UNIVERSIDADE FEDERAL DO AMAPÁ
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COORDENAÇÃO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

ERICK DA CUNHA SAMPAIO

UMA ABORDAGEM PARA AJUSTAR O CONTROLE DE VERSÃO À NATUREZA DAS APLICAÇÕES WEB

Macapá
2021

UNIVERSIDADE FEDERAL DO AMAPÁ
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COORDENAÇÃO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

ERICK DA CUNHA SAMPAIO

UMA ABORDAGEM PARA AJUSTAR O CONTROLE DE VERSÃO À NATUREZA DAS APLICAÇÕES WEB

Trabalho de Conclusão de Curso apresentado a Universidade Federal do Amapá como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Julio Cezar Costa Furtado

Macapá
2021

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Central da Universidade Federal do Amapá
Jamile da Conceição da Silva – CRB-2/1010

Sampaio, Erick da Cunha.
S192a Uma abordagem para ajustar o controle de versão à natureza das aplicações web / Erick da Cunha Sampaio. – 2021.
1 recurso eletrônico. 83 f.

Trabalho de conclusão de curso (Graduação em Ciência da Computação) – Campus Marco Zero, Universidade Federal do Amapá, Coordenação do Curso de Ciência da Computação, Macapá, 2021.
Orientador: Professor Doutor Júlio Cezar Costa Furtado

Modo de acesso: World Wide Web.
Formato de arquivo: Portable Document Format (PDF

Inclui referências.

1. Engenharia de Software. 2. Software - Desenvolvimento. 3. Word Wid Web – (Sistema de recuperação da informação) - Aplicações. 4. Web - Controle de versões. I. Furtado, Júlio Cezar Costa, orientador. II. Título.

Classificação Decimal de Dewey, 22 edição, 005.1

SAMPAIO, Erick da Cunha. **Uma abordagem para ajustar o controle de versão à natureza das aplicações web**. Orientador: Júlio Cezar Costa Furtado. 2021. 83 f. Trabalho de conclusão de curso (Graduação em Ciência da Computação) – Campus Marco Zero, Universidade Federal do Amapá, Coordenação do Curso de Ciência da Computação, Macapá, 2021.

UNIVERSIDADE FEDERAL DO AMAPÁ
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COORDENAÇÃO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

ATA DE DEFESA DE DE TCC

Realizou-se no dia 14 de dezembro de 2021, às 15h00, via videoconferência pelo Google Meet, adefesa do TCC intitulado: **"UMA ABORDAGEM PARA AJUSTAR O CONTROLE DE VERSÃO À NATUREZA DAS APLICAÇÕES WEB"**, da discente ERICK DA CUNHA SAMPAIO, matricula 201512200160. A Banca Examinadora foi composta pelo Prof. Dr. JULIO CEZAR COSTA FURTADO, presidente da banca e orientador; Prof. Me. ADOLFO FRANCESCO DE OLIVEIRA COLARES e Prof. Me. SAMUEL SILVA DE OLIVEIRA, examinadores. Concluída a defesa, foram realizadas as arguições e comentários. Em seguida, procedeu-se o julgamento pelos membros da Banca Examinadora, tendo o trabalho sido APROVADO com nota 7,0.

E, para constar, eu, Prof. Dr. JULIO CEZAR COSTA FURTADO, orientador e presidente da Banca Examinadora, lavrei a presente ata que, após lida e achada conforme, foi assinada por mim e demais membros da Banca Examinadora.

Macapá, 14 de dezembro de 2021.

Prof. Dr. JULIO CEZAR COSTA FURTADO
Orientador do TCC

ADOLFO FRANCESCO DE OLIVEIRA COLARES:74382080282
Assinado de forma digital por ADOLFO FRANCESCO DE OLIVEIRA COLARES:74382080282

Prof. Me. ADOLFO FRANCESCO DE OLIVEIRA COLARES
Examinador (UNIFAP)

Prof. Me. SAMUEL SILVA DE OLIVEIRA
Examinador (UNIFAP)

RESUMO

Este trabalho sugere uma abordagem para adequar o controle de versão à natureza das aplicações Web. Essas aplicações possuem uma natureza única e dinâmica, que envolve uma ampla diversidade de itens de configuração (p. ex. código fonte e outros documentos) e conteúdo (p. ex. texto, gráficos, imagens, áudio e vídeo). Por sua vez, o controle de versão é uma atividade imprescindível para o desenvolvimento de sistemas de software, pois estabelece o gerenciamento de múltiplas versões de código fonte e documentos. No entanto, as estratégias gerais do controle de versão precisam ser adaptadas para estar em conformidade com a natureza das aplicações Web. Para atingir esse objetivo, estratégias são estabelecidas a partir do modelo MPS.BR e de características das aplicações Web. Em termos de metodologia, a pesquisa é feita por meio de uma análise qualitativa. Nesse cenário, a pesquisa mostra que, além do uso de estratégias para a seleção e identificação de itens de configuração, o uso de sistemas de controle de versão e o uso complementar do sistema de gerenciamento de conteúdo consiste em uma base estratégica viável para adequar o controle de versão à natureza das aplicações Web.

Palavras-chave: controle de versão, aplicações Web, itens de configuração.

ABSTRACT

This work presents an approach to adjust version control to the nature of Web applications. These applications have a unique and dynamic nature, which involves a wide variety of configuration items (e.g., source code and documents) and Web content (e.g., text, graphics, images, video, audio). Version control is an essential activity to the software systems development, since it establishes the management of multiple versions of source code and documents. However, version control strategies need to be adapted to conform to the nature of Web applications. To achieve this goal, strategies are defined based on the MPS.BR model and characteristics of Web applications. In terms of methodology, the research is done through a qualitative analysis. Basically, the research shows that, in addition to the use of tactics for the selection and identification of configuration items, the use of version control systems and the complementary use of the content management system is a viable strategy to adapt version control to Web applications.

Keywords: version control, Web applications, configuration items.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CMMI – Capability Maturity Model Integration

CSWR – Client-Side Web Refactoring

GC – Gerenciamento de Configuração

GCO – Gerenciamento de Configuração

GCS – Gerenciamento de Configuração de Software

IC – Item de Configuração

IEEE – Institute of Electrical and Eletronics Engineers

MPS.BR – Melhoria de Processo do Software Brasileiro

SCV – Sistema de Controle de Versão

SGC – Sistema de Gerenciamento de Conteúdo

TI – Tecnologia da Informação

SUMÁRIO

1 INTRODUÇÃO

Este capítulo introduz os alicerces que formam o significado deste trabalho. A primeira seção do capítulo esclarece o contexto do trabalho. Por conseguinte, a segunda seção faz um delineamento sobre a motivação, justificativa e contribuição a área. A próxima seção apresenta o objetivo geral e os objetivos específicos do trabalho. Na seção seguinte, a metodologia é explicada. Por fim, a última seção descreve a estrutura do trabalho.

1.1 Contexto do Trabalho

Este trabalho sugere uma abordagem para adequar o controle de versão à natureza das aplicações Web. Em termos gerais, este trabalho é definido na esfera do Gerenciamento de Configuração (GC), que segundo Khan et al. (2019), consiste no processo de controle e gerenciamento de mudanças ao lidar com o desenvolvimento de projetos.

Nesse cenário, o GC envolve um conjunto de atividades, sendo uma delas o controle de versão, a qual este trabalho é direcionado. Como o nome indica, o controle de versão lida com o gerenciamento de múltiplas versões de um projeto, estabelecendo o rastreio de cada mudança realizada, além de fornecer a capacidade para reverter ou desfazer qualquer modificação, conforme posto por Tsitoara (2020).

O controle de versão é uma atividade imprescindível para o desenvolvimento de software, pois a mudança é parte do ciclo de vida do projeto e, lidar com o arranjo de mudanças sem procedimentos efetivos é algo que pode levar os esforços de uma equipe de desenvolvimento à desordem. É importante dizer que este trabalho está concentrado no controle de versão para aplicações baseadas na *World Wide Web*, ou posto de forma simples, aplicações Web. A razão para esse direcionamento é a natureza especial dessas aplicações.

Rio e Abreu (2017), explicam que as aplicações Web são normalmente mais complexas do que outros sistemas de software, já que possuem a parte servidor e a parte cliente, além de envolver uma variedade de linguagens de programação (p. ex. PHP, C#, JavaScript e Java) e linguagens de conteúdo e formatação (p. ex. HTML e CSS). Logo, manter essa base heterogênea de código fonte não é uma tarefa fácil.

Entre as diversas características das aplicações baseadas na Web, Jirapanthong (2015) menciona a natureza dinâmica, grande volume de informação, integração a banco de dados, evolução constante e alta performance.

Com base em características dessa natureza, Dawei et al. (2016) acrescentam que o desenvolvimento de aplicações Web é, de fato, diferente em relação ao de uma aplicação tradicional para desktop e sua demanda por mudanças é especial.

Nesse sentido, Pressman e Maxim (2015) deixam claro que embora as estratégias gerais do controle de versão possam ser utilizadas, táticas e ferramentas devem ser adaptadas para estar em conformidade com a natureza única das aplicações Web. Daí o sentido deste trabalho ao propor o ajuste do controle de versão, de modo a contornar as características das aplicações Web que possam afetar o arranjo de mudanças em projetos dessa natureza. É importante dizer que a abordagem proposta não rejeita os princípios, práticas e ferramentas do controle de versão. Em vez disso, inclui estratégias que moldam o controle de versão às necessidades das aplicações Web.

Para adequar as estratégias gerais do controle de versão à natureza das aplicações Web, este trabalho usa como base o modelo de software MPS.BR. Posto de forma simples, este modelo de software determina os resultados esperados necessários para o gerenciamento de configuração e suas atividades, incluindo o controle de versão. De fato, outros modelos de software como CMMI e IEEE 828 também abrangem o gerenciamento de configuração e o controle de versão, então por que escolher o modelo MPS.BR?

Devido a sua clareza e familiaridade, o modelo MPS.BR estabelece de forma objetiva quais são os resultados esperados para o controle de versão. Esses resultados esperados são essenciais para adequar o controle de versão à natureza das aplicações Web. É importante mencionar que o modelo MPS.BR envolve todas as atividades do gerenciamento de configuração, porém, este trabalho é direcionado apenas para o controle de versão. Dessa forma, somente os resultados esperados que envolvem o controle de versão são utilizados.

