UNIVERSIDADE FEDERAL DO PARANÁ

CAROLINE MUNHOZ CORRÊA PALESTINO

ESTUDO DE TECNOLOGIAS DE CONTROLE DE VERSÕES DE SOFTWARES

CURITIBA
2015

UNIVERSIDADE FEDERAL DO PARANÁ

CAROLINE MUNHOZ CORRÊA PALESTINO

ESTUDO DE TECNOLOGIAS DE CONTROLE DE VERSÕES DE SOFTWARES

Trabalho de conclusão de curso apresentado como critério de aprovação a disciplina de Pesquisa em Informação, do curso de Gestão da Informação, Setor de Ciências Sociais Aplicadas da Universidade Federal do Paraná.

Orientador: Professor, Dr. José Simão de Paula Pinto.

CURITIBA
2015

TERMO DE APROVAÇÃO

CAROLINE MUNHOZ CORRÊA PALESTINO

ESTUDO DE TECNOLOGIAS DE CONTROLE DE VERSÕES DE SOFTWARES

Trabalho apresentado como requisito parcial à obtenção do grau de Bacharel em Gestão da Informação no curso de graduação em Gestão da Informação, pela seguinte banca examinadora:

_____

Prof. Dr. José Simão de Paula Pinto
Orientador – Setor de Ciências Sociais Aplicadas da
Universidade Federal do Paraná, UFPR.

_____

Prof. Dr. Marcos Antonio Tedesch
Setor de Ciências Sociais Aplicadas da
Universidade Federal do Paraná, UFPR.

_____

Profa. Dra. Helena de Fátima Nunes Silva
Setor de Ciências Sociais Aplicadas da
Universidade Federal do Paraná, UFPR.

Curitiba, 02 de dezembro de 2015.

DEDICATÓRIA

A Deus por ser meu amigo fiel e verdadeiro, por me permitir realizar este trabalho, por me dar o dom da vida e por me dar a salvação por meio de seu único filho Jesus, simplesmente por me amar (João 3:16).

AGRADECIMENTOS

Ao meu esposo Saulo, a quem amo tanto, por estar sempre presente, por ser também meu melhor amigo, por orar por mim e por ser minha inspiração

Aos meus pais Carmelia e Francisco e ao meu irmão Guilherme por me amarem e me incentivarem para não desistir nos momentos de dificuldades.

Aos meus sogros Josias e Cleocy por me darem o maior presente, meu esposo e por me cobrirem de amor, incentivo e orações.

A todos os meus mestres do curso de Gestão da Informação (UFPR), em especial ao meu mestre e professor José Simão de Paula Pinto que mais que um orientador foi (e é) um amigo, me dando sempre palavras de apoio.

Ao meu gestor Luciano Johnson que esteve sempre pronto a me ajudar e orientar.

RESUMO

Para a Engenharia de Software, um grande desafio é a documentação de projetos de desenvolvimento de sistema. A documentação que precisa ser detalhada, porém concisa, não é gerada somente na versão final, ou seja, possui versões que precisam ser controladas para garantir a qualidade do projeto e do produto. Este trabalho tem como objetivo apresentar um estudo das vantagens do uso de tecnologias de controle de versões no desenvolvimento de sistemas de informação, por meio da pesquisa de tecnologias de controle de versões, do levantamento e da comparação das características oferecidas pelas ferramentas. Utilizando-se de uma pesquisa exploratória, aborda temas relacionados a Gestão de Documentos (como os tipos de documentos existentes e o clico de vida dos documentos) e o Versionamento de Documentos, apresentando a importância da documentação de projetos na área de desenvolvimento de sistemas, que trabalha com informações de importante grau de sigilosidade. Analisa os sistemas GIT/GITHUB, SVN (VisualSVN + TortoiseSVN) e DokuWiki. Compara o resultado dos mesmos e conclui que prefere-se o sistema GIT/GITHUB devido a sua simplicidade de instalação e que o DokuWiki é útil somente para realização de Gestão Eletrônica de Documentos.

Palavras-chave: Tecnologias de Controle de Versões. Gestão de Documentos. Desenvolvimento de Sistemas.

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Na área da Tecnologia da Informação (TI), constantemente, trabalha-se com informações sigilosas. Por exemplo, um desenvolvedor de sistemas de determinada empresa está trabalhando em um projeto de um cliente e ele tem acesso a informações de um servidor. Se essas informações e os códigos desenvolvidos por ele não forem devidamente documentados e guardados, o cliente pode ter um sério problema com a segurança de suas informações. Por esse motivo faz-se necessário a importância de um bom repositório de documentos.

Porém, um sistema que contém apenas a função de repositório de documentos não supre as necessidades da área de TI, especialmente para a área de Desenvolvimento de *Softwares*. É importante um sistema que possua também a função de versionamento desses documentos. O Controle de Versão é uma prática da Engenharia de *Software* cujo princípio está na organização de projetos por meio do gerenciamento de diferentes versões de um documento. Essa maneira de organizar bastante eficaz pois possibilita desenvolver paralelamente, acompanhar o histórico de desenvolvimento e até resgatar o sistema em um ponto que estava estável, isso sem mexer na versão principal. É a maneira mais confiável e correta de assegurar a saúde do código-fonte, principalmente se tratando de grandes equipes de projetos.

Pensemos na seguinte hipótese, utilizando o mesmo cenário já retratado: o desenvolvedor, a pedido do cliente, realizou diversas correções e mudanças no sistema, alterando assim a versão do mesmo. Porém, em determinado dia verificou que precisa retornar o sistema para uma versão que possuía determinada função. Se o código não foi bem registrado e o número da versão do registro não estiver correta (ou inapropriada), provavelmente o desenvolvedor terá muito trabalho para encontrar o registro de código correto e, se não encontra-lo, provavelmente terá que desenvolver um novo código, isto é, terá um grande retrabalho, podendo comprometer o projeto (prazos e até mesmo a qualidade do sistema). Justamente para esse (e outros) cenário que o uso de sistemas de controle de versão se faz necessário e importante.

Ao ler alguns artigos e fóruns na Internet sobre controle de versões, conversar informalmente com alguns colegas de trabalho e da faculdade pôde-se verificar que algumas empresas vêm utilizando *Wikis* como repositório de documentos. Durante o curso das disciplinas Estágio Supervisionado I (SIN113) e Estágio Supervisionado II (SIN117), foi desenvolvido um trabalho de implantação do Dokuwiki (por não possuir mais vínculos empregatícios com a empresa em que o trabalho foi aplicado, por questões éticas, preferiu-se não mencionar o nome da mesma). Durante o curso da disciplina Engenharia de *Software* (SIN135), ministrada pelo professor José Simão de Paula Pinto e por atuar na área de Tecnologia da Informação, houve o interesse em explorar mais o tema “Controle de Versões”. Surgiu-se então o desejo de analisar, além da DokuWiki (e por conhecer bem esse sistema, resolveu-se então avalia-lo para verificar se atende aos requisitos de um sistema de controle de versão de documentos), *softwares* gratuitos de controle de versão. Viu-se a necessidade de entender os possíveis conceitos relacionados ao tema, realizando assim uma pesquisa documental.

É importante compreender o termo *documento* considerado como uma categoria da Ciência da Informação. “Um documento é a transmissão de informação, indiferente de seu conteúdo, tanto que é costumeiro encontrar na literatura a tratativa de documento como sinônimo de informação” (DUMAS; PINTO, 2014, p.4). O documento pode então ser considerado como um objeto (material) ou uma inscrição de um objeto (imaterial) de comunicação entre um emissor e um receptor. Pode ser classificado como reais (um documento é inscrito ou inserido), irreais (podem ser classificados em duas categorias: o analógico/eletrônico e digital, sendo a natureza binária o principal diferencial entre eles) e virtuais (documento necessita de um suporte tangível; para acessar sua mensagem é preciso a utilização de recursos adequados).

A Gestão de Documentos é um processo pertencente a área da arquivologia que visa intervir no ciclo de vida dos documentos, com menor custo e maior eficiência e eficácia. Busca reduzir o número de documentos em proporções manipuláveis até que os mesmos tenham um destino final (essa destinação final pode ser a extinção ou o recolhimento para os arquivos de permanência). Resumidamente, é a administração, organização e o controle de documentos.

De acordo com Bernandes e Delatorre (2008), na análise documental podem ser identificados os elementos (suporte, formato, gênero, espécie, tipo, documento

simples e documento composto. Para Rhoads (1989), o ciclo total de vida dos documentos é representado por quatro fases: Elaboração/Criação/Produção do Documento, Utilização e Manutenção do Documento, Disponibilização do Documento e Gestão de Documentos de valor permanente.

A Engenharia de *Software* é uma disciplina do conhecimento humano da área de conhecimento da Tecnologia da Informação, cujo objetivo é definir processos, métodos, ferramentas e ambientes para construir um *software* a fim de satisfazer necessidades de cliente e usuário dentro de prazos e custos previsíveis. Uma das tarefas da Engenharia de *Software* é a documentação do projeto para gerenciamento da qualidade do produto e do processo. A documentação consiste no conjunto de manuais e de diagramas que explicam o funcionamento do *software*. Atualmente muitos produtos no mercado vêm com manual de instruções e com o *software* não é diferente, pois o usuário precisa saber como funciona o programa e a maneira ideal de se utilizar.

No desenvolvimento de *software* são produzidos muitos documentos, dentre eles documentos que descrevem processos, requisitos, modelos do sistema e documentos de apoio de uso do sistema (como manual do usuário, ajuda *on-line* e tutoriais). É necessário que a documentação tenha qualidade, pois propicia uma maior organização no processo de desenvolvimento de um sistema e além disso facilita as modificações e futuras manutenções no mesmo. De acordo com Sommerville (2003), existem vários processos (ou metodologias) para o desenvolvimento de *software*, mas há atividades comuns fundamentais a todos eles, como a Especificação de *Software*, o Projeto e Implementação, a Validação de *Software* e a Evolução de *Software*.

Há várias metodologias definidas na literatura da Engenharia de *Software,* mas é comum organizações criarem sua própria metodologia ou adaptarem algum processo à sua realidade. As principais abordagens de metodologias de desenvolvimento de *Software* são: Metodologia Estruturada, Metodologia Orientada a Objetos, Metodologias de Desenvolvimento Ágil.

A utilização de técnicas, metodologias e ferramentas da Engenharia de *Software* para desenvolvimento de um *software* não garante total qualidade do mesmo. Para que haja maior qualidade, é imprescindível a etapa de execução de procedimentos de testes, que pode ser considerada como “revisão” da especificação, do projeto e da codificação.

O Teste de *software* faz parte do processo de desenvolvimento de sistema de *software*. É a execução de um produto para verificar se ele atingiu suas especificações e se funcionou corretamente no ambiente para o qual foi projetado. Cada fase do desenvolvimento tem uma atividade de teste associada, essas fases são: Teste de Unidade, Teste de Integração, Teste de Sistema, Teste de Aceitação e Teste de Regressão.

O Controle de Versão é uma prática da Engenharia de *Software* cujo princípio está na organização de projetos por meio do gerenciamento de diferentes versões de um documento. Possibilita acompanhar o histórico de desenvolvimento, desenvolver paralelamente e até resgatar o sistema em um ponto que estava estável, isso sem mexer na versão principal. Portanto, Controle de Versão é administrar as variáveis da informação. Há muitos sistemas de controle de versão que podem proporcionar benefícios no sentido de recuperar e verificar versões anteriores; esses sistemas podem ser classificados como Local, Centralizado ou Distribuído. O versionamento de um sistema objetiva documentar as inclusões, alterações e exclusões de funcionalidades, possibilitando saber exatamente quando uma determina funcionalidade foi ao ar e resgatar uma versão anterior em caso de erros durante o processo de publicação. O Versionamento Semântico é uma proposta de um conjunto simples de regras e requisitos que ditam como os números das versões são atribuídos e incrementados.

Esta pesquisa contou com três etapas:

1. Etapa documental.
2. Etapa para uma pesquisa exploratória na Internet buscando-se encontrar sistemas gratuitos de controle de versão.
3. Etapa para realizar o *download* e a instalação desses sistemas em dois ambientes (ambos computadores pessoais).
4. Etapa para adaptar um roteiro de planejamento de testes, que foi disponibilizado na disciplina Engenharia de *Software* (SIN135), ministrada pelo professor José Simão de Paula Pinto. A partir do roteiro, os requisitos e estratégias de testes foram definidos. Avaliou-se então a funcionalidade e a usabilidade dos sistemas.
5. Etapa para elaboração de dois protocolos para avaliação das ferramentas.
6. Etapa para comparação dos resultados e considerações sobre os mesmos.

Inicialmente tentou-se instalar todos os sistemas mencionados, porém surgiram diversas dificuldades para instalação dos sistemas CoTeia e Mercurial. Acredita-se que as instalações não obtiveram êxito devido a alguma incompatibilidade (não identificada) com os sistemas operacionais das máquinas utilizadas para os testes (Ambiente 01 e Ambiente 02), por esse motivo esses sistemas foram descartados.

Os sistemas gratuitos explorados e apresentados neste trabalho foram:

a) GIT e GITHUB (sistema distribuído).
b) Subversion SVN (sistema centralizado. É a evolução do modelo do sistema CVS, por esse motivo o CVS não foi "descartado"). Para uso desse tipo de sistema foi necessário a instalação dos *software*s VisualSVN Server e TortoiseSVN, que podem ser utilizados em ambiente *Windows*.
c) DokuWiki.

1.1 OBJETIVO GERAL

O objetivo deste trabalho é apresentar as vantagens do uso de tecnologias de controle de versões no desenvolvimento de sistemas de informação.

1.2 OBJETIVOS ESPECÍFICOS

1. Pesquisar tecnologias de controle de versões.
2. Levantar características oferecidas pelas ferramentas.
3. Comparar as tecnologias com as necessidades.

2 REFERENCIAL TEÓRICO

2.1 O QUE É DOCUMENTO

Antes de discorrer sobre a gestão de documentação em controle de versões de sistemas, é necessário realizar uma sucinta abordagem epistemológica sobre o termo *documento* que é considerado como uma categoria da Ciência da Informação.

A palavra *documento* deriva da palavra latina *documentum* e é decorrente do verbo do latim *docēre* cujo significado é *ensinar* ou *demonstrar.* Representa o meio de transmissão da informação e/ou do conhecimento (DUMAS; PINTO, 2014).

Para Otlet (1934), o conceito *documento* não se limita apenas ao entendimento de livro, como abordava a Biblioteconomia moderna (SHERA, 1980), pois *livro* corresponde à palavra latina *liber* (que é uma adaptação da palavra graga *biblos* – radical das palavras *biblioteca* e *biblioteconomia* -) e representava todo objeto artificial em que se reconhecia algum aspecto informativo. Então, o documento assumiu a condição de categoria da Documentação.

Otlet teve um papel fundamental no movimento bibliográfico, no final do século XIX, pois incentivou a criação de publicações especializadas e contribuiu para reuniões científicas e fóruns de debates de caráter acadêmico. Criou também associações que foram fundamentais para que pudesse propor disciplinarmente a Documentação (BLANQUET, 1993).

A retomada da noção de *biblos* possibilitou Otlet avançar também como a comparação da noção de fonte documental da “História Positivista”, pois, para ele, o texto escrito não seria o único que representava um documento. Esse estudo comparativo trouxe como evidência a aproximação dos conceitos “Documentação” e “História Positivista”.

Devido a aproximação encontrada entre os conceitos, surgiu o primeiro momento da Documentação, de “Fase Positivista”, em que após a década de 1930, o conceito de *documento* foi formulado considerando os objetos produzidos pelo homem em direção ao conteúdo informacional. A Fase Hermenêutica foi o segundo momento da Documentação (após a década de 1950). Nessa fase diversos autores

argumentavam que nenhum objeto/suporte nasce como um documento, pois esse aspecto se constitui posteriormente.

De acordo com López (2008), há uma dificuldade para encontrar e apresentar um conceito atual de documento devido às diversas ciências que estudam e que se utilizam do conceito. Pedauque (2003) discorre que são poucos os artigos científicos que propõem uma definição mais atual sobre documento, ao afirmar que “Muy pocos artículos científicos proponen una definición actual del documento, y todavía es menor el número de quienes la discuten. (PEDAUQUE, 2003, p.1).

