

UNIVERSIDADE FEDERAL DE RORAIMA

CENTRO DE CIÊNCIA E TECNOLOGIA

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

CURSO DE CIÊNCIA DA COMPUTAÇÃO

LILIANE DE MATOS SOARES

**UM REPOSITÓRIO DIGITAL DE APOIO À PESQUISA USANDO XP EM SOFTWARE COMO SERVIÇO**

Boa Vista, RR

2016

LILIANE DE MATOS SOARES

**UM REPOSITÓRIO DIGITAL DE APOIO À PESQUISA USANDO XP EM SOFTWARE COMO SERVIÇO**

Anteprojeto apresentado ao Curso de Bacharelado em Ciência da Computação do Departamento de Ciência da Computação da Universidade Federal de Roraima.

Orientadora: Profº. Msc. Delfa Mercedes Huatuco Zuasnábar.

Boa Vista, RR

2016

LILIANE DE MATOS SOARES

**UM REPOSITÓRIO DIGITAL DE APOIO À PESQUISA USANDO XP EM SOFTWARE COMO SERVIÇO**

Anteprojeto apresentado ao Curso de Bacharelado em Ciência da Computação do Departamento de Ciência da Computação da Universidade Federal de Roraima. Área de concentração: Engenharia de software. Defendido em 04 de abril de 2016 e avaliada pela seguinte banca examinadora:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profa. Msc. Delfa Mercedes Huatuco Zuasnábar

Orientadora / Curso de Ciência da Computação – UFRR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profº. Msc. Miguel Raymundo Flores Santibanez

Curso de Ciência da Computação - UFRR

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Profº. Esp. Filipe Dwan Pereira

Curso de Ciência da Computação - UFRR

**AGRADECIMENTOS**

Agradeço primeiramente a Deus pela vida e pela oportunidade.

Agradeço imensamente a minha mãe, que com todo amor, exemplo de vida e força, mostrou-me o caminho para chegar até aqui, e sem a qual nada disso seria possível. Dedico à minha família a concretização desse sonho, aos meus irmãos, primos e tios que estiveram sempre me apoiando na longa trajetória da graduação.

Agradeço, também, à sábia orientadora que se dedicou ao meu acompanhamento e com sua sapiência pôde instruir-me.

Por fim, agradeço aos meus amigos, que na simplicidade da luz de nossa amizade me trouxeram felizes momentos e lembranças de uma época de estudos que com muito carinho e orgulho ei de lembrar.

Algumas pessoas acham que foco significa dizer sim para a coisa em que você vai se focar. Mas não é nada disso. Significa dizer não às centenas de outras boas ideias que existem.

(Steve Jobs)

**RESUMO**

Com o avanço da internet e da computação em nuvem, o paradigma de desenvolvimento de software também mudou, existem uma tendência em migrar as aplicações ou desenvolver as mesmas no ambiente em nuvem, onde não há necessidade de instalar programas, tudo pode ser acessado pelo navegador na Internet, essa é a característica de software como serviço. Nesse mesmo rumo os métodos ágeis têm sido muito discutidos e utilizados pelos seus resultados no desenvolvimento de software. Neste sentido, o presente trabalho visa aplicar as os métodos XP, BDD e TDD para o desenvolvimento de um repositório bibliográfico digital com o objetivo de apoiar a pesquisa acadêmica, que irá funcionar como um serviço acessado pela Internet.

**Palavras-chaves:** Computação em nuvem, software como serviço, XP, BDD, TDD.

**ABSTRACT**

With the advancement of Internet and cloud computing, software development paradigm has changed, there is a tendency to migrate the applications or develop them in the cloud environment, where there is no need to install software, everything can be accessed by the browser on the Internet, this is the software as a service feature. In the same way the agile methods have been widely discussed and used by their results in the development of software. In this sense, this work aims to apply the XP methods, BDD and TDD for the development of a digital repository literature with the aim of supporting academic research, which will run as a service accessed over the Internet.

**Keywords:** Cloud computing, software as a service, XP, BDD, TDD.

**LISTA DE SIGLAS**

BDD *Behavior Driven Development*

C3 *Chrysler Comprehensive Compensation*

CPU *Central Processing Unit*

IaaS *Infrastructure as a Service*

PaaS *Platform as a Service*

RUP *Rational Unified Process*

SaaS *Software as a Service*

TDD *Test-Driven Development*

TI Tecnologia da Informação

UML *Unified Modelling Language*

XP *Extreme Programming*

**LISTA DE FIGURAS**

[Figura 1 - Camadas de arquitetura da computação em nuvem. 16](#_Toc447231128)

[Figura 2 - Lista dos mais importantes provedores em cada área IaaS 17](#_Toc447231129)

[Figura 3 - Lista dos mais importantes provedores em cada área PaaS. 18](#_Toc447231130)

[Figura 4 - Lista dos mais importantes provedores em cada área SaaS. 18](#_Toc447231131)

[Figura 5 - Ciclo de vida do Modelo Cascata 25](#_Toc447231132)

[Figura 6 - Ciclo de vida do RUP 26](#_Toc447231133)

[Figura 7- Conversa expressa em RSpec 34](#_Toc447231134)

[Figura 8 - Estrutura básica de um projeto em Cucumber. 36](#_Toc447231135)

[Figura 9 - Curso Engenharia de Software como Serviço. 40](#_Toc447231136)

[Figura 10 - Extreme Cloud Programming. 44](#_Toc447231137)

[Figura 11 – Cronograma de atividades do TCC 47](#_Toc447231138)

**LISTA DE TABELAS**

[Tabela 1 - Vantagens do uso do SaaS para provedores 21](#_Toc447231139)

[Tabela 2 - Os Benefícios do SaaS para o cliente 22](#_Toc447231140)

[Tabela 3 - Etapas de desenvolvimento do RUP 26](#_Toc447231141)

[Tabela 4 - Valores dos métodos ágeis e dos métodos tradicionais 27](#_Toc447231142)

[Tabela 5 - Valores do XP 29](#_Toc447231143)

[Tabela 6 - Algumas práticas do XP 30](#_Toc447231144)

[Tabela 7 - Programação do curso Engenharia de software como serviço 39](#_Toc447231145)

[Tabela 8 - Tipos de bibliotecas 42](#_Toc447231146)

SUMÁRIO

[1 **INTRODUÇÃO** 12](#_Toc446341466)

[1.1 MOTIVAÇÃO 12](#_Toc446341467)

[1.2 OBJETIVOS 13](#_Toc446341468)

[1.2.1 **Objetivo Geral** 13](#_Toc446341469)

[1.2.2 **Objetivos Específicos** 13](#_Toc446341470)

[1.3 ORGANIZAÇÃO DO DOCUMENTO 13](#_Toc446341471)

[2 **FUNDAMENTOS TEÓRICOS** 14](#_Toc446341472)

[2.1 COMPUTAÇÃO EM NUVEM 14](#_Toc446341473)

[2.1.1 **Modelos de serviços em nuvem** 16](#_Toc446341474)

[2.1.2 **Modelos de implantação em nuvem** 23](#_Toc446341475)

[2.2 Engenharia de software 23](#_Toc446341476)

[2.2.1 **Métodos Tradicionais** 24](#_Toc446341477)

[2.2.2 **Métodos Ágeis** 27](#_Toc446341478)

[2.2.3 ***Extreme Programming* (XP)** 29](#_Toc446341479)

[2.2.4 ***Test-Driven Development* (TDD)** 32](#_Toc446341480)

[2.2.5 ***Behavior-Driven Development* (BDD)** 34](#_Toc446341481)

[2.3 RUBY 37](#_Toc446341482)

[3 **TRABALHOS CORRELATOS** 39](#_Toc446341483)

[3.1 ENGINEERING SOFTWARE AS A SERVICE 39](#_Toc446341484)

[3.2 BIBLIOTECAS NA NUVEM: O USO DA COMPUTAÇÃO EM NUVEM EM BILIOTECAS 41](#_Toc446341485)

[3.3 IMPACT OF WEB 2.0 AND CLOUD COMPUTING PLATFORM ON SOFTWARE ENGINEERING 43](#_Toc446341486)

[4 **PROPOSTA** 46](#_Toc446341487)

[5 **CRONOGRAMA PARA O DESENVOLVIMENTO DO TCC I E II** 47](#_Toc446341488)

[5.1 CRONOGRAMA 47](#_Toc446341489)

[REFERÊNCIAS 48](#_Toc446341490)

# INTRODUÇÃO

## MOTIVAÇÃO

Atualmente a computação em nuvem vem se tornando cada vez mais importante, não só pela capacidade de armazenamento, mas também pelo desenvolvimento de software. Durante muito tempo o desenvolvimento de software foi voltado para aplicações *desktops,* onde a aplicação era instalada e executada localmente no computador do cliente, ou seja, o acesso era limitado a onde o aplicativo estava instalado. A difusão da Internet fez com que o novo paradigma de aplicações pudesse ser acessado via navegador, independente de onde estivesse instalado, paradigma conhecido como Software como Serviço, ou SaaS (*Software as a Service*) (DOURADO, 2009).

O SaaS concede o software e os dados como um serviço na Internet, normalmente por meio de um programa leve, como um navegador que é executado em dispositivos do cliente, ao invés de um aplicativo que deve ser instalado e executado inteiramente no dispositivo, com isso o cliente não precisa se preocupar se o software é da marca certa ou rápido o suficiente, nem se possui a versão correta do aplicativo. E o desenvolvedor, que possui uma única cópia do software pode atualizá-lo frequentemente (FOX; PATTERSON, 2015).

No mesmo rumo, o desenvolvimento de software tem evoluído para novos paradigmas, como os métodos ágeis, que estão melhor preparados para o desenvolvimento de programas onde os requisitos costumam mudar com frequência, eliminam gastos com documentação excessiva e burocrática, dando ênfase à comunicação, colaboração com o cliente, etc. (SATO, 2007). A abordagem aqui utilizada será o XP (*Extreme Programming*), pois, segundo Fox e Patterson (2015), ele é o método que inclui iterações de uma a duas semanas, projeto guiado por comportamento, desenvolvimento guiado por testes e programação em pares.

O que impulsionou a realização deste trabalho foi entender como a Engenharia de Software está aplicando seus mais recentes métodos, técnicas e ferramentas para o desenvolvimento de software como serviço. Aplicaremos os métodos ágeis XP, juntamente com os métodos para teste BDD (*Behavior-Driven Development*), criada por Dan North em 2003 e TDD (*Test-Driven Development*) para implementar na linguagem Ruby um repositório bibliográfico digital que funcionará como um serviço. O repositório visa agregar as diversas bibliografias utilizadas por pesquisadores de forma organizada e centralizada.