Quais são esses resultados esperados? Em seu nível de maturidade F, o modelo MPS.BR define cinco resultados esperados:

- GCO 1 (a partir do nível F): Itens de configuração são identificados e seus níveis de controle são estabelecidos.
- GCO 2 (a partir do nível F): Um sistema para gerencia de configuração e controle de mudanças é estabelecido, mantido atualizado e utilizado.
- GCO 3 (a partir do nível F): Baselines são estabelecidas considerando entregáveis e liberações aos interessados.

- GCO 4 (a partir do nível F): Registros de itens de configuração e de modificações realizadas nestes itens são estabelecidos, mantidos atualizados e utilizados.
- GCO 5 (a partir do nível F): Auditorias de configuração são executadas para avaliar as baselines e o conteúdo do sistema de gerenciamento de configuração.

Apenas os resultados esperados GCO 1 e GCO 2 são utilizados como base, pois estão diretamente atrelados ao controle de versão e isso é delineado no decorrer do trabalho. Os demais resultados esperados envolvem outras atividades do gerenciamento de configuração. A partir disso, para adequar as estratégias gerais do controle de versão à natureza das aplicações Web, este trabalho utiliza os resultados esperados GCO 1 e GCO 2 do modelo MPS.BR e características específicas que formam as aplicações Web, incluindo escalabilidade, arquitetura, natureza assíncrona e distribuída, base diversificada de código fonte, diferentes formas de conteúdo, evolução contínua, segurança e dependências.

1.2 Motivação, Justificativa e Contribuição a Área

As mudanças são parte inevitável do ciclo de desenvolvimento de software e podem ter um alto custo, especialmente ao levar em conta a natureza das aplicações Web. Daí surge o significado deste trabalho ao propor uma adaptação do controle de versão para essas aplicações.

Como foi explicado, as aplicações Web incluem um conjunto de particularidades. Aplicações dessa natureza estão em evolução contínua e o imediatismo por informação Web é algo notável, levando a essas aplicações um célere volume de mudanças. Com efeito, essas aplicações envolvem políticas organizacionais em relação a parte servidor e a parte cliente, além de uma diversidade de tecnologias, incluindo documentos de hipertexto, folhas de estilo, scripts, templates, streaming de mídia e outras que adicionam complexidade ao lidar com mudanças, principalmente quando essas aplicações são colocadas a nível de escalabilidade e necessidade por alta performance.

Nesse sentido, realizar a integração de código fonte e recursos é uma tarefa complexa, especialmente quando muitas mudanças são realizadas por diferentes membros de uma equipe de desenvolvimento e as estratégias gerais do controle de versão não estão devidamente ajustadas a natureza das aplicações Web.

Não raro, a maior parte dos sistemas de software pode ser pensada como um conjunto de versões, de modo que, cada uma delas deve ser mantida e gerenciada, pois é fácil perder o controle de quais mudanças e versões de componentes foram incorporadas em cada versão do

sistema, conforme expõe Sommerville (2016). Com base nisso, controlar adequadamente as versões durante o ciclo de vida de produtos de software é algo essencial, ainda mais quando se trata de aplicações baseadas na Web, já que estas possuem características que podem influenciar o controle de versão.

Pressman e Maxim (2015) mencionam que sem controles efetivos, mudanças impróprias em aplicações Web podem levar a publicação não autorizada de novos produtos, funcionalidades erroneamente testadas que frustram os usuários, falhas de segurança que colocam em risco os sistemas internos da empresa e outras consequências economicamente desagradáveis ou mesmo desastrosas. Em virtude disso, para lidar com as mudanças no desenvolvimento de aplicações Web é necessário estabelecer controles específicos por meio de ferramentas apropriadas.

Para Zolkifli, Ngah e Deraman (2018), sem ferramentas efetivas para o controle de versão, os desenvolvedores de software são estimulados a manter várias cópias duplicadas de código fonte em seus computadores. Segundo os autores, isso é arriscado, pois é fácil modificar ou apagar um arquivo em cópias erradas do código. Logo, essas modificações podem resultar em prejuízos para projeto e perdas relacionadas a tempo e esforços.

Diante dessas características, é necessário adaptar as estratégias gerais do controle de versão para obter maior efetividade ao lidar com mudanças em projetos de aplicações Web. Portanto, fica claro a justificativa deste trabalho ao propor uma abordagem para adequar o controle de versão à natureza dessas aplicações.

Dito isso, este trabalho busca trazer contribuições para a comunidade acadêmica e desenvolvedores de software em relação a forma como o controle de versão é usado em aplicações Web. O controle de versão é uma atividade imprescindível para o desenvolvimento de software, pois constitui suas próprias terminologias, algoritmos, ferramentas e estratégias de utilização. Por outro lado, o desenvolvimento de produtos de software está, cada vez mais, sendo expandido para ser baseado em aplicações Web. Logo, adequar as práticas do controle de versão à natureza das aplicações Web é um procedimento útil para os desenvolvedores e para a engenharia de software.

1.3 Objetivos

Uma vez esclarecido o contexto do trabalho e a sua motivação, justificativa e contribuição à área, é definido a seguir os objetivos necessários para a sua realização.

1.3.1 Objetivo Geral

O objetivo geral deste trabalho consiste na composição de uma abordagem para ajustar o controle de versão à natureza das aplicações Web.

1.3.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho, são definidos os seguintes objetivos específicos:

a) Introduzir o gerenciamento de configuração e a sua atividade denominada controle de versão.
b) Esclarecer o significado atrelado à natureza das aplicações Web;
c) Explicar a razão para adequar as estratégias gerais do controle de versão diante de aplicações Web;
d) Definir quais são os resultados esperados do modelo MPS.BR em que as estratégias são baseadas;
e) Definir quais são os aspectos do controle de versão em que as estratégias são baseadas;
f) Definir quais são as características das aplicações Web em que as estratégias são baseadas;
g) Definir as estratégias;
h) Discutir as estratégias conforme os resultados esperados do modelo MPS.BR;

1.4 Metodologia

Este trabalho estabelece uma pesquisa qualitativa, que segundo Botelho e Cruz (2013) é aquela que busca entender um fenômeno específico em profundidade, por meio de descrições, comparações, interpretações e atribuição de significados possibilitando investigar informações, fatos e ocorrências que não são expressas de forma estatística. Dito isso, a implementação da pesquisa inclui quatro etapas.

A primeira etapa conduz a revisão na literatura sobre o gerenciamento de configuração e a sua atividade denominada controle de versão e, por conseguinte, sobre a natureza das aplicações Web, que também é parte essencial deste trabalho. Nesta etapa, o propósito é delinear de forma minuciosa a fundamentação teórica necessária para estabelecer um nível de esclarecimento sobre os conhecimentos que alicerçam a pesquisa.

A segunda etapa consiste em definir quais são os resultados esperados do modelo MPS.BR em que as estratégias são baseadas. Essa etapa é essencial para definir quais aspectos do controle de versão integram a abordagem.

A terceira etapa consiste em definir quais são os aspectos do controle de versão. Essa etapa é essencial para a definição de estratégias, pois cada aspecto do controle de versão envolve um conjunto específico de estratégias.

A quarta etapa define as características das aplicações Web. Essa etapa contribui para a formação das estratégias.

A quinta etapa é dedicada a definição de estratégias com base nos aspectos do controle de versão e nas características das aplicações Web. Neste ponto, os esforços da pesquisa estão concentrados em averiguar estratégias que estejam alinhadas aos resultados esperados do modelo MPS.BR e, então, adequar essas estratégias à natureza das aplicações Web.

A sexta e última etapa consiste na discussão das estratégias que formam a abordagem proposta. Essa discussão é feita através de uma análise qualitativa.

Para facilitar a visualização da metodologia deste trabalho, a seguir é apresentado uma breve lista que simplifica o conteúdo de cada etapa:

1. Fundamentação teórica;
2. Definição dos resultados esperados do modelo MPS.BR;
3. Definição dos aspectos do controle de versão;
4. Definição das características das aplicações Web;
5. Definição das estratégias;
6. Discussão sobre as estratégias por meio de uma análise qualitativa.

1.5 Estrutura do Trabalho

Ao passo que o primeiro capítulo realiza a introdução por meio de seções, seja para o contexto do trabalho, motivação, justificativa e contribuição a área, etc., os demais também seguem a partir de uma estrutura específica. Dito isso, os parágrafos seguintes descrevem brevemente a estrutura dos próximos capítulos.

O segundo capítulo faz um delineamento sobre a contextualização e terminologias do trabalho. A primeira seção deste capítulo introduz o gerenciamento de configuração e menciona suas atividades e tarefas. Por conseguinte, a segunda seção descreve brevemente os padrões CMMI, IEEE 828 e MPS.BR. O controle de versão é a única atividade do gerenciamento de configuração a receber detalhes neste trabalho. Nesse sentido, o capítulo fornece uma seção

específica para descrever o controle de versão e suas terminologias. O referido capítulo também inclui uma seção dedicada às aplicações Web, descrevendo características como escalabilidade, base heterogênea de código fonte, arquitetura cliente e servidor, etc. Por fim, o capítulo inclui uma seção para delinear os aspectos do controle de versão de forma contextualizada à natureza das aplicações Web.

O terceiro capítulo apresenta os trabalhos relacionados. Para isso o capítulo dispõe uma seção para cada um dos trabalhos. A primeira seção descreve o artigo “Live Versioning of Web Applications through Refactoring” de Grigera et al. (2018). A segunda seção explica a patente de Rudek et al. (2020), intitulada “Distributed Version Control for Tracking Changes in Web Applications”. Por fim, a terceira seção expõe o trabalho de Lunyov (2019), chamado “Detecting Changes in Web Applications”.

O quarto capítulo descreve a abordagem proposta. Para isso é feito a definição de estratégias alinhadas aos resultados esperados do modelo MPS.BR. Essas estratégias são formadas a partir de aspectos do controle de versão e são adaptadas conforme as características das aplicações Web. Assim, a primeira seção do capítulo mostra como essas estratégias são estabelecidas. Nas seções seguintes as estratégias são delineadas.