Dumas e Pinto (2014) afirmam que:

> A função precípua de um documento é a transmissão de informação, indiferente de seu conteúdo, tanto que é costumeiro encontrar na literatura a tratativa de documento como sinônimo de informação (DUMAS; PINTO, 2014, p.4).

Sob essa perspectiva a informação pode ser considerada como um objetivo tangível, quando apresentada por um documento. Há ainda alguns autores como Silva *et. al.* (2008) que abordam o conceito *documento* sob a perspectiva da comunicação:

> No interior do sujeito, em interacção permanente com o seu meio envolvente, reside a permanente construção de sentido, de significado, potenciando-se um movimento semântico e semiótico bipolar quem diz/escreve significa e quem recebe/descodifica/interpreta capta e refaz o significado. A essência da comunicação passa por esse movimento em espiral, contínuo e infindável, implicando emissores, meios/canais para a mensagem e receptores, mas o que importa sublinhar, aqui, é que a interioridade humana de um texto valoriza-o como autêntico, mas não necessariamente como verdadeiro. (SILVA *et. al.*, 2008, p.12).

Documento é um meio/suporte de transmissão de uma mensagem de um transmissor para um receptor, com a intenção de comunicar algo.

Dumas e Pinto (2014) observam três planos para distinguir um documento, são eles: existência, validade e eficácia. A *existência* gera o pressuposto de que um documento existe quando está posto no mundo independente de possuir validade ou eficácia. Quanto à *validade* pode-se afirmar que o documento deve estar de acordo com a ciência ou área em que se está sendo gerado, guardado ou aplicado. Já “a

análise da eficácia de um documento deverá ocorrer em conformidade com o fim a que ele se propõe no campo do conhecimento em que está oposto" (DUMAS; PINTO, 2014, p. 6).

2.1.1 Tipos de Documentos

O documento pode então ser considerado como um objeto (material) ou uma inscrição de um objeto (imaterial) de comunicação entre um emissor e um receptor. Os documentos podem ser classificados como reais, irreais e virtuais.

É importante observar que há alguns elementos materiais que compõe um documento, são eles:

- Materialidade: De acordo com Dumas e Pinto (2014), o suporte funciona como elemento de fixação da informação. "O documento não é um ato, senão uma coisa" (CARNELUTTI, 2005, p.187).
- Integridade: É a "proteção contra modificações, duplicação, inserção, remoção ou re-ordenamento [sic] de mensagens" (SILVA *et. al.*, 2008, p. 10). Dumas e Pinto (2014) afirmam que pode-se fazer uma ligação entre o requisito da integridade ao plano da existência de um documento, pois quando um documento é adulterado ele deixa de existir, surgindo em seu lugar um novo documento, nesse caso o adulterado.
- Permanência: De acordo com Reig Cruañes (2005), um suporte é uma estrutura com a capacidade de conter informação que, ao ser fixada, conserva por muito tempo sua unidade de significação intelectual. A permanência é a condição necessária para transmissão da informação.

A partir dos elementos materiais que compõe um documento e dos tipos de documentos (descritos nos itens a seguir), foi possível elaborar a Tabela 1:

Quadro 1: Elementos que compõe um documento

TIPO DE DOCUMENTO	ELEMENTOS MATERIAIS QUE COMPÕEM UM DOCUMENTO		
	MATERIALIDADE	INTEGRIDADE	PERMANÊNCIA
REAL	Microscopicamente sua infinidade de átomos formam algo que transmite uma informação. Pode ser	Microscopicamente identifica-se as alterações ocorridas de elementos intelectuais.	Tempo em que as informações ficam contidas e disponíveis.

	observada pelos nossos sentidos.		
IRREAL (Analógico/ Eletrônico)	Não pode ser observada por meio dos sentidos. Precisa-se de diferentes aparelhos eletrônicos para acessar o seu conteúdo.	Precisa-se de um processo mais aprofundado para verificação das possíveis adulterações do conteúdo fixado.	Com o passar do tempo há a dificuldade de tecnologia hábil para leitura.
IRREAL (Digital)	Não pode ser observada por meio dos sentidos. Precisa-se de diferentes aparelhos eletrônicos para acessar o seu conteúdo. Possui capacidade de duplicação.	Pode ser alterado sem deixar vestígios. Solução mais moderna para garantir a integridade: assinatura digital.	É perpétuo, pode ter seu conteúdo acessado indefinidamente através dos tempos, desde que não seja deteriorado.
VIRTUAL	Possui materialidade dispersa. A informação não se encontra fixada em um suporte único.	Difícil de ser garantida, pois trata de vários documentos isolados que se integram para formar um documento composto.	Bem destrutível e privativo, pelo fato de ter vários documentos isolados.

Fonte: Adaptado de Dumas e Pinto (2014)

O Documento Real pode ser considerado como suporte material em que um documento é inscrito ou inserido. É descodificável diretamente pelos sentidos humanos. Teve seu nascimento com o surgimento da escrita.

Para construção de um Documento Irreal é necessário a utilização de tecnologias específicas para a inserção do conteúdo e visualização do mesmo. De acordo com Dumas e Pinto (2014) há dois tipos de documentos irreais: o analógico/eletrônico e o digital, sendo a natureza binária o principal diferencial entre eles. O documento analógico ou eletrônico “conta com um suporte material, porém o usuário não pode acessar diretamente a informação nele contida: necessita de intermediação de aparelhos leitores” (BRAVO, 2002, p.113). O documento digital possui o seu formato em bits. Constitui-se de uma codificação binária e é necessária a utilização de um computador para lê-lo.

O Documento Virtual, de acordo com Bravo (2002), é aquele que não se faz necessário de um suporte tangível; para acessar sua mensagem é preciso a utilização de recursos adequados que permitam entrar em qualquer instante no depósito irreal onde se guardam todos os documentos.

2.2 GESTÃO DE DOCUMENTOS

A Gestão de Documentos é um processo pertencente a área da arquivologia que visa intervir no ciclo de vida dos documentos, com menor custo e maior eficiência e eficácia. Busca reduzir o número de documentos em proporções manipuláveis até que os mesmos tenham um destino final (essa destinação final pode ser a extinção ou o recolhimento para os arquivos de permanência). Para realiza-la há um conjunto de procedimentos e operações técnicas para planejamento, capacitação, controle, promoção, fluxo, tramitação, uso, avaliação, seleção, organização, arquivamento, manutenção, disponibilização, acesso e conservação dos documentos, priorizando os aspectos qualitativos dos documentos (AMARAL; MEDEIROS, 2010). Resumidamente, é a administração, organização e o controle de documentos.

Auxilia para saber basicamente onde estão os documentos, quando pode-se eliminá-los, a restrição de acesso aos mesmos, para encontrar e identificar os documentos de guarda permanente (históricos) e cuidar bem deles. É importante porque é necessário ter clareza de quais informações podem e não podem ser fornecidas para que ninguém seja prejudicado e para, principalmente, não se “afogar” num mar de documentos e não encontrar o que se precisa, ou seja, agilizar o acesso às informações.

De acordo com Bernardes (1989), a Gestão de Documentos administra, organiza e controla o documento em todas as suas fases, são elas: corrente, intermediária e permanente Na Fase Corrente (ou 1ª idade) os documentos são vigentes e frequentemente consultados. Na Fase Intermediária (ou 2ª idade) os documentos já estão com pouca frequência de uso e estão no final de sua vigência; aguardam prazos de prescrição, precaução e sua destinação final (eliminação ou guarda permanente). Na Fase Permanente (ou 3ª idade) os documentos perderam a

sua vigência administrativa, porém são preservados em definitivo por seu valor histórico, legal e probatório.

Abreu (2015) classifica as fases da Gestão de Documentos em criação, alteração, roteamento, aprovação, recuperação, arquivamento, descarte e segurança, conforme o Quadro 2.

Quadro 2: Fases de Gestão de Documentos

Fase	Atividade
Criação	Criação do documento a partir de uma aplicação (exemplo: editor de textos), captura por meio de digitalização ou importação de legado, classificação da informação por categoria ou tipo, indexação e definição do critério de segurança.
Alteração	Controle de versão durante o ciclo de vida documental, que deve implicar em publicar, somente, a versão mais recente, permitir o acesso às versões anteriores (apenas com devida autorização), comparar diferentes versões do mesmo documento, criar, automaticamente, novas versões com codificação própria em diversos níveis e, ainda, criar um histórico de referência para auditoria.
Roteamento	Redirecionamento do documento a outro usuário (ou grupo) antes de publicá-lo.
Aprovação	Disponibilização do documento para reutilização.
Recuperação	A pesquisa do documento que permita sua reutilização é feita, geralmente, por meio de atributos (também chamados propriedades ou índices) do documento, utilizando uma linguagem de consulta. Há, ainda, outras técnicas de recuperação de documentos como Texto integral (*full-text retrieval*), por proximidade semântica, por fragmento de texto, por fonética e por enciclopédia.
Arquivamento	O arquivamento pode ser feito *on-line*, *near-line*, *off-line* e utilizando múltiplas mídias como discos óticos, CD, DVD, fitas, discos magnéticos e, até, formatos analógicos como o microfilme e o papel.
Descarte	Acontece quando o documento deixa de ser necessário, tornando-se acordo com uma tabela de temporalidade pré-estabelecida.
Segurança	Controle do acesso em vários níveis como leitura, edição etc..

Fonte: Abreu (2015)

A Gestão Eletrônica de Documentos (GED) pode ser descrita como um conjunto de tecnologias que permite uma organização realizar o gerenciamento de seus documentos em forma digital. Esses documentos podem ser das mais diversas origens, como papel, microfilme, imagem, som, planilhas eletrônicas, arquivos de texto etc.

2.2.1 Ciclo de vida de um documento

De acordo com Bernardes e Delatorre (2008), na análise documental podem ser identificados os elementos característicos de cada documento, esses elementos são: suporte (material sobre o qual as informações são registradas), forma (estágio de preparação e transmissão dos documentos), formato (configuração física que assume um documento, de acordo com a natureza do suporte e o modo como foi confeccionado), gênero (configuração que assume um documento de acordo com a linguagem utilizado na comunicação de seu conteúdo), espécie (configuração que assume um documento de acordo com a disposição e a natureza das informações nele contidas), tipo (configuração que assume uma espécie documental de acordo com a atividade que a gerou), documento simples (os documentos são simples quando formados por um único item), documento composto (os documentos são compostos quando, ao longo de sua trajetória, acumulam vários documentos simples).

Para Rhoads (1989), o ciclo total de vida dos documentos é representado por quatro fases:

a) Elaboração/criação/produção do documento: produção de formulários, estipulação de diretrizes e fomento de sistemas de gestão e aplicação de tecnologia moderna aos processos.

b) Utilização e manutenção do documento: controle e utilização dos documentos necessários para realizar ou facilitar as atividades de uma organização. Envolve a criação e o aperfeiçoamento dos sistemas arquivisticos e a recuperação de dados, a gestão dos arquivos, a análise de sistemas de produção e a manutenção de programas de documentos vitais, o funcionamento dos centros de documentação e a automação dos processos.

c) Disponibilização do documento: identificação e descrição das séries documentais, estabelecimento de programas de retenção, disponibilização e destinação dos documentos e as decisões acerca de conservação.

d) Gestão de documentos de valor permanente: desenho e os equipamentos dos depósitos, os processos de conservação e preservação dos arquivos, o planejamento de políticas de acesso aos arquivos, os

procedimentos dos serviços de referência, a criação de novos arquivos e a informação sobre eles.

2.3 DOCUMENTAÇÃO NA ENGENHARIA DE SOFTWARE

A Engenharia de *Software* é uma área de conhecimento da Tecnologia da Informação voltada para a especificação, desenvolvimento e manutenção de sistemas de *software*, com a aplicação de tecnologias e práticas da Ciência da Computação e Gerência de Projetos. Seu objetivo é definir processos, métodos, ferramentas e ambientes para construir um *software* a fim de satisfazer necessidades de cliente e usuário dentro de prazos e custos previsíveis. Sommerville (2007) afirma que:

> A engenharia de *software* é uma disciplina de engenharia relacionada com todos os aspectos da produção de *software*, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que este entrar em operação. (SOMMERVILLE, 2007, p. 15)

A Engenharia de Software engloba três elementos (métodos, ferramentas e procedimentos) que permitem controlar o processo de desenvolvimento e oferece uma base sólida para a implementação de *softwares* com qualidade, de maneira produtiva. Os métodos detalham o que fazer para a construção do *software*, as ferramentas apoiam de forma automatizada ou semi-automatizada aos métodos e os procedimentos conectam os métodos e as ferramentas permitindo que o desenvolvimento do *software* seja racional e oportuno (PRESSMAN, 2006).

É importante também compreender o conceito de *Software*. Sommerville (2007) discorre que um *software* não é somente um programa de computador e afirma:

> *Software* não é apenas um programa, mas também todos os dados de documentação e configuração associados, necessários para que o programa opere corretamente. Um sistema de *software* consiste, geralmente, de um conjunto de programas separados; arquivos de configuração, que são utilizados para configurar esses programas; documentação do sistema, que descreve a estrutura do sistema; a

> documentação do usuário, que explica como usar o sistema; e *sitesWeb* por meio dos quais os usuários obtém informações recentes sobre o produto (SOMMERVILLE, 2007, p. 15)

Entre várias das Engenharia de *Software,* há a tarefa da documentação do projeto para gerenciamento da qualidade do produto e do processo. A documentação também contribui para futuras manutenções e aprimoramentos. A documentação consiste no conjunto de manuais e de diagramas que explicam o funcionamento do *software*. Atualmente, muitos produtos no mercado vêm com manual de instruções e com o *software* não é diferente, pois o usuário precisa saber como funciona o programa e a maneira ideal de se utilizar. Se um projeto não estiver bem documentado tende a ter problemas, tais como: desenvolvedores alterando um mesmo artefato ao mesmo tempo, não se refletir alterações nos artefatos impactados por um artefato em alteração, não se saber qual a versão mais atual de um artefato, incompatibilidade entre os grupos de desenvolvimento, inconsistências, retrabalho, atraso na entrega e insatisfação do cliente.

No desenvolvimento de *software* são produzidos muitos documentos, dentre eles documentos que descrevem processos, requisitos, modelos do sistema e documentos de apoio de uso do sistema (como manual do usuário, ajuda *on-line* e tutoriais). É necessário que a documentação tenha qualidade, pois propicia uma maior organização no processo de desenvolvimento de um sistema e além disso facilita as modificações e futuras manutenções no mesmo. A documentação também contribui para a redução do impacto da perda de membros da equipe, do tempo de desenvolvimento de fases posteriores e de manutenção, diminuindo assim os erros e aumentando a qualidade do processo e do produto gerado. Os documentos a serem gerenciados são aqueles previstos como saídas das atividades do processo.

De acordo com Sanches (2001), pode-se afirmar, então, que a criação da documentação é tão importante quanto a criação do *software* em si. Porém existe a necessidade do planejamento e da definição de um processo para controlar a documentação de uma organização. Se há um processo definido do projeto, o planejamento de sua documentação consiste apenas em selecionar quais artefatos serão submetidos à gerência de configuração de *software* e ao controle e garantia da qualidade.

Apesar de sua importância, muitas vezes a documentação é negligenciada devido aos prazos limitados, a falta de uma política organizacional que valorize essa tarefa e a falta de ferramentas para apoiar a documentação, fazendo com que desenvolvedores optem por deixar a documentação em segundo plano. Então é comum uma organização gastar de 20 a 30% de todo o esforço de desenvolvimento na elaboração de documentos.

Sistemas que facilitam a documentação permitem agilizar o desenvolvimento de *software*, realizando algumas tarefas automaticamente e permitem que modificações sejam realizadas com mais facilidade, simplificando a tarefa de manutenção. De acordo com Falbo *et. al.*, um problema bastante comum no uso de ferramentas de documentação é o pouco suporte oferecido ao processo de documentação em si, incluindo a identificação dos documentos e suas relações com as atividades do processo e padrões organizacionais, o projeto da estrutura dos documentos, incluindo a definição de modelos de documento, e a própria elaboração de documentos.

