcançar controle sobre o desenvolvimento, priorização de recursos e estimativas realistas (Striebeck, 2006).

- **ALM**: os seus conceitos foram eleitos para compor o ADE, pois oferecem a gestão do ciclo de vida da aplicação no âmbito de governança, desenvolvimento e manutenção (Chappel, 2014; Rossman, 2010). Além de ter também como base os princípios do ALM afirmados por Kääriäinen e Välimäki (2008): 1) criação e gestão dos artefatos gerados; 2) rastreabilidade dos artefatos durante o ciclo de vida; 3) relatórios de artefatos de ciclo de vida; 4) automação de processos; 5) integração das ferramentas; 6) comunicação. Salienta-se ainda que o ALM possibilita os envolvidos em um projeto de software visualizar continuamente o estado do fluxo de trabalho pela integração entre as ferramentas no ambiente (Kim et al., 2011). Ademais, o ALM possibilita o gerenciamento ágil, com a integração das ferramentas de gestão e desenvolvimento (Franky, 2011) para atingir a convergência inteligente definida por Fuggeta e Di Nitto (2014).
- **Integração de ferramentas de engenharia de software**: com a finalidade de promover a utilização sistemática de práticas e ferramentas ágeis, é essencial oferecer o máximo de automatização possível no processo de desenvolvimento, bem como a integração entre ferramentas (Kravchik, 2009). Serão consideradas as dimensões de integração de ferramentas de engenharia de software: dados, controle e processo (Thomas e Nejmeh, 1992). Justifica-se essas dimensões de integração por também estarem presentes nos requisitos para um ADS (Betemps, 2003), conforme apresentado no Capítulo 2.

As práticas que compõem o ADE foram selecionadas por serem comumente usadas pela indústria (VersionOne, 2015; Melo et al., 2012; Abrantes e Travassos, 2011), conforme apresentadas no Capítulo 2. São elas:

- Gerenciamento de projeto;
- Controle de versão de código;
- Revisão de código;
- Integração contínua;
- Análise de qualidade de código;
- Testes automatizados.

3.3. Elementos do ADE

Esta seção apresenta os elementos que compõem o ADE. Esses elementos apóiam as práticas selecionadas na Seção 3.2. São eles:

1. Gerenciador de projeto: responsável por realizar o gerenciamento das atividades de desenvolvimento, além do gerenciamento do projeto. Assim permite a definição do processo de desenvolvimento, viabiliza a recuperação de informações do projeto, proporciona a visibilidade dos que estão em andamento, atribui tarefas para as pessoas envolvidas no projeto, notifica os responsáveis por cada atividade para cada evento, possibilita que as informações necessárias para execução da atividade sejam encontradas, e ainda propicia o controle do projeto por parte dos gerentes. Para cada funcionalidade a ser desenvolvida, é criada uma tarefa no gerenciador de projeto, que possibilita fragmentá-la em tarefas menores. Através das tarefas no gerenciador de projeto, as seguintes funcionalidades são disponíveis:
    a. Rastreabilidade do código: é possível identificar todo código desenvolvido para cada funcionalidade através dos *commits*, que são os envios de alterações no código fonte para o controlador de versão, realizados no controlador de versão. Uma vez que cada *commit* possui o identificador da funcionalidade existente no gerenciador de projeto.
    b. Armazenamento de documentos: é possível armazenar os documentos gerados durante o ciclo de desenvolvimento de cada funcionalidade.
    c. Estimativas: permite inserir a estimativa definida para cada tarefa da funcionalidade e da própria funcionalidade.
    d. Apontamento de horas: permite o controle de horas consumidas para o desenvolvimento de cada funcionalidade.
    e. Priorização das funcionalidades: permite priorizar as funcionalidades criadas dentro do gerenciador de projeto. Possibilita até mesmo uma ordenação de precedência entre elas.
2. Controlador de versão: responsável pelo versionamento do código fonte do software para apoiar o rastreamento das alterações realizadas e a recuperação de versões anteriores.
3. Revisor de código: responsável por estruturar as revisões e possibilitar a identificação de possíveis erros e melhorias de códigos a serem realizadas

(Sommerville, 2010). A utilização de ferramentas para apoiar a revisão de código aumenta a rastreabilidade de erros e possibilita melhorias ao código (Rigby et al., 2012). Além disso, quando os erros são encontrados durante o desenvolvimento, o custo para as correções é menor, em relação a um defeito encontrado quando o software está em produção (Sommerville, 2010; McConnell, 2004).

4. Integração contínua: responsável por integrar o trabalho dos desenvolvedores frequentemente, uma vez que pode ser configurado para cada alteração no controlador de versão disparar uma execução de IC. Para cada integração, o serviço de integração continua busca o código fonte do software e após isso é verificado se existe algum erro de compilação através do *build* automatizado. Além disso, são executados os testes automatizados, o que proporciona a detecção de erros de integração de forma contínua (Fowler, 2006); não somente, mas também em cada integração uma nova versão do software é gerado.
5. Analisador de qualidade de código: responsável por monitorar o código fonte para produzir métricas de: acoplamento, coesão, código duplicado, cobertura de testes unitários e padrões de desenvolvimento. A visualização dessas métricas possibilita a análise e a tomadas de decisões para corrigir e adequar os problemas.
6. Testes automatizados: no ADE os testes automatizados não são considerados como serviços. Porém o ADE define duas etapas de testes:
    a. Testes automatizados: podem ser divididos em testes unitários e testes funcionais;
    b. Testes manuais: são os testes de aceitação de cada funcionalidade desenvolvida.

Neste ponto, o ALM tem por objetivo coordenar o uso sistemático dos elementos definidos no ADE. Isso é feito pela integração entre os elementos e automatização das etapas de desenvolvimento. A articulação dos elementos definidos, nesta seção, corrobora com o âmbito do ALM definido por (Chappel, 2014; Rossman, 2010):

- Governança: todo o gerenciamento do projeto, como gerenciamento dos requisitos, definição do processo de desenvolvimento, gerenciamento de horas gastas para cada funcionalidade e rastreabilidade de alterações por funcionalidade são realizadas pelo gerenciador de projeto. Tudo o que diz

respeito ao gerenciamento do projeto é definido dentro deste elemento, e dessa forma, possibilita a visualização e acompanhamento de todo o ciclo de vida da aplicação a qualquer momento.

- Desenvolvimento: para o desenvolvimento do software, usa-se todos os elementos definidos no ADE. O controlador de versão é integrado com o gerenciador de projeto para permitir a rastreabilidade de todo o código desenvolvido para cada funcionalidade. O revisor de código é integrado com o controlador de versão. Por fim, antes da finalização do desenvolvimento de uma funcionalidade, o elemento de IC executa os testes automatizados a cada *commit* e verifica problemas de compilação que possam ter sido introduzidos pelo *commit*. Ao final de cada execução realizada pelo elemento de IC, também é disparada a análise de qualidade de código e, se desrespeitado alguma regra de qualidade definida previamente, é criada uma nova tarefa, automaticamente pelo ADE, no gerenciador de projeto para que sejam feitas as correções necessárias.
- Manutenção: Consiste no acompanhamento do projeto em produção no cliente. A partir do momento da entrada da primeira versão do software em produção, é responsabilidade da área de manutenção monitorá-lo e realizar o processo de entrada de novas versões em produção. Para eventuais correções de problemas ou novas funcionalidades solicitadas, inicia-se o ciclo de desenvolvimento novamente. Tanto para problemas quanto para novas funcionalidades, deve-se criar uma nova tarefa no gerenciador de projeto. O ADE auxilia na área de manutenção através do elemento de IC, que a cada integração, gera uma nova versão do software, e assim realiza o processo de disponibilização de novas versões em produção.

A Tabela 3.1 exemplifica qual elemento está presente em cada âmbito do ALM.

Tabela 3.1: Elementos do ADE associados as áreas do ALM

Área do ALM	**Elementos**
Governança	Gerenciador de projeto, Controlador de versão, Integração contínua
Desenvolvimento	Gerenciador de projeto, Controlador de versão, Integração contínua, analisador de qualidade de código e Revisor de código
Manutenção	Gerenciador de projeto, Integração contínua

3.3.1. Arquitetura Lógica

A arquitetura lógica do ADE tem por objetivo mostrar os elementos que estão contidos no ADE. A integração entre todos os elementos disponíveis no ambiente é essencial para que possa automatizar o desenvolvimento do *software* e gerenciamento do projeto. A Figura 3.1 apresenta uma visão geral da arquitetura lógica do ADE.

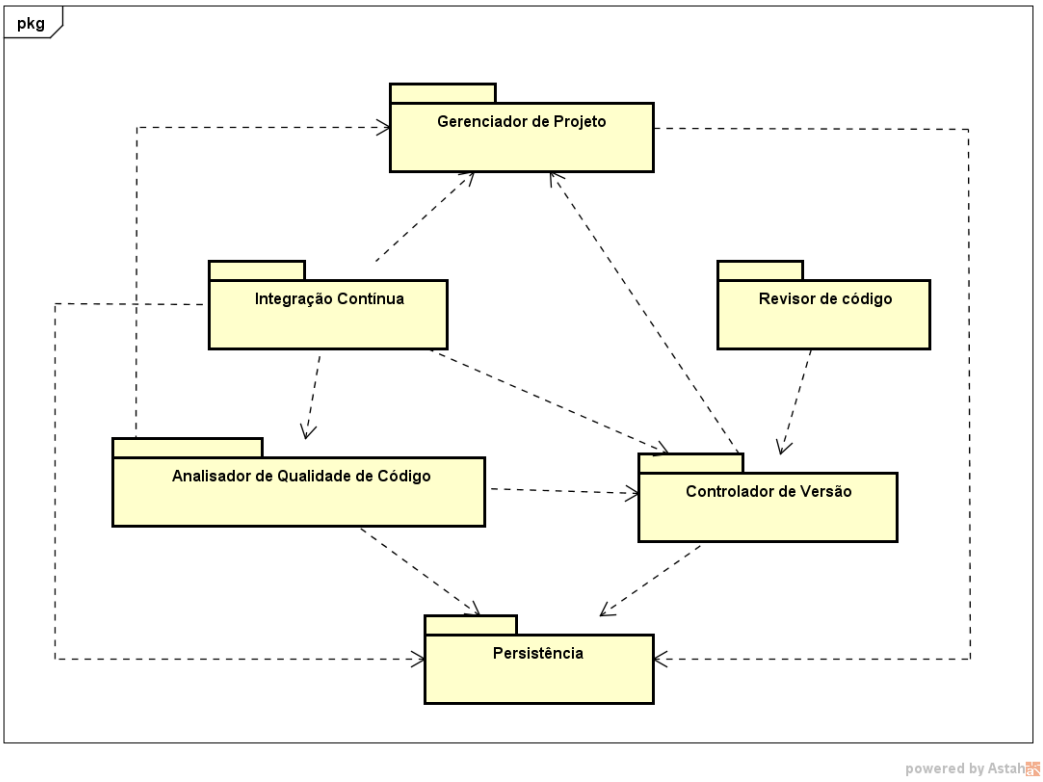


Figura 3.1: Visão geral da arquitetura lógica do ADE

A arquitetura foi definida com o objetivo de automatizar o ambiente juntamente com o processo de desenvolvimento, possibilitando a integração entre os elementos do ambiente e permitindo a construção de artefatos de software a cada alteração no controlador de versão. Dessa forma, corrobora com o método de gerenciamento, uma vez que, a cada *Sprint* uma nova versão do software deve ser gerada; soma-se a isso, o objetivo do ALM em realizar acompanhamento de todo o ciclo de vida da aplicação e visualização a qualquer momento. Na próxima seção, será definido o processo de desenvolvimento e como o ADE se integra com ele.

3.4. Processo de Desenvolvimento

O processo de desenvolvimento do ADE define a sequência das etapas a serem seguidas por uma equipe de desenvolvimento de software, para conceber e gerar uma aplicação, com base nos recursos disponíveis em ADE.

A Figura 3.2 apresenta o processo de desenvolvimento do ADE representado graficamente por um diagrama de atividades UML. Esta representação contém as etapas do processo e as informações manipuladas por elas, conforme descritas a seguir.

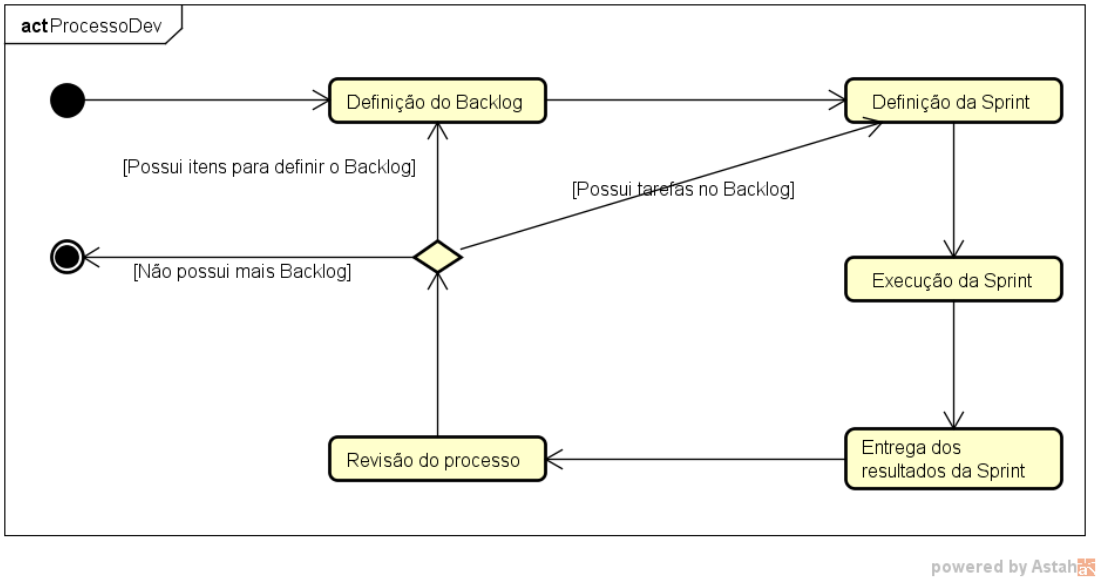


Figura 3.2: Representação gráfica do processo de desenvolvimento do ADE

Etapa 1 – Definição do *Backlog*

A etapa de definição do *Backlog* contém três subetapas: análise, criação do *Backlog* e priorização do *Backlog*. A primeira subetapa, a análise, visa compreender o problema para

que o PO tenha a visão das funcionalidades necessárias para resolução do problema. O resultado da análise é estabelecido em histórias de usuário, e cada história representa uma funcionalidade. Deve-se representar as histórias de usuário no gerenciador de projeto para permitir a gestão do desenvolvimento do software. Dentro do gerenciador, pode-se chamar uma história de usuário de tarefa e, além disso, as histórias podem ser fragmentadas em subtarefas. Em seguida, é possível conceber o *Backlog* considerando as histórias definidas. Por fim, deve-se realizar a priorização do *Backlog* de acordo com as definições feitas pelo cliente e a ordem de precedências entre as histórias.

Etapa 2 - Definição da *Sprint*

Nesta etapa, é realizada a reunião de planejamento da *Sprint*, na qual o PO, o *Scrum Master* e time de desenvolvimento selecionam as tarefas que irão compor o *Backlog* da *Sprint*. Através do gerenciador de projeto, são identificadas as tarefas que já foram analisadas e priorizadas pelo PO, na etapa anterior, para serem estimadas por meio da prática de *Planning Poker*. Cada tarefa selecionada é atualizada com a estimativa dada pelo time de desenvolvimento no gerenciador de projeto. Além disso, o time de desenvolvimento define como serão realizadas as implementações de cada tarefa e quem realizará. Ao final desta etapa, tem-se o *Backlog* da *Sprint*, o qual define as funcionalidades que serão entregues ao cliente ao final da *Sprint*.

Etapa 3 - Execução da *Sprint*

Nesta etapa, o *Backlog* da *Sprint* é implementado. Para o acompanhamento da *Sprint*, são realizadas reuniões diárias entre o time de desenvolvimento e o *Scrum Master* com o objetivo de identificar e corrigir possíveis problemas que podem atrapalhar a entrega da *Sprint*. Para a implementação de cada tarefa do *Backlog*, cinco passos devem ser executados. Os passos definidos para a implementação de cada tarefa estão representados na Figura 3.3, por meio de um diagrama de atividades UML.