O quinto capítulo estabelece os resultados e discussão. Esse capítulo discute as estratégias definidas no capítulo anterior em relação aos resultados esperados do modelo MPS.BR e a natureza das aplicações Web.

O último capítulo define a conclusão do trabalho. Neste capítulo são postos os argumentos que concluem o trabalho.

2 CONTEXTUALIZAÇÃO E TERMINOLOGIAS DO TRABALHO

Este capítulo define a fundamentação necessária para o entendimento da abordagem inicialmente proposta. Como ponto de partida é esclarecido o significado do gerenciamento de configuração e apresentado as suas atividades. Em seguida, são postos os principais padrões e modelos para o GC e, então, a sua atividade denominada controle de versão é delineada. Por conseguinte, o capítulo descreve importantes características sobre as aplicações Web. Por fim, são postos alguns dos aspectos do controle de versão de forma contextualizada à natureza das aplicações Web.

2.1 Gerenciamento de Configuração

Como ponto de início, é importante esclarecer que existem diferentes acrônimos para o gerenciamento de configuração, bem como GCS, GCO, GC, etc. Esses acrônimos podem mudar conforme o padrão de software e a literatura. Existem autores que utilizam o acrônimo GCS (Gerenciamento de Configuração de Software) e outros que utilizam simplesmente o acrônimo GC (Gerenciamento de Configuração). Por sua vez, o padrão de software MPS.BR usa o acrônimo GCO para fazer referência ao gerenciamento de configuração.

De modo geral, este trabalho utiliza o termo “gerenciamento de configuração” e o acrônimo “GC” para conduzir o assunto, pois além de grande parte da literatura usar essa terminologia, é uma forma simples e concisa de referência. Dada as disposições precedentes, a seguir é apresentado o conceito de GC.

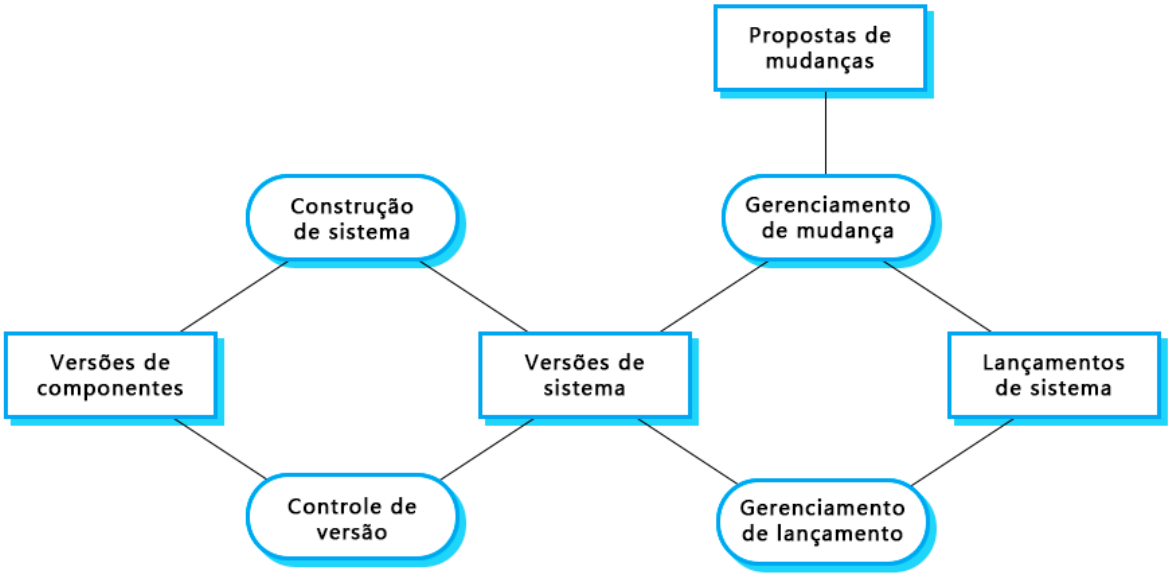
O’Regan (2019), descreve que o gerenciamento de configuração consiste na gestão e controle de mudanças no software e nas entregas de projetos, além de fornecer rastreabilidade completa das mudanças feitas durante o projeto através de um conjunto de procedimentos e ferramentas. De acordo com o autor citado, o GC fornece um registro do que foi modificado, bem como de quem o fez.

Neste ponto é pertinente esclarecer o significado de item de configuração (IC). Segundo Galin (2018), uma unidade de código fonte, um documento etc., que é projetado para o GC e tratado como uma unidade distinta no processo de gerenciamento de configuração é chamado de IC. A partir disso, o termo item de configuração será utilizado neste trabalho, embora haja autores que utilizem outros termos (p. ex. objetos de configuração).

Dito isso, o GC realiza a identificação de itens de configuração (incluindo seus números de versão), controla as mudanças para esses itens e mantém a sua integridade e rastreabilidade, permitindo o desenvolvimento ordenado de software, conforme expõe O'Regan (2019).

Sommerville (2016) explica que o GC envolve quatro atividades estreitamente relacionadas: controle de versão, construção de sistema, gerenciamento de mudança e gerenciamento de lançamento, conforme ilustra a Figura 1.

Figura 1 - Atividades do Gerenciamento de Configuração.

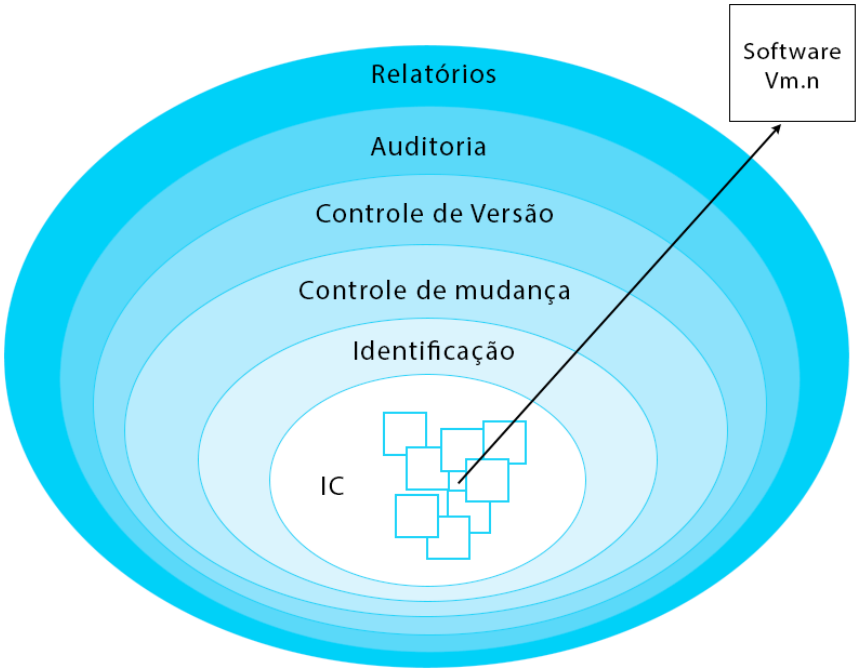


Fonte: Modificado de Sommerville (2016, p. 732).

Como podemos notar na Figura 1, as atividades do gerenciamento de configuração são representadas por retângulos com cantos arredondados. Essas atividades estão relacionadas a versões de componentes, versões de sistema, propostas de mudança e lançamentos de sistema, os quais são representados por retângulos normais. De acordo com Sommerville (2016), as propostas de mudança consistem nas mudanças a serem avaliadas, partindo dos custos e do seu impacto no projeto.

Outra forma de compreender o processo de gerenciamento de configuração é através de tarefas que podem ser visualizadas como camadas concêntricas. Como ilustra a Figura 2, essas tarefas incluem identificação, controle de mudança, controle de versão, auditoria de configuração e relatórios. É possível notar que ao centro estão dispostos os itens de configuração, os quais fluem de uma camada para outra ao longo do processo de gerenciamento de configuração.

Figura 2 - Camadas do Gerenciamento de Configuração.



Fonte: Modificado de Pressman e Maxim (2016, p. 633).

A acepção deste trabalho é voltada para o controle de versão. Portanto, as demais atividades do GC, ainda que estejam relacionadas entre si e tenham a sua devida relevância, não consistem no objeto de estudo a ser investigado, sendo apenas mencionadas por razões de esclarecimento ao contexto do trabalho.

Entre as tarefas do GC, apenas a identificação de itens de configuração é incluída na pesquisa, pois consiste em um aspecto do controle de versão essencial para o trabalho.

2.2 Padrões e Modelos para o Gerenciamento de Configuração

É importante dizer que o GC é implementável de várias formas e isso depende da natureza do projeto. Desse modo, as atividades do gerenciamento de configuração podem ser articuladas de forma ligeiramente confusa e isso pode criar dificuldades para o projeto.

Nesse sentido, há padrões e modelos para auxiliar na estruturação de processos do GC. Laporte e April (2018), pontificam padrões como IEEE 828 e o CMMI (Capability Maturity Model Integration). Outrossim, é importante mencionar o modelo MPS.BR que segundo a Softex estabelece resultados esperados necessários para o gerenciamento de configuração.

De acordo com a IEEE Standards Association (IEEE SA), o padrão IEEE 828-2012 estabelece requisitos para os processos do gerenciamento de configuração em sistemas e na engenharia de software, incluindo a identificação e aquisição dos itens de configuração, controle de mudanças, relatórios acerca do estado desses itens, além de abordar sobre a

construção de software e a engenharia de lançamento. Este padrão também define as áreas de conteúdo para um plano de GC e quais atividades devem ser realizadas, quando estas devem ocorrer no ciclo de vida do software e quais recursos são requeridos.

O CMMI é um modelo criado por profissionais da indústria, governo e do Software Engineering Institute (SEI). Este modelo trata sobre a coleta das características de processos efetivos, de modo a utilizar essa informação para fornecer um guia capaz de aprimorar e amadurecer os processos de uma organização. Assim, a finalidade é facilitar o desenvolvimento de soluções, por meio de melhorias no gerenciamento, aquisição e manutenção dos produtos ou serviços de uma organização, conforme expõe Chaudhary (2017).