2.4 A CRISE DO *SOFTWARE*

A “crise do *software”* surgiu entre as décadas de 1960 e 1970 quando os desenvolvedores começaram a ter dificuldades por causa da grande demanda. Era necessário o desenvolvimento de *softwares* mais complexos para acompanhamento das transformações ocorridas no mundo, porém a comunidade de *software* ainda não estava preparada para essas mudanças. Projetos tinham baixa qualidade e códigos de difícil manutenção. O *software* não era confiável e era entregue com atraso, seu custo ficava maior do que previam e, em muitos casos, seu desempenho era precário (SOMMERVILLE, 2007). Por isso surgiram técnicas para o desenvolvimento de sistemas com qualidade. Foi quando se viu necessário o surgimento de uma nova profissão, a Engenharia de *Software*.

Segundo Pressman (2006), o *software* passou por várias transformações, até na forma de seu desenvolvimento, pois antigamente o programador trabalhava sozinho e depois foi substituído por uma equipe de especialistas em *software*, cada

um se concentrando em uma pequena parte da aplicação, ou seja, o trabalho passou a ser dividido.

Em 1968 ocorreu na Alemanha a Conferência da Organização do Tratado do Atlântico Norte (OTAN) sobre Engenharia de *Software*, organizada para discutir sobre a chamada “Crise de Software”. O principal objetivo dessa reunião era o estabelecimento de práticas mais maduras para o processo de desenvolvimento. Foi aí que surgiu o termo “Engenharia de *Software*”, uma disciplina criada exclusivamente para estudar e aprimorar o desenvolvimento de *software* (SOMMERVILLE, 2007).

Em 1986 Alfred Spector (presidente da *Transarc Corporation*), foi coautor de um artigo que comparava a construção de pontes ao desenvolvimento de *software*. Ele afirmava que as pontes normalmente eram construídas no tempo planejado, no orçamento, e nunca caiam. No entanto, os *softwares* nunca ficavam prontos dentro do prazo e do orçamento, e ainda quase sempre apresentavam problemas. Em 1995 a *The Standish Group* publicou um estudo que revelou que 84% dos projetos de *software* não eram bem sucedidos (eram cancelados ou apresentavam falhas críticas). Analisando apenas os projetos mal sucedidos, o custo real era 189% maior que o estimado, e o tempo de conclusão 222% maior. Estimou-se que nesse ano, as agências governamentais e companhias privadas estadunidenses tenham gasto US$ 81 bilhões apenas em projetos cancelados, e mais US$ 59 bilhões em projetos concluídos fora do tempo previsto. A organização *The Standish Group* continuou publicando seu relatório nos anos seguintes. 35% dos projetos de *software* iniciados em 2006 foram obtidos com sucesso, porém ainda é um problema que dois terços de todos eles fracassaram. Um relatório desses estudos intitulado como “*Chaos Research”*, os projetos são enquadrados em três categorias distintas:

- **Mal sucedido**: O projeto é cancelado em algum momento do desenvolvimento por uma ou mais razões.
- **Bem sucedido**: O projeto é concluído dentro do orçamento estimado e do prazo previsto.
- **Comprometido**: O projeto é concluído, porém entregue com atraso, com orçamento além do estimado, e em alguns casos, o *software* é precário.

A Engenharia de *Software* tem proposto a cada dia novos métodos para controlar a complexidade dos *softwares*, começando pelas metodologias de desenvolvimento linear, até os processos ágeis de desenvolvimento que atualmente

têm sido muito utilizados. Entretanto, mesmo com todo esse esforço, ainda há muitos projetos de *software* sendo desenvolvidos com problemas, por esse motivo Pressman (1997) afirma que a Engenharia de *Software* está em um estado de "aflição crônica". Essa "aflição crônica" pode ser comprovada pelos estudos realizados pela *The Standish Group* (citados anteriormente).

A figura 1 faz um retrato sobre os problemas de um projeto de *software*:

Figura 1:

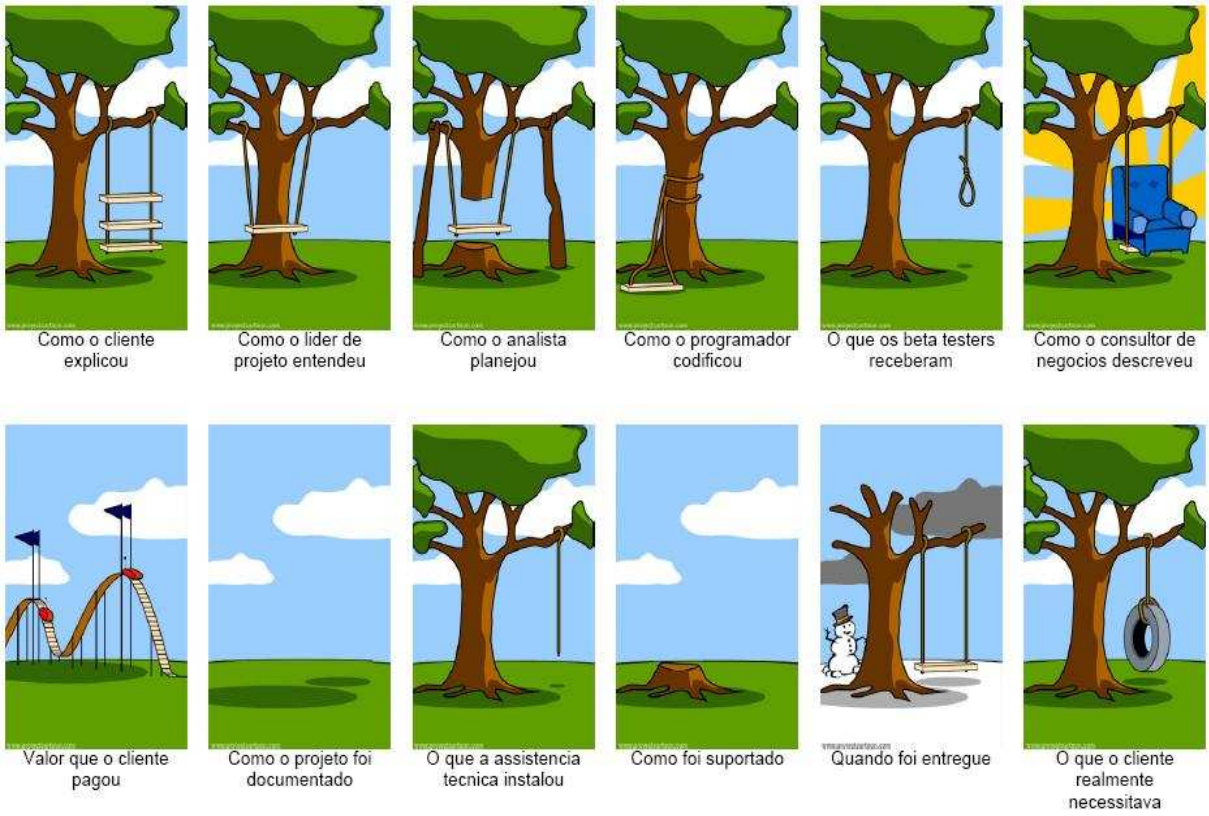


Figura 1. Fonte: Imagens Google - autor desconhecido.

2.5 METODOLOGIAS DE DESENVOLVIMENTO DE *SOFTWARE*

Há uma discussão na ciência sobre metodologia e método, pois essas palavras são muito utilizadas como sinônimos. Método pode ser entendido como um processo e a metodologia como o estudo de um ou vários métodos. Ambas as

palavras derivam do mesmo radical do Grego: “*métodos”* (caminho para chegar a um fim) e “*logia”* (estudo de).

Na Engenharia de *Software* alguns autores argumentam que método é um processo com uma série de passos para construir um *software* e a metodologia é a um conjunto de práticas recomendadas. Por tanto pode-se afirmar que uma metodologia é um conjunto estruturado de práticas utilizadas no processo de desenvolvimento de um *software*. De acordo com Caetano (2009), as metodologias de desenvolvimento de *software* são essenciais para não tornar a tarefa complexa. Porém, dependendo do projeto os métodos tradicionais podem deixar os desenvolvedores amarrados a requisitos desatualizados, que não correspondem às reais necessidades do cliente.

De acordo com Sommerville (2003), existem vários processos (ou metodologias) para o desenvolvimento de *software*, mas há atividades comuns fundamentais a todos eles, como a **Especificação de *Software*** (definição dos requisitos e das restrições), o **Projeto e Implementação de *Software*** (o *software* é produzido de acordo com as especificações. Nesta fase são propostos modelos por meio de diagramas aos quais são implementados em alguma linguagem de programação), a **Validação de *Software*** (o *software* é validado para garantir que todos os requisitos foram implementados) e a **Evolução de *Software*** (o *software* precisa evoluir para continuar sendo útil ao cliente).

Há várias metodologias definidas na literatura da Engenharia de *Software,* mas é comum organizações criarem sua própria metodologia ou adaptar algum processo à sua realidade. As principais abordagens de metodologias de desenvolvimento de *Software* são: Metodologia Estruturada, Metodologia Orientada a Objetos, Metodologias de Desenvolvimento Ágil.

As **Metodologias Estruturadas** baseiam-se em técnicas estruturadas de decomposição funcional (princípios semelhantes aos utilizados pelas linguagens de programação). Essas metodologias têm por objetivo a formalização do processo de identificação de requisitos para reduzir as possibilidades de má interpretação dos mesmos e para introduzir técnicas baseadas nas melhores práticas no processo de análise e desenho. Essas metodologias eram compostas por uma sequência de fases e atividades, com *inputs* e *outputs*, regras, intervenientes, técnicas, notações, ferramentas, documentação, técnicas de gestão, etc., com o objetivo focar mais no processo global e menos na programação. A maioria dessas metodologias adotaram

o modelo em cascata, em que cada atividade tem que ser completada e finalizada antes que a atividade seguinte possa ser iniciada. Nesse contexto surgiram os conceitos de ciclo de vida, e de metodologias de desenvolvimento de *software*. As metodologias estruturadas introduziram um conjunto de conceitos, são eles:

- Processo: é uma sequência de atividades, que processam vários *inputs* e produzem vários *outputs.*
- Fluxo de informação: toda a circulação de informação que ocorre numa organização de forma a executar os processos de negócio.
- Repositório de dados: conceitos que importam para a organização reter informação para utilização futura.
- Entidade: conceitos de negócio, entidades físicas ou abstratas, internas ou externas à organização, e que são relevantes para o desempenho adequado da função da organização.
- Evento: acontecimento que é desencadeado interna ou externamente ao sistema, ou pode representar simplesmente a passagem de tempo, e que desencadeia uma mudança de estado.

As **Metodologias Orientadas a Objetos** podem ser aplicadas na modelagem e no desenvolvimento de sistemas. Na modelagem, a orientação a objeto pode ser aplicada através das notações para modelagem de sistemas orientados a objetos, essas notações (como os métodos UML, OMT e Booch) são regras, conceitos e representações gráficas do sistema. E no desenvolvimento de sistemas através de linguagens de programação orientada a objetos (como Java, C++, Delphi). Nesse contexto pode-se definir objeto como uma abstração, com limites e significados bem definidos em relação ao problema considerado.

As **Metodologias de Desenvolvimento Ágil** tornaram-se conhecidas em 2001, quando especialistas em processos de desenvolvimento de *software* criaram a “Aliança Ágil” e o estabelecimento do “Manifesto Ágil”. - Pessoas e interações, ao contrário de processos e ferramentas. Os principais conceitos dessas metodologias são: *Software* executável (ao contrário de documentação extensa e confusa), colaboração do cliente (ao contrário de constantes negociações de contratos) e respostas rápidas para as mudanças (ao contrário de seguir planos previamente definidos). Uma característica das metodologias ágeis é que elas são adaptativas ao invés de serem preditivas. As metodologias ágeis trabalham com constante

feedback, o que permite adaptar rapidamente a eventuais mudanças nos requisitos. SCRUM, XP e FDD são alguns dos métodos ágeis mais utilizados atualmente.

Todas as metodologias geram documentos e esses documentos não são gerados somente na versão final, portanto possuem versões, necessitando então de um controle de versões.

2.6 TESTES DE SOFTWARE

A utilização de técnicas, metodologias e ferramentas da Engenharia de Software para desenvolvimento de um *software* não garante total qualidade do mesmo. Para que haja maior qualidade, é imprescindível a etapa de execução de procedimentos de testes, que pode ser considerada como “revisão” da especificação, do projeto e da codificação.

O processo de teste deve ser realizado de forma cuidadosa e criteriosa. Por esta razão, o esforço para realizar a etapa de teste pode chegar a 40% do esforço total empregado no desenvolvimento do *software*, principalmente pelo fato de que tudo deve ser documentado da minunciosamente. O processo de teste compõe-se de três partes: Planejamento de Testes, Execução de Testes e Controle de Testes. O Planejamento de Testes garante que os testes sejam preparados antes do fim da implementação do produto. A Execução de Testes executa os casos e procedimentos de teste especificados e comparar os resultados esperados e obtidos, registrando esses resultados. O Controle de Testes garante que os testes planejados sejam executados e que seus resultados sejam registrados por meio de constante monitoramento.

O Teste de *software* faz parte do processo de desenvolvimento de sistema de *software* e pode ser definido como a execução de um produto para verificar se ele atingiu suas especificações e se funcionou corretamente no ambiente para o qual foi projetado. O seu objetivo é revelar falhas em um produto, para que as causas das mesmas sejam identificadas e corrigidas, tudo isso antes da entrega final. Cada fase do desenvolvimento tem uma atividade de teste associada. As fases de testes são:

- Teste de Unidade: testa uma única unidade do sistema, identificando erros de lógica e de implementação em cada módulo, separadamente.
- Teste de Integração: testa a integração entre duas partes do sistema. Verifica se os componentes ou módulos do sistema, juntos, trabalham conforme as especificações do sistema e do projeto do programa.
- Teste de Sistema: avalia se o sistema funciona como um todo, com todas as unidades trabalhando juntas.
- Teste de Aceitação: valida com o cliente se o sistema está de acordo com suas expectativas e com as especificações do sistema e do projeto do programa.
- Teste de Regressão: os testes são refeitos para que novos defeitos não sejam introduzidos em módulos do *software* após o desenvolvimento de uma nova versão do produto.

Os testes podem também ser classificados como “Caixa Preta” ou “Caixa Branca”:

- Teste de Caixa Preta: verifica o funcionamento (incluindo a verificação de operacionalidade de todas as funções) do *software* através de suas interfaces. A maneira em que o sistema está organizado internamente não tem importância, mesmo que isso possa causar algum impacto na operação de alguma função observada em sua interface.
- Teste de Caixa Branca: examina a estrutura interna e os detalhes procedimentais. O “*status*” do programa pode ser examinado para eventual comparação com condições de estado esperadas para aquela situação.

Os processos de teste e de desenvolvimento devem estar separados, porém estão em um ciclo constante. Em outras palavras, pode ser definido como “o processo de analisar um item de *software* para detectar a diferença entre as condições desejadas e as condições existentes (ou seja, estado esperado e estado obtido) e avaliar uma característica/atributo de um item de teste” (LIMA *et. al.*, 2006).

Um processo de teste visa ocasionar falhas em um produto e não contribui diretamente para a sua construção, por isso dizemos que sua natureza é “destrutiva” e não “construtiva”. Um bom caso de teste é aquele que apresenta uma alta probabilidade de revelar um erro ainda não conhecido.

2.7 CONTROLE DE VERSÃO

O Controle de Versão é uma prática da Engenharia de *Software* cujo princípio está na organização de projetos por meio do gerenciamento de diferentes versões de um documento. Essa maneira possibilita acompanhar o histórico de desenvolvimento, desenvolver paralelamente e até resgatar o sistema em um ponto que estava estável, isso sem mexer na versão principal. Controle de versão é administrar as variáveis da informação. Muito usado por desenvolvedores, é o modo mais confiável e correto de assegurar a saúde do código-fonte, principalmente se tratando de grandes equipes de projetos.