1. Implementação: é realizada a codificação do software, de acordo com as decisões tomadas na etapa 2;
2. Testes Automatizados: este passo é responsável por realizar os testes automatizados. A transição entre os passos de implementação e testes automatizados deve ocorrer até que a atividade seja concluída, pois uma maior cobertura de teste aumenta a qualidade do código e diminui a geração de defeitos no *software* desenvolvido (Maximilien e Williams, 2003; George e Williams, 2004);

3. Testes Manuais: após a conclusão dos passos de implementação e testes automatizados, deve-se realizar testes manuais da funcionalidade adicionada. O ADE pode apoiar a identificação das funcionalidades afetadas por meio da ferramenta de revisão de código, na qual são destacados os trechos de código alterados;
4. Revisão de código: todo o código desenvolvido deve ser revisado por um profissional diferente para aumentar a probabilidade de identificar possíveis erros e melhorias a serem realizadas e assim diminuir o número de falhas do software produzido. O ADE apoia isso através do gerenciador de projeto por meio do processo definido; assim antes de finalizar uma tarefa, é necessário que a revisão de código seja realizada por um outro profissional que possui permissão para aprovar o código. Além disso, o serviço de revisão de código auxilia na execução deste passo ao destacar os trechos de código alterados e pelo analisador de qualidade de código, que retorna quais as regras de qualidade foram violadas;
5. Integração contínua: o código produzido deve ser integrado com os demais códigos que comporão o novo artefato. O processo de integrar o código, executar os testes automatizados, detectar os causadores dos erros quando houver e gerar uma nova versão executável do software, são funções da etapa de IC. Este passo é realizado automaticamente pela ferramenta de IC configurada no ADE. Este executável pode ser entregue para o cliente ao final da *Sprint*. Além dos itens descritos, o servidor de integração contínua também disparará o analisador de qualidade de código para realizar as análises de cobertura de testes, quantidade de duplicações, complexidade do software, quantidade de comentários, dívida técnica e padrões de códigos. Os resultados das análises devem ficar visíveis para toda a equipe de desenvolvimento, além de ser usado na etapa 5, revisão do processo. Caso as regras de qualidade sejam quebradas, o IC dispara a criação automática de uma tarefa no gerenciador de projeto, para que seja corrigido o código gerador das quebras de qualidade.

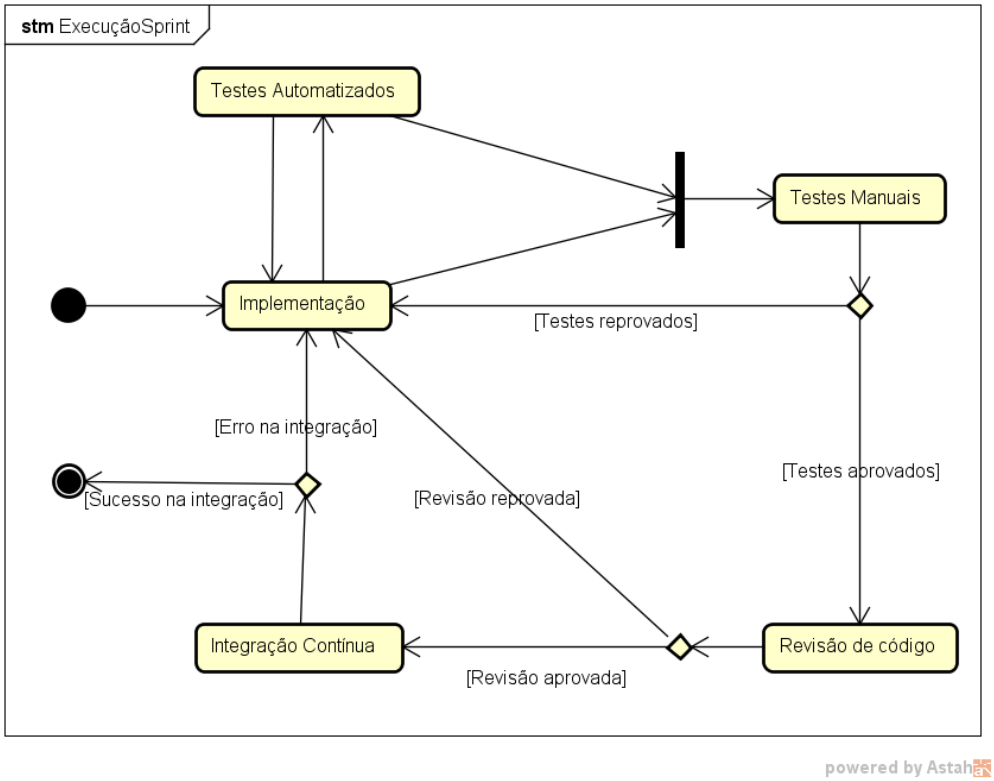


Figura 3.3: Representação gráfica da etapa Execução da Sprint

Etapa 4 - Entrega dos resultados da *Sprint*

Ao final da *Sprint*, uma nova versão do software com as novas funcionalidades deve ser gerada. Isso é feito pela integração contínua, a partir do último *commit* da *Sprint*. Assim, é realizada a entrega para o cliente e apresentadas as funcionalidades contidas nesta nova versão. As funcionalidades são identificadas por meio das tarefas finalizadas na *Sprint*, pelo gerenciador de projeto.

Etapa 5 - Revisão do processo

A última etapa do processo do ADE é realizada a retrospectiva da *Sprint* e assim permite que o PO, o *Scrum Master* e o time de desenvolvimento revisem os problemas encontrados na execução da *Sprint*, como problemas na comunicação com o cliente, nos requisitos, falha no processo, estouro das estimativas para as tarefas, aumento da dívida técnica, entre outros problemas que podem surgir no decorrer da *Sprint*. Por conseguinte, possibilita inspecionar e adaptar o processo para a próxima *Sprint*. Alguns dos problemas discutidos nesta etapa podem ser obtidos, pelos membros do time de desenvolvimento, *Scrum Master* e PO, através dos elementos ADE. A Tabela 3.2 exemplifica-os.

Tabela 3.2: Elemento do ADE no qual é possível recuperar informações sobre problemas

Elementos do ADE	**Problemas/Dificuldades**
Gerenciador de projeto	Falha na estimativa, problemas nos requisitos, rastreabilidade dos *commits*, falha na execução do processo
Analisador de qualidade de código	Dívida técnica, cobertura de testes, problemas de *design* do código, código fora dos padrões
Integração contínua	Quais *commits* causaram falhas no software

Ao final da etapa 5, caso ainda existam funcionalidades no *backlog*, uma nova *Sprint* é iniciada voltando à etapa 1. A existência de funcionalidades no *backlog* é encontrada no elemento gerenciador de projeto, onde possui todas as funcionalidades mapeadas em forma de tarefas, e caso existam o PO deve iniciar uma nova *Sprint*.

3.5. Integração no ADE

A Tabela 3.3 demonstra quais os elementos no ADE possuem integração entre si. São especificadas as integrações em relação às dimensões definidas por Thomas e Nejmeh (1992) por dados, por controle e por processo.

Tabela 3.3: Tipos de integrações entre os elementos do ADE

	Gerenciador de projeto	Controlador de versão	Revisor de código	Integração Contínua	Analisador de qualidade de código	
Gerenciador de projeto		X				Dados
		X		X	X	Controle
				X		Processo
Controlador de versão			X	X	X	Dados
				X	X	Controle
				X		Processo
Revisor de código						Dados
						Controle
						Processo
Integração Contínua					X	Dados
					X	Controle
					X	Processo
Analisador de qualidade de código						Dados
						Controle
						Processo

A integração por dados no ADE é realizada por meio de serviços *Web*; assim cada ferramenta que necessita de dados produzidos ou armazenados em outra ferramenta faz uma solicitação quando necessário. Assim, é responsabilidade de cada serviço interpretar dos dados disponíveis por outro serviço. Esta integração é transparente ao usuário, uma vez que já existem *plug-ins* para a integração entre as ferramentas e basta configurá-los. Paralelamente, ainda são disponibilizados os serviços *Web* para implementações mais específicas quando necessário. Não só o compartilhamento de dados é realizado; também é possível realizar alterações nos dados de outras ferramentas e inserir novos dados, através de serviços *Web*. Cada ferramenta possui seu mecanismo de armazenamento, seja ele um banco de dados ou sistema de arquivos. Dessa forma, não existe um repositório central para todos os artefatos gerados durante o desenvolvimento do software.

Em relação à integração de controle, ocorre pelas configurações nas ferramentas para que possibilite a ativação de outra ferramenta. Esta configuração não é realizada através da modelagem do processo no ADE, uma vez que ele apenas define o fluxo de trabalho e não possui o controle sobre as ferramentas. A Tabela 3.4 exemplifica as integrações por controle existentes no ADE.

Tabela 3.4: Descrição das integrações por controle no ADE

Elemento do ADE	**Caso de integração por controle**
Analisador de qualidade de código	Cria atividades no gerenciador de projeto quando quebrar as regras de qualidade
Controle de versão de código	Notifica o servidor de integração contínua que houve uma alteração no repositório
Integração contínua	Executa os testes automatizados, Ativa o analisador de qualidade de código, Cria atividades no gerenciador de projeto quando uma execução falhar

Em relação à integração por processo, o ADE não é baseado explicitamente por meio do processo; a definição dele no ADE é realizada dentro do gerenciador de projetos, e tem o objetivo de definir as etapas que devem ser realizadas para o desenvolvimento do software. Porém, o processo não define um controle em relação às ferramentas ou uma definição automatizada para que as demais ferramentas do ambiente sejam executadas apenas pela definição do processo. As alterações no processo de desenvolvimento podem ser realizadas de maneira dinâmica pelo gerenciador de projeto.

3.6. Exemplo de um Ambiente

Nas seções anteriores foram descritos os elementos do ADE, a arquitetura lógica, o processo de desenvolvimento e as integrações presentes entre os elementos. Para a instanciação do ADE, existem diversas ferramentas que podem ser selecionadas. Assim, esta seção tem o objetivo de exemplificar uma instanciação do ADE.

Para a seleção das ferramentas foi realizado uma pesquisa exploratória com o objetivo de analisar os projetos de software na indústria para reconhecer as ferramentas mais usadas pela indústria, praticantes de métodos ágeis, e entender como elas podem ser coordenadas para que possibilitam a integração das práticas ágeis. Os resultados desta pesquisa estão disponíveis no Apêndice A. Além dos resultados da pesquisa também foram consideradas as ferramentas encontradas na revisão bibliográfica, como são os casos do SonarQube (Ferenc et al., 2014), Gerrit (Rigby et al., 2012) e Redmine (VersionOne, 2015). A Tabela 3.5 apresenta qual ferramentas corresponde a qual elemento do ADE.

Tabela 3.5 Ferramentas correspondentes aos elementos do ADE

Elemento do ADE	**Ferramenta**
Gerenciador de projeto	Redmine
Controlador de versão	Git
Revisor de código	Gerrit
Integração contínua	Jenkins
Analisador de qualidade de código	SonarQube

Dessa forma, para cada elemento do ADE foi definido uma ferramenta correspondente. A Figura 3.4 demonstra a relação entre cada ferramenta. Os relacionamentos são equivalentes aos apresentados na Figura 3.1 a qual representa as integrações entre os elementos. Assim pode se observar as integrações existentes entre as ferramentas neste exemplo de instanciação do ADE.

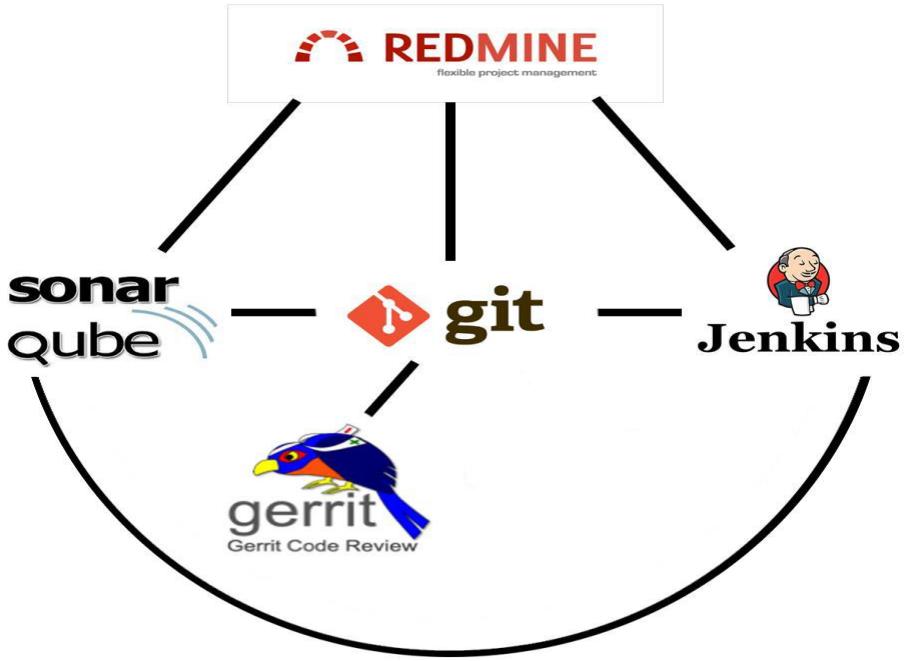


Figura 3.4: Exemplo da instanciação do ADE

As integrações entre cada ferramenta são realizadas através de *plug-ins* e configurações nas ferramentas. Não são necessárias novas implementações para realizar as integrações. Neste exemplo de instanciação, as ferramentas são compatíveis, porém antes de selecionar as ferramentas para compor o ADE é necessário validar se existe compatibilidade entre elas.

3.7. Considerações Finais

Neste capítulo, foram apresentados os princípios de projeto do ADE, a sua arquitetura, a descrição do processo do ADE, e por fim, as suas integrações. A definição deste ambiente, em harmonia com o *Scrum* e ALM, contribui para evidenciar como as práticas e técnicas ágeis são aplicadas ao processo de desenvolvimento, em como as ferramentas podem ser articuladas para conceber um ambiente integrado e automatizado, permitindo alcançar os conceitos do ALM.

É importante ressaltar que o projeto do ambiente apresentado, nesta pesquisa, contém as práticas e técnicas mais utilizadas pela indústria de software, conforme apresentado na Seção 2.4. Caso exista o desejo de adicionar novas ferramentas ao ADE, basta adicioná-la e realizar as integrações, quando possível. Para novas etapas no processo de desenvolvimento ou adaptações do processo apresentado, deve-se, após a alteração no processo, atualizar o fluxo de trabalho no gerenciador de projeto.

O próximo capítulo, o 4, apresenta a avaliação do projeto do ADE, uma comparação com os conceitos de PSEE, mediante um estudo qualitativo.

4

Avaliações

4.1. Considerações Iniciais

Este capítulo apresenta as avaliações do ADE, propostas nesta dissertação, e composta de duas partes. A primeira é uma comparação entre o ADE e aspectos relevantes propostos em relação aos PSEE. A segunda parte apresenta um estudo empírico qualitativo com a finalidade de avaliar a viabilidade do ADE proposto.

4.2. Comparação entre PSEE e o ADE

Os PSEE podem ser genericamente vistos como compostos de três partes (Bandinelli et al., 1996; Fuggetta, 1996):

1. Motor de processo;
2. Interface com o usuário;
3. Repositório central de dados.

Betemps (2003) também apresenta os requisitos necessários para um ambiente com suporte a processo de software. Para realizar a comparação entre os PSEE existentes na literatura e o ADE, os parâmetros escolhidos estão na Tabela 4.1, a qual também demonstram os requisitos necessários para um PSEE que cada parâmetro engloba.

Tabela 4.1: Parâmetros de comparação entre PSEE e ADE

Parâmetro	**Requisitos necessários para um PSEE**
Definição e evolução de processo	Suporte ao Gerenciamento, Suporte aos Eventos do Processo, Flexibilidade e Reusabilidade
Definição de responsabilidades da equipe	Colaboração
Controle de ferramentas	Verificação
Integração por dados	Consistência, Integração de Dados e Monitoramento
Integração por controle	Integração de Controle
Capacidade de evolução	Extensibilidade e Fácil de usar

A Tabela 4.2 ressalta as similaridades e diferenças entre o que foi apresentado no Capítulo 2 sobre PSEE e o ADE proposto no Capítulo 3.