## OBJETIVOS

### Objetivo Geral

Aplicar e analisar os métodos XP, BDD e TDD para o desenvolvimento de um repositório bibliográfica digital como serviço.

### Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos devem ser executados:

1. Pesquisar o estado da arte sobre os métodos ágeis, com foco em XP e nos métodos para teste, BDD e TDD;
2. Pesquisar o estado da arte sobre computação em nuvem, com foco em software como serviço;
3. Usar as ferramentas Cucumber e RSpec para auxílio nos testes;
4. Desenvolver o repositório bibliográfico digital para apoio às pesquisas na plataforma Cloud9;
5. Levantar e analisar os resultados obtidos.

## ORGANIZAÇÃO DO DOCUMENTO

No primeiro capítulo foram apresentados a motivação e os objetivos para o desenvolvimento do trabalho. O segundo capítulo traz a fundamentação teórica para a contextualização do leitor. O terceiro capítulo apresenta os trabalhos desenvolvidos por outros pesquisadores que se relacionam a esta pesquisa. O quarto capítulo expõe a proposta de desenvolvimento do trabalho e ideias que serão relevantes para o emprego dos métodos XP, BDD e TDD na construção do protótipo que funcionará como serviço. O quinto capítulo expõe a proposta de desenvolvimento de trabalho. E, por último, o capítulo seis possui o cronograma de atividades.

A seguir é apresentado o capítulo com referencial teórico.

# FUNDAMENTOS TEÓRICOS

Neste capítulo abordaremos sobre a fundamentação teórica deste trabalho que inclui os tópicos de computação em nuvem, seus modelos de serviço e implantação; sobre a engenharia de software e suas metodologias, tradicional e ágil, especificando o método XP e os métodos para teste TDD e BDD.

## COMPUTAÇÃO EM NUVEM

Cogo (2013) relata que a primeira definição associada à computação em nuvem é a de computação como utilidade, ou seja, ela passou ser cobrada como um serviço, assim como água e luz o são, e não mais como um produto comprado em sua totalidade. O usuário passaria a pagar somente aquilo que utilizasse.

Apesar do termo computação em nuvem ou *Cloud Computing* ser atual, o conceito já existe desde a década de 60, onde o cientista norte-americano John McCarthy definiu que, se os computadores, da forma como ele pensava, se tornassem os computadores do futuro, então a computação poderia ser organizada como um serviço público, assim como é a telefonia. E cada assinante pagaria apenas pelos recursos que ele realmente utilizasse e teria acesso a todos os recursos oferecidos pelas linguagens de programação de um grande sistema. Alguns assinantes conseguiriam oferecer serviços a outros assinantes. A computação com um serviço público poderia ser base de uma nova e importante indústria (CARISSIMI, 2015).

Mesmo com a popularidade da ideia transmitida, o conceito não teve a oportunidade de prosperar, porque as tecnologias facilitadoras necessárias eram incapazes de sustentar tal modelo de computação. Então, no final de 1990 o desenvolvimento de tecnologias de informática atingiu os níveis adequados de comercialização, dando o renascimento para a ideia de McCarthy na forma de provedores de serviços de aplicativos e computação em nuvem (CHURAKOVA; MIKHRAMOVA, 2010). A partir daí ela começa a revitalizar e emergir na área da tecnologia.

Computação em nuvem se refere tanto a aplicações entregues como serviço através da Internet, como ao hardware e aos sistemas de software nos centros de dadosque fornecem esses serviços (ARMBRUST et al, 2010). Ela revolucionou a indústria de TI (Tecnologia da Informação) ao oferecer como um serviço grandes quantidades de recursos computacionais (ALKMIN, CORDEIRO, 2014).

Em termos de pesquisa na área de ciência da computação a nuvem não é classificada como um novo paradigma. O que houve, na verdade, foi a evolução e a convergência de pesquisas realizadas nas diversas áreas da computação, como a computação utilitária, *clusters* e *grids*, etc, tornando-a no que é hoje. O sucesso que obteve em aplicações comerciais de grande escala são indicadores do seu potencial nas aplicações científicas de alto desempenho (ALKMIN, CORDEIRO, 2014).

Um dos grandes motivos do avanço da computação em nuvem foi a virtualização. Em computação, virtualização é um termo genérico, que se refere à abstração dos recursos do computador, mascarando as características físicas deles de forma que outros sistemas, aplicações ou usuários finais possam interagir com tais recursos (MARTINS, 2010). Ela melhora a utilização de recursos e eficiência energética - ajudando a reduzir substancialmente a sobrecarga de manutenção do servidor e proporcionando rápida recuperação de desastres e alta disponibilidade (GORELIK, 2013).

Além dos motivos anteriormente citados, a virtualização tem sido muito importante para a computação em nuvem porque isola software de hardware e assim fornece um mecanismo para redistribuir rapidamente aplicações em servidores com base nas demandas computacionais (GORELIK, 2013).

O funcionamento da computação em nuvem segue um modelo de negócio onde o usuário paga apenas pelo que consome de recursos e o provedor do serviço mantém uma infraestrutura física, amortiza os custos de manutenção e investimentos compartilhando e alugando seus recursos a diferentes usuários (CARISSIMI, 2015). Provedores como a *Amazon* e a *Microsoft* permitem que as empresas evitem grandes investimentos em infraestrutura e adquiram novos recursos de computação de forma dinâmica, conforme o necessário. Este modelo é particularmente viável para pequenas empresas e *star-ups[[1]](#footnote-1)*, que muitas vezes não tem condições de gastar grandes somas de dinheiro no início de sua jornada de negócios (GORELIK, 2013).

Na prática, ela é um modelo de negócio baseado no conceito de elasticidade. A elasticidade é a capacidade de alocar recursos conforme a demanda. Se durante a execução de uma aplicação for atingido um pico de utilização, provocando gargalos ou a degradação de desempenho, o cliente pode solicitar mais máquinas virtuais para auxiliar o processamento durante esse pico de demanda. Após a normalização da situação, essas máquinas virtuais podem ser liberadas (CARISSIMI, 2015). Dessa maneira, cada cliente paga apenas pela utilização efetiva dos recursos.

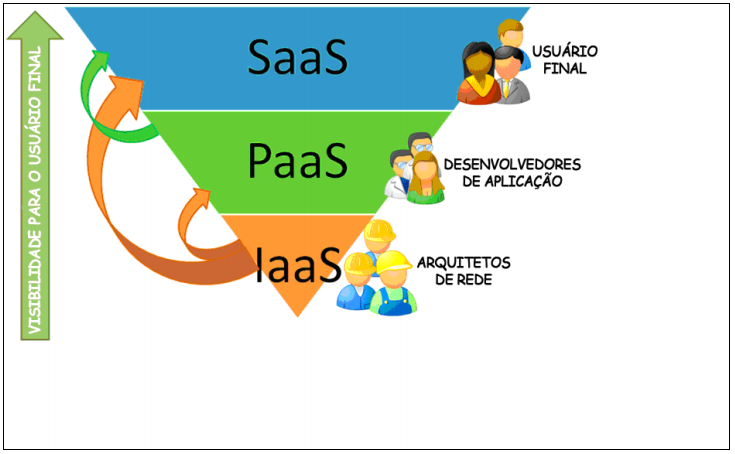
Porém existem barreiras para a adoção da mesma. Segundo Gorelik (2013) a segurança dos dados é uma das barreiras mais desafiadora porque os dados são os bem mais precioso de uma empresa, portanto elas se sentem mais seguras quando eles estão armazenados em seus próprios *datacenters,* de forma a ter total controle sobre eles. Embora não haja garantia de que os dados são melhor protegidos internamente comparando com a nuvem pública. Ainda segundo o autor, existe a possibilidade dos dados estarem melhor protegidos na nuvem justamente porque os provedores de nuvem investem mais em segurança de dados que o próprio cliente.

### Modelos de serviços em nuvem

Ao falar em computação em nuvem podemos pensar em camadas, ao invés de diferentes funcionalidades. Esse pensamento dá um caráter mais arquitetural e de integração entre os níveis, como mostra a Figura 1. Essas camadas são:

1. **Infraestrutura como serviço (IaaS);**
2. **Plataforma com serviço (PaaS);**
3. **Software como serviço (SaaS).**

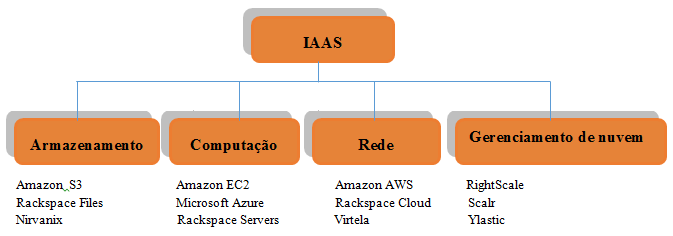
Figura 1 - Camadas de arquitetura da computação em nuvem.



Fonte: MERIAT (2015).

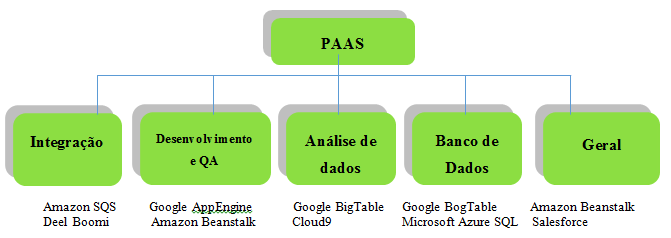
No modelo IaaS, a capacidade fornecida ao consumidor consiste na aquisição de processamento, armazenamento, redes e outros recursos computacionais fundamentais em que o consumidor é capaz de implantar e executar software arbitrário, incluindo sistemas operacionais e aplicativos. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, mas tem controle sobre os sistemas operacionais, armazenamento e aplicativos implantados. Além disso, o consumidor pode ser capaz de selecionar os componentes de rede, por exemplo, hospedar firewalls e balanceadores de carga[[2]](#footnote-2) (JAKONEN, 2011).

Figura 2 - Lista dos mais importantes provedores em cada área IaaS

Fonte: Adaptado de GORELIK (2013).

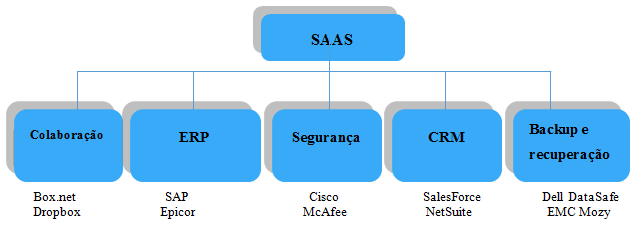
No modelo PaaS, o consumidor tem a capacidade de implantar para a infraestrutura da nuvem aplicações criadas para o cliente usando linguagens e ferramentas (por exemplo, .NET, Java, Python) suportadas pelo fornecedor. O consumidor não gerencia ou controla a infraestrutura subjacente da nuvem, rede, servidores, sistemas operacionais, ou armazenamento. No entanto o consumidor tem o controle sobre as aplicações implantadas e possivelmente ao ambiente de configuração onde a aplicação está hospedada (JAKONEN, 2011).