O versionamento de um sistema tem por objetivo documentar as inclusões, alterações e exclusões de funcionalidades. Isso possibilita saber exatamente quando uma determina funcionalidade foi ao ar e resgatar uma versão anterior em caso de erros durante o processo de publicação. Um sistema com versionamento transmite muito mais segurança a quem o utiliza. Cada sistema de desenvolvimento trata diferentemente o versionamento e por essa razão é essencial realizar a leitura da documentação da ferramenta utilizada para que se possa trabalhar o versionamento com exatidão

De acordo com Dias (2011), com a falta de controle de versão a empresa pode ter os seguintes problemas:

- Histórico: não há o registro da evolução do projeto e das alterações sobre cada arquivo. Com essas informações sabe-se quem fez o que, quando e onde. Também permite revisar o arquivo sempre que desejado.
- Colaboração**:** não possibilita que vários desenvolvedores trabalhem em paralelo sobre os mesmos arquivos, resultando na probabilidade de que um sobrescreva o código de outro, o que pode fazer com que apareçam defeitos e perda de funcionalidades.
- Variações no Projeto: não mantém linhas diferentes de evolução do mesmo projeto. Por exemplo, não mantendo uma versão 1.0 enquanto a equipe prepara uma versão 2.0.

Em um Sistema de Controle de Versão os arquivos do projeto e o histórico de suas versões ficam armazenados em um servidor. Ele é comumente utilizado em desenvolvimento de *software* para gerenciar o código fonte de um projeto. Os

desenvolvedores podem acessar e resgatar a última versão disponível e fazer uma cópia local, na qual poderão trabalhar em cima dela e continuar o processo de desenvolvimento. Toda e qualquer alteração pode ser enviada novamente para o servidor, atualizando assim a sua versão a partir outras feitas pelos demais desenvolvedores. O Sistema de Controle de Versão oferece ferramentas que mesclam o código para evitar conflitos, como por exemplo, a edição mútua de um mesmo arquivo.

2.7.1 Versionamento Semântico

O Versionamento Semântico é uma proposta de um conjunto simples de regras e requisitos que ditam como os números das versões são atribuídos e incrementados. Para que isso funcione corretamente é necessário declarar primeiro uma Interface de Programação de Aplicativos (API) pública que pode consistir de documentação ou ser determinada pelo próprio código.

O segundo passo é comunicar as mudanças com incrementos específicos para o número de versão.

Se não houver uma especificação formal os números de versão tornam-se inutilizáveis para gerenciamento de dependências. Os desenvolvedores responsáveis por um sistema certamente desejam certificar-se que qualquer atualização no pacote funcionará como foi inicialmente desenhado. O Versionamento Semântico fornece uma maneira sensata de lançar e atualizar pacotes sem precisar atualizar para novas versões de pacotes dependentes, poupando tempo e aborrecimento.

Considere o formato de versão X.Y.Z (Maior.Menor.Correção).

- **X** (*Release* Maior): quando ocorrem mudanças incompatíveis na API.
- **Y** (*Release* Menor): quando são adicionadas funcionalidades mantendo compatibilidade.
- **Z** (*Release* de Correção): quando são corrigidas as falhas mantendo compatibilidade.

Quando um *software* é lançado recebe a o número de versão 1.0. O *Release* Maior (ou *Major Version*) geralmente é utilizado quando a nova versão do *software*

recebe grandes modificações, tornando-se diferenciada ou incompatível com a versão anterior. O *Release* Menor (ou *Minor Version*) é utilizado quando o *software* na versão 1.0 recebe melhorias, passa então para a versão 1.1. E o *Release* de Correção (ou *Patch Version*) é utilizado quando um sistema recebe correções o sistema de controle de versões recebe o terceiro dígito. Ou seja, seguindo o *software* recebe o número de versão 1.1.1.

De acordo com Carvalho (2015), as regras básicas são:

- Qualquer *software* que utiliza Versionamento Semântico deve declarar uma API pública que deve ser objetiva e clara.
- Um número de versão normal deve ter a forma X.Y.Z, onde os números devem ser inteiros. Cada elemento deve aumentar numericamente. Exemplo: 1.1.0 -> 1.2.0 -> 1.3.0.
- Qualquer modificação em determinado conteúdo de uma versão, deve ser lançada em uma nova versão. Ou seja, a versão anterior não deve ser modificada.
- *Major Version* que possui começo igual a “0” (por exemplo 0.x.z) é para desenvolvimento inicial. Nesse estágio a API pública pode ser considerada instável.
- *Major Version* deve ser implementada se quaisquer mudanças que quebrem a compatibilidade com versões anteriores são introduzidas na API pública.
- *Major Version* pode incluir mudanças *minor* e *patch*.
- *Minor Version* deve ser implementada se as novas funcionalidades forem compatíveis com as versões anteriores e se forem introduzidas à API pública.
- *Minor Version* pode ser implementada se novas funcionalidades e/ou melhorias forem introduzidas no código privado.
- *Minor Version* pode incluir *patches*.
- *Patch Version* deve ser incrementada se *buf fixes* (mudanças internas que corrigem um comportamento incorreto) introduzidos são compatíveis com versões anteriores.

2.7.2 Tipos de Sistemas de Controle de Versão

O Sistema de Controle de Versão pode ser classificado em Local, Centralizado ou em Distribuído.

O sistema Local é um dos métodos preferidos de controle de versão por muitas pessoas. Baseia-se no ato de salvar arquivos em outro diretório. É muito comum por ser tão simples, mas é também muito suscetível a erros pois é fácil esquecer em qual diretório foi salvo e gravar no arquivo errado ou sobrescrever arquivos acidentalmente. Por esse motivo, programadores desenvolveram Sistemas de Controle de Versão locais que armazenam todas as alterações dos arquivos sob controle de revisão

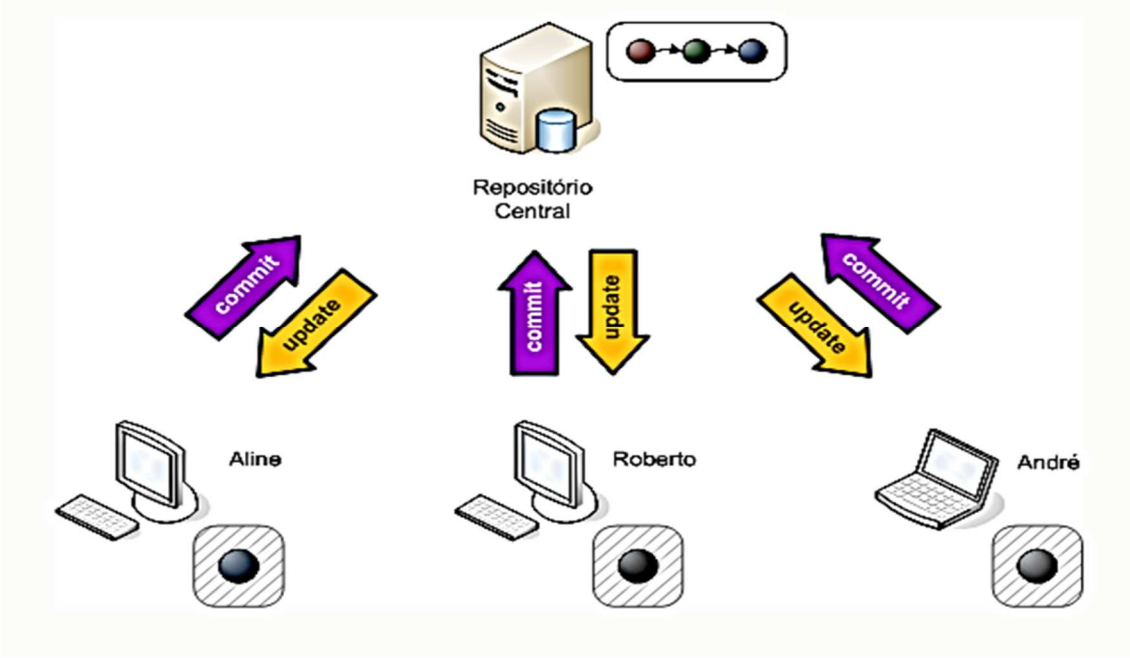
O sistema Centralizado (representado na Figura 2) possui apenas um único servidor central mas várias cópias de trabalho (uma para cada desenvolvedor), baseados na arquitetura cliente-servidor; é recomendado para empresas que trabalham com uma rede local. A comunicação é centralizada no servidor.

No controle de versão Distribuído (representado na Figura 3) há vários servidores autônomos e independentes (um para cada desenvolvedor) e as operações de *check-in e check-out* são feitas na própria máquina. Esse tipo de controle de versão é recomendado para equipes com muitos desenvolvedores e que se encontram em diferentes locais. É um sistema mais rápido, porém exige maior conhecimento da ferramenta e inicialmente isso pode atrapalhar os desenvolvedores.

As cópias de trabalho podem comunicar-se entre si sem a necessidade de um servidor como intermediador. A comunicação entre o servidor central e as áreas de trabalho funciona com outras duas operações, chamadas de *pull* e *push* (puxar e empurrar).

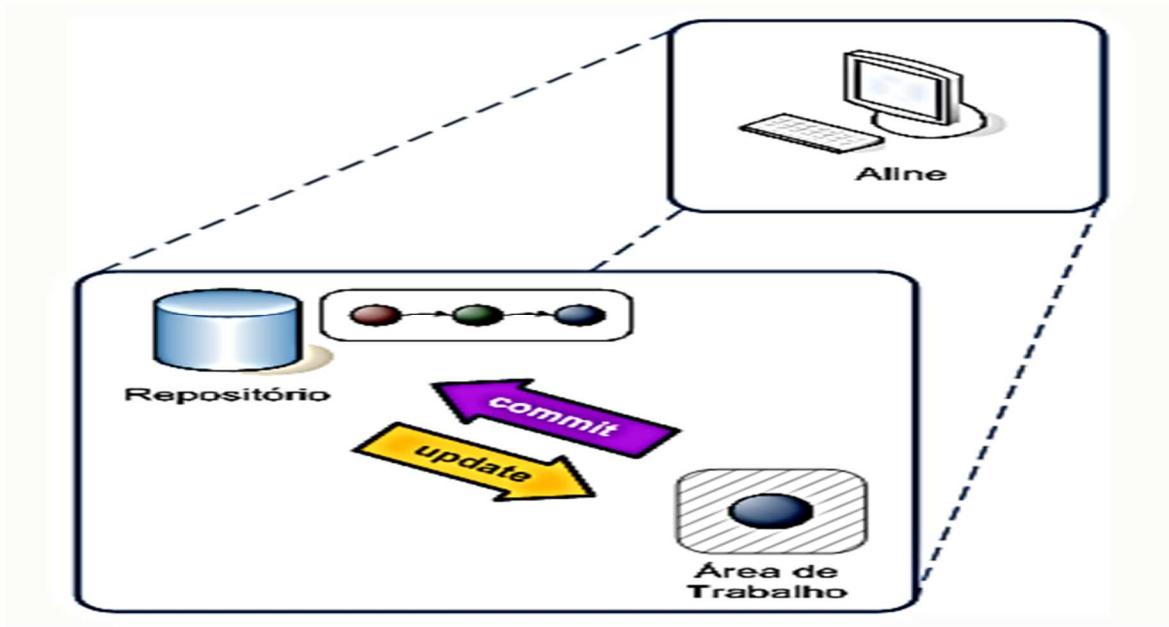
d) *pull*: operação que possibilita pegar a versão de outra área de trabalho e mesclar com a sua.
e) *push*: operação que envia para outra área a sua versão do projeto.

Figura 2: Representação de Sistema Centralizado



Fonte: Dias (2011)

Figura 3: Representação de Sistema Distribuido



Fonte: Dias (2011)

3 MÉTODO

Este trabalho utilizou-se de uma pesquisa exploratória e contou com 6 fases.

Na fase 1 foi realizada uma pesquisa em foi analisado e explorado como se dá a gestão e o versionamento de documentos em sistemas de controle de versão, para o desenvolvimento de *softwares*. Para isso verificou-se a necessidade de pesquisar sobre alguns temas em específico:

- Conceito, tipos de Documentos e o clico de vida dos mesmos.
- Gestão de Documentos.
- Documentação na Engenharia de *Software.*
- A Crise do *Software.*
- As metodologias de desenvolvimento de *Software*.
- Testes de *Software*.
- Controle de Versão, tipos de sistemas que possuem esse recurso e Versionamento Semântico.

Na segunda fase foi realizada uma pesquisa na Internet buscando-se encontrar sistemas gratuitos de controle de versão. Analisando e filtrando os inúmeros resultados de busca na Internet, verificou-se que os sistemas gratuitos mais utilizados são:

- CVS - Open Source Version Control;
- CoTeia;
- GIT e GITHUB;
- Mercurial;
- Subversion SVN.

A terceira fase foi realizado o *download* e a instalação desses sistemas nos seguintes ambientes (ambos computadores pessoais):

- Ambiente 01:
  - Edição do Windows: Windows 7
  - Processador: Intel ® Core™ i5- 2520M
  - Memória instalada (RAM): 16,0 GB
  - Tipo de sistema: Sistema Operacional de 64 Bits
- Ambiente 02:

- Edição do Windows: Windows 8
- Processador: Intel ® Core™ i5- 3230M
- Memória instalada (RAM): 4,00 GB
- Tipo de sistema: Sistema Operacional de 64 Bits

Na quarta fase verificou-se que os tipos de testes que se adequam para análise dos *softwares* são os “Teste de Aceitação” e “Teste de Caixa Preta” (que são significativamente parecidos, apesar do primeiro ser considerado como fase de teste e o segundo como uma classificação de tipo de teste). Nesse momento foi adaptado um roteiro de planejamento de testes, que foi disponibilizado na disciplina Engenharia de *Software* (SIN135), ministrada pelo professor José Simão de Paula Pinto. Durante a exploração do roteiro, decidiu-se analisar a funcionalidade e a usabilidade dos sistemas. O roteiro então ficou definido com cinco itens:

1 Requisitos de testes: Identificação de todos os itens que deverão ser alvo para teste. Esta lista de itens representará “o que” será testado.

2 Estratégia de testes: Descrição de “como” os itens identificados para teste serão testados.

3 Testes de funcionalidade: teste de instalação, teste de segurança e controle de acesso, teste de volume e teste de interface com o usuário.

4 Recursos: Descrição dos recursos que necessários para execução dos testes (banco de dados, equipamentos, etc.).

5 Resultado e Avaliação dos testes: Apresentação da diferença entre os resultados da execução dos testes dos sistemas.

A partir do roteiro, os **requisitos** e **estratégias** de testes foram definidos com os seguintes passos:

1. *Upload* de documento: enviar, fazer *upload* de um documento.
2. Arquivamento de documento: salvar o documento no sistema.
3. Versionamento semântico: verificar se o sistema gera automaticamente e corretamente a versão de um arquivo (de acordo com os critérios abordados sobre Versionamento Semântico). Verificar logo após um *upload* e uma alteração de um documento.
4. Visualização de documento: visualizar o documento no sistema.
5. *Download* de documento: baixar um documento do sistema.
6. Alteração de documento: alterar um documento do sistema e fazer o *upload,* verificando se substitui o documento anterior ou não.

7. Exclusão de documento: verificar comportamento do sistema ao excluir um documento.

Para avaliação da funcionalidade dos *softwares*, foram analisados os seguintes itens:

1. Configuração: verificar se *software* tema funciona corretamente nas configurações de hardware e software exigidos.
2. Aspecto Funcional: verificar a aceitação dos dados, do processamento, da resposta a esse processamento.
3. Instalação: verificar se o *software* pode ser instalado sob as condições de nova instalação (primeira instalação), atualização (já havia uma versão instalada) ou instalação personalizada (o usuário pode selecionar os componentes a serem instalados).
4. Segurança e controle de acesso: verificar se usuários são restringidos a funções específicas de acordo com o perfil que lhes foi atribuído.

Para avaliação da usabilidade foram utilizados os conceitos de Jakob Nielsen, apresentados pela profª dra. Maria do Carmo Freitas, durante o curso da disciplina Design da Informação (SIN115). Segundo Nielsen (2010), usabilidade é uma medida de qualidade que avalia o quão fácil é utilizar uma interface na visão do usuário, seja computadorizada ou não e está associada a alguns atributos como: Facilidade de aprender, Facilidade de lembrar, Eficiência, Erros e Satisfação. Quanto maior a usabilidade de uma interface, mais fácil será sua utilização. Foram definidos os seguintes itens para avaliação de usabilidade:

1. Facilidade de aprendizado: verifica se usuário pode realizar determinada tarefa sem precisar aprender novamente a utilizar o produto após um período sem utilizá-lo.
2. Rapidez no desenvolvimento de tarefas: verifica se o tempo que o usuário leva para executar determinada tarefa.
3. Baixa taxa de erros: os usuários cometem poucos erros ao usar o sistema e, quando cometem, sua correção é simples.
4. Linguagem: a linguagem utilizada no sistema deve ser considerada fácil de entender. Não deve ser confusa e nem redundante.
5. *Feedback*: é importante que usuário sempre esteja atualizado de suas ações através de *feedbacks* e esses, por sua vez, devem mostrar ao usuário sua atual situação.