Tabela 4.2: Similaridades e diferenças entre PSEE e o ADE

	PSEE	**ADE**
Definição e evolução de processo	Propõe uma PML para definição do processo e um motor de processo que a executa.	Adota o processo *Scrum* e utiliza um gerenciador de projeto por meio do qual as atividades são especificadas, mas não controla a execução. A manutenção das atividades é realizada pelo gerente de projetos.
Definição de responsabilidades da equipe	São definidas no próprio modelo de processo.	São definidas no gerenciador de projeto de forma compatível como o processo *Scrum.*
Controle de ferramentas	As ferramentas são ativadas conforme a definição do processo.	As ferramentas são ativadas pelos desenvolvedores conforme o conjunto escolhido.

	PSEE	**ADE**
Integração por dados	Os artefatos de desenvolvimento são armazenados em um repositório central seguindo padrões específicos que podem ser identificados pelas ferramentas.	Os artefatos de desenvolvimento são armazenados em sistemas de arquivos e as ferramentas são responsáveis por eventuais mecanismos de filtragem de dados necessários.
Integração por controle	As ferramentas são projetadas de modo a que elas possam ser ativadas a partir do processo definido e entre elas mesmas.	As ferramentas são ativadas pelos desenvolvedores ou por *scripts* programados.
Capacidade de evolução	A evolução depende da conformidade aos padrões de processo, ferramentas e armazenamento de dados	Existe flexibilidade de incorporação de ferramentas, pois os padrões de dados e controle são baseados em arquivos e na articulação entre as ferramentas

Com base nas informações da Tabela 4.2, fica claro que o ADE não é baseado na descrição explícita do processo. A definição do processo no ADE é realizada como uma descrição do fluxo de trabalho e não define um controle em relação às ferramentas ou uma definição automatizada, para que as demais ferramentas do ambiente sejam executadas.

Além disso, o ADE promove, pelas etapas do processo, que o gerenciador de projeto seja alimentado com as informações de desenvolvimento, como: apontamento de tempo gasto em cada tarefa, armazenamento de documentos relacionado a cada tarefa, atualização do estado de cada atividade de acordo com o andamento fluxo do processo. A partir destas informações, o ADE permite que informações do projeto sejam recuperadas no gerenciador de projeto e assim projetando a sua visibilidade.

4.3. Estudo Empírico Qualitativo

Esta seção apresenta uma avaliação empírica qualitativa com a finalidade de evidenciar a viabilidade do ADE proposto. A realização deste estudo foi importante, pois tornou possível analisar o *feedback*/respostas de especialistas para refinar o ADE.

Para ilustrar a metodologia de pesquisa aplicada neste estudo, a Figura 4.1 apresenta as respectivas etapas seguidas. A metodologia possui as etapas necessárias para a realização de um estudo empírico qualitativo.

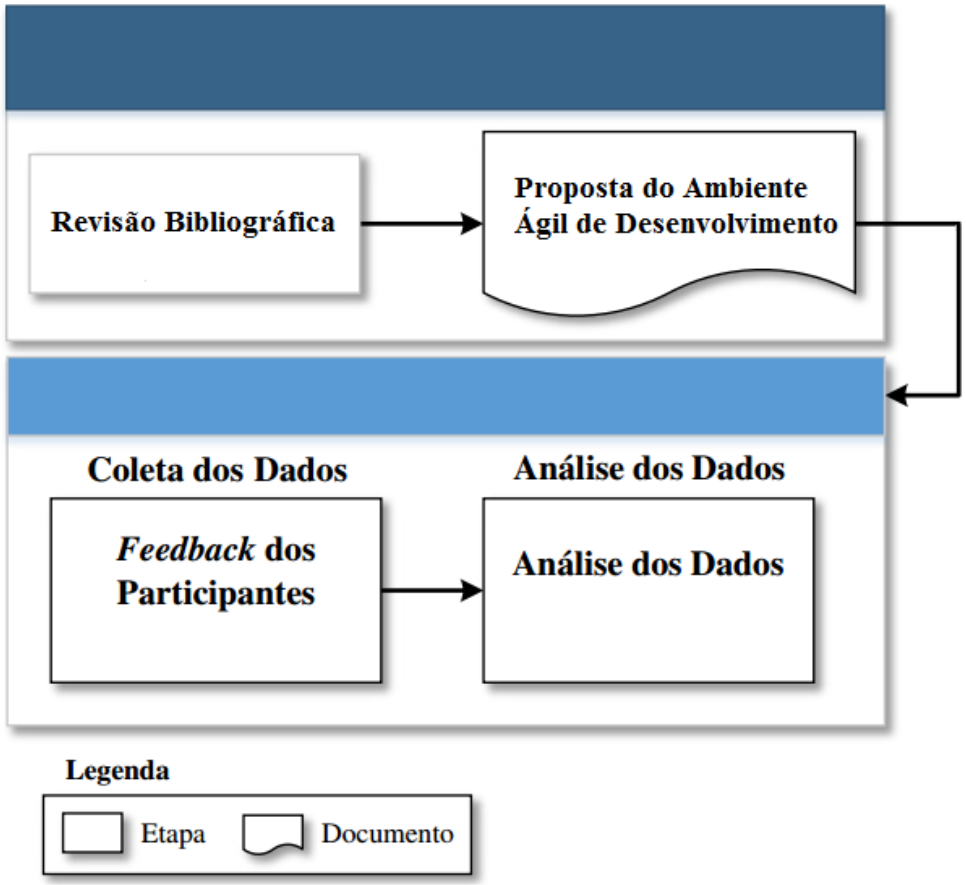


Figura Erro! Nenhum texto com o estilo especificado foi encontrado no documento.**4.1:** Metodologia de Pesquisa do Estudo Empírico Qualitativo

Este estudo é baseado em procedimentos da Teoria Fundamentada nos Dados (*Grounded Theory*), como Codificação (*Coding*) proposta por Corbin e Strauss (2008), que complementam a estratégia sequencial exploratória de métodos mistos (*mixed-methods*), conforme Seaman (1999), Dyba et al. (2011) e Creswell e Clark (2010). Os conceitos relevantes da pesquisa são representados por meio de códigos identificados nas respostas dos especialistas.

Um estudo em que os seus resultados não podem ser medidos por meio de métodos estatísticos, em qualquer tipo de escala numérica, é definido como qualitativo (Corbin e Strauss, 2008). Segundo Corbin e Strauss (2008), *Grounded Theory* é um método sistemático para coletar e analisar dados, para então gerar, elaborar e validar teorias substantivas. Este método fornece os meios para transformar a visão do pesquisador em realidade. Portanto, a análise ocorre pela interação entre os especialistas e os dados (Corbin e Strauss, 2008).

Na pesquisa qualitativa, três elementos básicos são fundamentais (Corbin e Strauss, 2008):

1. **Dados**: podem ser coletados de várias formas, como questionários, entrevistas, documentos, no caso deste estudo, por meio de questionário eletrônico;
2. **Procedimentos**: usados pelos pesquisadores para organizar e interpretar os dados obtidos;
3. **Relatórios**: que podem ser escritos e verbais, como artigos publicados em periódicos científicos.

Com o objetivo de conceitualizar, reduzir os dados e elaborar categorias em termos de sua propriedade e dimensões, a Codificação faz parte dos procedimentos deste estudo (Broberg, 2011). Para analisar os dados deste estudo, foram utilizados Codificação aberta (*Open Coding*) e Codificação Axial (*Axial Coding*) (Corbin e Strauss, 2008):

- Codificação Aberta é o processo analítico realizado manualmente com a leitura dos dados coletados, onde os conceitos são identificados de acordo com o conteúdo recuperado. O processo de codificação aberta pode classificar palavras, linhas ou sentenças, as quais possibilitam identificar as propriedades e dimensões dos dados;
- Codificação Axial é a segunda parte da codificação e tem por objetivo reagrupar as categorias encontradas na Codificação Aberta em categorias de maior abstração. Dessa forma, permite que novas subcategorias sejam geradas.

Exemplos completos de Codificação e *Grounded Theory* podem ser vistos nas dissertações Geraldi (2015) e Dias (2015).

4.3.1. Definição do Estudo

O propósito principal deste estudo foi analisar se o projeto do ADE é viável como mecanismo de apoio ao desenvolvimento com MA. Baseado no modelo *Goal Question Metric* (GQM) (Basili et al., 1994; van Solingen et al., 2002), este estudo tem por objetivo:

Analisar o projeto do ADE;

Com o propósito de identificar sua viabilidade;

Em relação à capacidade de gestão do projeto e apoio ao desenvolvimento de software.

Do ponto de vista de desenvolvedores e gerentes que atuam em projetos ágeis;

No contexto de profissionais com mais de cinco anos de experiência em desenvolvimento de software em projetos ágeis.

4.3.2. Planejamento do Estudo

As etapas correspondentes ao planejamento deste estudo empírico qualitativo são apresentadas a seguir:

Projeto Piloto: um estudo piloto foi realizado, com a participação de um especialista da indústria, que possui experiência em desenvolvimento de software com métodos ágeis, com o objetivo de avaliar a instrumentação utilizada, para adequá-la. Os dados obtidos no projeto piloto não foram utilizados como resultados deste estudo empírico, porém as considerações em relação a erros e melhorias na instrumentação proposta foram levadas em consideração.

Treinamento: para este estudo foi necessário apresentar aos especialistas o ADE, exemplificar a rotina de trabalho com o ADE, além dos documentos entregues na instrumentação.

Especialistas: os participantes deste estudo foram cuidadosamente selecionados pela experiência que possuem na indústria, com desenvolvimento de software e projetos ágeis. Dos participantes, um (1) é aluno de mestrado (14,3%), cinco (5) são pós-graduados lato sensu (71,4%) e um (1) é aluno de pós-graduação lato sensu (14,3%).

Instrumentação: todos os especialistas do estudo receberam um conjunto de documentos: (i) um termo de adesão ao estudo; (ii) um questionário de caracterização sobre a experiência do participante nos assuntos abordados; (iii) um documento contendo a modelagem e descrição do processo de desenvolvimento do ADE; e por fim, (iv) um documento contendo a descrição da arquitetura do ADE. Além destes documentos de apoio, foi realizado um treinamento explicando o processo, arquitetura e integrações do ADE. A instrumentação completa deste estudo está disponível no Apêndice A.

A instrumentação apresentada poderia ser utilizada pelos especialistas para consulta, com a finalidade de auxiliá-los a responderem o questionário eletrônico criado na plataforma do

Google Drive[1]. Para isso, foi criado um formulário eletrônico que continha sete perguntas. Cada especialista recebeu, por email, o endereço para o formulário e os documentos relacionados ao processo e arquitetura do ADE.

4.3.3. Execução do Estudo

Nesta seção, são apresentadas as etapas executadas para a obtenção dos resultados referentes à aplicação deste estudo empírico qualitativo.

Procedimentos de Participação: a participação de cada especialista ocorreu da seguinte forma:

1. O especialista recebe os documentos necessários ao estudo para preencher e assinar;
2. O especialista faz a leitura do termo, esclarece possíveis dúvidas e assina o Termo de Consentimento Livre e Esclarecido (TCLE);
3. O especialista faz a leitura do Questionário de Caracterização de Participante, esclarece possíveis dúvidas, preenche suas informações e o assina;
4. O experimentador/pesquisador ministra o treinamento presencial;
5. Após o treinamento, o experimentador/pesquisador envia a instrumentação para cada especialista, e esta pode ser utilizada para consulta ao responder o questionário eletrônico;
6. O experimentador/pesquisador explica o questionário eletrônico enviado a cada especialista, e esclarece possíveis dúvidas com relação ao formato, preenchimento e envio das respostas;
7. O especialista responde e envia o formulário eletrônico respondido.

As tarefas foram realizadas em igual número pelos especialistas. Portanto, a Tabela 4.3 apresenta as informações sobre cada especialista deste estudo, os quais foram ordenados (de forma crescente) de acordo com sua formação acadêmica. Os especialistas foram classificados com base nos seguintes fatores:

Formação Acadêmica: Especializando (En), Especialista (Es), Mestrando (Mn);

Experiência com Desenvolvimento de Software: Nenhuma (N), Básica (B), Moderada (M), Avançada (A); e

Experiência com Métodos Ágeis: Nenhuma (N), Básica (B), Moderada (M), Avançada (A).

[1] Disponível em *https://drive.google.com*

Tabela 4.3: Dados de Caracterização Detalhados dos Especialistas do Estudo

Nº do Especialista	**Formação Acadêmica**	**Experiência com Desenvolvimento de Software**	**Experiência com Métodos Ágeis**
Nº 1	En () Es(X) Mn()	N() B() M() A(X)	N() B() M() A(X)
Nº 2	En (X) Es() Mn()	N() B() M() A(X)	N() B() M() A(X)
Nº 3	En () Es() Mn(X)	N() B() M() A(X)	N() B() M() A(X)
Nº 4	En () Es(X) Mn()	N() B() M() A(X)	N() B() M() A(X)
Nº 5	En () Es(X) Mn()	N() B() M() A(X)	N() B() M() A(X)
Nº 6	En () Es(X) Mn()	N() B() M() A(X)	N() B() M() A(X)
Nº 7	En () Es(X) Mn()	N() B() M() A(X)	N() B() M(X) A()

4.3.4. Análise e Interpretação dos Resultados

Esta seção descreve os resultados obtidos com base nas respostas fornecidas por cada especialista, de acordo com a execução deste estudo. A análise dos dados coletados, neste estudo, foi realizada por meio dos procedimentos de *Grounded Theory*. A ferramenta *Dedoose*[2] foi utilizada para organizar e codificar os dados coletados dos especialistas por meio da codificação aberta. Deste modo, em seguida, possibilitou a categorização dos códigos pela codificação axial e assim permitindo representações gráficas dessas categorias.

As respostas dos especialistas foram classificadas de acordo com as categorias e códigos gerados, e são apresentadas na integra no Apêndice A. Por conseguinte, três categorias emergiram durante o processo de codificação em relação ao ADE, são elas: fornece benefícios na utilização, possíveis melhorias e dificuldades para adoção.

A Tabela 4.4 mostra a relação das categorias com as respostas dos especialistas. Para um melhor entendimento da Tabela 4.4 as categorias, respostas e os especialistas são classificados da seguinte forma:

- Categoria: para cada categoria é utilizado o nome da categoria
- Resposta: para cada resposta é utilizada a abreviatura Rn, tal que n é o número da pergunta referente a resposta (R1 até R7);
- Especialista: Para cada especialista será utilizada a abreviatura E + ID, por exemplo: E1, E2 até E7.

[2] Disponível em *http://dedoose.com/*

Tabela 4.4: Classificação das categorias com as respostas dos especialistas

Respostas	Fornece Benefícios na Utilização	Possíveis Melhorias	Dificuldade para Implantação
P1.R1		X	
P1.R2	X		
P1.R3	X		
P1.R4		X	
P1.R5	X		
P1.R7			X
P2.R2	X		
P2.R3	X		
P2.R5	X		
P2.R6	X		
P2.R7			X
P3.R2	X		
P3.R3	X		
P3.R4	X		
P3.R5	X		
P3.R6	X		
P3.R7			X
P4.R2	X		
P4.R3	X		
P4.R4	X		
P4.R6	X		
P4.R7			X
P5.R2	X		
P5.R4		X	
P5.R6	X		
P5.R7			X
P6.R2		X	
P6.R3	X		
P6.R4		X	
P7.R4	X		
P7.R6	X		
P7.R7	X		X

Deste modo, as descrições das categorias são apresentadas seguidas dos códigos que as compõem, bem como os trechos das respostas coletadas que apoiam sua categorização, as quais contribuíram para a avaliação do ADE.

Fornece Benefícios na Utilização: a análise das respostas dos especialistas evidenciou que o ADE fornece benefícios na utilização. A Figura 4.2 apresenta uma representação gráfica dos códigos que fazem parte desta categoria.