Figura 3 - Lista dos mais importantes provedores em cada área PaaS.

 Fonte: Adaptado de Gorelik (2013).

Gorelik (2013) define software como serviço como um modelo de prestação de serviços em nuvem que oferece uma assinatura de software on-line e sob demanda. Tal como acontece com outros modelos de prestação de serviços em nuvem, o SaaS oferece às empresas a oportunidade de reduzir os custos internos de suporte de TI e transferir a responsabilidade de manutenção para o provedor de SaaS.

Figura 4 - Lista dos mais importantes provedores em cada área SaaS.

Fonte: Adaptado de Gorelik (2013).

Na produção tradicional de software o lançamento de novas versões provoca necessariamente um impacto negativo nos parceiros de negócio, nos responsáveis de TI e muitas vezes nos próprios clientes. Isso não ocorre no SaaS, pois sua essência resolve o problema e liberta todo trabalho de distribuição e instalação dos produtos localmente nos clientes. O SaaS está continuamente em desenvolvimento, numa lógica chamada de “eternos Beta”, sem nunca precisar chegar em uma versão terminada para entrar em produção (PINHO, 2009).

Pinho (2009) explica que o SaaS é um modelo baseado em aplicações *Multi-tenant*, que permitem que vários usuários usem o mesmo aplicativo ao mesmo tempo. Porém, existem variantes nos quais o software fica instalado na maquina do cliente e a informação é sincronizada entre o cliente e os serviços alojados remotamente.

A hospedagem *multi-tenant* e a automação de serviços permite que os fornecedores de SaaS mantenham os custos baixos. Dependendo do provedor de SaaS e do tipo de aplicação, algumas aplicações personalizadas são disponíveis frequentemente, mas esta é tipicamente mais limitada do que com as soluções de computação em nuvem, PaaS e IaaS (GORELIK, 2013).

Na computação em nuvem, a ideia fundamental é reutilizar diferentes tipos de serviços baseados na Internet. Em SaaS, o software em si é alvo de reutilização e é implantado para os consumidores de serviços através da Internet. Normalmente, relacionamentos um-para-muitos são usados em serviços de entrega em SaaS. Por exemplo, o Google Maps é um serviço que fornece um conjunto de operações que utiliza informações compartilhadas em um mapa, e o serviço pode ser usado por vários clientes (JAKONEN, 2011).

Segundo Cheun (2009) o SaaS apresenta seis recursos principais:

* **Reusabilidade**: em SaaS, o software é objeto de reutilização e é implantado para os consumidores do serviço através da Internet;
* **Dados gerenciados pelo provedor**: o provedor de serviço é responsável pela instalação e gestão de dados do serviço no seu próprio servidor, sendo essencial que o fornecedor de serviços ofereça segurança de dados e função de confiabilidade para que os clientes do serviço não desconfiem do mesmo e, assim levaria a uma redução da utilização dos serviços;
* **Serviço personalizado***:* significa que os consumidores do serviço podem altera-lo com base em suas necessidades individuais. Esta é uma característica que permite que um prestador de serviços atenda as diferentes necessidades de cada cliente. É impossível para os prestadores personaliza-los para satisfazer as diferentes necessidades de cada cliente, porque na nuvem, vários consumidores com acesso à Internet podem se tornar potenciais utilizadores de serviços de nuvem. Por isso, eles devem ser personalizados pelos consumidores para seus próprios propósitos.
* **Disponibilidade***:* os clientes são capazes de acessar serviços de SaaS a partir de um navegador web através da Internet, não tendo nenhuma posse sobre o software que está implementado e em execução no servidor do fornecedor de serviços. Consequentemente, muitos fornecedores de SaaS centram sua atenção em alcançar a maior disponibilidade possível dos serviços. Se um SaaS não estiver disponível, os clientes não podem utilizar as funcionalidades do SaaS. Como exemplo, temos o serviço de e-mail do Google, Gmail, estava completamente para baixo em 01 de setembro de 2009. Por causa disso, muitos usuários do Gmail não foram capazes de utilizar o serviço, ou seja, ler ou escrever e-mail.
* **Escalabilidade***:* em engenharia de software recursos de escalabilidade habilitam o software a manipular uma porção crescente de trabalho ou de ser estendido facilmente. Devido à natureza caixa-preta dos serviços de *Cloud Computing*, clientes do serviço não podem controlar os recursos, como memória, rede, ou a utilização da CPU. Em outras palavras, um prestador de serviços é responsável por redimensionando os recursos com base em pedidos do cliente, sem notificá-lo dos detalhes.
* **Custo***:* As despesas para SaaS são estimados com base na utilização de serviços, tais como o número de chamadas de serviço ou a duração da utilização do serviço. Isso é diferente de modelo de negócio tradicional, em que as despesas são estimadas com base na propriedade. Os consumidores de serviço podem se conectar e utilizar o serviço, tanto quanto eles quiserem, e, em seguida, pagar apenas a quantidade de uso – assim com se utiliza eletricidade ou água.

Segundo Churakova e Mikhramova (2010), uma das questões importantes a considerar quanto à possibilidade de oferecer a aplicação sob a forma de um serviço em vez do software tradicional local é que é necessário um tipo de infraestrutura, a fim de entregar com sucesso o software para os clientes. Para tanto, três considerações fundamentais têm de ser levadas em conta pelo fornecedor do software durante o processo de configuração da infraestrutura:

* Visto o fato de que a entrega do serviço é realizado através de uma rede, a primeira atividade principal é a prestação de conectividade contínua entre o prestador, o software e o cliente.
* Em segundo lugar, o provedor de software deve garantir a continuidade dos negócios através da criação de infraestrutura de rede, servidor e armazenamento necessário.
* Finalmente, o fornecedor SaaS é responsável pela garantia da segurança durante o armazenamento de dados e prestação de serviços, o que implica que medidas como backup, recuperação de desastres e políticas de autenticação apropriados são realizadas.

As características tecnológicas do SaaS são: aumento do uso de conexão de rede, aumento do uso de navegadores como interface e aumento do uso de processamento do lado do servidor. O modo de entrega no SaaS é bastante diferenciado dos modelos anteriores de negócio de software e o usuário final tem acesso local independente das capacidades do software, enquanto que os fornecedores têm total controle sobre os aplicativos que residem em seu *datacenter* (LUOMA, 2013).

Na tabela 1 a seguir estão reunidas as principais vantagens que o SaaS proporciona para os provedores.

Tabela 1 - Vantagens do uso do SaaS para provedores

|  |  |
| --- | --- |
|  | Vantagens do uso do SaaS para Provedores |
| 1 | SaaS reduz os custos para os provedores, uma vez que é o único a lidar com as necessidades de todos os clientes a partir de sua localização, incluindo atualização e personalização de sistemas. |
| 2 | SaaS é completamente gerenciado por provedores; portanto, eles podem atualizar e corrigir o sistema sem visitar pessoalmente os sites dos clientes. |
| 3 | SaaS ajuda os provedores a agilizar a atualização do software para melhorar a sua competitividade no mercado. Devido ao fato do SaaS ser oferecido aos clientes através de uma plataforma baseada na Web, os provedores podem atualizar constantemente uma nova versão. Portanto, eles podem oferecer uma nova versão para os clientes mais rápido que o software tradicional, que precisam enviar uma nova versão em um disco de aplicação a todos os clientes para atualizações. |

Fonte: RUCHUTRAKOOL; ZHU (2009).

Já na tabela 2 estão reunidos os principais benefícios que ele trás para o cliente:

Tabela 2 - Os Benefícios do SaaS para o cliente

|  |  |
| --- | --- |
| Benefícios do SaaS para o cliente | |
| 1 | SaaS auxilia o cliente a economizar nos custos iniciais típicos, que inclui servidor, rede, licença, software de banco de dados e os custos de sistemas operacionais porque os prestadores do serviço são responsáveis por pagar os custos ao invés disso. |
| 2 | SaaS reduz os riscos decorrentes da aquisição de software. Como o fato de que os clientes não precisam implantar uma grande infraestrutura de TI local, portanto, quando estão desapontados com os resultados do serviço, eles têm a opção de abandonar este serviço e buscar um novo serviço sem ter de se preocupar com a dispendiosa infraestrutura de TI. |
| 3 | Clientes de SaaS podem facilmente expandir o número de usuários, porque SaaS é entregue em uma arquitetura multi-tenant. Essa arquitetura entrega mais escalabilidade para o cliente, ou seja, o acesso de clientes pode ser dimensionado para o numero de usuários de imediato, porque seus processos não afetam outros processos quando expande. |
| 4 | Clientes de SaaS adquirem facilmente recursos personalizados. Uma das vantagens do software orientado a serviço é a facilidade de ajuste. |
| 5 | SaaS ajuda os clientes a se concentrarem mais em suas competências principais de negócio, porque eles não tem preocupação com qualquer infraestrutura de TI, ela é de responsabilidade dos prestadores de serviços de SaaS. |
| 6 | SaaS ajuda os clientes a se concentrarem mais em suas competências principais de negócio, porque eles não tem preocupação com qualquer infraestrutura de TI, ela é de responsabilidade dos prestadores de serviços de SaaS. |
| 7 | SaaS auxilia os clientes para acessar os dados do aplicativo e de atualização, independentemente de onde estiverem. Eles só precisam de Internet e um navegador, a fim de fazer *logon* e usar o aplicativo. Além disso, esses recursos permitem que equipes trabalhem a partir de suas localizações, como de suas casas. Portanto, não precisa ir a sede a fim de acessar o sistema. Isso permite que trabalhe em qualquer lugar e em qualquer hora. |

Fonte: RUCHUTRAKOOL; ZHU (2009).

### Modelos de implantação em nuvem

Além dos modelos de serviço existem os modelos de implantação ou estruturais de computação em nuvem que fornecem a mesma para os usuários. São eles (COGO, 2013):

1. **Publico:** As aplicações, armazenamento e outros recursos são disponibilizados para o público pelo fornecedor. Os serviços, pagos ou gratuitos são iguais para todos os usuários, todos tem uma instância dele através da Internet. As características é que o usuário não tem domínio sobre os dados, nem sobre a possível personalização. As empresas que oferecem esse serviço geralmente possuem grandes capacidades de armazenamento e processamento.
2. **Privado:** Desenvolvido exclusivamente para uma organização, o modelo privado pode ser gerido pela própria empresa ou terceiros, internamente ou não. Os serviços são usados internamente pela empresa
3. **Híbrido:** É a composição entre nuvens públicas e privadas, agrupadas para oferecerem os benefícios de ambos os modelos. A característica desse modelo é que na híbrida a infraestrutura do serviço roda na nuvem privada, e a outra parte na pública, dando mais segurança e controle ao cliente.