6. Ajuda e documentação: é importante que o sistema possua recurso de ajuda ao usuário bem como um manual de utilização, de linguagem clara e objetiva.
7. Equilíbrio de cores: se as cores forem aplicadas adequadamente, podem ser utilizadas para identificar elementos que deverão atrair a atenção do usuário, podendo resultar em uma rápida e correta assimilação da informação que se quer passar. O uso incorreto delas pode distrair o usuário e diminuir sua produtividade.
8. Utilização de elementos visuais: utilização de elementos e que ajudam a visualizar melhor suas ações, como ícones e pictogramas.
9. Ícones: verifica se existem ícones e se sua aparência é simplificada. Os elementos mais significativos do ícone não devem ser muito pequenos em relação ao tamanho total. O ícone deve ser memorizado e relembrado.
10. Consistência: é recomendável que a interface deve se apresente sempre da mesma forma, com a mesma apresentação visual e comportamento, para evitar frustração causada por elementos inesperados. A consistência também pode ser medida pelo uso de terminologia, layout gráfico, conjunto de cores e fontes padronizadas.
11. Metáforas: pois aproveitam o conhecimento que o usuário tem do "mundo real". Por exemplo, o ícone de lixeira para representar local de arquivos excluídos.
12. Rótulos: uso de rótulos deve ser consistente.
13. Janelas: deve-se evitar abrir novas janelas ao clicar em um link ou botão, pois polui a tela e cobre a tela que o usuário está utilizando.

A análise do sistema *Dokuwiki* baseou-se em uma nas experiências obtidas durante o curso das disciplinas Estágio Supervisionado I (SIN113) e Estágio Supervisionado II (SIN117), sob orientação do professor Mauro José Belli. Pôde-se então verificar se uma ferramenta *Wiki* atende ou não às necessidades levantadas neste trabalho, para controle de versão de documentos.

Na quinta fase foram elaborados dois protocolos para avaliação das ferramentas. Com os **requisitos** e **estratégias** de testes já definidos, foi elaborado um protocolo (Quadro 3) simples para verificar se o sistema atende ou não às necessidades. O segundo protocolo (Quadro 4) avalia os sistemas com 18 itens (4

para funcionalidade e 14 para usabilidade. Cada um valendo 1 ponto) de 1 a 5 (sendo 1 péssimo e 5 excelente) de acordo com a sua funcionalidade e usabilidade:

Quadro 3: Procolo 01

Funções	Operou corretamente?	
	Sim	Não
Upload de documento		
Arquivamento de documento		
Versionamento semântico		
Visualização de documento		
Download de documento		
Alteração/Atualização de documento		
Exclusão de documento		

Fonte: A autora.

Quadro 4: Protocolo 02

NOME DO SISTEMA							
CATEGORIA	ITENS DE AVALIAÇÃO	1 (péssimo)	2 (ruim)	3 (regular)	4 (bom)	5 (excelente)	Observações
TESTE DE FUNCIONALIDE	Configuração						
	Funcional						
	Instalação						
	Segurança e controle de acesso						
TESTE DE USABILIDADE	Facilidade de aprendizado						
	Rapidez no desenvolvimento de tarefas						
	Baixa taxa de erros						
	Linguagem						
	Feedbacks						
	Ajuda e documentação						
	Equilíbrio de cores						
	Utilização de elementos visuais						
	Fontes						
	Ícones						
	Consistência						
	Metáforas						

	Rótulos						
	Janelas						

Fonte: A autora.

Na última etapa, os resultados da avaliação da tabela de usabilidade e funcionalidade foram compilados da seguinte maneira:

- Funcionalidade:

  - Contagem total de pontos de “péssimo”, “ruim”, “regular”, “bom” e “excelente”.

  - Cálculo da porcentagem dos pontos de “péssimo”, “ruim”, “regular”, “bom” e “excelente” em relação ao total de itens da categoria (total de 4 itens).

- Usabilidade:

  - Contagem total de pontos de “péssimo”, “ruim”, “regular”, “bom” e “excelente”.

  - Cálculo da porcentagem dos pontos de “péssimo”, “ruim”, “regular”, “bom” e “excelente” em relação ao total de itens da categoria (total de 14 itens).

- Funcionalidade + Usabilidade:

  - Contagem total de pontos de “péssimo”, “ruim”, “regular”, “bom” e “excelente”.

  - Cálculo da porcentagem dos pontos de “péssimo”, “ruim”, “regular”, “bom” e “excelente” em relação ao total de itens das duas categorias (total e 18 itens).

Ao término das contagens e pontuações, realizou-se uma simples comparação ente os resultados de pontuação dos sistemas.

4 RESULTADOS

Inicialmente tentou-se instalar todos os sistemas mencionados, porém surgiram diversas dificuldades para instalação dos sistemas CoTeia e Mercurial. Acredita-se que as instalações não obtiveram êxito devido a alguma incompatibilidade (não identificada) com os sistemas operacionais das máquinas utilizadas para os testes (Ambiente 01 e Ambiente 02), por esse motivo esses sistemas foram descartados.

Os sistemas gratuitos explorados e apresentados neste trabalho foram:

- GIT e GITHUB (sistema distribuído);
- Subversion SVN (sistema centralizado. É a evolução do modelo do sistema CVS, por esse motivo o CVS não foi "descartado"). Para uso desse tipo de sistema foi necessário a instalação dos *software*s VisualSVN Server e TortoiseSVN, que podem ser utilizados em ambiente *Windows*.
- DokuWiki.

Cada sistema foi testado seguinte o roteiro e critérios estipulados.

4.1 Tecnologias de Controle de Versão

A seguir apresenta-se as ferramentas analisadas e suas respectivas avaliações, com base nos requisitos e descritos no método.

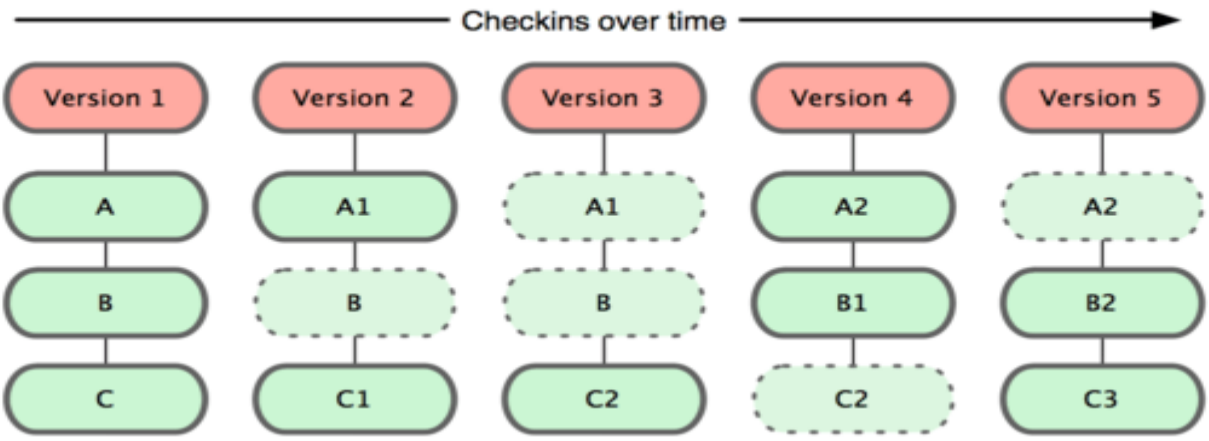
4.1.1 GIT e GITHUB

O GIT é um programa de controle de versões de documentos que possibilita trabalhar sempre em um mesmo diretório, fazendo alterações em seu projeto, gravando documentação e comentários. O sistema registra tudo o que for salvo e aprovado.

No GIT é possível também criar áreas completamente separadas para testes ou para projetos diferentes, desfazer alterações que estão com problemas, voltando para a versão que estava estável. Também possibilita o trabalho em equipe de maneira mais simples e segura, podendo criar e editar documentos de projetos na qual diversas pessoas podem contribuir simultaneamente, sem o risco de suas alterações serem sobrescritas. Por mais complexo que isso possa parecer, o GIT mantém tudo em ordem para evitar problemas para os desenvolvedores.

A maioria dos outros sistemas de controle de versão armazena informação como uma lista de mudanças por arquivo, tratando as informações e mantendo-as como um conjunto de arquivos e as mudanças feitas a cada arquivo. Um grande diferencial do GIT em relação à maioria dos sistemas é que ele não armazena as informações dessa forma, pois considera que os dados são como um conjunto de *snapshots* de um mini-sistema de arquivos. Para isso, no GIT há a possibilidade de criar, a qualquer momento, vários *snapshots* (ou “branch”. Espécie de “cópia espelho”. Captura de algo em um determinado instante, como em uma foto) do projeto. Pode-se supor que em um projeto de um sistema é preciso realizar determinada alteração no código fonte, mas não é desejável que essa alteração não esteja disponível para mais ninguém, a não ser somente para o desenvolvedor responsável pela alteração, até que tudo esteja certo. Então, o *snapshot* é uma boa solução para esse caso, pois o desenvolvedor pode trabalhar nesse “espelho” até acertar todos os detalhes do sistema e quando o mesmo estiver pronto, pode-se fazer um ***merge*** para o projeto original. Cada vez que se salva (ação conhecida tecnicamente como “*commit”*) um projeto, é como se ele tirasse uma foto de todos os seus arquivos naquele momento e armazenasse uma referência para essa captura. Se nenhum arquivo foi alterado, a informação não é armazenada novamente. A Figura 4 apresenta como o GIT trata os dados por meio dos *snapshots*.

Figura 4: Tratamento dos dados - GIT



Fonte: Site Git.

Também há disponível o GITHUB, que é um local de armazenamento em nuvem dos arquivos enviados via GIT. Com ele não é necessário instalar o GIT em diversos computadores toda vez que precisar acessar determinado arquivo (quando se está em viagem, por exemplo). O GITHUB é um serviço *web* que oferece diversas funcionalidades extras aplicadas ao GIT. Resumindo, pode-se usar, gratuitamente, o GITHUB para armazenamento de projetos pessoais e também, em outras palavras, pode-se dizer que o GITHUB é um complemento do GIT. Além disso, quase todos os projetos/*frameworks*/bibliotecas sobre desenvolvimento *open source* estão no GITHUB, o que torna possível acompanhá-los por meio de novas versões, contribuindo ao informar *bugs* ou até mesmo enviando código e correções.

A maior parte das operações no GIT precisam apenas de recursos e arquivos locais para operar. Por essa razão, o GIT é rápido, a maioria das operações são quase instantâneas. Para visualizar o histórico de um projeto, por exemplo, ele simplesmente lê diretamente do banco de dados local e não precisa requisitar ao servidor o histórico para que possa apresentar as informações.

Nesse sistema, a maioria das operações apenas acrescentam dados à base do GIT. É muito difícil fazer qualquer coisa no sistema que não seja reversível ou remover dados de qualquer forma. Então há o risco de perder ou bagunçar mudanças que ainda não foram salvas, mas depois de fazer um *commit* de um *snapshot* no GIT, é muito difícil perder essas alterações.

O GIT faz com que seus arquivos sempre estejam em um dos três estados fundamentais: consolidado (ou "*committed",* quando os dados estão seguramente armazenados em sua base de dados local), modificado (ou "*modified",* quando determinado arquivo que sofreu mudanças mas que ainda não foi consolidado na

base de dados) e preparado (ou "*staged",* quando um arquivo é já foi como preparado quando e é marcado como modificado na versão atual para que ele faça parte do *snapshot* do próximo *commit*). Para isso então há três seções principais de um projeto do GIT: o diretório do GIT (G*it Directory, Repository*), o diretório de trabalho (*working directory*), e a área de preparação (*staging area*), conforme ilustração a seguir:

Figura 5:Seções principais de um projeto do GIT

Fonte: Site Git.

Basicamente o *workflow* do GIT resume-se da seguinte maneira:

1. Modificação de arquivos em um diretório de trabalho.
2. Seleção dos arquivos, adicionando *snapshots* deles para uma área de preparação.
3. *Commit* que leva os arquivos como eles estão na área de preparação e os armazena permanentemente no diretório GIT.

No GIT não existe um repositório central. É possível escolher um local para essa finalidade, mas cada repositório, mesmo o da máquina do desenvolvedor, contém uma cópia completa e funcional do repositório. Uma desvantagem desse modelo é que a clonagem inicial do repositório pode demorar, pois não é realizado somente a transferência da cópia atual, mas também do histórico, *tags* e *branches*. Outra dificuldade está relacionada ao gerenciamento centralizado e um controle de acesso mais efetivo, porque os repositórios ficam distribuídos em vários ambientes.

Apesar das dificuldades encontradas pela falta de um repositório central, há certas vantagens, tais como:

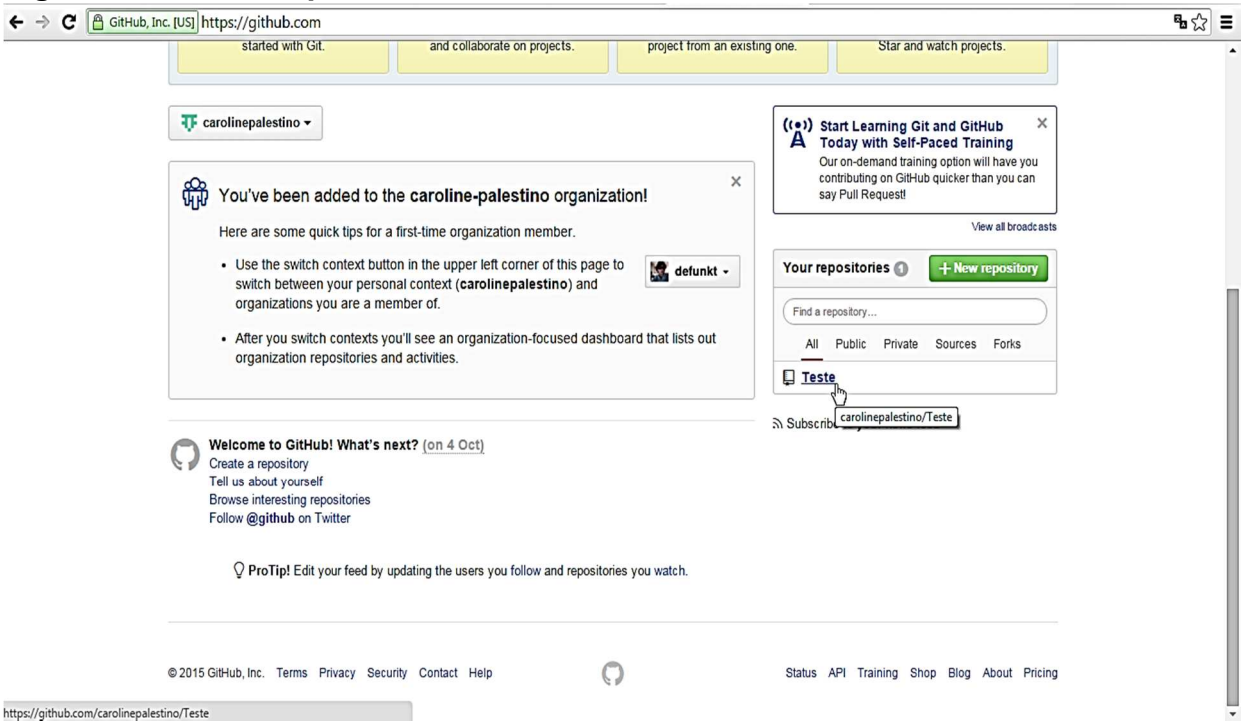
1. *Pull* (comando que recupera atualizações do repositório remoto para o local inicial, eles possuem mais velocidade do que os sistemas centralizados).