Segundo a Associação para Promoção da Excelência do Software Brasileiro (Softex) o modelo MPS.BR tem como base os requisitos de processos definidos nos modelos de melhoria do processo e que atende a necessidade de implantar os princípios provenientes da engenharia de software de forma adequada ao contexto das empresas, estando em conformidade com as principais abordagens internacionais para definição, avaliação e melhoria dos processos de software. Neste modelo, o GC é abordado no nível de maturidade F – Gerenciado, que define o propósito do gerenciamento de configuração, como sendo estabelecer e manter a integridade de todos os produtos de trabalho de um processo ou projeto e, a partir disso, disponibilizá-los a todos os envolvidos.

Embora haja um número de modelos de software relacionados ao GC, é importante dizer que este trabalho estabelece estratégias para o controle de versão que seguem os critérios estabelecidos nos resultados esperados do modelo MPS.BR.

2.3 Controle de Versão

À medida que um projeto evolui, muitas versões de itens de configuração podem existir simultaneamente. Por essa razão, Pressman e Maxim (2016) explicam que o controle de versão combina procedimentos e ferramentas para gerenciar diferentes versões de itens de configuração criados durante o processo de software.

Magana e Muli (2018) reforçam esse entendimento ao descrever o controle versão como uma atividade que oferece suporte ao rastreamento de mudanças para um item de configuração. Em outras palavras, isso significa a possibilidade de reverter as alterações feitas em um item de configuração e registrar as mudanças introduzidas em uma base de código fonte.

Segundo Wazlawick (2013), a versão de um item de configuração é um estado particular desse item durante o desenvolvimento de um sistema de software e, normalmente é identificada

por um número. De acordo com o autor citado, o controle de versão mantém o arranjo sobre o trabalho paralelo de vários desenvolvedores através das seguintes funcionalidades:

1. Manter e disponibilizar cada versão já produzida de IC;
2. Ter mecanismos para disponibilizar diferentes ramos (branches) de desenvolvimento;
3. Estabelecer uma política de sincronização de mudanças que evite esforços desnecessários;
4. Fornecer um registro de mudanças para cada item de configuração.

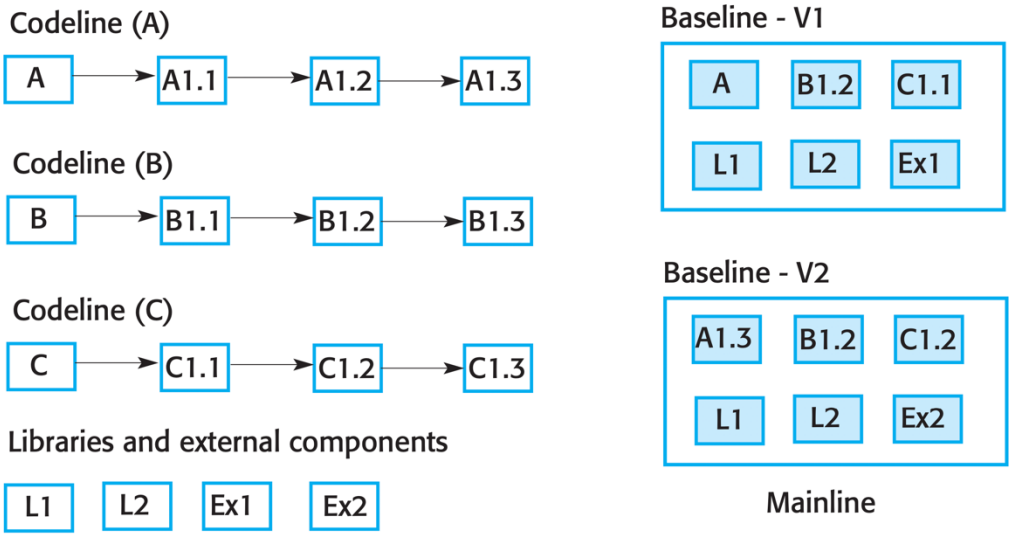
Como notado por Del Sole (2017), o controle de versão basicamente cria um histórico para cada arquivo de forma que os desenvolvedores possam obter versões específicas posteriormente. Segundo o autor, o controle de versão não é limitado a arquivos de código fonte, porém esse é provavelmente o seu uso mais comum.

Sommerville (2016) ressalta que o controle de versão envolve o processo de gerenciar codelines e baselines, sendo que:

1. Codeline é a uma sequência de versões de um componente de software e outros itens de configuração de que esse componente depende;
2. Baseline é uma definição específica de um sistema, ou seja, uma coleção de versões de componentes que fazem um sistema;

Para tornar nítido esses conceitos a Figura 3 ilustra as codelines e baselines, além da mainline, que segundo Sommerville (2016), consiste em uma sequência de baselines representando diferentes versões de um sistema.

Figura 3 - Codelines e baselines.



Fonte: Modificado de Sommerville (2016, p. 736).

Wazlawick (2013), menciona que todos os arquivos referentes aos itens de configuração ficam em um local chamado repositório, sob a guarda do Sistema de Controle de Versão (SCV). Diante do que expõe o autor, o repositório pode ser entendido como um local (diretório) em que as diferentes versões de cada item são mantidas e identificadas.

Jyani e Bawar (2020), descrevem o sistema de controle de versão como uma importante ferramenta na área do desenvolvimento de software que é utilizada, principalmente, para rastrear e gerenciar mudanças em arquivos do projeto (especialmente código fonte). A saber, Vemula (2017), esclarece que muitos SCV estão disponíveis e os mais populares são: Git, Subversion e Azure DevOps Server (anteriormente conhecido como Team Foundation Server).

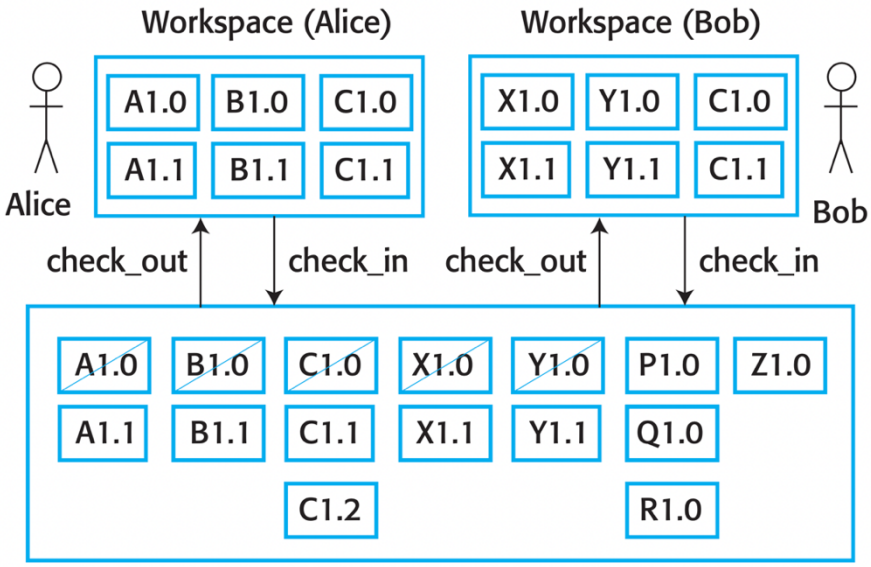
De acordo com Neelakantam e Pant (2017), o sistema de controle de versão é útil para os desenvolvedores e projetistas, pois permite salvar e armazenar cada versão distinta de seus arquivos (itens de configuração). Conforme descrevem os autores, nenhum projeto estará completamente perdido devido a erros, esquecimentos ou travamentos, pois através do SCV é possível retornar a qualquer uma das versões anteriores caso haja um problema na versão em desenvolvimento.

Zolkifli, Ngah e Deraman (2018), explicam que através do SCV desenvolvedores podem trabalhar em qualquer arquivo e a qualquer momento sem sobrepor o trabalho uns dos outros. De acordo com os autores, se dois desenvolvedores, por exemplo, realizem mudanças no mesmo arquivo, o SCV irá mesclar (merge) ou avisá-los de que alguns dos códigos são conflitantes.

Uma vez posto o significado de SCV, Deepa et al. (2020) acrescentam que existem dois tipos de arquiteturas para essas ferramentas:

3 Sistemas centralizados: dependem de um único repositório para armazenar a maior parte dos dados do projeto, bem como os dados de controle de versão;
4 Sistemas distribuídos: colocam uma cópia de todo o repositório no computador de cada usuário, o que facilita o trabalho independente e offline.

Figura 4 - SCV centralizado.



Fonte: Modificado de Sommerville (2016, p. 736).

A Figura 4 mostra o funcionamento de um SCV centralizado e a utilização de workspaces. Neste ponto é essencial esclarecer o significado de workspaces para a engenharia de software. Segundo Pressman e Maxim (2015), workspaces são áreas de armazenamento que engenheiros de software e desenvolvedores usam para criar, modificar, testar e integrar código fonte e outros itens de configuração.