Alguns autores especificam mais modelos, porém para o entendimento deste trabalho já é suficiente.

## Engenharia de software

Desenvolver software não é uma atividade trivial, pois o mesmo se trata de algo abstrato e intangível. Existem muito problemas que podem ocorrer no processo de criação do software, principalmente no que se refere ao desenvolvimento profissional, como alto custo, alta complexidade, desproporção entre o que o cliente desejava e o produto entregue, etc.

Diante desses problemas e acreditando-se que o processo utilizado era uma das causas, a Engenharia de Software surgiu com o objetivo de enfatizar o quanto era preciso que o desenvolvimento de software fosse baseado nos princípios teóricos e nas disciplinas práticas tradicionalmente conhecidos dos diversos ramos da engenharia (SATO, 2007).

Nas seções a seguir serão apresentados como a engenharia de software evoluiu ao longo dos anos com os métodos tradicionais e os mais recentes, métodos ágeis.

### Métodos Tradicionais

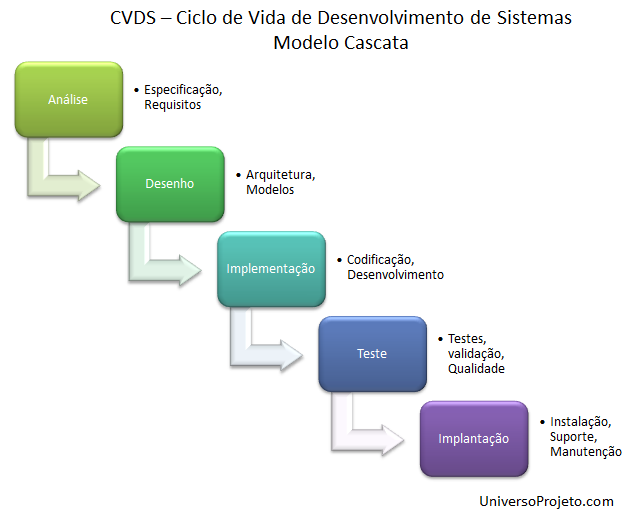
Os métodos tradicionais de desenvolvimento têm como característica a precedência da especificação de todos os requisitos antes de qualquer coisa. Eles surgiram da necessidade de se organizar o processo de desenvolvimento visto que os programadores adotavam as próprias técnicas e abordagens. Cada produto era feito sob medida para um propósito particular. Dessa forma a manutenção, confiabilidade e a produtividade se tornavam um grande desafio de garantir. Surgiu, então, um novo paradigma centrado no produto e na combinação de componentes padronizados, seguindo uma linha de produção industrial (UCHÔA, 2013).

Uchôa (2013) explica que esses modelos tradicionais, orientados a planos e a processos, foram desenvolvidos com o proposito de criar projetos grandes, caros, de complexidade elevada e de alto risco. Nessas situações, havia grande dependência de contratos e rigorosos procedimentos de controle, pois as condições de execução não favoreciam a confiança entre clientes e desenvolvedores.

O processo de desenvolvimento em cascata foi um dos primeiros e surgiu na década de 70, Winston Royce é comumente citado como o primeiro autor a descrevê-lo. Era composto por fases distintas, do levantamento de requisitos até a implantação, ou seja, todas elas eram encadeadas. Na década seguinte surge o modelo em espiral que serviu como base para os atuais processos iterativos e incrementais como o *Rational Unified Process*, o RUP, e também os Métodos Ágeis (SATO, 2007).

A figura 5 mostra os estágios do modelo em cascata e fica evidente o porquê dele ser chamado dessa maneira.

Figura 5 - Ciclo de vida do Modelo Cascata



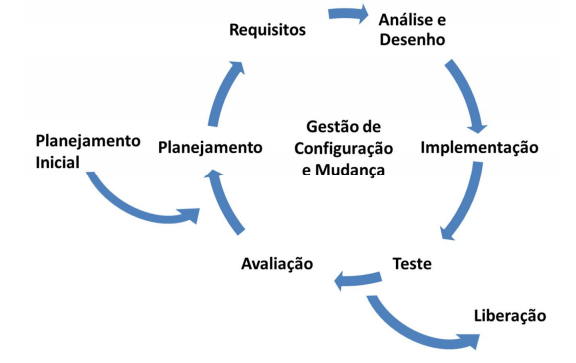
Fonte: Universo Projeto (2016).

O RUP foi desenvolvido com o intuito de complementar a UML (do inglês *Unified Modelling Language*), método para modelagem de software. Adota uma abordagem iterativa no desenvolvimento de sistemas. As práticas enfatizadas pelo RUP são (UCHOA, 2013):

1. Desenvolver de forma iterativa;
2. Gerenciar os requisitos;
3. Usar arquiteturas baseadas em componentes;
4. Representar visualmente;
5. Verificar continuamente a qualidade;
6. Controlar alterações.

O método do RUP é um ciclo iterativo de seis etapas, conforme é apresentado na figura abaixo:

Figura 6 - Ciclo de vida do RUP



Fonte: Adaptado de Uchôa (2013).

O método também possui etapas, que são especificadas na tabela abaixo (FOX; PATTERSON, 2015):

Tabela 3 - Etapas de desenvolvimento do RUP

|  |  |
| --- | --- |
| Etapas de Desenvolvimento do RUP | |
| 1. Iniciação | Etapa onde é definido o negócio do software, delimitado o escopo para definir o cronograma e orçamento. |
| 1. Elaboração | Nessa etapa ocorre o trabalho com as partes interessadas para identificar os casos de uso, projetar uma arquitetura de software, estabelecer o plano de desenvolvimento e construir um protótipo inicial. |
| 1. Construção | Etapa da codificação e testes do produto. |
| 1. Transição | É onde o produto passa do desenvolvimento para o ambiente real, incluindo teste de aceitação e treinamento do usuário. |

Fonte: (FOX; PATTERSON, 2015)

Uchôa (2013) esclarece que o RUP pode ser usado tanto no desenvolvimento tradicional quanto em projetos ágeis. Porém, ele não é considerado ágil, pois foi originalmente criado para cobrir processos inteiros de desenvolvimento de software, com orientações extensas que não são típicas de um ambiente ágil.

### Métodos Ágeis

Em Somerville (2011) vemos como nos dias de hoje as empresas operam em um ambiente global, onde as mudanças ocorrem de forma rápida. Em sistemas de desenvolvimento de software que pretendem especificar completamente todos os requisitos, projetando, construindo e testando, em seguida, não estão adaptados para o desenvolvimento rápido de software. Esses processos convencionais, como cascata ou baseado em especificações, não estão preparados para mudanças que podem acontecer, como nos requisitos, no projeto, ou implementação, caso o projeto precise ser refeito ou retestado. Isso torna o processo mais demorado, o que gera atrasos na entrega do software.

De acordo com Sommerville (2011) os métodos ágeis surgiram na década de 90 como uma proposta dos desenvolvedores insatisfeitos com as abordagens tradicionais e pesadas da engenharia de software. Eles se reuniram e formaram uma aliança, a Aliança Ágil, e lançaram em fevereiro de 2001 o **manifesto ágil**, onde propunham um ciclo de vida de software mais leve (FOX; PATTERSON, 2015).

Na tabela seguir é apresentado os quatro valores definidos no manifesto ágil, onde os itens à esquerda agregam mais valor que os itens à direita, que representam as metodologias tradicionais:

Tabela 4 - Valores dos métodos ágeis e dos métodos tradicionais

|  |  |
| --- | --- |
| Valores: Métodos Ágeis *vs* Métodos Tradicionais | |
| Métodos Ágeis | **Métodos Tradicionais** |
| Indivíduos e interações | Processos e ferramentas |
| Software em funcionamento | Documentação abrangente |
| Colaboração com o cliente | Negociação de contratos |
| Responder a mudanças mais | Seguir um plano |

Fonte: Adaptado de Beck et al (2001).

Para que entendesse o foco dos métodos ágeis foram disponibilizados 12 princípios onde detalham a filosofia contida no manifesto e onde os métodos ágeis devem se adequar (BECK et al, 2001):

1. A maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software;
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente, dando aos clientes vantagens competitivas;
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
4. Equipes de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto;
5. Construir projetos em torno de indivíduos motivados, dando o ambiente, suporte necessário, confiando neles para fazer o trabalho;
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face;
7. Software funcionando é a medida primária de progresso;
8. Os processos ágeis promovem desenvolvimento sustentável. Todos devem manter um ritmo de desenvolvimento constante;
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade;
10. Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial;
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis;
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Fox e Patterson (2015) explicam que métodos ágeis são modelos de desenvolvimento que incentivam a prática de abraçar as mudanças como um fato da vida, em que os desenvolvedores devem aperfeiçoar continuamente um protótipo incompleto, mas em funcionamento, com o *feedeback* do cliente em cada iteração, até que ele esteja feliz com o resultado.

Segundo Sommerville (2011), métodos ágeis são bem-sucedidos em duas situações: no desenvolvimento de um produto pequeno ou médio e no desenvolvimento de um sistema personalizado, em que o cliente participa ativamente, sem muitas regras e regulamentos que afetem o software. No entanto, enfatiza que os princípios básicos dessa metodologia são, muitas vezes, difíceis de concretizar para grandes sistemas, principalmente em sistemas críticos, onde há a necessidade de proteção, segurança e análise de confiança, sendo que seu uso exigiria grandes modificações nos métodos ágeis.

Sob os valores e princípios do manifesto ágil aparecem diversas abordagens mais específicas, com diferentes ideias, comunidades e líderes, onde formam um grupo distinto, porém seguindo os mesmos princípios (SATO, 2007).

Dentre os métodos ágeis podemos citar XP, *Scrum*, Família *Crystal, Lean Software Development, Feature Driven Development* (FDD), *Dynamic System Development Method* (DSDM), entre outras.

### *Extreme Programming* (XP)

O *Extreme Programming* (XP) foi um dos métodos ágeis que mais recebeu atenção na virada do século. Seu objetivo é a excelência no desenvolvimento de software, visando o baixo custo, poucos defeitos, alta produtividade e alto retorno de investimento (SATO, 2007).