2. Diversas operações não necessitam de acesso à rede. Por isso o usuário pode trabalhar *offline*, sincronizando com o repositório remoto apenas quando achar necessário.
3. O usuário pode trabalhar em modo privado, gerando *tags*, *branches* e versões que podem ser descartadas.
4. Cada cópia do repositório funciona como um *backup* do repositório "principal".
5. *O merge* é automático somente quando há conflitos.

O GIT não necessita de *webservices*, não há requisitos “complexos” para instalação. Basta somente fazer o *download* do GIT e instalá-lo. Para utilização do GITHUB também não há limitações, pois pode-se acessá-lo de qualquer navegador.

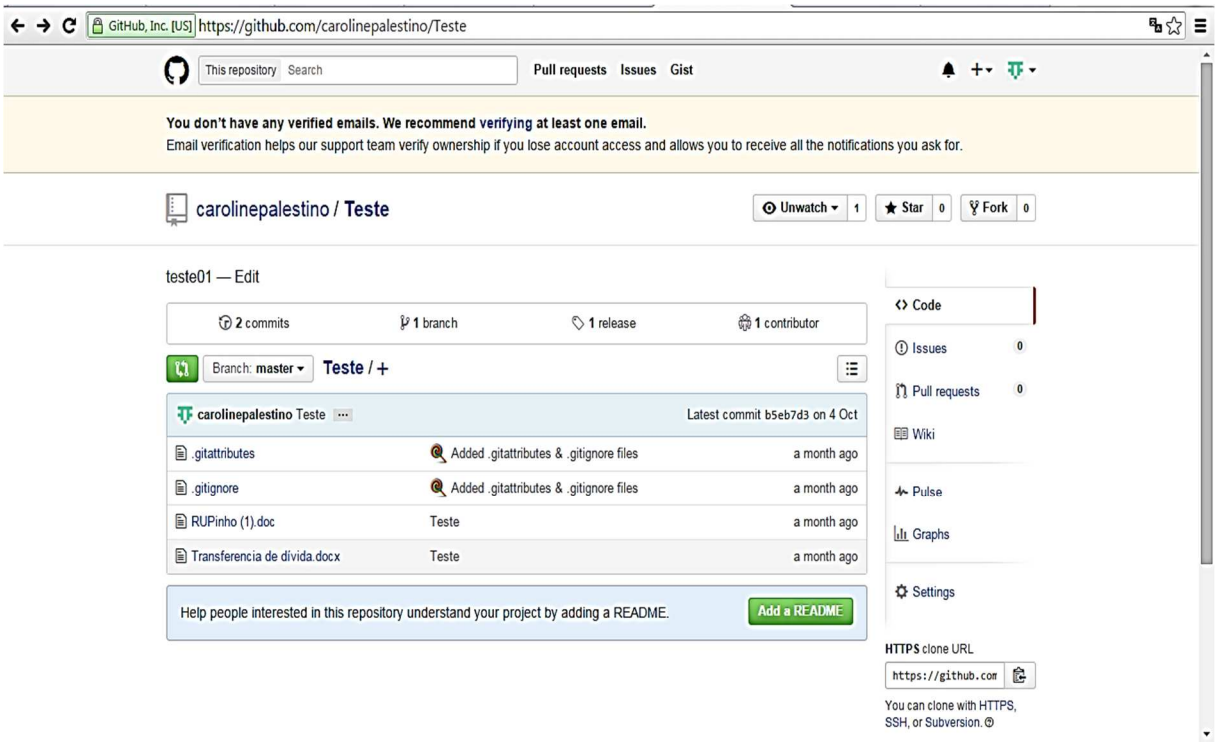
A Figura 5 trata de um *print* (cópia/foto) da tela de acesso ao repositório “Teste” na interface *web* do GITHUB e a Figura 6 a trata de um *print* da tela de visualização dos arquivos do repositório “Teste”.

Figura 5: Print de repositório criado no GIT



Fonte: A autora – *print* de tela do sistema GIT

Figura 6: Visualização dos arquivos em um diretório



Fonte: A autora – *print* de tela do sistema GIT

4.1.1.1 Avaliação GIT e GITHUB

Com base nos requisitos e critérios de avaliação, os sistema GIT e GITHUB foram avaliados da seguinte maneira:

Quadro 5: Protocolo 01 – GIT e GITHUB

Funções	Operou corretamente?	
	Sim	Não
Upload de documento	X	
Arquivamento de documento	X	
Versionamento semântico	X	
Visualização de documento	X	
Download de documento	X	
Alteração/Atualização de documento	X	
Exclusão de documento	X	

Fonte: A autora.

Quadro 6: Protocolo 02 – GIT e GIHUB

GIT E GITHUB							
CATEGORIA	**ITENS DE AVALIAÇÃO**	**1**	**2**	**3**	**4**	**5**	**Observações**
		(péssimo)	**(ruim)**	**(regular)**	**(bom)**	**(excelente)**	
TESTE DE FUNCIONALIDE	Configuração				X		
	Funcional				X		
	Instalação				X		
	Segurança e controle de acesso				X		
TESTE DE USABILIDADE	Facilidade de aprendizado				X		
	Rapidez no desenvolvimento de tarefas			X			
	Baixa taxa de erros				X		
	Linguagem			X			Somente em Inglês
	Feedbacks			X			
	Ajuda e documentação			X			
	Equilíbrio de cores					X	
	Utilização de elementos visuais				X		
	Fontes					X	
	Ícones				X		
	Consistência				X		
	Metáforas				X		
	Rótulos			X			
	Janelas					X	

Fonte: A autora.

A avaliação do sistema GIT/ GITHUB aponta que o mesmo atende a todas as necessidades. Quanto ao seu nível Usabilidade e Funcionalidade os resultados foram:

- Funcionalidade:

  - Total pontos “péssimo”: 0 (0%)

  - Total pontos “ruim”: 0 (0%)

  - Total pontos “regular”: 0 (0%)

  - Total pontos “bom”: 4 (100%)

  - Total pontos “excelente”: 0 (0%)

- Usabilidade:
  - Total pontos “péssimo”: 0 (0%)
  - Total pontos “ruim”: 0 (0%)
  - Total pontos “regular”: 5 (36%)
  - Total pontos “bom”: 6 (43%)
  - Total pontos “excelente”: 3 (21%)
- Funcionalidade + Usabilidade:
  - Total pontos “péssimo”: 0 (0%)
  - Total pontos “ruim”: 0 (0%)
  - Total pontos “regular”: 5 (28%)
  - Total pontos “bom”: 10 (55%)
  - Total pontos “excelente”: 3 (17%)

4.1.2 DokuWiki

Wikis são *sites* que podem servir como enciclopédias livres, são utilizados para identificar um tipo específico de coleção de documentos, alguns também servem como comunidades virtuais e reúnem somente um tipo de assunto que é abordado somente por pessoas com o mesmo interesse. Outros *Wikis* reúnem notícias que os usuários postam a respeito de determinado tema. Porém, algumas empresas optaram por esse tipo de sistema para a finalidade de repositório de documentos, pois os documentos podem ser criados e editados em comunidade e frequentemente podem ser revisados por administradores do *site*. O Dokuwiki foi escolhido, dentre diversas opções de W*iki*¸ pelos seguintes motivos:

a) Foi desenvolvido em PHP (linguagem conhecida devido às disciplinas da área de Tecnologia da Informação ministradas no curso de Gestão da Informação – UFPR).

b) A sintaxe é simples e dificilmente é preciso consultar a documentação para compreendê-la.

c) Cada página criada é salva como um arquivo de texto simples, o que facilita o *backup* (que é realizado ao compactar o diretório do Dokuwiki).

d) Não utiliza banco de dados, que faz com que tenha grande velocidade para criação, edição e leitura.

e) Permite o *upload* de imagens e uso de referências externas.

f) Suporta mais de cinquenta idiomas.

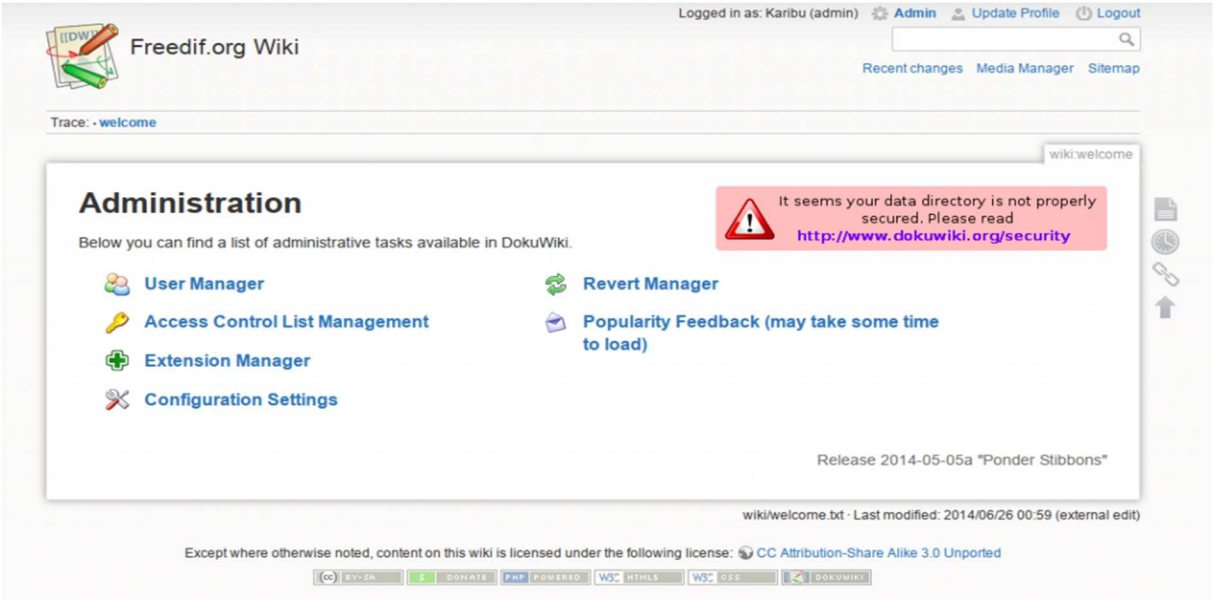
As principais características do Dokuwiki são:

- Ferramenta simples (voltada para criação de documentação de qualquer tipo). Voltado para grupo de trabalho e pequenas companhias.
- Sintaxe simples.
- Revisões de páginas ilimitado.
- *Links interwikis* customizáveis.
- Conteúdo pode ser categorizado em *nomespaces*, fácil de navegar pelo índex automático
- Roda em qualquer servidor que suporta *PHP, Apache, II, Lighttpd e Nginx.*
- Pode utilizá-la em qualquer navegador recente.
- Pode ser customizado conforme os desejos e necessidades do usuário.

A Dokuwiki possui um eficiente controle de permissões de acesso, em que o administrador pode estipular diferentes níveis de acesso, como: somente leitura, leitura e ***download***, leitura e edição (inclui ***upload***), edição e exclusão. Com isso, somente usuários registrados podem acessar ao sistema com seus devidos perfis.

A Figura 7 apresenta tela de configurações que somente usuário administrador possui acesso no DokuWiki.

Figura 7: Tela de configurações de acesso admistrador - DokuWiki



Fonte: http://freedif.org/dokuwiki-host-your-own-wiki-on-your-server/

4.1.2.1 Avaliação DokuWiki

Com base nos requisitos e critérios de avaliação, o sistema DokuWiki foi avaliado da seguinte maneira:

Quadro 7: Protocolo 01 - DokuWiki

Funções	Operou corretamente?	
	Sim	Não
Upload de documento	X	
Arquivamento de documento	X	
Versionamento semântico		X
Visualização de documento	X	
Download de documento	X	
Alteração/Atualização de documento	X	
Exclusão de documento	X	

Fonte: A autora.

Quadro 8: Protocolo 02 - DokuWiki

DOKUWIKI							
CATEGORIA	ITENS DE AVALIAÇÃO	1 (péssimo)	2 (ruim)	3 (regular)	4 (bom)	5 (excelente)	Observações
TESTE DE	Configuração			X			

FUNCIONALIDE	Funcional			X			
	Instalação			X			Necessita de webservices com suporte a PHP
	Segurança e controle de acesso					X	
TESTE DE USABILIDADE	Facilidade de aprendizado			X			
	Rapidez no desenvolvimento de tarefas				X		
	Baixa taxa de erros				X		
	Linguagem				X		
	Feedbacks				X		
	Ajuda e documentação			X			Vários documentos disponíveis, porém não são integrados a ferramenta.
	Equilíbrio de cores					X	
	Utilização de elementos visuais				X		
	Fontes				X		
	Ícones				X		
	Consistência			X			
	Metáforas					X	
	Rótulos					X	
	Janelas					X	

Fonte: A autora.

Baseado nas experiências com o sistema DokuWiki, não há dúvidas que ele é ótimo para realização de GED. Porém a avaliação aponta que o mesmo não atende a necessidade para controle de versão de documentos de desenvolvimento de softwares.

Quanto ao seu nível Usabilidade e Funcionalidade os resultados foram:

- Funcionalidade:

  - Total pontos “péssimo”: 0 (0%)

  - Total pontos “ruim”: 0 (0%)

- Total pontos “regular”: 3 (75%)
- Total pontos “bom”: 0 (0%)
- Total pontos “excelente”: 1 (25%)

- Usabilidade:
  - Total pontos “péssimo”: 0 (0%)
  - Total pontos “ruim”: 0 (0%)
  - Total pontos “regular”: 3 (21%)
  - Total pontos “bom”: 7 (50%)
  - Total pontos “excelente”:4 (29%)
- Funcionalidade + Usabilidade:
  - Total pontos “péssimo”: 0 (0%)
  - Total pontos “ruim”: 0 (0%)
  - Total pontos “regular”: 6 (33%)
  - Total pontos “bom”: 7 (39%)
  - Total pontos “excelente”: 5 (28%)

4.1.3 Apache Subversion (SVN)

O Apache Subversion (SVN) é um *software* de controle de versão e revisão de informações. Ele é um *software* livre desenvolvido para manter todo o histórico corrente de versões de pastas, documentação. Por ser um *software* livre seu código é aberto para desenvolvedores.

Um servidor de SVN (que pode ser denominado como "repositório") mantém sempre uma versão atualizada de todo o código. Cada pessoa que participa do desenvolvimento (usuário) pode acessar o servidor SVN utilizando um *software* que é chamado cliente SVN.

O SVN é a evolução do modelo do sistema CVS que possuía várias limitações. O SVN comparado ao CVS, dispõe os comandos *Rename* (função para renomear arquivos) e o *Move (*função para mover arquivos*)*. Tem também a capacidade de guardar metadados dos arquivos e diretórios. O SVN também

consegue rastrear arquivos renomeados, é mais rápido do que o CVS. O *commit* do CVS é por arquivo e o SVN consegue agrupar as mudanças de um *commit*, possibilitando voltar a uma revisão anterior (facilitando encontrar em qual *commit* o código foi quebrado).

O SVN necessidade de um repositório central para que os usuários façam o *checkout* e *commit* dos arquivos versionados. Possui grande capacidade de gerenciamento e usa seu repositório unificado para administrar conteúdo, *logs* e gerar informações estatísticas para uma informação armazenada. Servindo como um repositório de modificações, o SVN permite retornar o código a um estado anterior, auxiliando o analista na verificação das implementações combinadas de períodos diferentes criando uma única versão.

Quando o servidor SVN recebe uma nova versão realiza várias comparações do novo código em relação a última versão. O servidor SVN tentará resolver o conflito. Simões (2012) afirma que “Se arquivos diferentes foram alterados, ou se classes diferentes de um mesmo arquivo foram alteradas, por exemplo, o servidor resolverá automaticamente o conflito, de forma transparente para os usuários, realizando um *mearge* dos arquivos, e produzindo automaticamente um *head version* que é a união de ambos. Caso os dois desenvolvedores tenham, por exemplo, alterado as mesmas linhas nas mesmas classes de um código, o servidor pode não conseguir resolver o conflito. Neste caso, o usuário será alertado (no momento do *committ*) sobre o conflito, e será solicitado a resolvê-lo manualmente”.