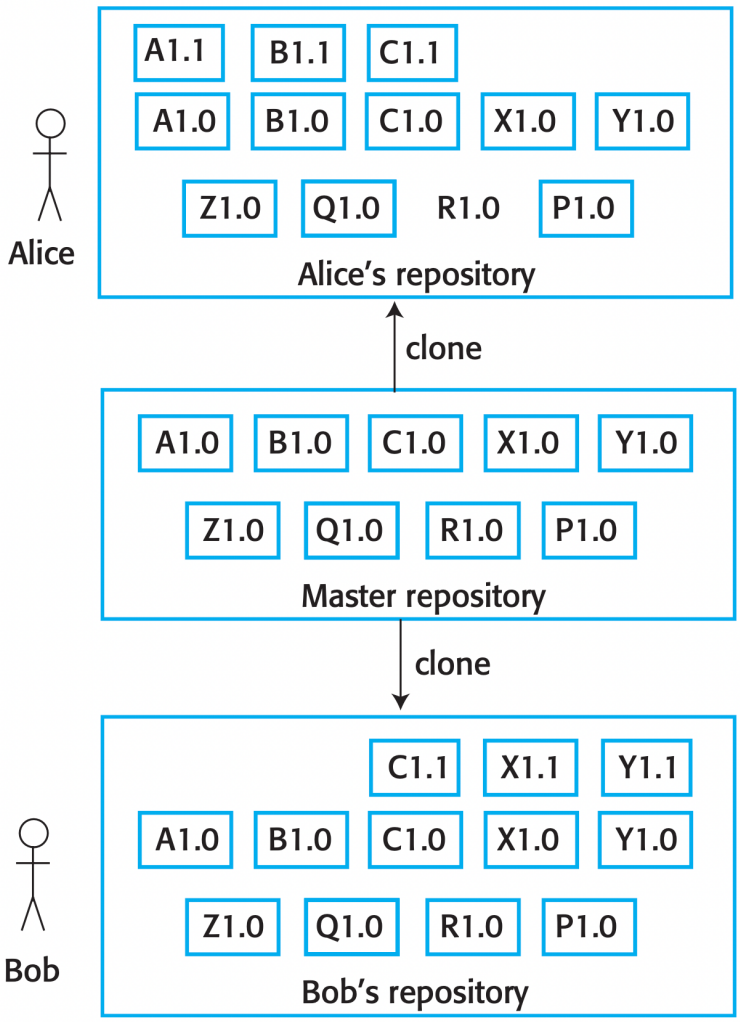
Nesse cenário, os desenvolvedores Alice e Bob copiam (check-out) apenas os componentes que serão modificados para os seus respectivos workspaces. Ao completar suas modificações, eles retornam (check-in) os componentes para o repositório do projeto e, isso cria uma nova versão para esses itens de configuração.

Logo, um SCV centralizado funciona basicamente armazenando as mudanças em um único servidor no qual os clientes (desenvolvedores) possam se conectar. Porém, Tsitoara (2020) explica que o principal problema de um SCV centralizado é que um erro no servidor pode custar todo o trabalho da equipe de desenvolvimento, além de requerer conexão à rede, já que o repositório do projeto é mantido em um servidor remoto.

Por essa razão, há outro modelo de arquitetura para os sistemas de controle de versão. Diante do que expõe Tsitoara (2020), um SCV distribuído é geralmente mais rápido que outros tipos de SCV porque não precisa de acesso a rede para um servidor remoto. Em outras palavras, quase tudo é feito localmente. Segundo o autor, um SCV distribuído não possui um servidor principal que mantém todo o histórico de mudança. Em vez disso, cada cliente possui um clone do repositório, conforme ilustra a Figura 5. Nesse exemplo, os desenvolvedores Alice e Bob

criam um clone do repositório em seus computadores e, quando finalizam suas alterações, eles confirmam (commit) essas mudanças e atualizam seu repositório de servidor privado.

Figura 5 - SCV distribuído.



Fonte: Modificado de Sommerville (2016, p. 738).

De acordo com Xu et al. (2019), um SCV distribuído pode tornar a cooperação entre os membros de uma equipe de desenvolvimento mais flexível, já que cada desenvolvedor tem um repositório completo e modifica os dados localmente. Segundo os autores, isso significa que cada membro do projeto é livre para revisar os dados do projeto de modo privado e escolher o conjunto de mudanças a ser publicada em seu próprio repositório de servidor público com base em qualquer versão compartilhada.

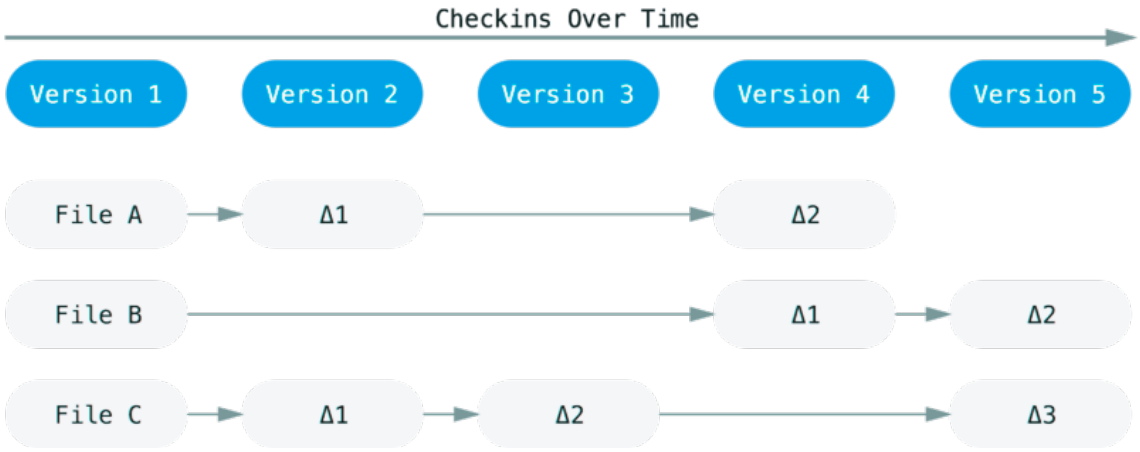
Li, Tsukiji e Takano (2016), mencionam que através do uso generalizado de sistemas de controle de versão distribuídos, muitos projetos estão sendo compartilhados e gerenciados em um ambiente Web. Para os autores, isso permite que muitos desenvolvedores, incluindo líderes e apoiadores de projetos, os quais escrevem funções importantes, solucionam erros e

fornecem código-fonte fundamental para aliviar gargalos, possam avançar de forma colaborativa em seus projetos.

Desse modo, o controle de versão distribuído é altamente adequado e amplamente usado para o desenvolvimento de projetos de código aberto (open source), pois a comunidade envolvida é caracterizada por grandes equipes de desenvolvimento que trabalham de forma colaborativa em projetos de larga escala e que estão geograficamente distribuídos em todo o mundo, conforme expõe Jyani e Barwar (2020).

Ao fazer uso das práticas e ferramentas do controle de versão é essencial conhecer quais são as principais formas utilizadas para o armazenamento de arquivos e versões. Conceitualmente, Chacon e Straub (2014), esclarecem que a maior parte dos sistemas de controle de versão armazenam a informação como uma lista de arquivos baseados em mudanças (isso é comumente descrito como controle de versão baseado em deltas). De acordo com Kiatphao e Suwannasart (2017), o método delta mantém somente uma versão completa e recria as outras versões a partir das diferenças entre cada versão. A Figura 6 ilustra o controle de versão baseado em deltas ao longo do tempo.

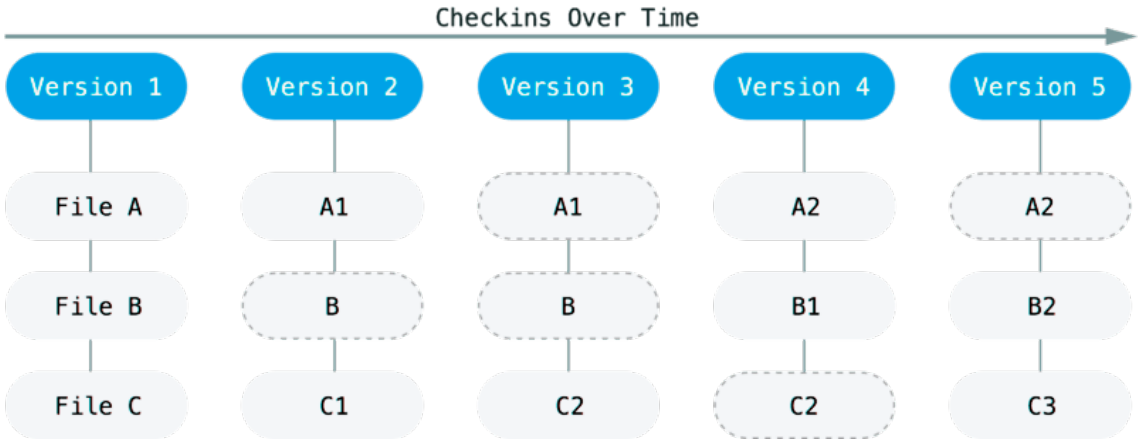
Figura 6 - Armazenamento de dados baseado em deltas.



Fonte: Modificado de Chacon e Straub (2014, p. 14).

Entretanto, Chacon e Straub (2014), discorrem que o Git lida com os seus dados de forma diferente, pois a cada vez que o desenvolvedor envia, ou salva o estado de seu projeto, o Git basicamente captura uma imagem (snapshot) de como todos os seus arquivos parecem naquele momento e armazena uma referência para esse snapshot. Diante do que expõe os autores, o Git pensa em seus dados mais como um fluxo de snapshots. Para esclarecer a ideia geral dessa solução, a Figura 7 mostra o armazenamento de dados como snapshots.

Figura 7 - Armazenamento de dados como snapshots do projeto ao longo do tempo.



Fonte: Modificado de Chacon e Straub (2014, p. 14).

Através da Figura 7 é possível notar que se os arquivos não tiverem mudado ao criar uma nova versão do projeto, o Git não armazena novamente o arquivo. Em vez disso, o Git cria apenas um link para o arquivo que já está armazenado em uma versão anterior (isso é indicado pelas linhas tracejadas).

2.4 Aplicações Web

Segundo Yadav et al. (2018), aplicações Web são sites ativos, compostos de programas baseados em servidor que fornecem a interação com o usuário e várias outras funcionalidades. Este trabalho está concentrado em determinadas características das aplicações Web que podem influenciar as mudanças e a percepção do controle de versão em um projeto, as quais são delineadas a seguir.

Como ponto de partida, Bukhari et al. (2018) esclarecem que as aplicações baseadas na Web são desenvolvidas de uma maneira diferente, à medida que envolvem diversos stakeholders, e variam em seu tamanho e na ideia a ser implementada. Segundo os autores, devido as aplicações Web possuírem uma natureza diferente em relação a aplicações convencionais, os modelos tradicionais de desenvolvimento de software não se adequam completamente ao desenvolvimento de tais aplicações.

Em projetos de aplicações Web os requisitos são alterados rapidamente e essas mudanças influenciam nas relações entre os itens de configuração, conforme descrito por Jirapanthong (2015).

De acordo com Mesbah (2016), as aplicações Web possuem tecnologias heterogêneas com interdependências dinâmicas entre elas, além de uma natureza assíncrona e distribuída que

tornam o desenvolvimento, teste e manutenção dessas aplicações um esforço notório para os membros de uma equipe de desenvolvimento.