Kent Beck foi o criador do XP e um dos que lançaram o manifesto ágil. Beck foi chamado para ajudar no projeto C3, *ou Chrysler Comprehensive Compensation System*, para a empresa *DaimlerCrysler*, que após anos de fracassos ficou pronto em um ano, usando as práticas que vieram a se tornar o XP (FAGUNDES, 2005).

Em sua segunda edição o XP segue um conjunto de valores, princípios e práticas para serem adotados durante o processo de desenvolvimento. As práticas são as técnicas utilizadas no dia-a-dia por membros da equipe de XP, que só fazem sentido no contexto da programação; os valores são critérios mais amplos e universais usados para julgar uma situação, eles dão razão às práticas, enquanto que as práticas colocam em evidência os valores; os princípios funcionam como ferramentas para tradução dos valores em práticas (SATO, 2007).

Os valores do XP expresso por Beck e Andres (2004) são explicitados na tabela abaixo:

Tabela 5 - Valores do XP

|  |  |
| --- | --- |
| Valores do XP | |
| 1. Comunicação | Tem como objetivo a comunicação rápida e ao mesmo tempo eficaz entre os membros da equipe e o cliente. |
| 1. Simplicidade | Esse objetivo expressa que o software precisa ser continuamente simplificado. |
| 1. *Feedback* | Todo problema deve ser evidenciado o mais rápido possível tanto pela equipe quanto pelo cliente para que possa ser corrigido. |
| 1. Coragem | A coragem torna-se extremamente valiosa quando usa a combinação dos três valores anteriores. Precisa-se de coragem para pedir ajuda, |

Fonte: BECK; ANDRES (2004).

Em relação às práticas, Beck e Andres (2004) explica que elas não se mantêm por si só, o que faz elas funcionarem é a cooperação do seu conjunto. Algumas práticas são especificadas na tabela a seguir:

Tabela 6 - Algumas práticas do XP

|  |  |
| --- | --- |
| Prática | Definição |
| Jogo do planejamento | Consiste na decisão do que deve ser feito e no que pode ser adiado, já que o XP baseia-se em requisitos atuais e não em requisitos futuros. |
| Entregas frequentes | As entregas são feitas ao término de iteração, devendo ser tão pequena quanto possível. O foco está no monitoramento do progresso do software tanto pelos clientes quanto pelos programadores. |
| Histórias | O planejamento no XP é feito com histórias escritas em pequenos cartões, sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento do software. |
| Projeto simples | Manter o projeto mais simples possível o faz permanecer ágil e flexível, mas para isso é preciso uma revisão contínua. |
| Testes | No XP os programadores escrevem testes de unidade e os clientes testes de aceitação. No primeiro, os testes são escritos antes da codificação e só após são codificados. No segundo, os testes são escritos para conferir se uma determinada história está funcionando corretamente. |
| Refatoração | É a prática onde é possível simplificar o módulo atual sem perder nenhuma funcionalidade. Isso mantém o código simples e manutenível. |
| Programação em pares | A programação é feita em dupla, que trabalham em um único computador. |
| Propriedade coletiva | O código do projeto pertence a todos os membros da equipe, ou seja, todos assumem responsabilidade por todo o código. |
| Integração contínua | Quando um trabalho é concluído ele é integrado ao sistema como um todo. Essa é uma prática que funciona bem porque fica óbvio quem deve fazer as correções quando os testes falharem. Essa prática é facilitada com o uso de apenas uma máquina de integração. |
| Semana de 40 horas | O número de horas dedicado ao projeto deve ser realista, não sendo recomendo fazer horas extras por períodos maiores que uma semana, pois o cansaço do excesso de trabalho pode ocasionar no desenvolvimento de código com erros que darão trabalho mais tarde. |
| Cliente presente | Essa prática faz com que as pessoas que serão afetadas pelo sistema façam parte da equipe. Os clientes devem entender as necessidades do negócio e conhecer os verdadeiros usuários do sistema para poder escrever histórias, definir prioridades, etc. |
| Código compartilhado | O repositório do código fonte é compartilhado por toda a equipe. Qualquer um que observe uma oportunidade de agregar valor a alguma parte do código deve fazê-lo. O time todo se torna responsável pelo sistema inteiro. Dessa forma, adquire-se uma visão ampla do sistema e isso facilita a execução de refatorações. |

Adaptado de FAGUNDES (2005) e SATO (2009).

O XP faz uma seleção de práticas antigas levando os princípios óbvios do senso comum para níveis extremos, por exemplo, se iterações curtas são boas, então, faça-as mais curtas possíveis, prefira minutos, horas a dias, meses ou anos. Essas são as principais características da metodologia XP.

### *Test-Driven Development* (TDD)

Desenvolver software é um processo complexo com tarefas e requisitos que tendem a mudar constantemente. Dado esse alto grau de incerteza e as constantes mudanças, os métodos ágeis focam em *feedback* contínuo, tanto da equipe em relação ao cliente quanto do cliente à equipe. Dessa forma, muitas das práticas sugeridas visam aumentar a quantidade e a qualidade desse *feedback*. O Desenvolvimento Guiado por Testes, ou do inglês, *Test-Driven Development* (TDD), criado por Kent Beck em 2003, é uma dessas práticas onde o foco é o retorno de respostas do processo de desenvolvimento(ANICHE, 2012).

Segundo Murata e Nogueira (2013), o TDD é uma forma de programar que encoraja o bom *design* e um processo disciplinado que auxilia a evitar erros de programação através da escrita de pequenos testes automatizados. Ele é uma prática de sucesso do XP (*Extreming Programming*). Quando ele é utilizado para criar novos códigos, ele é, às vezes, chamado de teste a *priori*, já que os testes ganham existência antes que qualquer código comece a ser testado. O TDD também pode ser usado para estender ou modificar código já existente, ou seja, novos testes podem ser criados para esses códigos.

O TDD é uma atividade de desenvolvimento de software baseado na repetição de um curto ciclo de operações. O fluxo de trabalho do mesmo é composto por três partes – escrevendo um teste, escrevendo o código do programa e refatorando. Vermelho e verde são as cores utilizadas pela maioria das ferramentas de teste para indicar falhas de teste e que o teste passou, respectivamente. As três palavras - vermelho, verde e refatorar - descrevem assim o processo do TDD (KOUDELIA, 2011).

Fox e Patterson (2015) explicam que antes de escrever qualquer código novo, deve-se escrever um teste para apenas um aspecto do comportamento que ele deve ter. Como o código testado ainda não existe isso força a pensar em como se deseja que o código se comporte e interaja com os colaboradores. A seguir é descrita cada etapa do TDD:

1. Etapa vermelha: Nessa etapa o teste é executado e verifica-se que ele falha, já que ainda não foi programado o código necessário para fazê-lo funcionar.
2. Etapa verde: O código deve ser escrito o mais simples possível que permita que o teste passe sem quebrar qualquer outro código existente.
3. Etapa refatoração: Deve-se buscar oportunidades para refatorar o código e os testes – alterar a estrutura do código para eliminar redundâncias, repetições ou qualquer outra feiura que possa ter aparecido como resultado do acrescimento do novo código. Os testes asseguram que sua refatoração não introduzirá nenhum erro.

Cada etapa deve ser repetida até que todos os comportamentos necessários para fazer a que a etapa do cenário passe estejam completos.

Aniche (2012) aponta como benefícios atingidos com o TDD, a bateria de testes de unidade alcançados que ajudam na prevenção de erros de regressão e em segurança nas constantes refatorações. O fato de ter essa mudança na ordem do ciclo de desenvolvimento ainda traz mais alguns benefícios, como maior simplicidade, menor acoplamento[[3]](#footnote-3) e maior coesão das classes criadas.

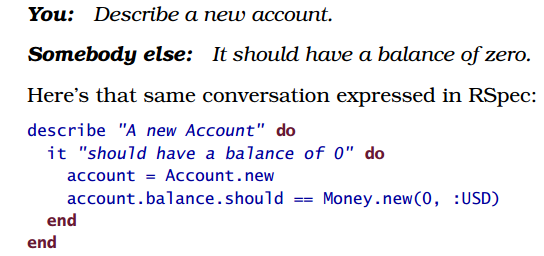
O programador deve dar pequenos passos, sempre escrever testes com menor funcionalidade possível, escrever o código mais simples que faça o teste passar, e fazer somente uma refatoração por vez. Porque quanto maior o passo, mais tempo vai levar para concluir, ou seja, mais tempo sem receber *feedback* sobre o código também. Além do programador não criar soluções mais complexas do que elas precisam ser. (ANICHE, 2012).

Murata e Nogueira (2013) explicam que as dúvidas e as dificuldades enfrentadas por quem usa o TDD estão, principalmente, no pensamento errôneo de que o TDD se foca em testar. Na verdade, testes são apenas efeitos colaterais positivos, pois o foco se encontra no comportamento esperado do software. Por exemplo, ao invés de escrever testes para cada método de uma classe, passa-se a escrever especificações sobre o que se espera do código, ou seja, especifica-se o comportamento do código antes de escrevê-lo.

A ferramenta que será utilizada para a prática do TDD será o RSpec para escrever testes de unidade. RSpec torna possível escrever poderosos testes automatizados para para aumentar a confiabilidade do código.

O RSpec fornece uma linguagem específica de domínio para especificar o comportamento de objetos. Ela abraça a metáfora de escrever o comportamento de maneira que possa expressá-lo como se estivesse falando com um cliente ou outro desenvolvedor CHELIMSKY (2010). A figura 7 mostra um trecho do que seria esse tipo de conversa:

Figura 7- Conversa expressa em RSpec



Fonte: CHELIMSKY (2010).

O método *describe* define um grupo de exemplo. A cadeia entre aspas representa a faceta do sistema que queremos descrever (a nova conta). O método *it* define um exemplo de código, a sequência passada a ele descreve um comportamento específico que estamos interessados. Esse é um exemplo somente para se ter uma ideia geral do funcionamento da ferramenta.

### *Behavior-Driven Development* (BDD)

O ciclo de vida ágil foi criado para remediar vários problemas presentes em muitos tipos de software, como por exemplo, o software não ser aquilo que o cliente esperava, atrasos no processo, ultrapassagem do orçamento original, etc. Por isso, ele envolve trabalhar de maneira próxima e contínua com os *stakeholders* para desenvolver requisitos e testes. Envolve também a manutenção de um protótipo funcional enquanto novos recursos são implantados para os *stakeholders* terem uma noção das próximas funcionalidades a serem adicionadas e se o sistema é o que eles estão querendo.

O *Behavior-Driven Development* (BDD) é uma técnica de desenvolvimento ágil de software que surgiu originalmente em 2003, criada por Dan North. Combina técnicas e princípios gerais do TDD com ideias vindas do *Domain-Driven Design* (DDD).