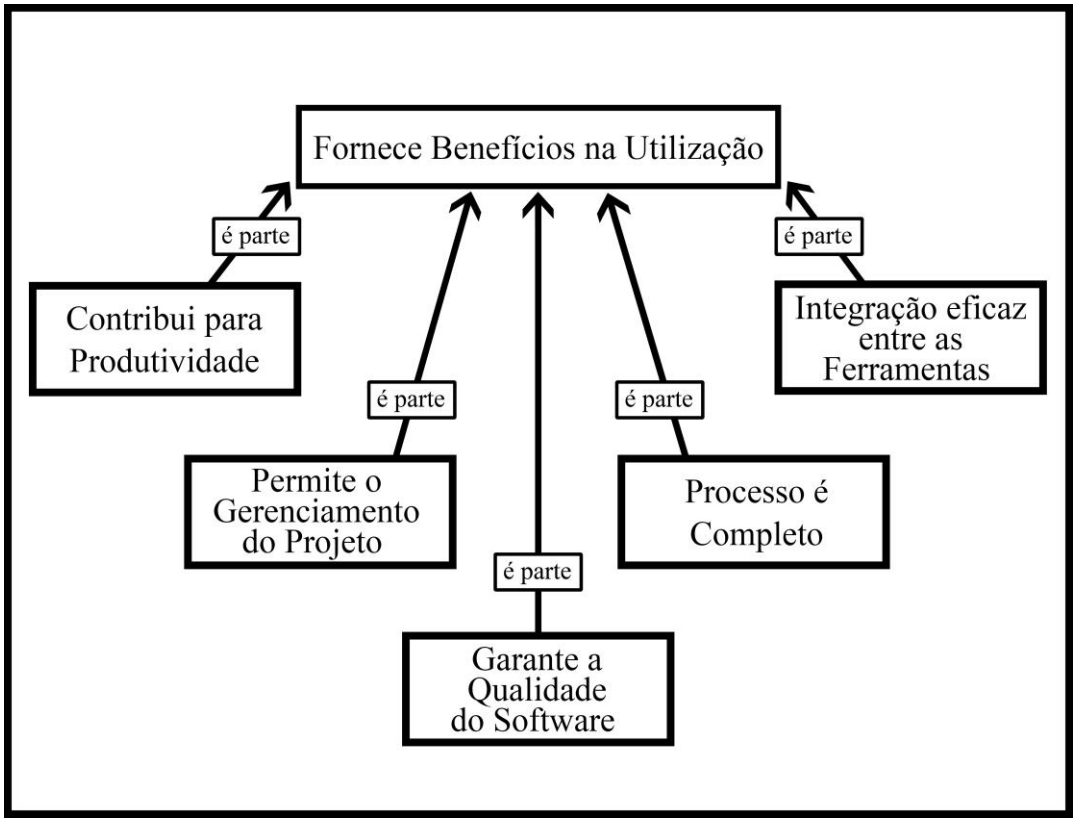


Figura 4.2: Representação gráfica com as associações relacionadas à categoria "Fornece Benefícios na Utilização"

O ADE contribui para aumentar a produtividade do desenvolvimento do software de acordo com o código Contribui para Produtividade atribuído com base na resposta do especialista P2.R5: "*Sim.*"; do P3.R5: "*Sim, com as ferramentas bem alinhadas e automatizadas, o ganho de produtividade do desenvolvimento de software é uma consequência.*"; do P1.R1: "*Sim por conta das ferramentas de execução de cenários e de integração contínua, além da atividade de revisão da tarefa.*"; e do P7.R7: "*Sim, acredito que favorece a produtividade.*".

Além disso, o ADE fornece apoio para o gerenciamento do projeto com base no código Permite o Gerenciamento do Projeto, atribuído de acordo com as respostas dos especialistas P2.R2 e P5.R2: "*Sim.*"; do P4.R2: "*Sim. Para o gerenciamento do projeto de software sim.*"; do P3.R2: "*... as outras ferramentas produzem informações suficientes para manter um bom gerenciamento de um projeto de software.*"; e do P1.R2: "*Sim, desde que as*

seguintes informações sejam armazenadas: Dados sobre o cronograma do projeto, Lista das tarefas e suas dependências, e Custo dos recursos necessários na execução das tarefas".

Ademais, o ADE contribui para promover a qualidade do software desenvolvido de acordo com o código Garante a Qualidade do Software, atribuído com base na resposta do especialista P1.R3: "*Sim, porque existem etapas específicas para a definição do propósito do projeto (definição do escopo) e a validação da entrega (Revisão).*"; do P2.R3: "*Sim.*"; do P3.R3: "*Sim, pois através da ferramenta de ADE podemos amarrar todo o código versionado e gerações de versões para o gerenciamento do ciclo de vida da aplicação. Com isso, gerando uma base histórica e bem documentada da aplicação.*"; do P4.R3: "*Sim, aumentam. Mas este aumento só ficará evidenciado com a ação de pessoas que entendam a qualidade a ser produzida.*"; e do P6.R3: "*Sim. Elas automatizam vários processos que poderiam gerar transtorno se feitas manualmente, e também adicionam insumos com code review e analisador de qualidade de código, integração contínua que não são realidades em vários centros de desenvolvimentos.*".

Além disso, alguns especialistas afirmaram que o processo definido no ADE é completo com base no código Processo é Completo, atribuído de acordo com a resposta do especialista P3.R4: "*Sim*"; do P4.R4: "*Não removeria e nem adicionaria nenhuma etapa no momento. Os gráficos contemplam com excelência o processo de desenvolvimento e entrega de funcionalidades.*"; e do P7.R4: "*Acredito ser um processo completo, porém a sua aplicação por completo depende muito do projeto que será implementado, quantidade de pessoas, custo e prazo.*".

Outrossim, a integração entre as ferramentas do ADE são eficazes para permitir a evolução contínua do sistema de desenvolvimento de software de acordo com o código Integração eficaz entre as Ferramentas, com base na resposta do especialista P1.R6: "*Sim, para a etapa de desenvolvimento.*"; dos P2.R6, P5.R6 e P7.R6: "*Sim.*"; do P3.R6: "*Sim, o rastreio das informações legado deixado pelas integração acredito ser o fator mais importante para o acompanhamento para o desenvolvimento do software.*"; e do P4.R6: "*Sim. Com este cenário é possível integrar o ambiente de software.*".

Possíveis Melhorias: foram identificadas algumas possíveis melhorias para o ADE. A Figura 4.3 representa graficamente esta categoria juntamente com os códigos que a compõe.

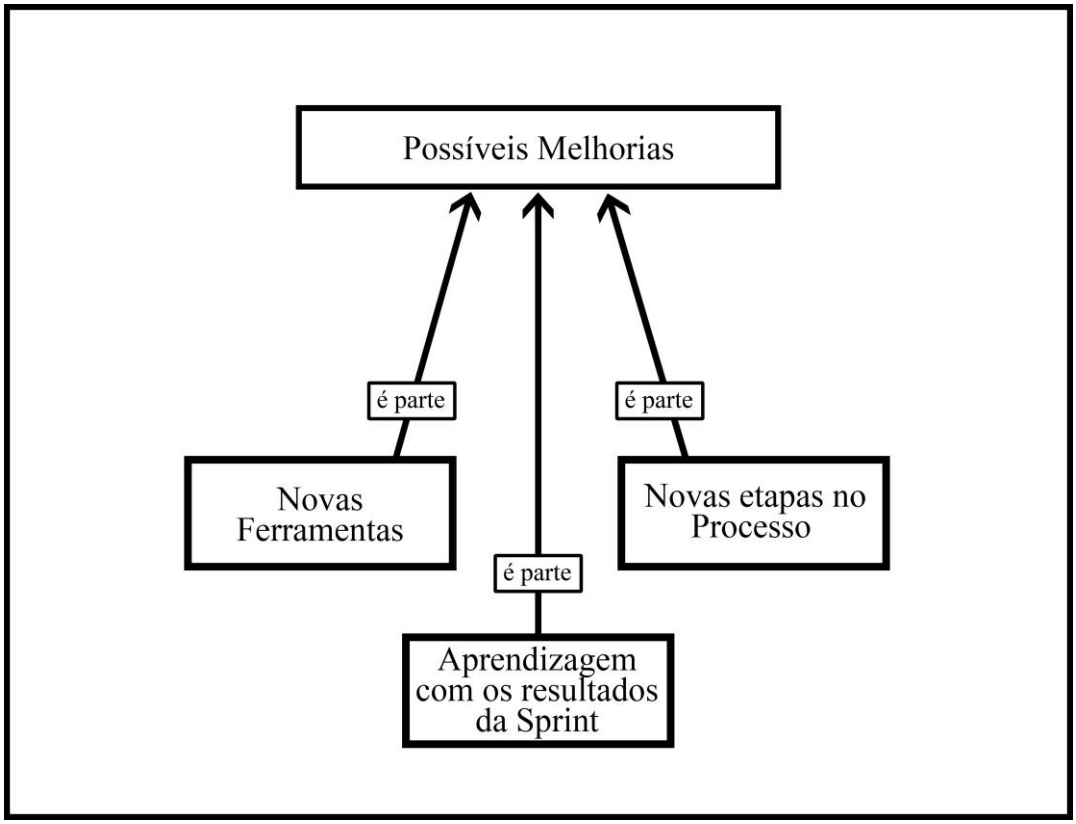


Figura 4.3: Representação gráfica com as associações relacionadas à categoria "Possíveis Melhorias"

O ADE requer possíveis melhorias, como incluir outras ferramentas com base no código Novas Ferramentas, atribuído de acordo com as respostas do especialista P1.R1: "*Ferramenta que gerencie e faça versões das configurações realizadas no banco de dados durante os processos de implantação. O ideal é que essa ferramenta permitisse a seleção e configuração de demandas específicas, facilitando assim a replicação das configurações em ambientes secundários como de homologação.*", "*Mapeamento dos módulos/requisitos: Ferramenta que realize o rastreamento dos módulos ou requisitos do sistema, de modo a registrar quais são as partes mais alterados, seus custos, e recursos que possuem conhecimento nessas partes.*", "*Ferramenta de virtualização que emule o ambiente de produção*" e P1.R4: "*... indicadores de monitoramento do desenvolvimento.*"; e do P6.R4: "*... acho primordial alguma ferramenta que nos entregue métricas que nos permitam melhorar o processo pelo aprendizado contínuo, e armazenar o histórico destes avanços.*".

Paralelamente, foi sugerido que o ADE fornecesse uma maneira de aprendizagem a partir dos resultados das *Sprint* de acordo com o código Aprendizagem com os resultados da Sprint, atribuído com base nas respostas do especialista P6.R2: "*Sinto falta de informações de resultados da sprint que possibilitam o aprendizado contínuo.*" e "*Proporia alteração*

adicionando alguma ferramenta para métrica das sprints desenvolvidas, para que seja possível a visualização dos resultados obtidos, e a aprendizagem contínua a partir dos resultados.".

Os especialistas ainda sugeriram alterações no processo de desenvolvimento que estão presentes no código Novas etapas no Processo, atribuído de acordo com a resposta do especialista P1.R4: "*Incluir a etapa de mobilização da equipe*."; e do P5.R4: "*... senti falta do pair programming.*".

Dificuldade para Implantação: foi analisado, de acordo com as respostas dos especialistas, quais seriam as barreiras para adoção do ADE. A Figura 4.4 apresenta uma representação gráfica desta categoria e os códigos que a compõem.

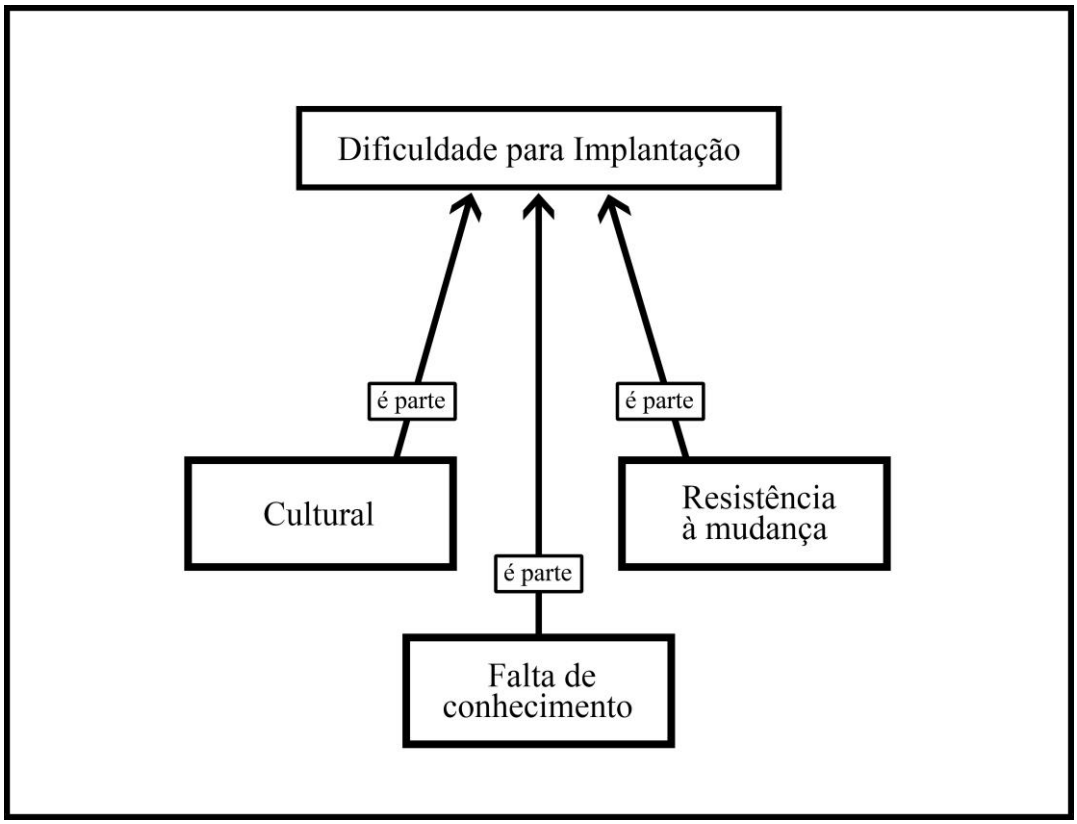


Figura 4.4: Representação gráfica com as associações relacionadas à categoria "Dificuldade para Implantação"

Segundo os especialistas, uma dificuldade que pode ser encontrada para adoção do ADE é a cultura da empresa, de acordo com o código Cultural atribuído com base na resposta do especialista P1.R7: "*Culturais por parte da gestão que precisa enxergar as atividades de revisão de código como um ponto de ganho de desempenho e qualidade, e não como um desperdício de recurso*."; do P3.R7: "*Uma das dificuldades é a mudança de cultura da equipe de desenvolvimento para usar esse ambiente. No início, alguns desenvolvedores não são*

adeptos a essas mudanças, isso pode causar dados inconsistentes na ferramenta, como por exemplo, não avançar as tarefas na ferramenta de gerenciamento."; do P5.R7: "*Cultura da empresa.*"; e do P7.R7: "*... a principal barreira aqui é a cultura organizacional, uma empresa não acostumada a investir em testes automatizado, por exemplo, pode relutar em adotar as ferramentas propostas.*".

Salienta-se ainda que a falta de conhecimento pode ser outra dificuldade para adoção do ADE, com base no código Falta de conhecimento, atribuído de acordo com a resposta do especialista P1.R7: "*Intelectuais por parte da dificuldade em encontrar profissionais que saibam criar cenários de teste automatizados, muito menos modelar uma solução de negócio de modo a facilitar a construção dos cenários de teste.*"; e do P4.R7: "*Falta de pessoas capacitadas.*".

Por fim, de acordo com o código Resistência à mudança, a última barreira sugerida pelos especialistas é a resistência a mudança, que foi atribuído com base na resposta do especialista P2.R7: "*Resistência a mudança, preconceitos já existentes.*"; e do P5.R7: "... *aversão à mudanças das pessoas.*".

A partir da análise e interpretação das respostas dos especialistas e com os resultados obtidos, é possível fornecer indícios de que o ADE é viável e existem benefícios na sua utilização, como (i) promove a qualidade do software; (ii) contribui para gerar mais produtividade no desenvolvimento do software; (iii) permite a gestão do projeto; (iv) as integrações são eficazes para evolução contínua do sistema de desenvolvimento de software; e (v) o processo definido no ADE é suficientemente completo.

4.3.5. Avaliação da Validade do Estudo

As ameaças consideradas relevantes e identificadas com relação aos impactos para tal estudo são apresentadas nesta seção.

4.3.5.1. Ameaças à Validade de Conclusão

A principal ameaça para este estudo está relacionada ao número de participantes/especialistas, que foram 7. Porém independentemente de a quantidade de participantes ser relativamente pequena, a colaboração dos especialistas, a partir dos seus conhecimentos, foi significativa. Além disso, a qualificação dos especialistas foi priorizada, pela característica própria do estudo.

4.3.5.2. Ameaças à Validade de *Constructo*

Este estudo foi definido e testado com base em um projeto piloto realizado, refinando o questionário. Os dados obtidos em tal projeto foram descartados.

Embora o nível de conhecimento exigido sobre os conceitos de desenvolvimento de software em projetos ágeis e ferramentas de desenvolvimento demonstraram-se satisfatórios para análise da técnica, uma possível ameaça de *constructo* é a falta de testes para validar a compreensão dos participantes em relação aos questionários.