Algumas das características e ações disponíveis do SVN são:

a) Repositório: possui local de armazenagem dos arquivos do projeto, no banco de dados do SVN.
b) *Working Copy*: é uma cópia do trabalho local em que o programador atua, este é criado automaticamente ao fazer o *checkout* de algum projeto.
c) *Checkout*: processo de *download* de um projeto para a máquina local, desde que seus arquivos estejam ligados ao SVN e passíveis de alteração.
d) *Import*: fazer o *upload* de arquivos de um novo projeto no repositório. Após o i*mport* o sistema obriga o usuário a fazer um *Checkout* para que o *Working Copy* seja vinculado ao SVN.
e) *E*xport: ação para obter o projeto pelo repositório sem vínculo com o SVN.
f) *Commit*: ação para enviar as modificações realizadas no computador local para o servidor do SVN.

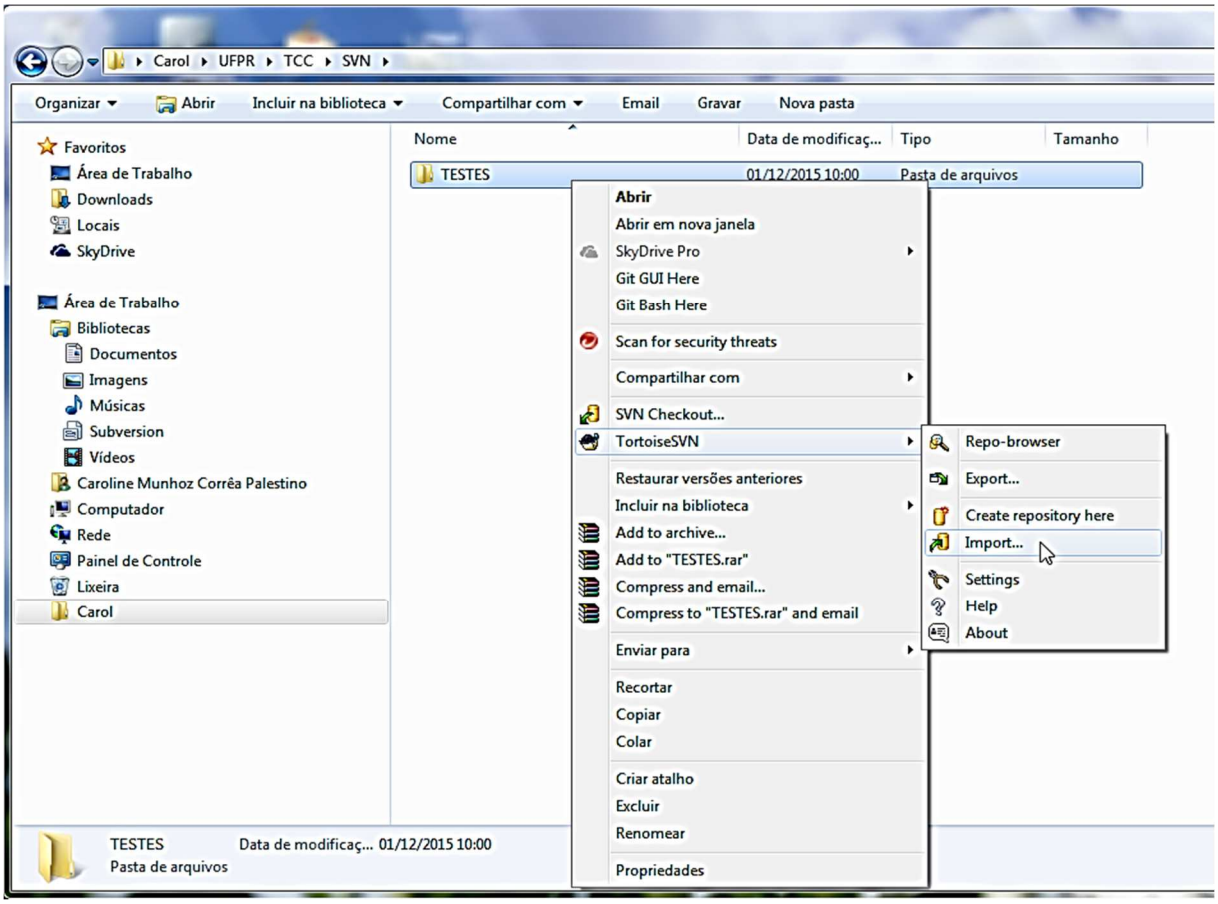
g) *Update*: são as atualizações do servidor do SVN.
h) *Revision*: é o número/versionamento de identificação obtido a partir de uma sequência, pela qual é compartilhada por todos os diretórios do repositório. Este expõe as alterações ou conjunto de alterações realizadas.
i) *HEAD*: é a revisão mais recente do repositório.
j) *Trunk*: armazenar a versão ativa mais recente de desenvolvimento.
k) *Branches*: armazena versões de desenvolvimento paralelo oriundas do *trunk*, porém isoladas deste. Deve ser utilizado quando uma implementação trazer o risco de afetar a integridade do *trunk*.
l) *Tags*: armazena etiquetas para facilitar a localização de revisões. Cada etiqueta possui um nome único que a identifica, sendo criada como um diretório, sempre através do *trunk*.
m) *Branch/Tag*: refere-se à geração de *branches* ou *tags* a partir de um *trunk* ou geração de um *branch* a partir de uma *tag* ou outro *branch*.
n) *Merge*: refere-se a mesclagem de revisões entre os diretórios especiais. Sempre deve ser realizada com a *working copy* apontando para o destino do merge.
o) *Switch: a*lteração do repositório utilizado por uma *working copy*. É realizada uma atualização ou mesclagem dos arquivos para assegurar que a *working copy* contenha exatamente o conteúdo do novo repositório mais quaisquer alterações locais.
p) *Relocate*: realocação do endereço de um repositório. Apenas atualiza o endereço, sem realizar nenhum tipo de atualização nos arquivos.

Durante o processo de pesquisa sobre o Subversion SVN, em específico, verificou-se a necessidade de uma ferramenta para instalar em servidor Windows (o Ambiente 01 foi utilizado como servidor e para instalação do *software cliente*). Ao realizar algumas pesquisas na Internet, resolveu-se utilizar o sistema VisualSVN Server que, de acordo com Passos (2010), é uma ferramenta para instalação e manutenção simplificada do Subversion SVN para Windows, juntamente com o servidor *Web Apache*, utilizado pelo Subversion, que só possui uma aplicação de linha de comando para ser usada pelo DOS. Fez-se necessário também instalar o sistema TortoiseSVN para ser utilizado como cliente (também instalado no Ambiente 01). TortoiseSVN é um cliente Apache SVN e foi escolhido por ser bastante intuitivo,

por ser um *software* livre (até para ambiente comercial). Ambos os sistemas são muito utilizados e bem reconhecidos.

A Figura 8 apresenta funções do TortoiseSVN ao clicar com botão direito do *mouse* sobre determinado arquivo.

Figura 8: Funções do TortoiseSVN



Fonte: A autora – *print* de tela.

4.1.3.1 Avaliação SVN

Com base nos requisitos e critérios de avaliação, o sistema SVN (VisualSVN + TortoiseSVN) foi avaliado da seguinte maneira:

Quadro 9: Protocolo 01 - SVN

Funções	Operou corretamente?	
	Sim	Não
Upload de documento	X	
Arquivamento de documento	X	

Versionamento semântico	X	
Visualização de documento	X	
Download de documento	X	
Alteração/Atualização de documento	X	
Exclusão de documento	X	

Fonte: A autora.

Quadro 10: Protocolo 2 - SVN

SVN							
CATEGORIA	**ITENS DE AVALIAÇÃO**	**1 (péssimo)**	**2 (ruim)**	**3 (regular)**	**4 (bom)**	**5 (excelente)**	**Observações**
TESTE DE FUNCIONALIDE	Configuração				X		
	Funcional				X		
	Instalação			X			Necessita de servidor
	Segurança e controle de acesso					X	Segurança pode ser baseada em permissão do servidor AD
TESTE DE USABILIDADE	Facilidade de aprendizado				X		
	Rapidez no desenvolvimento de tarefas					X	
	Baixa taxa de erros			X			
	Linguagem				X		
	Feedbacks				X		
	Ajuda e documentação			X			Vários documentos disponíveis, porém não são integrados a ferramenta.
	Equilíbrio de cores				X		
	Utilização de elementos visuais				X		
	Fontes					X	
	Ícones					X	
	Consistência				X		
	Metáforas					X	
	Rótulos					X	
	Janelas					X	

Fonte: A autora.

A avaliação do sistema SVN aponta que o mesmo atende a todas as necessidades. Quanto ao seu nível Usabilidade e Funcionalidade os resultados foram:

- Funcionalidade:
  - Total pontos “péssimo”: 0 (0%)
  - Total pontos “ruim”: 0 (0%)
  - Total pontos “regular”: 2 (25%)
  - Total pontos “bom”: 2 (50%)
  - Total pontos “excelente”: 2 (25%)
- Usabilidade:
  - Total pontos “péssimo”: 0 (0%)
  - Total pontos “ruim”: 0 (0%)
  - Total pontos “regular”: 2 (14%)
  - Total pontos “bom”: 6 (43%)
  - Total pontos “excelente”: 6 (43%)
- Funcionalidade + Usabilidade:
  - Total pontos “péssimo”: 0 (0%)
  - Total pontos “ruim”: 0 (0%)
  - Total pontos “regular”: 3 (17%)
  - Total pontos “bom”: 8 (44%)
  - Total pontos “excelente”: 7 (39%)

4.1.4 Análise comparativa das ferramentas

Elaborou-se o Quadro 11 para auxílio na comparação entre os sistemas:

Quadro 11: Comparação de Resultados

SISTEMA	PONTUAÇÃO FUNCIONALIDADE + USABILIDADE (%)				
	Péssimo	Ruim	Regular	Bom	Excelente
GIT/GITHUB	0	0	28	55	17

DOKUWIKI	0	0	33	39	28
SVN	0	0	17	44	39

Fonte: A autora.

Pôde-se observar que o sistema que possui menor pontuação em “Regular” é o SVN (VisualSVN + TortoiseSVN), deixando GIT/GITHUB em “segundo lugar” e o DokuWiki em “último lugar”.

5 CONCLUSÃO

Compreender a epistemologia do termo documento possibilita a reflexão sobre a importância e o valor da Informação (que resumidamente é um conjunto de dados que possuem um significado). Realizar uma gestão de informações compete ter compreensão sobre o valor das mesmas. Vive-se na “Era da Informação”, informações são criadas a todo momento e é preciso trata-las para absorver somente as que possuem qualidade, confiabilidade e veracidade.

Esta pesquisa tornou possível interligar diversos temas e conceitos aprendidos durante o curso de graduação de Gestão da Informação da UFPR, resultando no anseio de estar sempre atualizando o conhecimento. Vê-se como a melhor maneira de estimular a busca pelo aprendizado.

O Método utilizado para esta pesquisa foi de extrema valia para o alcance dos objetivos (entre eles, o principal: apresentar as vantagens do uso de tecnologias de controle de versões no desenvolvimento de sistemas de informação). Pôde-se concluir que existe uma grande “rivalidade” entre o modo centralizado e o modo distribuído de controlar versões. Verifica-se que o SVN e o GIT (juntamente com o GITHUB) são os mais utilizados para controle de versões em desenvolvimento de sistemas e atendem às necessidades. A maior diferença entre os dois é o modo de controle. Particularmente prefere-se o sistema GIT/GITHUB (apesar do sistema centralizado ser extremante simples também) devido a sua instalação simples, não necessitando de um servidor, o que possibilita também acessá-lo em qualquer lugar. O DokuWiki é excelente para realização de GED, pois possui funções específicas para tal, porém conforme a análise e avaliação realizadas, pode-se afirmar que o mesmo não atende ao quesito controle de versão de documentos, principalmente para a área de desenvolvimento de *softwares*. Se determinada empresa optou/opta pela utilização do DokuWiki devido a sua interface *web,* recomenda-se então a exploração e utilização do GIT/GITHUB.

Com esse estudo surgiu o interesse em realizar futuramente novas pesquisas para propor um método de melhorar a automatização de versionamento de documentos de sistemas de controle de versões.

REFERÊNCIAS

ABREU, S. **Gerenciamento do ciclo de vida de um documento.** Universidade Anhembi Morumbi. Disponível em: http://www2.anhembi.br/html/ead01/gestao_eletr_documentos/aula6.pdf. Acesso em 22 abr. 2015.

AMARAL, C. M. G.; MEDEIROS, N. L. A Representação do ciclo vital dos documentos: uma discussão sob a ótica da gestão de documentos. **Em Questão.** Porto Alegre, v.16, n.2, p. 297 -310, jul/dez. 2010.

BARRICO, C. **Evolução das Metodologias de Desenvolvimento de Software**. Disponível em: http://www.di.ubi.pt/~cbarrico/Disciplinas/EngenhariaSoftware/Downloads/Cap%203%20-%20Evolucao%20das%20Metodologias%20de%20Desenvolvimento%20de%20Software.pdf. Acesso em 06 jun. 2015.

BECHTLUFFT, R. **Criando sites não colaborativos com o DokuWiki**. 2009. Disponível em: <http://www.hardware.com.br/tutoriais/dokuwiki/>. Acesso em 01.nov.2015.

BERNARDES, I.P. **Como Avaliar Documentos de Arquivo**. São Paulo: Arquivo de Estado, 1998. Disponível em: <http://www.arqsp.org.br/arquivos/oficinas_colecao_como_fazer/cf1.pdf> Acesso em 01.mar.2015.

BERNARDES, I.P.; DELATORRE, H. Gestão Documental Aplicada. **Arquivo Público do Estado de São Paulo**. São Paulo, 2008. Disponível em: http://www.arquivoestado.sp.gov.br/site/assets/publicacao/anexo/gestao_documental_aplicada.pdf. Acesso em 22 abr. 2015.

BLANQUET, M.-F. La fonction documentaire: etude dans une perspective hístórique. **Documentaliste-Sciences de Information**, v.30, n.4-5, p. 199-204, 1993.

BRAVO, B. R. **El documento: entre la tradición y la renovación**. Getafe: Ediciones Trea, 2002.

CAETANO, R. **Metodologias de desenvolvimento:** qual a mais adequada?, 2009. Disponível em: http://computerworld.com.br/gestao/2009/08/05/metodologias-de-desenvolvimento-qual-a-mais-adequada Acesso em: 22 maio 2015.

CARNELUTTI, F**. A Prova Civil.** 4. ed. Campinas: Bookseller, 2005.

CARVALHO, B. M. de. **Padronizando o versionamento do seu software com Semantic Versionning**. Disponível em: http://planeta-globo.com/2010/08/18/padronizando-o-versionamento-do-seu-software-com-semantic-versioning/ Acesso em: 09 set. 2015.

CARVALHO, M. **O que é Git e Github?**. 2015. Disponível em: <http://www.blogdainformatica.com.br/o-que-e-git-e-github/>. Acesso em: 17.out.2015.

CRUAÑES, J. R. Para una teoría informativa del documento: extensión y aplicabilidad del concepto. **Revista Interamericana de Bibliotecologia.** Colombia, v. 28, n. 1, jan.-jun. 2005.

DIAS, A. F. **Conceitos Básicos de Controle de Versão de Software — Centralizado e Distribuído**, 2011. Disponível em: http://www.pronus.eng.br/artigos_tutoriais/gerencia_configuracao/conceitos_basicos_controle_versao_centralizado_e_distribuido.php?pagNum=0 Acesso em: 01 mai. 2015.

Diferenças entre Git, SVN e CVS. Disponível em: < http://pt.stackoverflow.com/questions/8315/diferen%C3%A7as-entre-git-svn-e-cvs>. Acesso em: 19.out.2015.

DUMAS, M. N.; PINTO, J. S. P. Uma busca por um conceito genérico de documento: tipos e suportes. **Processo Judicial Eletrônico.** Ordem dos Advogados do Brasil – Conselho Federal. Brasília, DF – 2014, p. 411 – 434.

FALBO, R. A. et. al. O. **Apoio à Documentação em um Ambiente de Desenvolvimento de *Software***. Disponível em: http://www.inf.ufes.br/~falbo/download/pub/2004-IDEAS-1.pdf. Acesso em: 17 mai. 2015.