Por sua vez, Rio e Abreu (2017) afirmam que a complexidade das aplicações Web em relação a outros projetos de desenvolvimento de software se dá porque essas aplicações envolvem a parte servidor (que pode executar em um servidor remoto ou servidor na nuvem) e a parte cliente (que é exibida e executada em um navegador). Outrossim, Rio e Abreu (2017) explicam que essa complexidade também é decorrente da diversidade de linguagens de programação que as aplicações Web abrangem (p. ex. PHP, C#, JavaScript e Java) e linguagens de conteúdo e formatação (p. ex. HTML e CSS).

A partir disso, Aniche (2015) ressalta que não é simples lidar com essa base heterogênea de código fonte das aplicações Web, uma vez que os desenvolvedores precisam integrar diferentes tecnologias e lidar com as suas interdependências. Não obstante, Horcas et al. (2018) explicam que devido a essa natureza multilíngue, as aplicações Web estão sujeitas a um alto grau de variabilidade refinada que geralmente afeta a maioria dos itens de configuração.

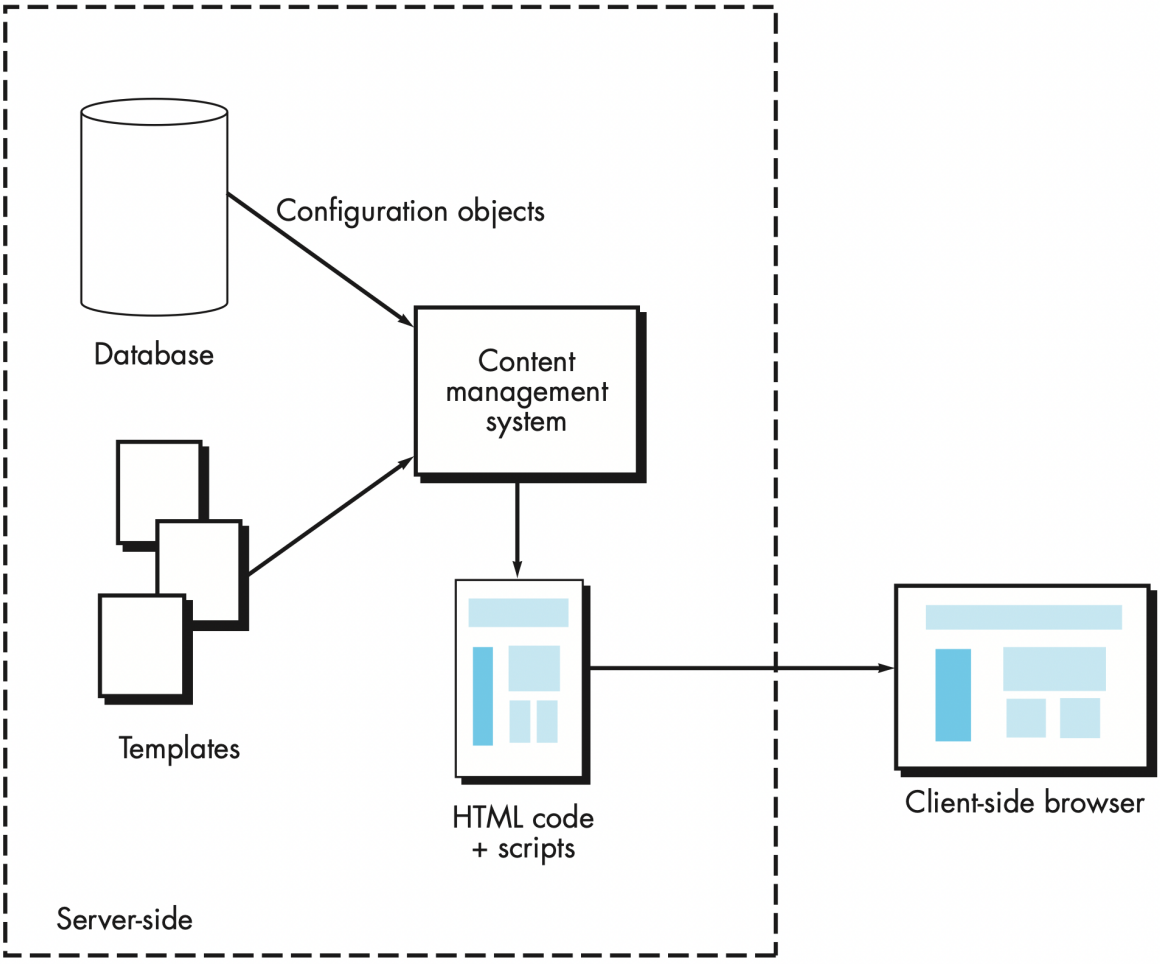
Diversidade e quantidade de conteúdo são características determinantes para aplicações Web. Nesse sentido, Harwani (2015) afirma que as aplicações Web possuem não apenas diversas formas de conteúdo, mas também uma grande quantidade. Segundo o autor, quanto maior a quantidade de conteúdo, mais difícil é manter uma aplicação Web. Esse conteúdo inclui texto, vídeo, áudio, gráficos imagens e outras formas de conteúdo da web.

Posto isso, Jirapanthong (2015) ressalta importantes características das aplicações baseadas na Web, dentre as quais vale mencionar:

a) Natureza dinâmica, isto é, a informação muda rapidamente conforme o tempo e as necessidades do usuário;
b) Grande volume de informação;
c) Integração a banco de dados e sistemas de rastreamento;
d) Equipe de desenvolvimento com especialidades diversificadas;
e) Evolução constante e alta performance;
f) Necessidades de metodologias e processos de desenvolvimento promissores.

Diante disso, Pressman e Maxim (2015) explicam que as aplicações Web criam páginas dinamicamente. Primeiro, o usuário consulta a aplicação solicitando uma informação específica. Por conseguinte, a aplicação consulta o banco de dados do servidor, formata a informação adequadamente e, então apresenta a página ao usuário. Em suma, a página resultante é construída no lado servidor e passada para o lado cliente para ser visualizada. Esse procedimento é ilustrado na Figura 8.

Figura 8 - Representação genérica para uma aplicação Web.



Fonte: Modificado de Pressman e Maxim (2016, p. 644).

Levando em conta aspectos de segurança, Laverdière e Merlo (2018), explicam que as aplicações baseadas na Web regularmente passam por manutenção e evolução, de modo que a sua segurança pode ser afetada por mudanças no código fonte entre os lançamentos dessas aplicações. Nesse cenário, os autores ressaltam que os desenvolvedores precisam de processos de garantia de qualidade para prevenir falhas de segurança e garantir que as modificações no código fonte tenham apenas um impacto planejado e desejável na segurança. Daí a o significado do GC e do controle de versão para gerenciar essas mudanças.

2.5 Aspectos do Controle de Versão e Aplicações Web

Esta seção descreve alguns dos principais aspectos do controle de versão de forma contextualizada à natureza das aplicações Web, os quais incluem:

A. Seleção de itens de configuração;

B. Identificação de itens de configuração;

C. Escolha do SCV;
D. Escolha do SGC.

As seções seguintes descrevem cada um desses aspectos do controle de versão.

A. Seleção de Itens de Configuração

Esta seção descreve a seleção de itens de configuração. Posto de forma simples, a seleção de itens de configuração consiste em definir quais itens devem estar sobre a guarda do controle de versão. Para realizar a seleção de itens de configuração é preciso conhecer quais são os tipos de itens de configuração. Galin (2018), menciona que os tipos mais comuns de itens de configuração são:

1. Documentos de projeto:
   a) Documento de requisitos de software;
   b) Especificações de design da interface;
   c) Descrição do banco de dados;
   d) Documento de descrição de versão.
2. Código de software:
   a) Código fonte;
   b) Código objeto;
   c) Protótipo de projeto.
3. Arquivos de dados (data files):
   a) Casos de teste e scripts de teste;
   b) Parâmetros.
4. Ferramentas de desenvolvimento de software:
   a) Compiladores e depuradores;
   b) Geradores de aplicação.

Em projetos de aplicações Web, o código de software é uma categoria amplamente diversificada e requer atenção especial. Horcas et al. (2018), ressaltam que o desenvolvimento dessas aplicações envolve o gerenciamento de uma grande diversidade de arquivos e recursos, como código, páginas ou folhas de estilo, implementadas em diferentes linguagens. De acordo com os autores, a natureza multilíngue dessas aplicações inclui diversos tipos de código fonte (p. ex. Java, Groovy e JavaScript), templates (p. ex. HTML, Markdown e XML), folhas de estilo (p. ex. CSS e suas variantes, assim como SCSS) e outros tipos de arquivos (p. ex. JSON,

YML e shell script). Como resultado, os autores explicam que essas aplicações produzem variabilidades que geralmente afetam grande parte de seus itens de configuração.

Além disso, Paez (2018) explica que em algumas situações é comum obter dados relacionados à configuração de variáveis de ambiente em tempo de execução. Segundo o autor, esses valores devem estar sobre o controle de versão. Logo, também é importante reconhecer esses dados ao realizar a seleção de itens de configuração em projetos de aplicações Web.

Devido a essa base diversificada de código fonte, é fundamental selecionar cuidadosamente os artefatos de código a serem incluídos como itens de configuração. Para isso, Galin (2018) menciona que o principal critério para a classificação e inclusão de um item em um sistema de controle de versão, é o seu potencial de contribuição para o desenvolvimento posterior da aplicação e do processo de manutenção.

Laporte e April (2018) explicam que o gerente do projeto e sua equipe de desenvolvimento devem decidir quais itens devem estar sobre o controle de versão. Segundo os autores, assim que um item de configuração é escolhido, ele estará sujeito a revisão formal e aceitação por um número de indivíduos. Diante do que expõe os autores, escolher a quantidade apropriada de itens de configuração é uma tarefa fundamental, pois revisões formais exigem que um processo seja seguido e que os defeitos sejam rastreados, corrigidos e verificados. Com base nesses pontos, Laporte e April (2018), indicam que os seguintes critérios podem ajudar nessa decisão:

- Número de mudanças previstas;
- Complexidade e dimensões;
- Segurança;
- Impacto no cronograma de desenvolvimento;
- Impacto no cronograma da implementação;
- Itens comerciais que não são modificados;
- Itens fornecidos pelo cliente;
- Manutenção realizada por diferentes fornecedores;
- Locação – quando componentes são desenvolvidos em muitos sites;
- Uso múltiplo – quando um componente é usado por muitas aplicações.