4.3.5.3. Ameaças à Validade Interna

As seguintes dificuldades foram identificadas:

Como a amostra de tal estudo foi pequena, algumas poucas variações com relação as habilidades dos especialistas estavam evidentes, sendo assim, os especialistas n~ao foram divididos em grupos. Logo, as tarefas executadas pelos especialistas foram realizadas de maneira similar.

Diferenças entre os especialistas: devido ao pequeno tamanho da amostra, as variações entre as habilidades dos especialistas foram poucas. Logo, as tarefas executadas pelos especialistas foram realizadas de maneira similar.

Acurácia das respostas dos participantes: uma vez treinados nos conceitos do ADE, somando-se ao nível de conhecimento que os especialistas possuem, considera-se que as respostas fornecidas são válidas e significantes.

Efeito de Fadiga: Com o intuito de reduzir os efeitos de fadiga dos participantes, o treinamento foi realizado individualmente e foi enviada a instrumentação por email para os especialistas. O prazo de quinze dias foi concedido a eles para preencher os questionários; de modo que os efeitos da fadiga foram reduzidos.

Outros fatores importantes: A possível influência entre os especialistas foi amenizada com os questionários eletrônicos, uma vez que foram distribuídos de forma individual e respondidos em datas e horários distintos.

4.3.5.4. Ameaças à Validade Externa

Uma ameaça foi identificada referente aos especialistas. O objetivo inicial do estudo era buscar profissionais, mestres e doutores com experiência prática em desenvolvimento de

software em projetos ágeis e ferramentas de desenvolvimento. Porém, a obtenção de mestres e doutores foi uma das grandes dificuldades deste estudo. Assim, profissionais com pós-graduação, com conhecimento sobre tais assuntos foram convidados.

4.3.6. Considerações Finais

Foi permitido identificar três categorias e seus respectivos códigos, a partir dos resultados obtidos do estudo qualitativo, para avaliar a viabilidade do ADE como ambiente de desenvolvimento de software. Além disso, os resultados contribuíram para identificação de possíveis melhorias e dificuldades que podem dificultar a sua implantação.

Acrescenta-se que as respostas dos especialistas forneceram tanto indícios, quanto evidências de que o ADE é viável, pois afirmaram que o ADE promove a qualidade do software desenvolvido e fornece insumos para o gerenciamento do projeto. Ademais, as integrações entre as ferramentas do ADE são eficazes para permitir a evolução contínua do sistema de desenvolvimento de software. Em adição, o ADE contribui com relação à produtividade do desenvolvimento de software e possui o processo de desenvolvimento completo.

Por fim, compreende-se que novos estudos devem ser conduzidos no ambiente acadêmico e na indústria para avaliar os resultados gerados pelo ADE. De acordo com os resultados alcançados neste estudo empírico qualitativo, pode-se pensar em estudos quantitativos, a fim de verificar a sua eficiência e eficácia, tomando como base o corpo de conhecimento inicial construído nesta pesquisa.

5

Conclusão

Esta dissertação de mestrado apresentou o projeto de um ambiente de desenvolvimento de software no contexto de métodos ágeis, denominado ADE. Partiu-se do princípio que, com a introdução dos MA, é imprescindível a utilização de ferramentas apropriadas para se obter um gerenciamento ágil de projetos e impô-las aos colaboradores (Franky, 2011). Assim, tornou-se necessário um ambiente de desenvolvimento de software que articule o método de gerenciamento e as práticas ágeis.

O ADE é baseado no *Scrum* como método de gerenciamento, pois estudos demonstram vantagens em sua aplicação (Striebeck, 2006) e por ser o método mais utilizado pela indústria (VersionOne, 2015; Melo et al., 2012). As práticas que compõem o ADE foram selecionadas com base nas pesquisas de VersionOne, (2015), Melo et. al. (2012) e Abrantes e Travassos (2011). Além disso, o ADE foi apoiado nos princípios de ALM para articular o uso do *Scrum* e as práticas de desenvolvimento devido ao fato dos seus conceitos oferecerem a gestão do ciclo de vida do software no âmbito de governança, desenvolvimento e manutenção (Chappel, 2014; Rossman, 2010). Ademais, o ALM possibilita o gerenciamento ágil, com a integração das ferramentas de gestão e desenvolvimento (Franky, 2011).

O ADE evidenciou as práticas e técnicas ágeis aplicadas ao processo de desenvolvimento e como as ferramentas podem ser articuladas para conceber um ambiente integrado e automatizado, e assim permitir alcançar os conceitos do ALM.

Outrossim, foi evidenciado que o ADE é viável para utilização no desenvolvimento de software e, com base na avaliação realizada por meio do estudo qualitativo, foi demonstrado

que o ADE produz benefícios na sua utilização como: promover a qualidade do software, contribuir para geração de produtividade no desenvolvimento do software, permitir o gerenciamento do projeto, promover integrações eficazes para evolução contínua do sistema de desenvolvimento de software e possuir um processo que contemple as fases de desenvolvimento de software.

5.1. Contribuições

A principal contribuição desta pesquisa é a descrição de um ambiente ágil de desenvolvimento de software baseado em *Scrum*, apoiado nos princípios de ALM. Este ambiente facilitará o desenvolvimento de software em projetos ágeis e auxiliará as organizações, especialmente as pequenas empresas, a estruturar seu ambiente de desenvolvimento

Entre as contribuições deste trabalho, destacam-se:

- O projeto de um ADS que facilite a utilização sistemática de práticas ágeis no desenvolvimento de software e como elas são integradas no gerenciamento do projeto (Capítulo 3);
- Comparação entre o que foi proposto nas pesquisas existentes sobre ADS e o que está sendo usado na indústria de software (Seção 4.2);
- Avaliação empírica qualitativa do ADE (Seção 4.3);
- Estabelecimento de uma pesquisa que aproxima conceitos de pesquisa em ADS com experiências na indústria (Capítulo 3 e ficou evidente pela Seção 4.3);
- Oferecimento de uma referência para pequenas empresas definirem um ambiente de desenvolvimento de software, pois elas não possuem recursos financeiros para pesquisar como devem usar as práticas ágeis e conceber um ADS que articula uma MA com as práticas (Capítulo 3).

5.2. Limitações

Entre as limitações desta pesquisa, está a pequena quantidade de participantes no estudo qualitativo realizado. Neste estudo, os especialistas com qualificação adequada tinham tempo limitado para participar do estudo de forma efetiva. Porém, esta limitação pode ser reduzida por meio da obtenção de amostras maiores de participantes com o planejamento e condução

de novos estudos empíricos e na condução de replicações dos estudos empíricos realizados nesta pesquisa.

Outra limitação deste trabalho é que não foram realizados estudos a partir da observação de equipes de desenvolvimento e como usam os seus ambientes de desenvolvimento. Contudo, esta limitação pode ser reduzida, pois foram usadas as experiências do autor em projetos da indústria com a utilização de um ambiente de desenvolvimento.

Por fim, outra limitação deste trabalho é que o ADE não foi avaliado no ambiente industrial e também não foram feitas simulações com várias ferramentas alternativas. Acredita-se que empresas de tecnologia de informação poderiam contribuir, em muito, com a evolução da descrição do ambiente de desenvolvimento em projetos ágeis. Porém, esta limitação pode ser mitigada por meio da condução de estudos empíricos mais simplificados e ágeis, com o intuito de permitir a participação de colaboradores de diferentes empresas, já que o principal problema observado está relacionado a adequação do tempo.

5.3. Trabalhos Futuros

De acordo com a experiência adquirida com a especificação do ADE somado ao estudo empírico, e visando a continuidade do trabalho desenvolvido, a seguir, são apresentadas as principais perspectivas de trabalhos futuros relacionadas a este trabalho:

- Avaliar empiricamente o ADE baseado no *Scrum* em projetos reais na indústria;
- Investigar como o ADE pode auxiliar nas definições arquiteturais no desenvolvimento de software;
- Compartilhar o ADE por meio da instanciação de uma versão apenas com ferramentas de projetos *Open Source*, com o princípio de permitir que pesquisadores e empresas interessados no ambiente contribuam com a sua evolução e façam uso em seus projetos;
- Realizar as requisições de melhorias apresentadas pelos especialistas para evolução do ADE;
- Análises quantitativas dos benefícios identificados de acordo com especialistas para a análise da real efetividade do ADE;
- Investigar quais as métricas de gerenciamento podem ser extraídas e como usá-las para melhorar o desenvolvimento de software a partir do ADE;

- Investigar outras alternativas para realização de estimativas e como o ADE pode auxiliar na automatização disso;
- Investigar como o ADE pode auxiliar nas definições de requisitos do software;
- Analisar o uso de outros MA, encontradas na literatura, como método de gerenciamento no ADE.

Referências

Abbas, N.; Gravell, A. M.; Wills, G. B., Using Factor Analysis to Generate Clusters of Agile Practices (A Guide for Agile Process Improvement). In: *Agile Conference (AGILE)*, Orlando, 2010, p. 11 20.

Abrantes, J. F.; Travassos, G. H., Common Agile Practices in Software Processes. In: *Empirical Software Engineering and Measurement (ESEM)*, Banff, 2011, p. 355 358.

Agarwal, N.; Deep, P. Obtaining better software product by using test first programming technique. In: *Confluence The Next Generation Information Technology Summit (Confluence)*, Noida, 2014, p.742 747.

Ambriola, V.; Conradi, R.; Fuggeta, A. Assessing Process - Centered Software Engineering Environments. *ACM Transactions on Software Engineering and Methodology*, New York, v. 6, n. 3, p. 283 - 328, 1997.

Baessa, H. J. O. *Gestão do Ciclo de Vida das Aplicações: Análise do SAP Solution Manager*. Dissertação de Mestrado. Lisboa: ISEGI – UNL, 2011.

Bakota, T.; Hegedus, P.; Ladanyi, G.; Kortvelyesi, P.; Ferenc, R.; Gyimothy, T. A cost model based on software maintainability. In: *28th IEEE International Conference Software Maintenance (ICSM)*, Trento, 2012, p. 316 325.

Bandinelli, S.; Di Nitto, E.; Fuggetta, A. Supporting Cooperation in the SPADE - 1 Environment. *ACM Transactions on Software Engineering*, New York, v. 22, n. 12, 1996.

Barnes, A.; Gray, J. Cots, Workflow, and Software Process Management: An Exploration of Software Engineering Tool Development. In: *Australian Software Engineering Conference*,

Australia, 2000.

Basili, V. R.; Caldiera, G.; Rombach, H. D. The GQM approach. *Encyclopedia of Software Engineering (Wiley)*, 1994.

Bass, J. M. Influences on Agile Practice Tailoring in Enterprise Software Development. In: *AGILE India (AGILE INDIA)*, Bengaluruvol, 2012, p. 1 9.

Bassi, L.B.F. *Experiências com desenvolvimento ágil*. Dissertação de Mestrado. São Paulo: IME – USP, 2008.

Batra, D.; Xia, W.; Vandermeer, D.; Dutta, K. Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry. *Communications of the Association for Information Systems*, vol. 27, p. 379 - 394, 2010.

Beck, K. *Extreme Programming Explained: Embrace Change*. Reading. Addison-Wesley, 2004.

Beck, K.; Beedle, M.; Van bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Kern, J.; Marick, B.; Martin, R. C.; Mellor, S.; Schwaber, K.; Sutherland, J.; Thomas, D. Manifesto para desenvolvimento ágil de software. 2001. Disponível em http://agilemanifesto.org/iso/ptbr/manifesto.html. Acesso em: 22 out. 2014.

Betemps, C. M. *Ambientes de Desenvolvimento de Software Baseados em Workflow*. Dissertação de Mestrado. Porto Alegre: Instituto de Informática - Universidade Federal do Rio Grande do Sul, 2003.

Bosua, R. e Brinkkemper, S. Realisation of an integrated software engineering environment through heterogeneous CASE-tool integration. In: *Proceedings of the Software Engineering Environments Conference*, Noordwijkerhout, 1995, p. 152 159.

Broberg, L. L. *A Grounded Theory Approach to Examining Design and Usability Guidelines for Four-year Tribal College Web Sites*. Capella University, 2011.

Brown, A. W. An Examination of the Current State of IPSE Technology. In: *International Conference on Software Engineering (ICSE)*, Baltimore, 1993

Callahan, J.; Ramakrishnam, S. Software Project Management and Measurement on the World-Wide-Web (WWW). In: *Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Stanford, 1996.

Chan, D. K. C.; Leung, K. R.P.H. A Workflow Vista of the Software Process. In: *International Workshop on Database and Expert Systems Applications (DEXA)*, Toulouse, France, 1997.

Chan, D. K. C. Form Management in the Vicomte Workflow System. In: *International Conference on System Sciences (HICSS)*, Maui, 1999.

Chappel, D. *What is Application Lifecyle Management?* Relatório técnico, David Chappel & Associates, 2014. Disponível em http://www.davidchappell.com/writing/white_papers/What-is-ALM--Chappell.pdf. Acesso em: 5 mai. 2015.

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

Collins, E.; Dias-Neto, A.; de Lucena, V. F. Strategies for Agile Software Testing Automation: An Industrial Experience. In: *Computer Software and Applications Conference Workshops (COMPSACW)*, Izmir, 2012, p. 440 445.

Corbin, J. M.; Strauss, A. L. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. 3 ed. Sage Publications, 2008.

Creswell, J. W.; Clark, V. L. P. *Designing and Conducting Mixed Methods Research*. 2 ed. SAGE, 2010.

Dias J. W. *Evidência Empírica das Abordagens Composicional e Anotativa para Gerência de Variabilidades em Linhas de Processo de Software*. Dissertação de Mestrado, Universidade Estadual de Maringá, Brasil, 2015.

Dyba, T.; Prikladnicki, R.; Rönkkö, K.; Seaman, C.; Sillito, J. Qualitative Research in Software Engineering. *Empirical Software Engineering*, v. 16, n. 4, p. 425–429, 2011.

Ferenc, R.; Lango, L.; Siket, I.; Gyimothy, T.; Bakota, T. Source Meter Sonar Qube Plug-in. In: *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working*

Conference, Victoria, 2014, p.77 82.

Finkelstein, A.; Kramer, J.; Nuseibech, B. *Software Process Modelling and Technology*. Chichester, UK: John Wiley & Sons: Research Studies Press, 1994.

Franky, M. C. Agile Management and Development of Software Projects Based on Collaborative Environments. In: *Software Engineering. Notes (SIGSOFT)*, v. 36, n. 3, pp. 1-6, 2011. Disponível em: http://doi.acm.org/10.1145/1968587.1968605. Acesso em: 10 Abr. 2015.

Fowler, M. Continuous Integration. 2006. Disponível em: http://martinfowler.com/articles/continuousIntegration.html. Acesso em: 15 mar. 2015.

Fowler, M. *UML essencial: um breve guia para a linguagem-padrão de modelagem de objetos*. 3. Ed. Porto Alegre: Bookman, 2005.

Fowler, M.; Beck K.; Brant J.; Opdyke W.; Roberts D. *Refactoring: Improving the design of existing programs*. Addison-Wesley, 1999.

Fuggetta, A. Functionality and Architecture of PSEE. *Information and Software Technology*, v. 38, p. 289 - 293, 1996.

Fuggetta, A.; Di Nitto, E. Software process. *ACM Press*, p. 1 – 12, 2014. Disponível em http://dl.acm.org/citation.cfm?doid=2593882.2593883. Acesso em: 2 fev. 2015.

Geraldi, R. *SMartyCheck: uma Técnica de Inspeção baseada em Checklist para Diagramas de Casos de Uso e de Classes da Abordagem SMarty*. Dissertação de Mestrado, Universidade Estadual de Maringá, Brasil, 2015.

Gimenes, I. M. S. O Processo de Engenharia de Software: ambientes e formalismos. In: *Sociedade Brasileira de Computação*, Caxambú, 1994.

Gimenes, I. M. S.; Weiss, G. M.; Huzita, E. H. M. Um Padrão para Definição de um Gerenciador de Processos de Software. In: Jornadas Iberoamericanas de Ingenieria de Requisitos y Ambientes de Software (IDEAS), Alajuela, 1999.

Gandomani, T.J.; Zulzalil, H.; Ghani, A. A. A.; Sultan, A. B. M.; Nafchi, M. Z. Obstacles in

Moving to Agile Software Development Methods: At a Glance. *Journal of Computer Science*, v. 9, p. 620 - 625, 2013.