O BDD se refere a todo o comportamento, ou seja, tudo sobre o que um sistema deve fazer. Todo comportamento está lá porque acrescenta concretamente um valor de negócio para o sistema como um todo. Ele é impulsionado por valor de negócio, ou seja, pelo beneficio para o negócio que resulta uma vez que a aplicação está em produção (DUARTE; FERNANDES, 2013).

A prática faz questionamentos sobre o comportamento de uma aplicação antes e durante o desenvolvimento. Dessa forma as partes interessadas estão menos inclinadas a falhas de comunicação, os requisitos são continuamente aprimorados para ter certeza de que o sistema esteja de acordo com o esperado por eles (FOX; PATTERSON, 2015). O objetivo dos requisitos do BDD é, principalmente, construir a coisa certa.

Murata e Nogueira (2013) explicam que antes de iniciar o ciclo de desenvolvimento e entrega de um projeto, realiza-se uma reunião envolvendo todos os *stakeholders*, com o objetivo de definir a visão do projeto, ou seja, propósito. Em seguida, identificam-se os resultados esperados do projeto, que devem seu estar alinhados com a visão prévia que foi estabelecida para saber quando o projeto atingiu seu propósito.

No BDD, os requisitos são chamados de **histórias de usuários**. Uma versão mais leve dos requisitos e mais adequada aos métodos ágeis, que descrevem como se espera que a aplicação seja usada. Elas ajudam os *stakeholders* a planejar e priorizar o desenvolvimento. O processo começa pelas histórias de usuário (FOX; PATTERSON, 2015).

As histórias de usuário se referem a uma única funcionalidade com um ou mais cenários que mostram diferentes maneiras de utilizá-la. As palavras-chave **Funcionalidade** e **Cenário** identificam os respectivos componentes. E cada cenário é composto de sequências de 3 a 8 passos.

Boloz (2014) fundamenta que os cenários por sua vez são escritos do ponto de vista de um usuário, o esquema abaixo exemplifica o formato de um cenário:

**Dado** algum contexto inicial

**Quando** ocorre um evento

**Então**, descreve o resultado esperado.

**E ou Mas**

Essas palavras-chave ajudam os membros da equipe a decidir que tipo de comportamento é necessário para ser implementado na sequência. Cada frase é um passo do cenário e começa com sua própria palavra-chave. Os passos que começam com **Dado** geralmente estabelece condições. Passos que começam com **Quando** geralmente representa um usuário apertando um botão. Já os que começam com **Então** irão verificar se uma condição é verdadeira. Resumindo, Dado é usado para identificar o estado atual, **Quando** para identificar a ação e **Então** para identificar a consequência da ação. **E** e **Mas** são para permitir versões mais complexas da palavras anteriores.

A ferramenta que será utilizada para a prática do BDD será o Cucumber. Fox e Patterson (2015) descreve o Cucumber como uma admirável ferramenta que transforma histórias de usuário, que clientes podem facilmente entender, em testes de aceitação (verifica se o programa integrado atende as especificações) que garantem que o cliente está satisfeito, e em testes de integração (garantem que as interfaces entre as unidades tenham objetivos consistentes e se comuniquem corretamente), que garantem que as interfaces entre os módulos seguem o mesmo conjunto de hipóteses e se comunicaram corretamente. Essa ferramenta atua entre o cliente e o desenvolvedor, já que as histórias de usuário são claras para o cliente, mas ao mesmo tempo segue um padrão bem definido. A figura 6 abaixo mostra um exemplo de uma funcionalidade feita no Cucumber:

Figura 8 - Estrutura básica de um projeto em Cucumber.



Fonte: PEREIRA (2015).

Assim como Fox e Patterson (2015), Boloz (2014), dizem que a principal razão de muitas empresas usarem o BDD é que ele torna possível encontrar um caminho de comunicação entre os clientes e os desenvolvedores. No BDD os *stakeholders* estão menos propensos às falhas de comunicação porque são feitos questionamentos sobre o comportamento de uma aplicação antes e durante o desenvolvimento, os requisitos são aprimorados para que o software desenvolvido esteja de acordo com a vontade dos interessados. Logo, ao se concentrar no comportamento da aplicação ao invés de se concentrar na sua implementação, fica mais fácil reduzir mal-entendidos entre os *stakeholders.*

O BDD é uma metodologia de “fora para dentro”. Ela começa no exterior identificando os resultados do negócio, e, em seguida, perfura para dentro do conjunto os recursos que permite alcançar esses resultados. Cada recurso é capturado como uma história, que define o âmbito da aplicação do recurso, juntamente com seus critérios de aceitação. Embora, provavelmente, seja muito cedo para ser considerado a “melhor prática”, deve-se enfatizar que o BDD traz uma reformulação das boas práticas existentes (DUARTE; FERNANDES, 2013).

## RUBY

Ruby foi concebida em 1991 por Yukihiro Matsumoto, mundialmente conhecido como Matz. Ela é uma linguagem de programação com suporte à orientação a objetos, interpretada, multiparadigma, de tipagem forte e dinâmica. Também é reflexiva, permite que o código inspecione a si mesmo em tempo de execução, e suporta metaprogramação, que permite a geração e execução de código novo também em tempo de execução. A linguagem suporta programação funcional, especialmente o uso de blocos, onde trechos de código parametrizados chamados de expressões *lambda* carregam consigo seu escopo, fazendo com que sejam fechamentos (FOX; PATTERSON, 2015).

Como falado no parágrafo acima Ruby é uma linguagem orientada a objeto genuína, ou seja, quando você escreve código orientado a objeto, você está normalmente modelando conceitos do mundo real em seu código. Em ruby, objetos são criados chamando um construtor, que é um método especial associada a uma classe, o construtor padrão é chamado *new* (MATSUMOTO, 2001).

Como Ruby possui bibliotecas ricas em funções, mas existem poucos mecanismos, ela é considerada uma linguagem minimalista. Existem três características implícitas a esses mecanismos (FOX; PATTERSON, 2015):

1. Tudo é um objeto. Ruby dá suporte aos tipos básicos habituais – inteiros, números de ponto flutuante, cadeias de caracteres, vetores lineares e vetores associativos ou tabelas de *hash*. Mas ao contrário do Java, Ruby é **dinamicamente tipada:** o tipo de uma variável geralmente não é inferível até o tempo de execução.
2. Toda operação é uma chamada a um método de algum objeto e devolve um valor.
3. Toda programação é metaprogramação: o código é criado em tempo de execução que define novos métodos, ou seja, classes ou métodos podem ser adicionados ou modificados a qualquer momento, mesmo quando o programa está em execução.

A sintaxe do Ruby é fortemente influenciada pela sintaxe da linguagem Eiffel. Isso facilita a vida do programador porque a sintaxe é muito simples. Encoraja a escrever programas mais legíveis de uma forma mais elegante e facilitando que outro programador realize a manutenção de código posteriormente. Ela possui instruções nativas para *multithread* (processo paralelo) que são independentes de plataforma e seu coletor de lixo libera automaticamente toda a memória que foi utilizada durante a execução do programa (OLIVEIRA et al, 2006).

Ruby é a linguagem base das ferramentas utilizadas neste trabalho.

# TRABALHOS CORRELATOS

Neste capitulo serão apresentados alguns trabalhos relacionados que envolvem computação em nuvem, software como serviço e métodos ágeis.

## ENGINEERING SOFTWARE AS A SERVICE

Este trabalho trata-se de um curso onde Armando Fox e David Patterson (2015) lecionam em vídeo-aulas os fundamentos de engenharia de software usando técnicas ágeis para desenvolver Software como serviço (SaaS).

O curso ocorre na plataforma online aberta edX[[4]](#footnote-4), que mantém vários cursos de diversas áreas. Essa plataforma foi criada pela *Harvard University* e pelo *Massachusetts Institute of Technology*- MIT, com o intuito de oferecer cursos de nível superior gratuitos para qualquer um acessar virtualmente, em que os mesmos professores dos cursos presenciais ministram os online. A edX, assim como outras plataformas, ajudam a fortalecer os MOOCs (*Massive Open Online Course*), que são cursos online gratuitos onde qualquer pessoa pode fazer, além de serem oferecidos pelas mais renomadas universidades. Participam desse projeto universidades como *University of California* em Berkeley, Georgetown, etc.

O curso proposto por Fox e Patterson é composto por vídeos de palestras, que são separados em pequenos pedaços, geralmente, de oito a doze minutos cada. Segue abaixo, na tabela 7, a grade do curso com a programação dividida por semana.

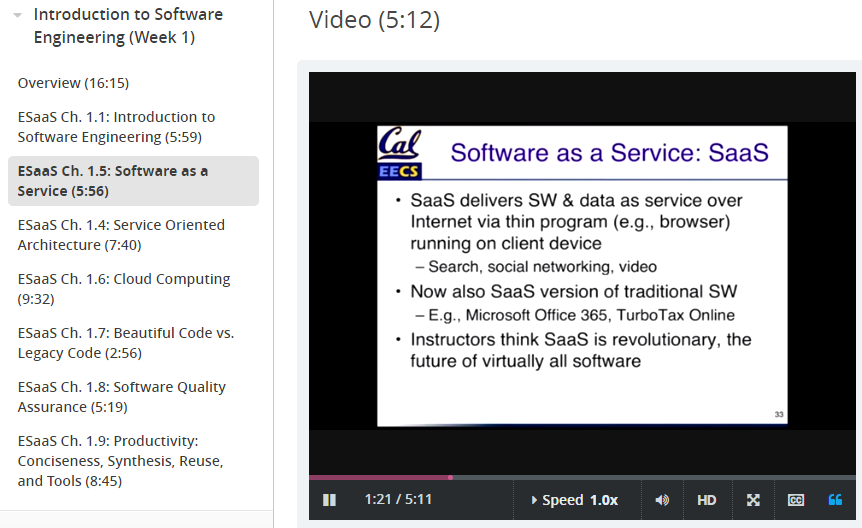
Tabela 7 - Programação do curso Engenharia de software como serviço

|  |
| --- |
| 1ª Semana |
| * Introdução a engenharia de software; * Planeje e documente x ágil, introdução ao ruby; |
| 2ª Semana |
| * Ruby; * Introdução ao BDD, TDD e SaaS; |
| 3ª Semana |
| * Arquitetura SaaS e REST; * Introdução aos Rails; |
| 4ª Semana |
| * Metodologias ágeis; * BDD com Cucumber e Capybara; |
| 5ª Semana |
| * TDD com RSpec. |

Fonte: Edx (2015)

A figura 9 abaixo dá uma visão do conteúdo abordado no curso.