Esses critérios são relevantes ao estabelecer a seleção de itens de configuração em projetos de aplicações Web. Nesse cenário, a seleção de itens de configuração envolve não apenas quais itens devem estar sobre o controle de versão, mas também a quantidade a ser definida, pois como notado por Pressman e Maxim (2016) à medida que uma aplicação Web

evolui muitas versões podem existir simultaneamente. Logo, sabendo que os itens de configuração estão sujeitos a revisões formais que podem consumir tempo e esforços, é preciso traçar uma seleção que observe os critérios mencionados.

B. Identificação de Itens de Configuração

Esta seção faz um delineamento sobre a identificação de itens de configuração. Para controlar adequadamente os itens de configuração é necessário que cada item seja unicamente identificado. O' Regan (2019), explica que isso é feito adotando uma convenção de nomenclatura para o código fonte e outros itens do projeto. De acordo com o autor, uma abordagem simples é o uso de rótulos mnemônicos e números de versão para identificar unicamente os itens de configuração. Por exemplo, uma especificação de requisitos de usuário para o projeto 005 na área de Finanças pode ser representada por:

FIN_005_URS

Diante do que expõe O'Regan (2019), o rótulo FIN corresponde a área de finanças, o número 005 indica o número de versão do projeto e o rótulo URS (User Requirements Specification), corresponde a especificação de requisitos do usuário. No entanto, os identificadores criados por sistemas de controle de versão normalmente não atribuem essa semântica aos itens de configuração.

Nesse caso, Paez (2018) explica que cada sistema de controle de versão atribui um identificador para cada versão, mas esse identificador não possui significado associado. Segundo o autor, em algumas ferramentas como Subversion, cada versão é identificada com um número sequencial que permite reconhecer a ordem das versões, mas alguns outros sistemas como o Git usam um hash para identificar as versões. Para agregar algum significado adicional, o autor sugere o uso regras de controle semântico para nomear versões ou tags. Com base nisso, o autor explica que o controle de versões semântico propõe o uso de três números para identificar cada versão, onde cada número possui um significado específico:

- Maior: esse número deve ser incrementado quando a nova versão incluir mudanças incompatíveis com as anteriores;
- Menor: esse número deve ser incrementado quando a nova versão adiciona funcionalidades de maneira compatível com versões anteriores;
- Patch: esse número deve ser incrementado quando a nova versão inclui correções de bug compatíveis com versões anteriores.

A partir desse modelo, cada versão possui a forma: “Maior.Menor.Patch”, conforme expõe Paez (2018). É importante ressaltar que diferentes modelos podem ser criados para estabelecer uma convenção de nomenclatura capaz de atribuir significado útil para as versões de itens de configuração. Por essa razão, é preciso entender as regras de negócio definidas para uma aplicação Web.

Gupta e Dhir (2016) afirmam que as aplicações Web fornecem produtividade, concorrência e rápido acesso as demandas online para atingir objetivos específicos. Logo, é interessante criar estratégias para lidar com as regras de negócio dessas aplicações. Outrossim, é importante pensar em soluções de nomenclatura para os múltiplos tipos de itens de configuração que as aplicações Web envolvem.

Como sugestão, Tsui, Karam e Bernal (2018) consideram os seguintes pontos ao criar um modelo:

- Um nome único de produto de software;
- Múltiplos tipos de artefatos (p. ex. documentos de projeto, código fonte, código executável, scripts de teste, dados de teste);
- Diversos formatos de artefatos (p. ex. texto, trechos de áudio, imagens);
- Um esquema que requer pelo menos dois níveis de agrupamento. O nível de versão de software é o fator primário. Dentro de um nível de lançamento, pode haver vários níveis de versões;
- Versões de software traduzidos para vários países.

De acordo com Tsui, Karam e Bernal (2018), a partir dessas suposições é possível formar o seguinte modelo:

PP. CC. RRR. VVV. TT. FF

Onde:

- PP (duas posições): é o código de produto do software (Product). P é alfanumérico, então há 36 caracteres para cada posição. Logo, PP corresponde a 36^2 possibilidades de código de produto;
- CC (duas posições): representa o código do país (Country). Neste caso, permitido 26 caracteres alfabéticos para cada posição. Assim, as duas posições permitirão 26^2, ou 676 países;
- RRR (três posições): representa o número de lançamento do software (Release). Assumindo apenas números de 0 a 9, RRR permite 10^3, isto é, 1000 lançamentos. Segundo Tsui, Karam e Bernal (2018), considerando que um produto possui dois

lançamentos por ano, três posições (RRR) permite manter uma aplicação por 500 anos;

- VVV (três posições) representa o controle versões em cada lançamento (Versioning). VVV é reiniciado a cada lançamento. Definindo, uma solução numérica de 0 a 9, VVV é capaz de fornecer 1000 versões dentro de cada lançamento;
- TT (duas posições): Descreve o tipo de artefato (Type). Pode ser um documento de requisitos, tabelas de banco de dados, código fonte, dados de teste etc. Neste caso, T pode ser alfabético, assim fornece 26 caracteres. Logo, as duas posições TT fornecem 26^2, ou 676 tipos diferentes de artefatos;
- FF (duas posições): Representa o formato do item (Form). O item de configuração pode estar em um formato de documento, código executável ou de imagem jpeg. Considerando que a posição F pode ser um número de 0 a 9, FF fornece 100 tipos diferentes de formatos.

O modelo de nomenclatura acima pode mudar conforme as combinações definidas. Por exemplo, é possível incluir caracteres maiúsculos e minúsculos, mudar os campos puramente numéricos para alfanumérico e ainda aumentar o número de posições para expandir o alcance desse modelo. Essas possibilidades são importantes ao considerar os múltiplos tipos de itens de configuração que as aplicações Web possuem.

C. Escolha do Sistema de Controle de Versão

Esta seção descreve a escolha do sistema de controle de versão. Galin (2018), explica que os SCV diferem em seu nível de compreensão, modelo estrutural de seu repositório, flexibilidade da aplicação e usabilidade.

Em relação ao modelo estrutural de repositório, Adam e Károly (2020) ressaltam que os SCV atuais adotam uma de duas arquiteturas, a centralizada ou a distribuída. De fato, é preciso estar ciente das diferenças entre um SCV centralizado e distribuído para escolher a ferramenta mais adequada as necessidades do projeto. Essas diferenças são explicadas na seção 2.3 deste trabalho. Relembrando alguns pontos, Tsitoara (2020) menciona que o principal problema de um SCV centralizado é que um erro no servidor pode prejudicar todo o trabalho da equipe de desenvolvimento, além de requerer conexão à rede, uma vez que o repositório do projeto é mantido em um servidor remoto. No caso do SCV distribuído, seu funcionamento não depende de um servidor central para manter todo o histórico de mudança, pois cada desenvolvedor

possui um clone do repositório, além de ser mais rápido do que outros tipos de SCV, já que não precisa de acesso à rede para um servidor remoto.

Como notado por Adam e Károly (2020), SCV modernos suportam o desenvolvimento paralelo de itens de configuração usando ramificações e mesclagens (branches e merges). Em virtude disso, os SCV utilizam diferentes técnicas para possibilitar o controle de versões de IC e proporcionar o desenvolvimento colaborativo. Na seção 2.3, foram delineadas as principais técnicas usadas no controle versão (deltas e snapshots). Relembrando, Kiatphao e Suwannasart (2017) explicam que o método "delta" mantém somente uma versão completa e recria as outras versões a partir das diferenças entre cada versão. Em relação a técnica de "snapshots" do Git, Chacon e Straub (2014) explicam que a cada vez que o desenvolvedor envia, ou salva o estado de seu projeto, o Git basicamente captura uma imagem (snapshot) de como todos os seus arquivos parecem naquele momento e armazena uma referência para esse snapshot. Nesse cenário, saber qual é a técnica de controle de versão usada pela ferramenta é um fator a considerar na escolha do SCV, pois como explicado cada método possui suas peculiaridades.

Conforme já mencionado na seção 2.3, há uma diversidade de SCV disponíveis. Logo, é preciso conhecer quais são as principais ferramentas e as suas características para estabelecer uma escolha adequada a natureza do projeto. De acordo com Uzunbayir e Kurtel (2018), alguns dos principais SCV são Apache Subversion (SVN), IBM Rational Synergy, Azure DevOps Server (anteriormente chamado de TFS), Mercurial, BitKeeper e Git. As ferramentas mencionadas são classificadas por tipo de SCV:

1) SCV centralizado:
    a) Apache Subversion (SVN);
    b) IBM Rational Synergy;
    c) Azure DevOps Server.
2) SCV distribuído:
    a) Mercurial;
    b) BitKeeper;
    c) Git.