George B.; Williams L. A Structured Experiment of Test-Driven Development. *Information and Software Technology*, v. 46, n. 5, p. 337 - 342, 2003. Disponível em: http://www.sciencedirect.com/science/article/pii/S0950584903002040. Acesso em: 05 mar. 2015.

Gregory, P.; Barroca, L.; Taylor, K.; Salah, D.; Sharp, H. Agile challenges in practice: a thematic analysis. In: *16th International Conference on Agile Software Development (XP 2015)*, Helsinki, 2015.

Haugen, N. C. An empirical study of using planning poker for user story estimation. In: *Agile Conference (AGILE)*, Minneapolis, 2006, p. 34 43.

Humble J.; Farley D. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

Jarke, M.; Nissen, H.; Pohl, K. Tool Integration in Evolving Information Systems Environment. In: Gi Workshop Information Systems and Artificial Intelligence: Administration and Processing of Complex Structures, Hamburg, 1994.

Kalermo, J. E Rissanen, J. Agile software development in theory and practice. Dissertação de Mestrado, Universidade de Jyväskylä, Finlândia, 2002.

Kääriäinen, J. E Välimäki, A. Impact of Application Lifecycle Management – A Case Study. In: *Conference on Interoperability of Enterprise, Software and Applications (IESA)*, Berlin, 2008, p. 55 67.

Kim, J. A.; Choi, S. Y.; Hwang, S. M. Process & Evidence enable to automate ALM (Application Lifecycle Management). In: *Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops (ISPAW, 2011)*, Busan, 2011, p. 348 351.

Kovair M. *ALM and Integrated ALM. 2016*. Disponível em http://www.kovair.com/What-are-ALM-and-Integrated-ALM.pdf. Acesso em: 5 mai. 2016.

Kravchik, M. *Application Lifecycle Management Environments: Past, Present and Future.* Dissertação de Mestrado, The Open University of Israel, 2009

Highsmith, J.; Cockburn, A. Agile software development: the business of innovation. *Computer*, v. 34, n. 9, p. 120 - 127, 2001.

Lacheiner, H.; Ramler, R. Application Lifecycle Management as Infrastructure for Software Process Improvement and Evolution: Experience and Insights from Industry. In: *EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA, 2011)*, Oulu, 2011, p. 286 293.

Lee, K. How Application Lifecycle Management can address eLearning Software Challenges. *International Journal of The Computer, the Internet and Management*, v. 12, p. 154 - 161, 2004. Disponível em http://www.ijcim.th.org/past_editions/2004V12N2(SP)/pdf/p154-161-KarenLee-Borland-eLearning_Software_Challenges.pdf. Acesso em: 15 ago. 2016.

Manzoni, L. Uso de Sistema de Gerência de Workflow para Apoiar o Desenvolvimento de Software Baseado no Processo Unificado da Rational Estendido para Alcançar Níveis 2 e 3 do Modelo de Maturidade. Dissertação de Mestrado, Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2001.

Maurer, F.; Succi, G.; Holz, H.; Kötting, B.; Goldmann, S.; Dellen, B. Software Process Support over the Internet. In: *International Conference on Software Engineering (ICSE)*, Los Angeles, 1999.

Maximilien, E. M.; Williams, L. Assessing test-driven development at IBM. In: *Software Engineering*, 2003, p. 564 569.

Mcconnell, S. C. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press, 2004.

Melo, C. O.; Santos, V. A.; Corbucci, H.; Katayama, E.; Goldman, A.; Kon, F. *Métodos Ágeis no brasil: Estado da prática em times e organizações*. Relatório técnico RT-MAC-2012?03, Departamento de Ciência da Computação. IME - USP, 2012. Disponível em http://ccsl.ime.usp.br/agilcoop/artigos. Acesso em: 20 jan. 2015.

Notari, D. *Uma Enciclopédia de Ambientes Distribuídos de Desenvolvimento de Software*. Dissertação de Mestrado. Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000.

Oinas-Kukkonen, H.; Rossi, G. On Two Approaches to Software Repositories and Hypertext Functionality. *Journal of Digital Information*, v. 1, n. 4, 1999. Disponível em: https://journals.tdl.org/jodi/index.php/jodi/article/view/14/13. Acesso em: 15 jun. 2015.

Opdyke, W. F. *Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks*. Tese de Pós Doutorado. University of Illinois, 1992. Disponivel em: http://www.ai.univ-paris8.fr/~lysop/opdyke-thesis.pdf. Acesso em: 21 mar. 2015.

Ortigosa, A. M. *Proposta de um Ambiente Adaptável de Apoio ao Processo de Desenvolvimento de Software*. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1995.

Perin, M. G. *Um Sistema de Gerenciamento de Hiperdocumentos para Ambientes de Desenvolvimento de Software*. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1992.

Pressman R. S. *Engenharia de Software, Uma abordagem profissional*. 7. Ed. New York: McGraw-Hill, 2011.

Razak, R. A.; Fahrurazi, F. R. Agile testing with Selenium. In: *Software Engineering (MySEC)*, Johor Bahru, 2011, p. 217 219.

Rigby, P.C.; Cleary, B.; Painchaud, F.; Storey, M.; German, D.M. Contemporary Peer Review in Action: Lessons from Open Source Development. *Software, IEEE*, vol. 29, no. 6, p. 56 - 61, 2012.

Rossberg, J. *Pro Visual Studio Team System Application Lifecycle Management*. New York: Apress, 2008.

Rossman B. *Application Lifecycle Management - Activities, Methodologies, Disciplines, Tools, Benefits, ALM Tools and Products*. New York: Tebbo, 2010.

Runeson, P. A survey of unit testing practices. *Software, IEEE* , vol.23, no.4, p. 22 - 29, 2006.

Seaman, C. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering (TSE)*, v. 25, n. 4, p. 557–572, 1999.

Schwaber, K. e Beedle, M. *Agile software development with Scrum*. Prentice-Hall, 2002.

Schwaber, K.; Sutherland, J., *Guia do Scrum*. 2013. Disponível em http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf. Acesso em: 10 nov. 2014.

Silva, G. C. *DiSEN-CI: Um servidor de integração contínua para modelos*. Dissertação de Mestrado. Maringá: Universidade Estadual de Maringá, 2013.

Sommerville I. *Software Engineering*. Addison Wesley, 2010.

Striebeck, M. Ssh! We are adding a process... [agile practices]. *Agile Conference*, Minneapolis, 2006, p. 185 193.

Swebok. *Guide to the Software Engineering Body of Knowledge*. 2014 Version. Disponível em: http://www.computer.org/web/swebok/v3. Acesso em: 20 nov. 2014.

Thomas, I.; Nejmeh, B. A. Definitions of Tool Integration for Environments. *IEEE Software*. p. 29 – 35, 1992.

van Solingen, R.; Basili, V.; Caldiera, G.; Rombach, H. D. *Goal Question Metric (GQM) Approach*. John Wiley and Sons, 2002.

Versionone. *VersionOne 10th annual state of agile survery*. 2015. Disponível em http://www.versionone.com. Acesso em: 20 ago. 2015.

Vonken, F.; Zaidman, A. Refactoring with Unit Testing: A Match Made in Heaven? In: *Reverse Engineering (WCRE, 2012)*, Kingston, 2012, p. 29 38.

Wasserman, A. Tool Integration in Software Engineering Environments. In: International Workshop on Environments Proceedings - Software Engineering Environments, Springer-Verlag, 1989, p. 137 149.

Williams, L.; Cockburn, A., Agile software development: it's about feedback and change. *Computer*, v. 36, n. 6, p. 39 - 43, 2003.

Apêndice A

A.1. Questionário de Caracterização do Participante

Questionário de Caracterização de Participante em Estudo de Caso Empírico Qualitativo

"Avaliação de Efetividade do ADE"

ID PARTICIPANTE

Nas perguntas a seguir, quando duas ou mais alternativas forem válidas, marque a alternativa que mais se aplica ao seu caso.

1 - Qual o seu nível de formação?

[ ] Graduando
[ ] Pós-graduando (Especialização)
[ ] Mestrando
[ ] Doutorando

[ ] Graduado
[ ] Pós-graduado (Especialização)
[ ] Mestre
[ ] Doutor

2 - Em qual setor atua?

[ ] Acadêmico (ensino)
[ ] Industrial (empresarial/corporativo)

3 - Qual o nome da empresa/universidade que atua?

_____

4 - Quanto tempo possui de experiência na área que atua?

_____ anos.

5 - Quanto tempo possui de experiência com processos ágeis?

_____ anos.

6 - Qual a sua experiência com Desenvolvimento de Software?

[ ] **Eu nunca ouvi falar a respeito sobre Desenvolvimento de Software.**
[ ] **Já lí, de forma superficial, algo a respeito sobre Desenvolvimento de Software.**
[ ] **Minha experiência com Desenvolvimento de Software é básica.**
Eu conheço os conceitos com relação ao cenário de desenvolvimento de software: análise de requisitos, implementação, padrões de projeto, arquitetura de software, testes e revisão de código.
[ ] **Minha experiência com Desenvolvimento de Software é moderada.**
Eu conheço os conceitos da opção anterior e possuo a experiência da participação do desenvolvimento de pelo menos uma aplicação.
[ ] **Minha experiência com Desenvolvimento de Software é avançada.**
Eu conheço os conceitos da opção anterior, além da experiência do desenvolvimento de mais de uma aplicação.

7 - Qual a sua experiência com Desenvolvimento de software com Métodos Ágeis?
[ ] **Eu nunca ouvi falar a respeito sobre Métodos Ágeis.**
[ ] **Já lí, de forma superficial, algo a respeito sobre Métodos Ágeis.**
[ ] **Minha experiência com Métodos Ágeis é básica.**
Eu conheço os conceitos com relação ao cenário de métodos ágeis: manifesto ágil, Scrum, Extreme Programming, testes automatizados, integração contínua.
[ ] **Minha experiência com Desenvolvimento de Software é moderada.**
Eu conheço os conceitos da opção anterior e possuo a experiência da participação de pelo menos um projeto usando métodos ágeis.
[ ] **Minha experiência com Desenvolvimento de Software é avançada.**
Eu conheço os conceitos da opção anterior, além da experiência em mais de um projeto que usa métodos ágeis.

ASSINATURA PARTICIPANTE	LOCAL E DATA
_____	_____,___de_____de____

A.2. Documentos de Apoio

Elementos do Ambiente Ágil de Desenvolvimento (ADE)

A arquitetura lógica do ADE tem por objetivo mostrar os elementos que estão contidos no ADE. A integração entre todos os elementos disponíveis no ambiente é essencial para que possa automatizar o desenvolvimento do *software* e gerenciamento do projeto. A Figura 1 apresenta uma visão geral da arquitetura lógica do ADE.

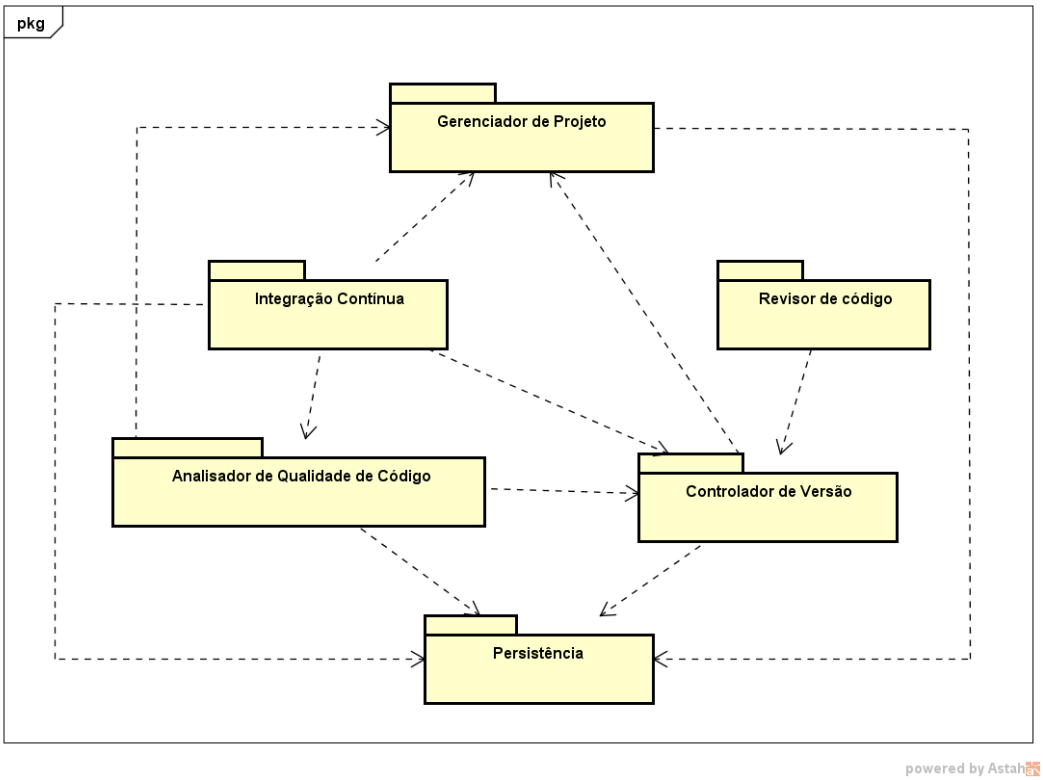


Figura 1. Visão geral da arquitetura lógica do ADE

A arquitetura foi definida com o objetivo de automatizar o ambiente juntamente com o processo de desenvolvimento, possibilitando a integração entre os elementos do ambiente e permitindo a construção de artefatos de software a cada alteração no controlador de versão. Dessa forma, corrobora com o método de gerenciamento, uma vez que, a cada *Sprint* uma nova versão do software deve ser gerada; soma-se a isso, o objetivo do ALM em realizar acompanhamento de todo o ciclo de vida da aplicação e visualização a qualquer momento. Na próxima seção, será definido o processo de desenvolvimento e como o ADE se integra com ele.

Processo de Desenvolvimento

O processo de desenvolvimento do ADE define a sequência das etapas a serem seguidas por uma equipe de desenvolvimento de software, para conceber e gerar uma aplicação, com base nos recursos disponíveis em ADE.

A Figura 2 apresenta o processo de desenvolvimento do ADE representado graficamente por um diagrama de atividades UML. Esta representação contém as etapas do processo e as informações manipuladas por elas, conforme descritas a seguir.

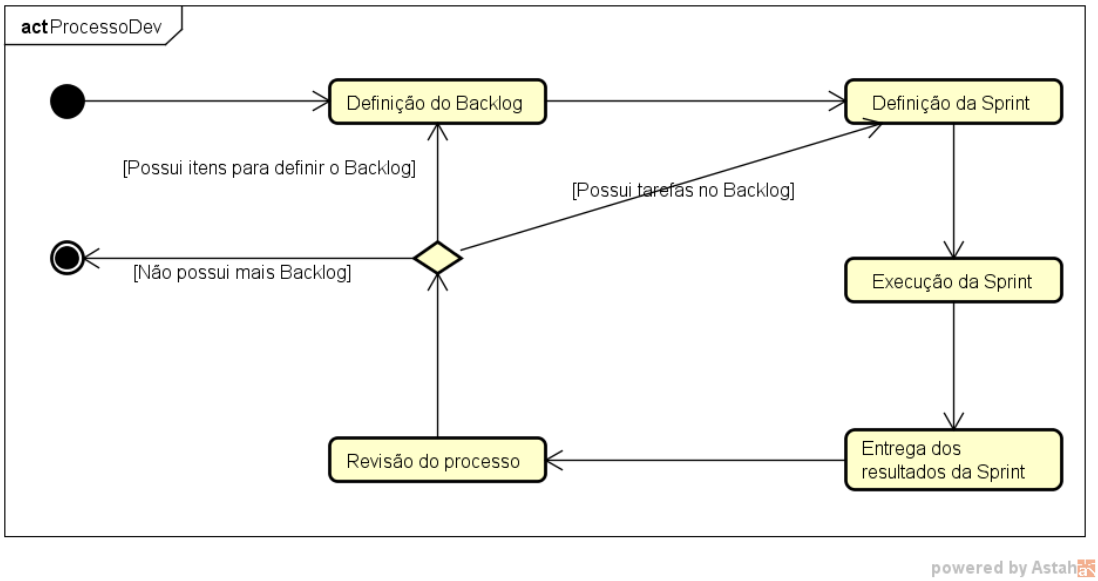


Figura 2: Representação gráfica do processo de desenvolvimento do ADE

Etapa 1 – Definição do *Backlog*

A etapa de definição do *Backlog* contém três subetapas: análise, criação do *Backlog* e priorização do *Backlog*. A primeira subetapa, a análise, visa compreender o problema para que o PO tenha a visão das funcionalidades necessárias para resolução do problema. O resultado da análise é estabelecido em histórias de usuário, e cada história representa uma funcionalidade. Deve-se representar as histórias de usuário no gerenciador de projeto para permitir a gestão do desenvolvimento do software. Dentro do gerenciador, pode-se chamar uma história de usuário de tarefa e, além disso, as histórias podem ser fragmentadas em subtarefas. Em seguida, é possível conceber o *Backlog* considerando as histórias definidas. Por fim, deve-se realizar a priorização do *Backlog* de acordo com as definições feitas pelo cliente e a ordem de precedências entre as histórias.