Figura 9 - Curso Engenharia de Software como Serviço.



Fonte: EDX (2015)

Antes de começar o curso eles apresentam uma breve explicação sobre os requisitos necessários antes de iniciar o curso: É recomendado o conhecimento básico em Ruby, já que ela é a linguagem de programação utilizada no curso, e caso não queira instalar uma pilha de software a pessoa pode criar uma conta no *Cloud9*[[5]](#footnote-5)*,* que é uma plataforma de desenvolvimento com suporte as diversas linguagens de programação.

## BIBLIOTECAS NA NUVEM: O USO DA COMPUTAÇÃO EM NUVEM EM BILIOTECAS

Neste trabalho Martins (2012) apresenta as alternativas que a computação em nuvem pode oferecer às bibliotecas no que se refere à qualificação, modernização, preservação, acessibilidade e atualização de seus acervos e ainda à redução de investimentos na implantação, treinamento e manutenção dos sistemas de informática. Além de um cenário de como o assunto se encontra na esfera brasileira.

A autora faz um estudo por meio de uma pesquisa bibliográfica de análise qualitativa, em que o problema da pesquisa consiste no rumo que as bibliotecas estão seguindo quanto ao uso da computação em nuvem na disponibilização de seus acervos, e como a instituição se beneficia com essa tecnologia.

A autora propõe como objetivo:

1. Investigar a computação em nuvem como uma solução consolidada no que diz respeito aos modelos e arquiteturas, se ela é uma solução vantajosa para utilizar.
2. Examinar as possibilidades de aplicação dos serviços em nuvem em bibliotecas se existe motivos para o livro estar na nuvem, e quais soluções em nuvem existem para bibliotecas. Além de explorar o uso da *cloud computing* em bibliotecas, quais existem na nuvem. Cita por exemplo, a biblioteca Municipal de Palo Alto na Califórnia, Estados Unidos.
3. Investigar o cenário brasileiro em relação a computação em nuvem em bibliotecas, onde apresenta uma perspectiva da mesma para o futuro do livro digital em bibliotecas.

Apesar dos avanços tecnológicos preocuparem alguns especialistas, a autora enxerga eles como forma de preservar e democratizar o acesso aos acervos, já que as obras podem se apresentar em diversos formatos, como PDF, HTML, etc. O livro digital não será desvinculado da biblioteca, pois, segundo a autora, fica evidente sua pertinência e o justifica.

Martins (2012) discute que a migração para a nuvem ainda é um processo decisivo muito difícil e exige um bom planejamento que definirá as condições para um uso seguro da instituição. A migração para a nuvem não deve ocorrer de forma repentina. O planejamento inclui o estudo das necessidades da instituição, conhecer os serviços fornecidos no mercado, assim como possibilidade de serviços gratuitos, além de, é claro, garantia de privacidade e segurança dos dados.

Martins (2012) explica que a computação em nuvem se apresenta como um modelo consolidado, com modelos de serviço e de implantação bem definidos, que permitem a criação de serviços conforme a demanda. A *cloud computing* acaba se moldando às necessidades de qualquer instituição, isso faz com que a terceirização acabe sendo uma boa opção para pequenas e grandes associações. Em relação aos riscos, ela ressalta que são os mesmos como em qualquer outro serviço, e que os provedores/prestadores possuem equipes que para garantir que a nuvem seja segura para seus clientes, a mesma com certeza apresenta mais benefícios que riscos. Mas, adverte para um bom planejamento para definir o fornecedor e analisar os itens de segurança.

Existe um novo conceito de biblioteca, chamado biblioteca sem parede, que diferente das convencionais facilitam a disseminação e o aceso em tempo real. Dentre os tipos de bibliotecas temos: biblioteca eletrônica, biblioteca digital, biblioteca virtual, biblioteca híbrida e futuramente a biblioteca na nuvem (MARTINS, 2012). A tabela abaixo dá a definição de cada uma delas:

Tabela 8 - Tipos de bibliotecas

|  |  |
| --- | --- |
| Bibliotecas sem Paredes | |
| Biblioteca eletrônica | Ampla utilização de máquinas, principalmente microcomputadores, facilitando a construção de índices online, na busca de textos completos e na recuperação e armazenagem de registros. |
| Biblioteca digital | Diferencia-se por se construir de um acervo estritamente digital. Dispõe de todos os recursos de uma biblioteca eletrônica, Oferece texto completo, vídeo, áudio, etc., sendo o aceso local e remoto. |
| Biblioteca virtual | A “ciberteca” conceituada como um tipo de biblioteca que, para existir, depende da tecnologia da realidade virtual – sistemas computadorizados por meio de redes – que criaria um abiente com salas, estantes, etc. |
| Biblioteca híbrida | Agrega diferentes suportes tecnológicos, apresentando coleções impressas, digitais e acessos via rede eletrônica. Seria a transição das bibliotecas do impresso e do virtual |
| Biblioteca na nuvem | São acervos e aplicativos mantidos em *datacenters* terceirizados, que não ocupam espaços nos terminais e servidores locais, permitindo o aceso remoto das informações, fornecendo os serviços de um sistema operacional interno, mas em ambiente virtual. |

Fonte: MARTINS (2012).

O usuário hoje procura uma biblioteca onde possa ser atendido num ambiente virtual, visualizar em seu equipamento eletrônico. Dessa forma as bibliotecas veem significativas alterações em suas estruturas, como forma de armazenamento, por exemplo. Nisso a computação em nuvem surge como uma promessa revolucionária na forma como os sistemas computacionais serão administrados e utilizados, dispersos geograficamente, etc (MARTINS, 2012).

Como a instituição não tem o controle total de armazenamento, então existe certo receio quanto ao serviço que envolve nuvem. A ressalva feita pela autora é que mudanças sempre geram manifestações de medo e hoje vemos como os avanços das novas tecnológicos podem intervir na criação e ampliação das bibliotecas. Essa é a motivação para se discutir o rumo das mesmas em tempos de Internet. Espera-se com isso promover um debate, visto que, existe pouca discussão sobre isso no Brasil.

Como trabalhos futuros Martins (2012) sugere examinar o impacto da computação em nuvem aliada aos e-books nas bibliotecas universitárias. Outra pesquisa seria como as bibliotecas estão vendo as perspectivas da aplicação da *cloud computing*? O próximo trata-se um estudo sobre a aplicação SaaS na redução de custos na operação e manutenção de serviços em bibliotecas.

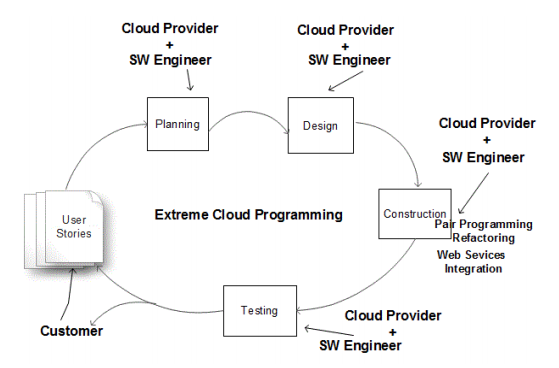
## IMPACT OF WEB 2.0 AND CLOUD COMPUTING PLATFORM ON SOFTWARE ENGINEERING

Neste artigo Guha (2010) analisa a forma como a computação em nuvem, tendo como plano de fundo a web 2.0, vai impactar o processo de Engenharia de Software para o desenvolvimento de software de qualidade. O autor faz os seguintes questionamentos: Como o provedor de nuvem é uma entidade externa ou de terceiros, o quão difícil será a interação com eles? Como separar os papéis de engenheiros de software e provedores de nuvem? O artigo amplia o tradicional modelo ágil *Extreme Programming* (XP) e integra a interação com o provedor de nuvem para facilitar a aceitação da computação em nuvem.

Guha (2010) explica que a economia de todos os países desenvolvidos depende de software de qualidade, e o custo do software é maior que o do hardware. Como o desenvolvimento de software envolve muitas partes, ele é inerentemente um processo complexo, diante disso, a tendência é que o projeto falhe devido à falta de comunicação e coordenação entre todas as partes envolvidas.

A tese principal do trabalho de Guha (2010) é que os modelos de processo de software prevalentes devem envolver o provedor de nuvem em cada tomada de decisão no desenvolvimento do ciclo de vida do software para tornar o projeto um sucesso. Então, propõe uma versão estendida do *Extreme* *Programming* (XP), um modelo de processo ágil, para a plataforma de computação em nuvem, nomeando-o de *Extreme Cloud Programming*, como mostra a figura 9.

Figura 10 - Extreme Cloud Programming.



Fonte: GUHA (2010).

Concluindo seu trabalho Guha (2010) explica que a computação em nuvem é uma mudança de paradigma sobre a forma tradicional de desenvolvimento de software, pois é preciso interagir com um novo elemento chamado provedor de nuvem. Mesmo a quantidade de trabalho para o desenvolvimento de software diminua, haverá a crescente exigência de comunicação e coordenação com o provedor de nuvem que faz o projeto de desenvolvimento de software mais complexo. Com isso os modelos prevalentes devem adicionar esse novo modo de interação com o provedor, e separar os papeis de engenheiros de software e provedores. Essa interação pode atenuar todos os desafios de desenvolvimento de software na plataforma de computação em nuvem e torná-la mais vantajosa no que se refere à implantação e desenvolvimento de software na nuvem.

# PROPOSTA

Como muitos autores afirmam construir software não é uma atividade trivial. Desenvolver um sistema bem sucedido depende de vários fatores e a Engenharia de Software apoia o processo de desenvolvimento oferecendo alguns métodos, técnicas e ferramentas para ter bons resultados e evitar problemas como, estouro de custos, de prazos, além de aumentar qualidade do produto disponível. Existe uma tendência para o desenvolvimento de software na nuvem, conhecido como Software como Serviço. Da mesma forma esse tipo de desenvolvimento necessita de mais estudos, visto que na literatura ainda não tem muitos trabalhos direcionados para essa área. Esse trabalho visa contribuir aplicando alguns métodos recentes da Engenharia de Software voltados ao desenvolvimento de software como serviço.

Essa pesquisa pretende utilizar os métodos XP, TDD e BDD para o desenvolvimento de um repositório bibliográfico digital de apoio à pesquisa, que funcione como uma base colaborativa e centralizada de bibliografias para estar à disposição do pesquisador. Segundo Dourado (2009), a necessidade desse tipo de aplicação se dá pelo fato de existir uma grande quantidade de publicações e é preciso um lugar que reúna e organize-as.