FALBO, R. A. **Engenharia de Software – Notas de Aula.** Universidade Federal do Espírito Santo (UFES), 2005. Disponível em: http://www.inf.ufes.br/~falbo/download/pub/2004-IDEAS-1.pdf. Acesso em: 17 maio 2015.

FERREIRA, S.B.L.; LEITE, J.C.S.P. Avaliação da Usabilidade em Sistemas de Informação: o Caso do Sistema Submarino. **Revista de Administração Contemporânea (RAC)**, v. 7, n. 2, p. 115-136. Abr./Jun. 2003.

GONÇALVES, A. S. **A Crise do Software.** Disponível em: https://www.portaleducacao.com.br/informatica/artigos/55859/a-crise-do-software Acesso em 17 mai. 2015

GONÇALVES, J. **Metodologia XP (eXtreme Programming) – A Crise do Software.** Disponível em: http://hiperbytes.com.br/artigos/metodologia-xp-extreme-programming-a-crise-do-software/. Acesso em 20 mai. 2015

LACERDA, E. C. **Sistemas de Controle de Versão**. Disponível em: http://www.devmedia.com.br/sistemas-de-controle-de-versao/24574. Acesso em 29 abr. 2015

LÓPEZ Y. J. Notas acerca del concepto y evolución del documento contemporâneo**. JORNADAS CIENTÍFICAS SOBRE DOCUMENTATIÓN CONTEMPORÂNEA, 7 Anais**. Madrid, Departamento de Ciencias y Técnicas Historiográficas, UCM – Universidad Complutense de Madri, 2008, p. 275-282. Disponível em: https://www.ucm.es/data/cont/docs/446-2013-08-22-9%20notas.pdf. Acesso em 29 mar. 2015.

LIMA, G.M.P.S. et. al. **Metodologia para Planejamento, Execução e Controle de Teste de Software**. 2006. Disponível em: < http://webx.sefaz.al.gov.br/posengsoft/documentos/vvt/2-teste_gladys_travassos_planejamento.pdf>. Acesso em 22. maio 2015

NIELSEN, J. Usability 101: **Introduction to Usability**. Disponível em: <http://www.useit.com/alertbox/20030825.html>. Acesso em: 13 mar. 2010.

PASSOS, T. **Instalando e Utilizando o Subversion com VisualSVN Server 2.1.4 e RapidSVN 0.12 no Windows**. 27.nov.2010. Disponível em: <http://blog.tiagopassos.com/2010/11/27/instalando-e-utilizando-o-subversion-com-visualsvn-server-2-1-4-e-rapidsvn-0-12-no-windows/>. Acesso em: 15.out.2015.

PEDAUQUE, R.T. **Documento:** forma, signo y médio, re-formulaciones de lo digital, 2003. Disponível em: http://archivesic.ccsd.cnrs.fr/file/index/docid/62458/filename/sic_00001160.pdf. Acesso em: 29 mar.2015.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Pearson Makron Books, 1997, Reimpressão 2007.

PRESSMAN, R. S. **Engenharia de Software** : 6 ed. São Paulo: McGraw Hill/Nacional, 2006.

OTLET, P. **Traité de documentation:** le livre sur le livre : théorie et pratique. Bruxelles : Mundaneum, 1934. Disponíve em:<http://lib.ugent.be/fulltxt/handle/1854/5612/Traite_de_documentation_ocr.pdf >. Acesso em 01.mar.2015.

Primeiros passos - Noções Básicas de Git. Disponível em: <https://git-scm.com/book/pt-br/v1/Primeiros-passos-No%C3%A7%C3%B5es-B%C3%A1sicas-de-Git>. Acesso em 18.out.2015

RHOADS, J. B. **La Funcion de la gestion de documentos y archivos en los sistemas nacionales de información:** un estudio del Ramp. Paris: UNESCO, 1989. Disponível em: <http://unesdoc.unesco.org/ images/0008/000847/084735so.pdf>. Acesso em: 22 abr. 2015.

RODRIGUES, Y. W. S. **Uma Avaliação da Informação e Gerenciamento Acadêmico**. Universidade Federal de Pernambuco. Recife, 9.jul.2010. Disponível em: <http://www.cin.ufpe.br/~tg/2010-1/ywsr.pdf>. Acesso em 01 out. 2015.

SANCHES, R. Documentação de Software. **Qualidade de Software: Teoria e Prática, Prentice Hall**. São Paulo, 2001. p. 54-59.

SHERA, J. H. Sobre biblioteconomia, documentação e ciência da informação. In: GOMES, H. E. (Org.). **Ciência da informação ou informática?** Rio de Janeiro: Calunga, 1980. p. 91- 105.

SILVA, A.B.M. **Documento e informação: as questões ontológica e epistemológica.** Estudos em homenagem ao Professor Doutor José Marques, p. 327 – 355. 2006. Disponível em: <http://repositorio-aberto.up.pt/bitstream/10216/8742/2/4815.pdf>. Acesso em: 30 mar. 2015.

SILVA, L.G.C. et. al. **Certificação digital:** conceitos e aplicações**.** Rio de Janeiro: Ciência Moderna, 2008.

SIMÕES, A. S. **O que é Subversion (SVN)?**. 2012. Disponível em: < http://www.sorocaba.unesp.br/#!/programas-especiais/pet-eca/pesquisa/o-que-e-svn/>. Acesso em 19.out.2015.

Site GIT. Disponível em: <https://git-scm.com/book/pt-br/v1/Primeiros-passos-No%C3%A7%C3%B5es-B%C3%A1sicas-de-Git>. Acesso em Acesso em 19.out.2015.

SOARES, M. S**. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software**. Universidade Presidente Antônio Carlos (UNIPAC). Disponível em: <https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CB0QFjAA&url=http%3A%2F%2Fjeltex.googlecode.com%2Fsvn%2Ftrunk%2FJeltex%2FMaterial%2520de%2520Leitura%2FUML%2FCompara%25C3%25A7%25C3%25A3o%2520entre%2520metodologias.PDF&ei=8C1jVeSlGMeigwT53oOoAw&usg=AFQjCNHtEXvEI3kI89n2_GiwklUt2TWL5Q&sig2=FgPwyRXeqBH6-FU-m3Ip1A>. Acesso em: 22 mai. 2015.

Sommerville, I. **Engenharia de Software**. Editora Addison-Wesley, 2003. p, 592.

GLOSSÁRIO

A

Apache: servidor da *Web* disponível gratuitamente e que é distribuído sob uma licença "*open source*". É executado na maioria dos sistemas operacionais baseados em *UNIX* (como Linux, Solaris, Digital UNIX e AIX), em outros sistemas derivados do POSIX UNIX (como Rhapsody, BeOS , e BS2000 / OSD) e em Window. De acordo com um levantamento realizado pela *Netcraft* (ww.netcraft.com) 60% dos *sites* na Internet estão usando Apache (mais precisamente 62% incluindo os derivados do Apache), tornando-o o mais utilizado.

Definição extraída de: <http://searchsoa.techtarget.com/definition/Apache>. Acesso em 15.nov.2015

Armazenamento em nuvem: ou "*cloud storage"* são formados por modelo de concentração de dados armazenados *on-line* em sistema virtualizados e especializados em armazenamento (estoque) de dados físicos.

Definição extraída de: <http://cavas.com.br/web-serie/web-serie-o-que-e/armazenamento-em-nuvem-ou-cloud-storage/>. Acesso em 15.nov.2015

B

Backup: atividade de copiar arquivos ou bancos de dados de modo que eles sejam preservados em caso de falha do equipamento ou outra catástrofe. *Backup* é normalmente uma parte da rotina da operação de grandes empresas com *mainframes*, bem como os administradores de computadores de empresas menores.

Definição extraída de: < http://searchstorage.techtarget.com/definition/backup>. Acesso em 15.nov.2015

Booch: "*Design Grady Booch"* orientada a objetos (OOD), também conhecido como Análise Orientada a Objetos e Projeto (OOAD), é um precursor para a Modelagem Unificada (UML). Inclui seis tipos de diagramas: classe, objeto, estado de transição, de interação, de módulos e de processo.

Definição extraída de: <https://www.edrawsoft.com/Booch-OOD.php>. Acesso em 15.nov.2015

Branches: é uma cópia do código derivado de um certo ponto do *trunk (*tronco de desenvolvimento, nele que o projeto deve se basear sempre) que é utilizado para a aplicação de mudanças no código, preservando a integridade do código no *trunk*. Se as mudanças funcionam de acordo com o planejado, elas geralmente são mescladas de volta para o *trunk*. É muito usado para experimentos e para desenvolvimentos paralelos.

Definição extraída de: <http://pt.stackoverflow.com/questions/20989/o-que-branch-tag-e-trunk-realmente-significam>. Acesso em 15.nov.2015

Bugs: trata-se de erro de codificação em um programa de computador.

Definição extraída de: <http://searchsoftwarequality.techtarget.com/definition/bug>. Acesso em 15.nov.2015

C

C++: linguagem de programação " orientada a objeto " criado por Bjarne Stroustrup e lançado em 1985. Ele implementa " abstração de dados ", usando um conceito chamado de "classes", juntamente com outras características para permitir programação orientada a objetos.

Definição extraída de: <http://www.hitmill.com/programming/cpp/whatiscpp.html>. Acesso em 15.nov.2015

Código-fonte: consiste nas instruções de programação que são criados por um programador com um editor de texto ou uma ferramenta de programação visual e, em seguida, salvo em um arquivo.

Definição extraída de: <http://searchsoa.techtarget.com/definition/source-code>. Acesso em 15.nov.2015

D

Delphi: abordagem de programação visual orientada a objetos para desenvolvimento de aplicativos. Com base em objeto linguagem de programação Pascal, a última versão do Delphi inclui instalações para a rápida construção ou conversão de uma aplicação em um serviço *Web*.

Definição extraída de: <http://searchsoa.techtarget.com/definition/Delphi>. Acesso em 15.nov.2015

DOS: O sistema operacional MS-DOS é responsável pela comunicação entre o usuário e o computador, recebendo ordens do usuário e enviando as respostas através do vídeo. Utiliza uma linguagem própria denominada linguagem de controle.

Definição extraída de: <http://www.inf.ufsc.br/~barreto/cca/sisop/msdos.html>. Acesso em 15.nov.2015

F

FDD: "*Feature Driven Development*" (Desenvolvimento Guiado por Funcionalidades), é uma metodologia ágil para gerenciamento e desenvolvimento de *software*. Ela combina as melhores práticas do gerenciamento ágil de projetos com uma abordagem completa para Engenharia de *Software* orientada por objetos.

http://www.heptagon.com.br/fdd

Definição extraída de: <http://www.heptagon.com.br/fdd>. Acesso em 15.nov.2015

I

Interface de programação de aplicativos (API): é um conjunto de instruções e padrões de programação para acesso a um aplicativo de software baseado na Web. Com uma API, os aplicativos se conversam sem intervenção ou conhecimento dos

usuários. Funciona por meio da comunicação entre diversos códigos, interligando diversas funções em um site (por exemplo, notícias, artigos, busca de imagens, etc.) de modo que seja possível utilizá-las em outras aplicações.

Definição extraída de: <http://canaltech.com.br/o-que-e/software/o-que-e-api/>. Acesso em 15.nov.2015

Inputs: conceito da língua inglesa, de uso frequente no âmbito da informática para se referir aos dados entrantes de um processo.

Definição da autora.

J

Java: linguagem de programação e plataforma computacional lançada pela primeira vez pela Sun Microsystems em 1995. Existem muitas aplicações e sites que não funcionam sem o *Java* instalado. O Java é rápido, seguro e confiável. De *laptops* a *datacenters*, *consoles* de games a supercomputadores científicos, telefones celulares à Internet, o Java está em todos os lugares.

Definição extraída de: < https://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em 15.nov.2015

L

Lighttpd: servidor *web* projetado para otimizar ambientes de alta performance. A utilização de memória é baixa se comparada a outros servidores *web*, possui um bom gerenciamento de carga da CPU e opções avançadas como CGI, FastCGI, SCGI, SSL, reescrita de URL, entre outras.

Definição extraída de: < https://www.oficinadanet.com.br/artigo/servidores/lighttpd >. Acesso em 15.nov.2015

Links interwikis: *links* para páginas em outros sites *wiki*, utilizando um estilo de *link* interno prefixado. Links interwiki tornam possível vincular páginas como, por exemplo, a *Wikipedia*.

Definição extraída de: < http://creationwiki.org/pt/Ajuda:Links>. Acesso em 15.nov.2015

N

Nomespaces: Na linguagem XML, é permitido que desenvolvedores definam seus próprios elementos. Por causa disso pode ocorrem de dois ou mais desenvolvedores de linguagens distintas, escolherem nomes iguais para seus elementos. O uso de "namespace" é uma forma de distinguir tais elementos que apesar de possuírem nomes iguais pertencem a vocabulários diferentes.

Definição extraída de: < http://imasters.com.br/artigo/161/dotnet/entendendo-namespaces/>. Acesso em 15.nov.2015

Nginx: servidor *web* e *proxy* reverso, que oferece um excelente desempenho e baixo consumo de memória. Isso faz com que ele seja muito usado por serviços de alta demanda, como o *Wordpress.com*, permitindo que um único servidor atenda facilmente a 10.000 conexões ou mais.

Definição extraída de: < http://www.hardware.com.br/noticias/2012-01/nginx2.html>. Acesso em 15.nov.2015

O

Open Source: refere-se a algo que pode ser modificado e compartilhado porque seu *design* é acessível ao público. Pode ser chamado de como "código aberto".

Definição extraída de: < http://opensource.com/resources/what-open-source>. Acesso em 15.nov.2015

Outputs: conceito da língua inglesa, de uso frequente no âmbito da informática para se referir aos dados resultantes de um processo.

Definição extraída de: < http://conceito.de/output>. Acesso em 15.nov.2015

P

PHP: criado por Rasmus Lerdorf - nascido na Groelândia - Dinamarca não é um Padrão *Web*, é uma tecnologia de código aberto. PHP não é uma linguagem de programação no sentido estrito da palavra, mas sim uma tecnologia que permite a inserção de *scripts* nos seus documentos.

Definição extraída de: < http://opensource.com/resources/what-open-source>. Acesso em 15.nov.2015

S

SCRUM: processo de desenvolvimento iterativo e incremental para gerenciamento de projetos e desenvolvimento ágil de *software*. É utilizado para trabalhos complexos nos quais é impossível predizer tudo o que irá ocorrer.

Definição extraída de: <http://www.brq.com/metodologias-ageis/>. Acesso em 15.nov.2015

T

Tags: em inglês quer dizer “etiqueta”. As *tags* na Internet são palavras que servem justamente como uma etiqueta e ajudam na hora de organizar informações, agrupando aquelas que receberam a mesma marcação, facilitando encontrar outras relacionadas.

Definição extraída de: <http://www.tecmundo.com.br/navegador/2051-o-que-e-tag-.htm>. Acesso em 15.nov.2015

U

UML: acrônimo para a expressão "*Unified Modeling Language*". É uma linguagem que define uma série de artefatos que nos ajuda na tarefa de modelar e documentar os sistemas orientados a objetos que desenvolvemos.

Definição extraída de: <http://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408#ixzz3rrQg39wQ>. Acesso em 15.nov.2015

Upload: ato de enviar dados do servidor para o cliente é chamado de *download*. Já o caminho inverso, quando a máquina do usuário envia algum conteúdo para o "server" na internet, é chamado de *upload*.

Definição extraída de: <http://www.tecmundo.com.br/conexao/1148-o-que-e-upload-.htm>. Acesso em 15.nov.2015

W

Workflow: é a tecnologia que possibilita automatizar processos, racionalizando-os e potencializando-os por meio de dois componentes implícitos: organização e tecnologia. Transforma radicalmente a maneira de toda empresa executar processos, atividades, tarefas, políticas e procedimentos.

Definição extraída de: < http://www.unipbrasiliacst.com.br/pos/AEWorkflow.pdf>. Acesso em 15.nov.2015

X

XP: “*Extreme Programming*” é uma metodologia de desenvolvimento de *software*, criada nos Estados Unidos ao final da década de 90. Vem fazendo sucesso em diversos países, por ajudar a criar sistemas de melhor qualidade, que são produzidos em menos tempo e de forma mais econômica que o habitual.

Definição extraída de: < http://www.desenvolvimentoagil.com.br/xp/>. Acesso em 15.nov.2015.