Etapa 2 - Definição da *Sprint*

Nesta etapa, é realizada a reunião de planejamento da *Sprint*, na qual o PO, o *Scrum Master* e time de desenvolvimento selecionam as tarefas que irão compor o *Backlog* da *Sprint*. Através do gerenciador de projeto, são identificadas as tarefas que já foram analisadas e priorizadas pelo PO, na etapa anterior, para serem estimadas por meio da prática de *Planning Poker*. Cada tarefa selecionada é atualizada com a estimativa dada pelo time de desenvolvimento no gerenciador de projeto. Além disso, o time de desenvolvimento define como serão realizadas as implementações de cada tarefa e quem realizará. Ao final desta

etapa, tem-se o *Backlog* da *Sprint*, o qual define as funcionalidades que serão entregues ao cliente ao final da *Sprint*.

Etapa 3 - Execução da *Sprint*

Nesta etapa, o *Backlog* da *Sprint* é implementado. Para o acompanhamento da *Sprint*, são realizadas reuniões diárias entre o time de desenvolvimento e o *Scrum Master* com o objetivo de identificar e corrigir possíveis problemas que podem atrapalhar a entrega da *Sprint*. Para a implementação de cada tarefa do *Backlog*, cinco passos devem ser executados. Os passos definidos para a implementação de cada tarefa estão representados na Figura 3, por meio de um diagrama de atividades UML.

1. Implementação: é realizada a codificação do software, de acordo com as decisões tomadas na etapa 2;
2. Testes Automatizados: este passo é responsável por realizar os testes automatizados. A transição entre os passos de implementação e testes automatizados deve ocorrer até que a atividade seja concluída, pois uma maior cobertura de teste aumenta a qualidade do código e diminui a geração de defeitos no *software* desenvolvido (Maximilien e Williams, 2003; George e Williams, 2004);
3. Testes Manuais: após a conclusão dos passos de implementação e testes automatizados, deve-se realizar testes manuais da funcionalidade adicionada. O ADE pode apoiar a identificação das funcionalidades afetadas por meio da ferramenta de revisão de código, na qual são destacados os trechos de código alterados;
4. Revisão de código: todo o código desenvolvido deve ser revisado por um profissional diferente para aumentar a probabilidade de identificar possíveis erros e melhorias a serem realizadas e assim diminuir o número de falhas do software produzido. O ADE apoia isso através do gerenciador de projeto por meio do processo definido; assim antes de finalizar uma tarefa, é necessário que a revisão de código seja realizada por um outro profissional que possui permissão para aprovar o código. Além disso, o serviço de revisão de código auxilia na execução deste passo ao destacar os trechos de código alterados e pelo analisador de qualidade de código, que retorna quais as regras de qualidade foram violadas;
5. Integração contínua: o código produzido deve ser integrado com os demais códigos que comporão o novo artefato. O processo de integrar o código,

executar os testes automatizados, detectar os causadores dos erros quando houver e gerar uma nova versão executável do software, são funções da etapa de IC. Este passo é realizado automaticamente pela ferramenta de IC configurada no ADE. Este executável pode ser entregue para o cliente ao final da *Sprint*. Além dos itens descritos, o servidor de integração contínua também disparará o analisador de qualidade de código para realizar as análises de cobertura de testes, quantidade de duplicações, complexidade do software, quantidade de comentários, dívida técnica e padrões de códigos. Os resultados das análises devem ficar visíveis para toda a equipe de desenvolvimento, além de ser usado na etapa 5, revisão do processo. Caso as regras de qualidade sejam quebradas, o IC dispara a criação automática de uma tarefa no gerenciador de projeto, para que seja corrigido o código gerador das quebras de qualidade.

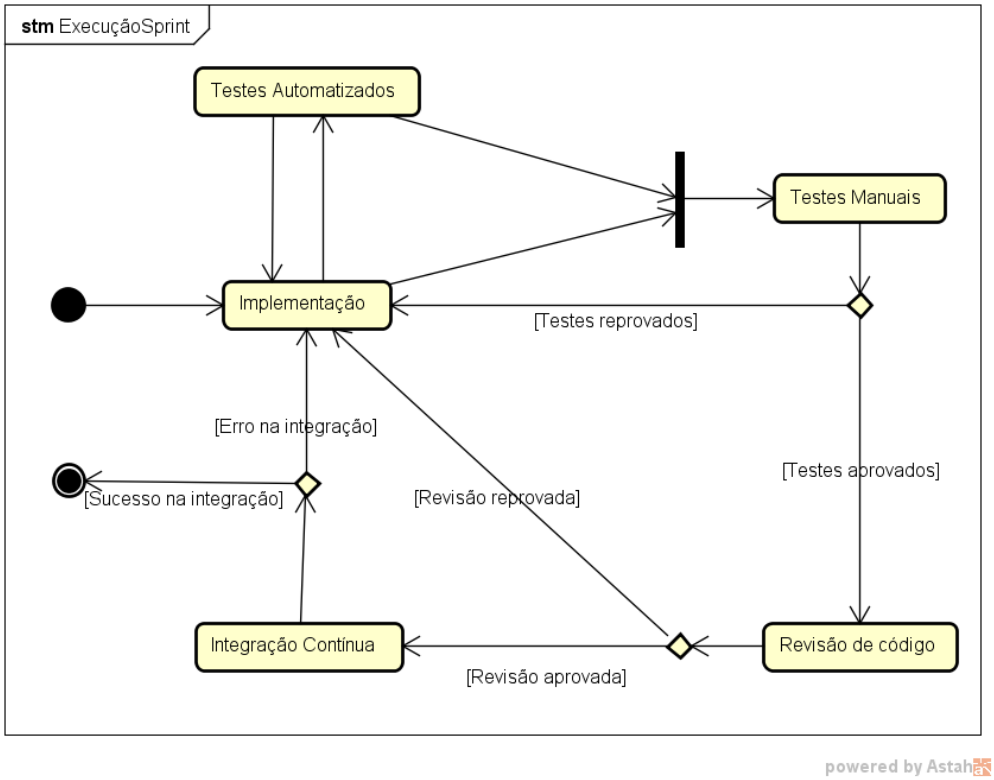


Figura 3: Representação gráfica da etapa Execução da Sprint

Etapa 4 - Entrega dos resultados da *Sprint*

Ao final da *Sprint*, uma nova versão do software com as novas funcionalidades deve ser gerada. Isso é feito pela integração contínua, a partir do último *commit* da *Sprint*. Assim, é realizada a entrega para o cliente e apresentadas as funcionalidades contidas nesta nova

versão. As funcionalidades são identificadas por meio das tarefas finalizadas na *Sprint,* pelo gerenciador de projeto.

Etapa 5 - Revisão do processo

A última etapa do processo do ADE é realizada a retrospectiva da *Sprint* e assim permite que o PO, o *Scrum Master* e o time de desenvolvimento revisem os problemas encontrados na execução da *Sprint*, como problemas na comunicação com o cliente, nos requisitos, falha no processo, estouro das estimativas para as tarefas, aumento da dívida técnica, entre outros problemas que podem surgir no decorrer da *Sprint*. Por conseguinte, possibilita inspecionar e adaptar o processo para a próxima *Sprint*. Alguns dos problemas discutidos nesta etapa podem ser obtidos, pelos membros do time de desenvolvimento, *Scrum Master* e PO, através dos elementos ADE. A Tabela 1 exemplifica-os.

Tabela 1: Elemento do ADE no qual é possível recuperar informações sobre problemas

Elementos do ADE	**Problemas/Dificuldades**
Gerenciador de projeto	Falha na estimativa, problemas nos requisitos, rastreabilidade dos *commits*, falha na execução do processo
Analisador de qualidade de código	Dívida técnica, cobertura de testes, problemas de *design* do código, código fora dos padrões
Integração contínua	Quais *commits* causaram falhas no software

Ao final da etapa 5, caso ainda existam funcionalidades no *backlog*, uma nova *Sprint* é iniciada voltando à etapa 1. A existência de funcionalidades no *backlog* é encontrada no elemento gerenciador de projeto, onde possui todas as funcionalidades mapeadas em forma de tarefas, e caso existam o PO deve iniciar uma nova *Sprint*.

Integração no ADE

A integração por dados no ADE é realizada por meio de serviços *Web*; assim cada ferramenta que necessita de dados produzidos ou armazenados em outra ferramenta faz uma solicitação quando necessário. Assim, é responsabilidade de cada serviço interpretar dos dados disponíveis por outro serviço. Esta integração é transparente ao usuário, uma vez que já existem *plug-ins* para a integração entre as ferramentas e basta configurá-los. Paralelamente, ainda são disponibilizados os serviços *Web* para implementações mais específicas quando

necessário. Não só o compartilhamento de dados é realizado; também é possível realizar alterações nos dados de outras ferramentas e inserir novos dados, através de serviços *Web*. Cada ferramenta possui seu mecanismo de armazenamento, seja ele um banco de dados ou sistema de arquivos. Dessa forma, não existe um repositório central para todos os artefatos gerados durante o desenvolvimento do software.

Em relação à integração de controle, ocorre pelas configurações nas ferramentas para que possibilite a ativação de outra ferramenta. Esta configuração não é realizada através da modelagem do processo no ADE, uma vez que ele apenas define o fluxo de trabalho e não possui o controle sobre as ferramentas. A Tabela 2 exemplifica as integrações por controle existentes no ADE.

Tabela 2: Descrição das integrações por controle no ADE

Elemento do ADE	**Caso de integração por controle**
Analisador de qualidade de código	Cria atividades no gerenciador de projeto quando quebrar as regras de qualidade
Controle de versão de código	Notifica o servidor de integração contínua que houve uma alteração no repositório
Integração contínua	Executa os testes automatizados, Ativa o analisador de qualidade de código, Cria atividades no gerenciador de projeto quando uma execução falhar

Em relação à integração por processo, o ADE não é baseado explicitamente por meio do processo; a definição dele no ADE é realizada dentro do gerenciador de projetos, e tem o objetivo de definir as etapas que devem ser realizadas para o desenvolvimento do software. Porém, o processo não define um controle em relação às ferramentas ou uma definição automatizada para que as demais ferramentas do ambiente sejam executadas apenas pela definição do processo. As alterações no processo de desenvolvimento podem ser realizadas de maneira dinâmica pelo gerenciador de projeto.

A.3. Respostas dos Especialistas

1 - Quais etapas você removeria, adicionaria ou alteraria do processo de desenvolvimento do ADE?

Resposta Especialista Nº 1: Senti falta de ferramentas que auxiliem os seguintes processos:

1) Mapeamento dos módulos/requisitos: Ferramenta que realize o rastreamento dos módulos ou requisitos do sistema, de modo a registrar quais são as partes mais alterados, seus custos, e recursos que possuem conhecimento nessas partes.

2) Ferramenta que gerencie e versione as configurações realizadas no banco de dados durante os processos de implantação. O ideal é que essa ferramenta permitisse a seleção e configuração de demandas específicas, facilitando assim a replição das configurações em ambientes secundários como de homologação

3) Ferramenta de virtualização que emule o ambiente de produção

Resposta Especialista Nº 2: Sim.

Resposta Especialista Nº 3: Não alteraria nenhum processos. Esses processos são suficientes para garantir o mínimo de qualidade tanto no código como no gerenciamento de todo o processo de desenvolvimento

Resposta Especialista Nº 4: Creio que esteja completo. Não vejo alteração nas ferramentas. Talvez a inclusão de plugins dentro do redmine para controle agil das demandas e exibição de backlog. No redmine padrão, as tarefas são exibidas com a visão do grafico GANT.

Resposta Especialista Nº 5: Acredito que esteja completo.

Resposta Especialista Nº 6: Proporia alteração adicionando alguma ferramenta para métrica das sprints desenvolvidas, para que seja possível a visualização dos resultados obtidos, e a aprendizagem contínua a partir dos resultados.

Resposta Especialista Nº 7: Não tenho alterações para propor.

2 - As técnicas e práticas contidas do ADE são suficientes para garantir a qualidade e o gerenciamento do código desenvolvido?

Resposta Especialista Nº 1: Sim, desde que as seguintes informações sejam armazenadas:

- Dados sobre o cronograma do projeto
- Lista das tarefas e suas dependências
- Custo dos recursos necessários na execução das tarefas.

Resposta Especialista Nº 2: Sim.

Resposta Especialista Nº 3: Sim, pois o código bem escrito por si só um uma fonte de documentação do sistema, as outras ferramentas produzem informações suficientes para manter um bom gerenciamento de um projeto de software

Resposta Especialista Nº 4: Sim. Para o gerenciamento do projeto de software sim, mas para o gerenciamento de um projeto como todo não. Não há evidências para o gerenciamento de comunicação, riscos e custos (que é caracteristica de um projeto).

Resposta Especialista Nº 5: Sim.

Resposta Especialista Nº 6: Nunca trabalhei com a ferramenta REDMINE, mas acredito que ela possa fornecer todas as informações necessárias.

Resposta Especialista Nº 7: Sinto falta de informações de resultados da sprint que possibilitam o aprendizado contínuo.

3 - Na sua opinião, as práticas, técnicas e ferramentas de apoio contidas no ADE aumentam a qualidade do software a ser desenvolvido?

Resposta Especialista Nº 1: Sim, porque existem etapas específicas para a definição do própósito do projeto (definição do escopo) e a validação da entrega (Revisão). Considero também que parte do sucesso do projeto está relacionado à alocação de equipe qualificada durante a execução. Não identifiquei nenhuma fase do processo onde as atividades de seleção e alocação da equipe esteja presente, se não estiver sendo considerado, esse pode ser um pode que afete a qualidade da entrega.

Resposta Especialista Nº 2: Sim.

Resposta Especialista Nº 3: Sim, pois através da ferramenta de ADE podemos amarrar todo o código versionado e gerações de versões para o gerenciamento da ciclo de vida da aplicação. Com isso gerando uma base histórica e bem documentada da aplicação.

Resposta Especialista Nº 4: Sim, aumentam. Mas este aumento só ficará evidenciado com a ação de pessoas que entendam a qualidade a ser produzida.

Resposta Especialista Nº 5: Acredito que a adaptação para a realidade da empresa das técnicas e ferramentas de apoio contidas no ADE aumentam a qualidade do software. ADE não pode ser uma caixa preta.

Resposta Especialista Nº 6: Sim. Elas automatizam vários processos que poderiam gerar transtorno se feitas manualmente, e também adicionam insumos com code review e analisador de qualidade de código, integração contínua que não são realidades em vários centros de desenvolvimento.

Resposta Especialista Nº 7: Acho indiferente, pois na minha opinião as práticas, técnicas e ferramentas por si só não aumentam a qualidade do código. Conhecimento adquirido através de estudos, ou através de compartilhamento com outras pessoas aumentam a qualidade do código. Elas podem aumentar talvez a produtividade.

4 - Na sua opinião, o processo de desenvolvimento definido é suficientemente completo? Você adicionaria, removeria ou alteraria o processo proposto?

Resposta Especialista Nº 1: Incluir a etapa de mobilização da equipe e indicadores de monitoramento do desenvolvimento, apresentando dados do projeto como: Orçamento estimado x Realizado. Progresso estimado x Realizado.

Resposta Especialista Nº 2: Incluiria um analisador estático de código.

Resposta Especialista Nº 3: Sim, talvez removeria a ferramenta de revisor de código, caso essa ferramenta exigisse muito processo manual, caso contrário, se fosse totalmente automatizada, não.

Resposta Especialista Nº 4: Não removeria e nem adicionaria nenhuma etapa no momento. Os gráficos contemplam com excelência o processo de desenvolvimento e entrega de funcionalidades. Talvez só colocaria alguma definição para uma ferramenta que torne público o andamento das sprints, isto atenderia os PO´s e geraria mais credibilidade a sequência dos trabalhos.