No ambiente SaaS é fácil implantar novos recursos rapidamente (FOX; PATTERSON, 2015), por esse motivo utilizamos o método ágil XP porque adota as mudanças como parte do processo e não como exceção. O XP também dá grande ênfase à escrita de testes antes de qualquer codificação, dessa forma utilizaremos também os métodos TDD e BDD voltados para essa finalidade, sendo adotados nas ferramentas RSpec e Cucumber, respectivamente. Como linguagem de programação será utilizada o Ruby por ser uma linguagem interpretada multiparadigma, considerada uma das melhores linguagens para SaaS, segundo Fox e Patterson (2015).

Para o desenvolvimento da aplicação existem as plataformas em nuvem Cloud9 e o *Google Cloud Plataform*, ambas oferecem suporte para a linguagem Ruby e tem acesso gratuito. Para a hospedagem da aplicação utilizaremos o servidor de aplicações em nuvem Heroku, que é um PaaS, rodando sobre o Amazon EC2, que é um IaaS.

# CRONOGRAMA PARA O DESENVOLVIMENTO DO TCC I E II

Esse capítulo apresentada o cronograma de desenvolvimento das atividades que serão realizadas no TCC I e II.

## CRONOGRAMA

As atividades que serão desenvolvidas no TCC I e II estão listadas abaixo e na Tabela 10:

1. Pesquisar e estudar o referencial teórico;
2. Escrever o TCC I;
3. Revisar e concluir o TCC I
4. Apresentar/Defender o TCC I
5. Desenvolver o protótipo;
6. Experimentar e avaliar o protótipo;
7. Analisar os resultados alcançados
8. Redigir o TCC II
9. Revisar e concluir o TCC II
10. Publicar artigo em evento específico
11. Apresentar/Defender o TCC II

Figura 11 – Cronograma de atividades do TCC

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Atividades | Out 15/Nov 15 | | Dez 15/Jan 16 | | Fev/Mar 16 | | Abr/Maio 16 | | Jun/Jul 16 | | Ago/16 |
| 1 |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |  |  |

# REFERÊNCIAS

ALKMIN, Cassio P.; CORDEIRO, Daniel. SIM MYCLOUD, simulando o gerenciamento de recursos virtualizados em plataformas de Computação em Nuvem. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUIDOS, 32, 2014, Florianópolis. **Artigo.** p. 977-984.

ANICHE, Mauricio Finavaro. **Como a prática do TDD influencia o projeto de classes em sistemas em sistemas orientados a objetos.** 2012. 75p. Dissertação (Mestre em Ciência da Computação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2012.

ARMBRUST, Michael et al. A View of Cloud Computing. **Communications of the ACM.** 53, 04, p. 50-58, april 2010.

BECK, Kent. et al. **Manifesto para Desenvolvimento Ágil de Software**. Ano2001. Disponível em: <http://www.agilemanifesto.org/>. Acesso em: 23 de dezembro 2015.

BECK, Kent; ANDRES, Cynthia. **Extreme Programming Explained:** Embrace Change. Massachusetts: Addison Wesley Professional, 2004. 224p.

BOLOZ, Dawid Mateusz. **Improving use of Behaviour-Driven Develoment in Websphere Commerce Projects.** 2014. 42p. Thesis (Bachelor´s Thesis). Degree Programme Software Engineering, JAMK – University of Applied Scienses, Jyväskylä, Finland, 2014.

CARISSIMI, Alexandre. **Desmistificando a Computação em Nuvem.** Disponível em: < www.lbd.dcc.ufmg.br/colecoes/erad-rs/2015/002.pdf >. Acesso em: 06 de outubro de 2015.

CHEUN, Du Wan et al. A Quality Model for Evaluating Software-as-a-Service in Cloud Computing. In: International Conference on Software Engineering Research, Management and Applications, 07, 2009, Montreal. **Artigo**. p. 261-266.

CHELIMSKY, David. **The RSpec Book.** 1 ed. Dallas: The Pragmatic Bookshelf, 2010. 400p.

CHURAKOVA, Inna; MIKHRAMOVA, Ramilja. **Software as a Service: Study and Analysis of SaaS Business Model and Innovation Ecosystems.** 2010. 151p. Thesis (Master in Economic Sciences: Business Engineering) - Economics Faculty, Gent University, Belgium, 2010.

COGO, Gabriel Silva. **Análise da Intenção de Adoção da Computação em Nuvem por profissionais da Área de TI**. 2013. 109p. Dissertação (Mestre em Administração) – Faculdade de Administração, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2013.

CRESWELL, John W.; CLARK, Vicki L. Plano. **Pesquisa de Métodos Mistos.** Tradução de Magda França Lopes. 2 ed. São Paulo: Penso, 2013. 288p.

DOURADO, Antonio Miguel Batista. O Uso de Metodologias Ágeis no Desenvolvimento de Software como Serviço: Base Bibliográfica Digital. Trabalho de Conclusão de Curso (Graduação) – Faculdade de Ciência da Computação, Marília, 2009.

DUARTE, César; FERNANDES, Amílcar. **Behavior-Driven Development.** Faculdade de Engenharia da Universidade de Porto, Portugal. Disponível em: < http://independent.academia.edu/C%C3%A9sarDuarte >. Acesso em: 27 de outubro de 2015.

EDX. Engineering Software as a Service. Disponível em: < https://courses.edx.org/courses/>. Acesso em: 15 de outubro de 2015.

FAGUNDES, Priscila Basto. **Framework para comparação e análise dos métodos ágeis.** 2005. 134p.Dissertação (Mestre em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2007.

FOX, Armando; PATTERSON, David. **Engineering Software as a Service.** 1. ed. Tradução Daniel Cordeiro e Fábio Con. Berkeley: Strawberry Canyon LLC, 2015. 556p.

GORELIK, Eugene. **Cloud Computing Models.** 2013. 82p. Thesis (Master in Management and Engineering) - Massachusetts Institute of Technology, Cambridge, Massachusetts, EUA, 2013.

GUHA, Radha; AL-DABASS, David. In: International Symposium on Electronic System Design, 2010, Denton, Texas. **Artigo**, 2010. p. 213-218.

JAKONEN, Markus. **When to Utilize Software as a Service.** 2011. 121p. Thesis (Master in Technology) - University Of Turku, Finland, 2011.

KOUDELIA, Nikolai. **Acceptance Test-Driven Development.** 2011. 102p. Thesis (Master in Software Engineering) – University of Jyväskylä, Finland, 2011.

LUOMA, Eetu. **Examining Business Models of Software-as-a-Service Companies.** 2013. 88p. Academic Dissertation (Graduation) - Faculty of Information Technology, University of Jyväskylä, Finland, 2013.

MARTINS, Adriano. Fundamentos de Computação Nuvem para Governos**.** In: Congresso da Sociedade Brasileira de Computação, 30, 2010, Belo Horizonte. **Artigo.** 13p.

MARTINS, Luziane Graciano. **Biblioteca em nuvem**: o uso da computação em Nuvem em bibliotecas. 2012. 121p. Trabalho de Conclusão de Curso (Graduação) – Faculdade de Biblioteconomia e Comunicação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2012.

MATSUMOTO, Yukihiro. **Programming Ruby - The Pragmatic Programmers Guide**. Disponível em: < http://ruby-doc.com/docs/ProgrammingRuby/ >. Acesso em: 10 março 2016.

MERIAT, Vitor. **Modelos de Serviço na Nuvem**: IaaS, PaaS e Saas. Disponível em: < http://www.vitormeriat.com.br/2011/07/08/modelos-de-servio-na-nuvem-iaas-paas-e-saas/>. Acesso em: 10 de novembo de 2015.

MURATA, Renato Yoshio; NOGUEIRA, Marcelo. Manutenibilidade em Projetos de Software: Uma Abordagem Ágil Utilizando Behaviour-Driven Development. In: International Conference on Engineering and Computer Education, 7, 2011, Guimarães. **Artigo**. p. 168-172.

OLIVEIRA, Bruno G. et all. Estudo de Caso do Uso de Linguagens de Script em Jogos. **Anais SULCOMP**, 02, 2006.

PEREIRA, Thiago. **A arte de desenvolver testes - Cucumber + Capybara.** Disponível em < http://agiletesters.com.br/topic/103/a-arte-de-desenvolver-testes-cucumber-capybara>. Acesso em: 05 de outubro de 2015.

PINHO, Vitor Pedro Figueiredo. **SaaS**: Análise de impacto na transformação da investigação e desenvolvimento de produto para serviço. 53p. Dissertação (Mestre em Engenharia de Gestão) – Faculdade de Engenharia, Universidade de Porto, Porto, 2009.

RUCHUTRAKOOL, Wiphop; ZHU, Minpeng. **Customer relationship management software as a service**: What is the providers’ view of advantages of CRM as a service? 2009. 84p. Thesis (Master in Informatic) - Lund University, Scania, Sueden, 2009.

SATO, Danilo Toshiaki. **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**. 2007. 155p. Dissertação (Mestre em Ciências) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2007.

SIGNIFICADO. Disponível em: < http://www.significados.com.br/startup/>. Acesso em: 20 de março de 2016.

SOMMERVILLE, Ian. **Engenharia de Software**. Traduzido por Ivan Bosnic e Kalinka G. de O. Gonçalves. 9ª ed. São Paulo: Pearson, 2011. 529p.

UCHÔA, Alexandre Prestes. **Contratação de Desenvolvimento Ágil de Software Pelo Governo**. 2013. Dissertação (Mestre em Engenharia de Sistemas e Computação) - Programa de Pós-graduação em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2013.

UNIVERSO PROJETO. **Qual o melhor entre o Modelo Ágil e o Tradicional**. Disponível em: < https://universoprojeto.wordpress.com/tag/cascata/>. Acesso em: 15 de janeiro de 2016.

WYNNE, Matt; HELLESOY, Aslak. **The Cucumber Book**: Behaviour-Driven Development

for Testers and Developers. 1 ed. Dallas: The Pragmatic Bookshelf, 2012. 315 p.

1. Startup significa o ato de começar algo, normalmente relacionado com companhias e empresas que estão no início de suas atividades e que buscam explorar atividades inovadoras no mercado (SIGNIFICADO, 2016). [↑](#footnote-ref-1)
2. Balanceadores de carga dividem a carga de trabalho equitativamente, podem ser tanto dispositivos de hardware especializado, quanto um servidor web especialmente configurado para isso (FOX; PATTERSON, 2015). [↑](#footnote-ref-2)
3. Acoplamento é o grau de dependência entre dois métodos. Menor acoplamento tem pouca ou nenhuma dependência. [↑](#footnote-ref-3)
4. edX site: https://courses.edx.org [↑](#footnote-ref-4)
5. https://c9.io/ [↑](#footnote-ref-5)