Resposta Especialista Nº 5: Acredito que Planning Poker não seja a única maneira de realizar estimativas. Nesta etapa poderiam ter mais opções. Também, senti falta do pair programming.

Resposta Especialista Nº 6: Acredito que não, acho primordial alguma ferramenta que nos entregue métricas que nos permitam melhorar o processo pelo aprendizado contínuo, e historize estes avanços.

Resposta Especialista Nº 7: Acredito ser um processo completo, porém a sua aplicação por completo depende muito do projeto que será implementado, quantidade de pessoas, custo e prazo.

5 - Você avalia que o ADE favorece a produtividade do desenvolvimento de software?

Resposta Especialista Nº 1: Sim por conta das ferramentas de execução de cenários e de integração contínua, além da atividade de revisão da tarefa. A implantação de uma ferramenta que possibilite emular o ambiente de produção no ambiente de desenvolvimento, também traria ganho de produtividade dado que minimiza as possibilidades de construir, testar e revisar parte do escopo do projeto em um cenário inexistente no ambiente real de produção.

Resposta Especialista Nº 2: Sim.

Resposta Especialista Nº 3: Sim, com as ferramentas bem alinhadas e automatizadas o ganho de produtividade do desenvolvimento de software é uma consequência.

Resposta Especialista Nº 4: Em partes. No meu entendimento, a produtividade não acontece sem a supervisão humana. O ambiente da ADE traz uma caixa de ferramentas e indicadores para a tomada de decisão. Cabe a uma pessoa pegar estas informações e transformar em produtividade (removendo empecilhos, organizando trabalho, gerenciando comunicação).

Resposta Especialista Nº 5: Como já dito anteriormente, favorece a produtividade desde que adaptado a realidade da empresa.

Resposta Especialista Nº 6: Se bem implementado sim, caso contrário pode gerar vários pontos de gargalos.

Resposta Especialista Nº 7: Sim, acredito que favorece a produtividade.

6 - Na sua opinião, as integrações entre as ferramentas no ADE são eficazes para garantir a automação do acompanhamento e desenvolvimento do software?

Resposta Especialista Nº 1: Sim para a etapa de desenvolvimento. No entanto, senti falta de ferramentas que automatizem o processo de monitoramento e controle do projeto.

Resposta Especialista Nº 2: Sim.

Resposta Especialista Nº 3: Sim, o rastreio das informações legado deixado pelas integração acredito ser o fator mais importante para o acompanhamento para o desenvolvimento do software.

Resposta Especialista Nº 4: Sim. Com este cenário é possível integrar o ambiente de software.

Resposta Especialista Nº 5: Sim. Integração é essencial.

Resposta Especialista Nº 6: Sim, faltando a automação que nos permita a evolução continua do sistema de desenvolvimento de software.

Resposta Especialista Nº 7: Sim.

7 - Quais seriam as barreiras e dificuldades na adoção do ADE?

Resposta Especialista Nº 1: Culturais por parte da gestão que precisa enxergar as atividades de revisão de código como um ponto de ganho de performance e qualidade, e não como um desperdício de recurso. Outro ponto de entrave é a dificuldade do cliente conseguir prever a data fim e custo total da construção de todo seu backlog. Intelectuais por parte da dificuldade em encontrar profissionais que saibam criar cenários de teste automatizados, muito menos modelar uma solução de negócio de modo a facilitar a construção dos cenários de teste.

Resposta Especialista Nº 2: Resistência a mudança, preconceitos já existentes.

Resposta Especialista Nº 3: Uma das dificuldade é a mudança de cultura da equipe de desenvolvimento para usar esse ambiente. No início alguns desenvolvedores não são adeptos a essas mudanças isso pode causar dados inconsistentes na ferramenta, como por exemplo não avançar as tarefas na ferramenta de gerenciamento.

Resposta Especialista Nº 4: Falta de hardware e tempo para investir em inovação. Falta de pessoas capacitadas e interessadas em investir nisto. Hoje os projetos são imediatistas.

Resposta Especialista Nº 5: Cultura da empresa e aversão a mudanças das pessoas.

Resposta Especialista Nº 6: Realizar a integração entre as ferramentas de maneira sustentável e que não gere gargalo. Manter esta estrutura.

Resposta Especialista Nº 7: Acredito que a principal barreira aqui é a cultura organizacional, uma empresa não acostumada a investir em testes automatizados por exemplo pode relutar em adotar as ferramentas propostas.

A.4. Pesquisa Exploratória sobre Ferramentas

O objetivo desta pesquisa exploratória é analisar os projetos de software na indústria para reconhecer as ferramentas mais usadas pela indústria, praticantes de métodos ágeis, e entender como elas podem ser coordenadas para que possibilitam a integração das práticas ágeis. Dessa forma constituindo um ambiente de desenvolvimento de software produtivo.

Busca-se informações sob a perspectiva do que é efetivamente realizado na indústria, em termos de ambiente de desenvolvimento de software. Opiniões sobre como tais atividades devem (ou poderiam) acontecer, em um sentido estritamente teórico, não são desejáveis para o intuito desta pesquisa.

A audiência alvo desta pesquisa são os profissionais com experiência em projetos ágeis. Podem fazer parte desta população aqueles que exercem os papéis de:

- Engenheiro de software
- Programador de sistemas
- Gerente de projeto
- Arquiteto de software
- Líder de projeto

Esses respondentes são caracterizados por já terem participado ou ainda fazerem parte de equipes de projetos de desenvolvimento de software cujo gerenciamento é conduzido sob

as guias de algum método ágil.

Especial cuidado com respostas de consultores e *coaches* para projetos ágeis que podem, eventualmente, sugerir como arquitetura deve ser conduzida, ao invés de como é efetivamente realizada.

As perguntas ficaram disponíveis por 30 dias no Google Forms. Os respondentes decidem se querem participar (ou não) da pesquisa. Foram convidados a participar os membros das seguintes comunidades:

- Grupos ágeis indexados pela *Agile Alliance*;
- Grupo "*Agile and Lean software development*" do *Linkedin*.

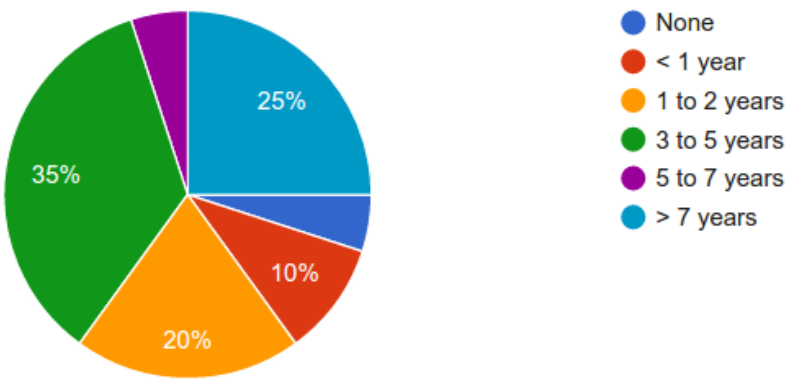
A *Agile Alliance* (https://www.agilealliance.org/) é uma organização sem fins lucrativos, com alcance global, que visa fomentar o avanço das práticas e princípios do desenvolvimento ágil. Ela mantém uma lista de grupos ágeis atuantes em alguns países nos cinco continentes. A lista completa e os e-mails de contatos dos responsáveis está disponível em http://agilealliance.org/resources/agile-user-groups/.

O *Linkedin* (https://www.linkedin.com) é uma plataforma social que reúne redes de profissionais. Nela existe o grupo "*Agile and Lean software development*" que agrega entusiastas e praticantes de vários métodos ágeis, sendo esta a maior comunidade de praticantes de métodos ágeis do *Linkedin*. O grupo pode ser acessado por https://www.linkedin.com/grp/home?gid=37631.

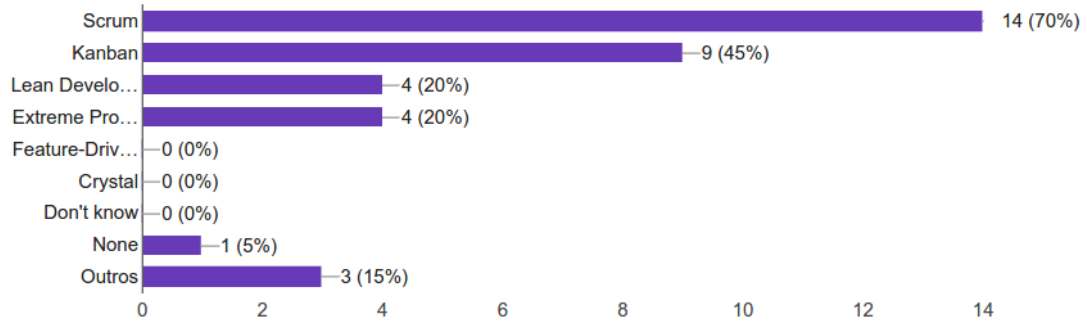
Resultados

A quantidade total de questionários respondidos foram 20. As respostas para o questionário estão descritas a seguir.

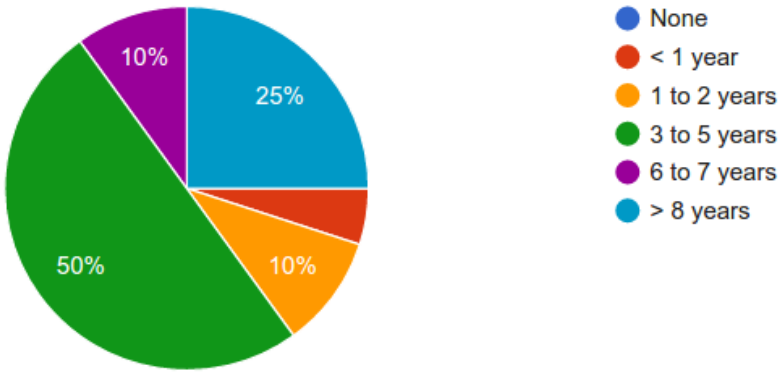
Aproximadamente quantos anos sua empresa tem praticado métodos de desenvolvimento ágil?



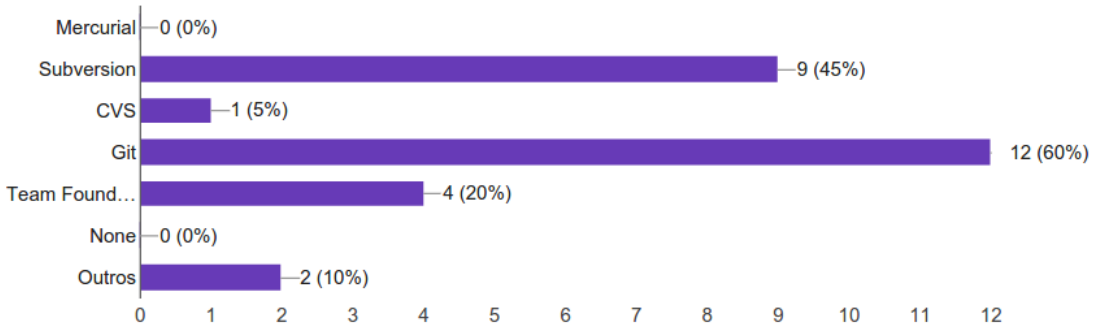
Qual é a metodologia ágil mais utilizada em sua empresa? (Marque todas as que se aplicam)



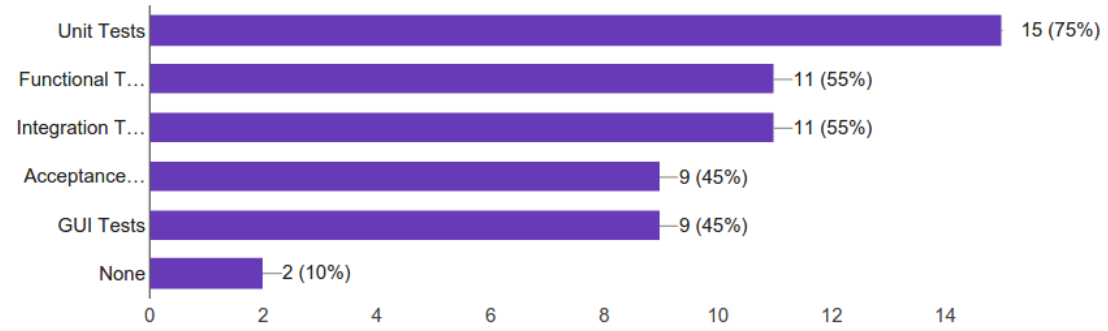
Avalie sua experiência pessoal com práticas de desenvolvimento ágil



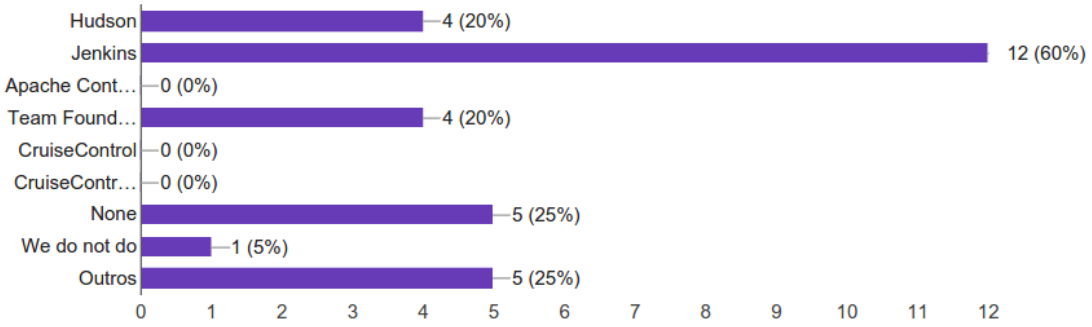
Qual ferramenta de controle de versão é usada em sua empresa atual? (Marque todas as que se aplicam)



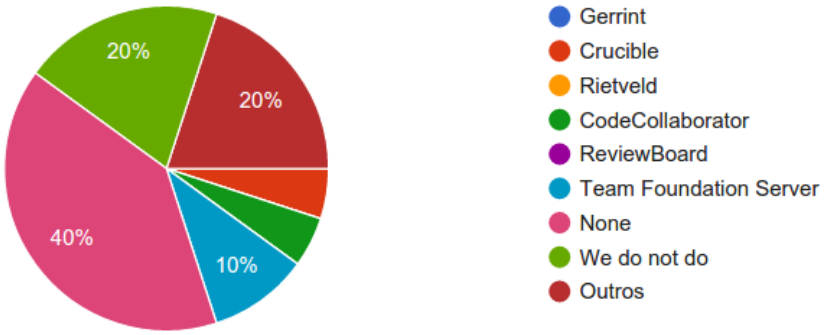
Que testes automatizados são usados em sua empresa? (Marque todas as que se aplicam)



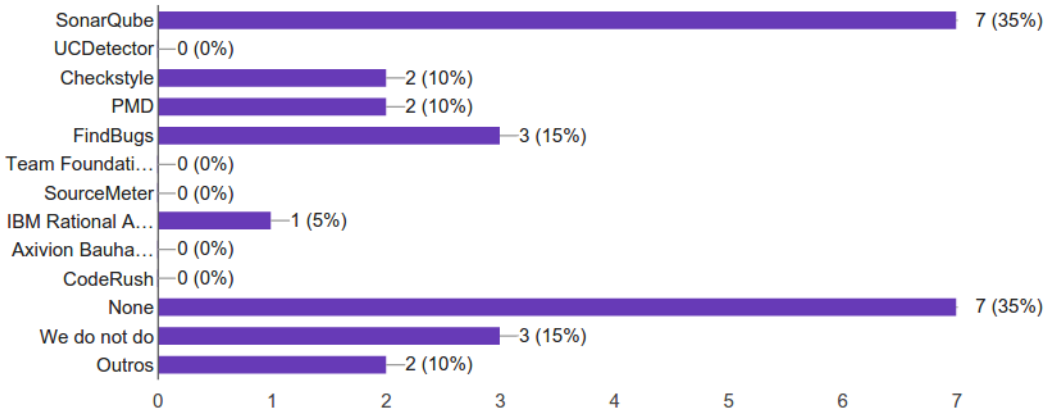
Que ferramenta de integração contínua é usada em sua empresa? (Marque todas as que se aplicam)



Qual ferramenta de revisão de código é usada em sua empresa?



Qual ferramenta é usada para codificar o analisador de qualidade? (Marque todas as que se aplicam)



Qual ferramenta de gerenciamento é usada em sua empresa? (Marque todas as que se aplicam)