De modo geral, os SCV são ferramentas de linha de comando, isto é, são utilizados por meio do terminal, através de comandos específicos. Embora grande parte dos SCV não possua uma interface gráfica de usuário, tais sistemas podem ser incluídos em ambientes de desenvolvimento integrado para obter maior produtividade. Por essa razão, este trabalho não inclui figuras para ilustrar os SCV mencionados. Nos tópicos seguintes cada uma dessas ferramentas é brevemente apresentada:

- Subversion: Deepa (2020), explica que o Subversion é um SCV centralizado de código aberto que está sobre a licença Apache. Segundo o autor, a ferramenta fornece mecanismos de ramificação e marcação para suportar workflows não lineares, além de possuir o recurso de rastreamento de mesclagem e a capacidade de gerenciar listas de mudanças para organizar commits (isto é, envios) em grupos específicos.
- IBM Rational Synergy: É um SCV centralizado que segundo Uzunbayir e Kurtel (2018), fornece o gerenciamento de código fonte para todos os artefatos no processo de desenvolvimento (p. ex. documentos, imagens e bibliotecas), além de permitir o gerenciamento de mudanças através do software Rational Change. Conforme expõe o autor, ambas as ferramentas se complementam para fornecer um ciclo de vida de gerenciamento de código fonte estável.
- Azure DevOps Server: Arora e Shigihalli (2019), explicam que o Azure DevOps Server (conhecido anteriormente como Team Foundation Server) é um conjunto de ferramentas que pode fazer parte do seu ambiente de desenvolvimento integrado. Com o Azure DevOps Server, membros de uma equipe podem trabalhar de maneira eficaz em projetos de todos os tamanhos, criar versões para seu código fonte, enquanto obtém rastreabilidade completa e visibilidade de suas atividades de desenvolvimento.
- Mercurial: Adam e Károly (2020), esclarecem que o Mercurial fornece funcionalidades de um sistema de controle de versão distribuído e possui como características fácil uso e boa documentação, além de utilizar a linguagem de programação Python. No entanto, os autores mencionam que a instalação do Mercurial pode ser um pouco complicada devido a dependências de outros pacotes e módulos.
- BitKeeper: É um SCV distribuído que oferece suporte para grandes projetos e possui recursos para renomear arquivos e diretórios, além de ser rápido e facilitar a realização de funções como commits e mesclagens, conforme expõe Deepa (2020). De acordo com o autor, todas as mudanças no BitKeeper são snapshots e existe um suporte offline verdadeiro, de modo que, os desenvolvedores podem trabalhar ponto a ponto sem envolver uma cópia mestre. Outro recurso mencionado pelo autor é que o BitKeeper automaticamente verifica a integridade do sistema para alertar o usuário.
- Git: De acordo com Tsitoara (2020), o Git é um SCV distribuído que funciona muito bem para o rastreamento de mudanças, pois fornece recursos como: ir e voltar entre

> versões, analisar as diferenças entre essas versões, verificar o histórico de mudança de um arquivo e marcar uma versão específica. Além disso, o autor explica que o Git também é uma ótima ferramenta para o trabalho em equipe, pois oferece suporte a troca de um conjunto de mudanças entre repositórios e a revisão de mudanças feitas por outros desenvolvedores.

Embora possuam funcionalidades importantes como o rastreamento de mudanças e recursos para ramificação, Tsui, Karam e Bernal (2018) afirmam que não há uma única ferramenta que possa gerenciar todos os diferentes itens de configuração e suas interdependências, e pode ser necessário o uso de muitas ferramentas em conjunto.

Levando em consideração a natureza das aplicações Web é importante notar que dependendo do contexto, itens de configuração podem requerer atenção especial. Com base nisso, Paez (2018) explica que é preciso definir se todos os itens de configuração devem ser armazenados com a mesma ferramenta, incluindo código fonte da aplicação, código fonte da infraestrutura, arquivos de configuração e arquivos de entrega.

Em relação à segurança, Paez (2018) acrescenta que em determinados casos há parâmetros de configuração que são sensíveis e geralmente são chamados de “secrets” (isto é, segredos) como senhas e chaves de acesso. Segundo o autor, há organizações que não colocam esses “secrets” sobre o controle de versão, já que a recomendação nesses casos é o armazenamento por meio de criptografia.

D. Escolha do Sistema de Gerenciamento de Conteúdo

Conforme dito na seção 2.4 deste trabalho, as aplicações Web podem alcançar uma grande quantidade de conteúdo. Nesse sentido, ferramentas específicas devem ser usadas para estabelecer o gerenciamento de conteúdo. Pressman e Maxim (2016), explicam que em muitos casos um processo convencional para o controle de versão pode ser muito complicado para as aplicações Web. De acordo com os autores, para lidar com esse aspecto há uma nova geração de ferramentas que são especificamente projetadas para essas aplicações. Essas ferramentas são chamadas de Sistemas de Gerenciamento de Conteúdo (SGC).

Harwani (2015), explica que um SGC é um sistema de software que permite criar, editar e publicar diferentes tipos de documentos. De acordo com o autor, esses documentos incluem arquivos de áudio e vídeo, arquivos de imagens e muitas outras formas de conteúdo da Web.

George (2015), esclarece que um SGC faz o uso de banco de dados e sistemas de arquivos para fornecer o armazenamento flexível e escalável do conteúdo de uma aplicação

Web. Segundo o autor, um sistema de gerenciamento de conteúdo é capaz de lidar com aplicações que envolvem pouco conteúdo, até plataformas de publicação massiva de conteúdo com milhões de páginas.

De acordo com Pressman e Maxim (2016), o gerenciamento de conteúdo está relacionado ao gerenciamento de configuração no sentido de que um SGC estabelece um processo (suportado por ferramentas apropriadas), que adquire o conteúdo existente (a partir de uma ampla variedade de itens de configuração de aplicações Web), estrutura esse conteúdo de forma que possa ser apresentado ao usuário final, e então fornece o conteúdo obtido para ser exibido no lado cliente (client-side). Esse processo é ilustrado na Figura 8 da seção 2.4.

Posto isso, Barker (2016) explica que um sistema de gerenciamento de conteúdo fornece funções de controle como:

1. Permissões:
   a) Quem pode ver o conteúdo?
   b) Quem pode modificá-lo?
   c) Quem pode excluí-lo?
2. Gerenciamento de trabalho e fluxo de trabalho (workflow):
   a) Este conteúdo está publicado?
   b) O conteúdo está em rascunho?
   c) O conteúdo foi arquivado e removido do público?
3. Controle de versões:
   a) Quantas vezes este conteúdo mudou?
   b) Como era há três meses?
   c) Como essa versão difere da versão atual?
   d) Posso restaurar ou republicar uma versão mais antiga?
4. Gerenciamento de dependência:
   a) Qual conteúdo está sendo usado por outro conteúdo?
   b) Se esse conteúdo for excluído, como isso afetará outro conteúdo?
   c) Qual conteúdo atualmente não está sendo utilizado?
5. Pesquisa e organização:
   a) Como encontrar uma parte específica do conteúdo?
   b) Como encontrar todo o conteúdo que se refere a X?
   c) Como agrupar e relacionar o conteúdo de modo que facilite o gerenciamento?

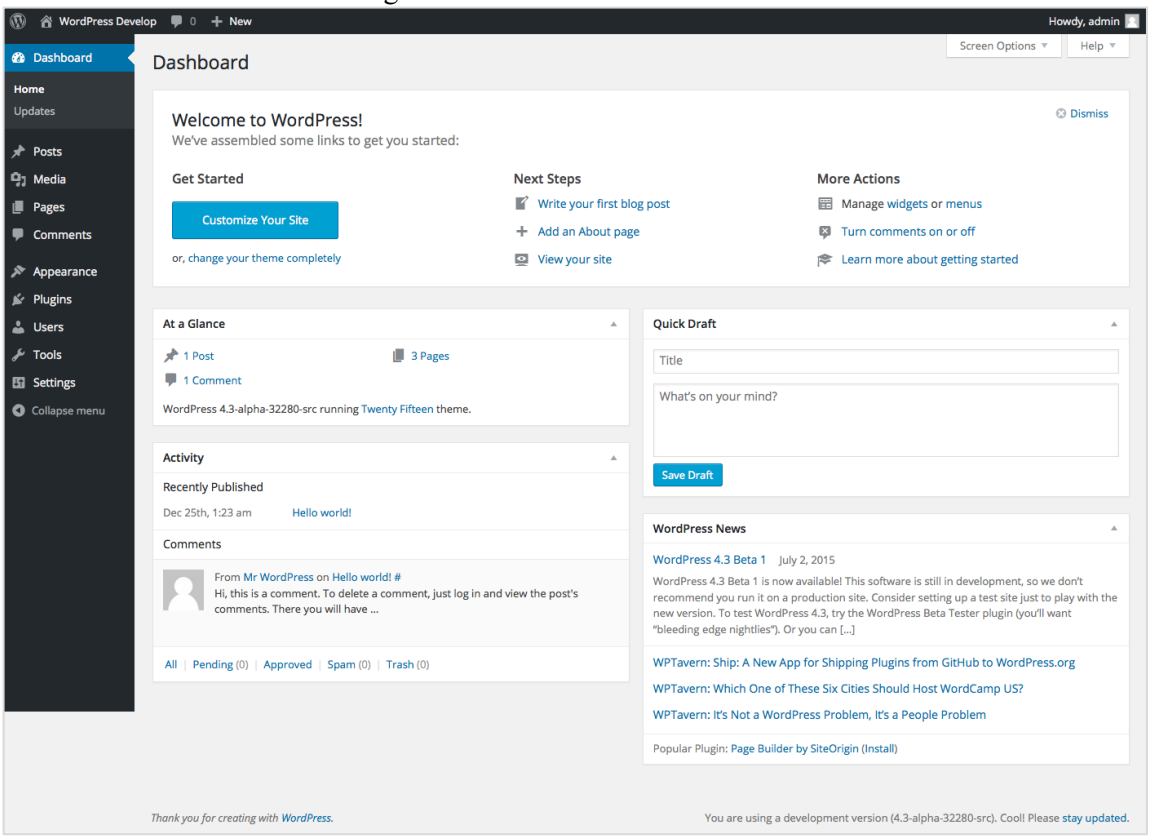
Essas funcionalidades incluem uma forma refinada de controle de versão para manter o conteúdo proveniente de aplicações baseadas na Web, pois segundo Barker (2016) essas

funções aumentam o nível de controle sobre o conteúdo e ajudam a reduzir riscos de exclusões acidentais desse conteúdo.

Kudaisi (2017), esclarece que há muitos sistemas de gerenciamento de conteúdo disponíveis e cada um deles possui diferentes propósitos e funcionalidades, incluindo: Joomla, DotNetNuke, TextPattern, Umbraco, ModX, WordPress, RefineryCMS, TinyCMS, Magnolia, Liferay, Ametys Liferay e Ph7CMS Liferay. De acordo com Horsman (2018), dentre essa variedade de SGC, os mais populares são WorPress, Joomla! e Drupal, os quais são brevemente apresentados nos parágrafos seguintes.

Williams, Tadlock e Jacoby (2020), explicam que o WordPress é o sistema de gerenciamento de conteúdo de código aberto mais popular. Uma das principais razões para essa popularidade é a facilidade para personalizar e estender os recursos do WordPress através de plugins. Segundo os autores, um plugin no WordPress é basicamente um script baseado em PHP que estende ou altera as funcionalidades do WordPress. A Figura 9 ilustra o Dashboard do WordPress.

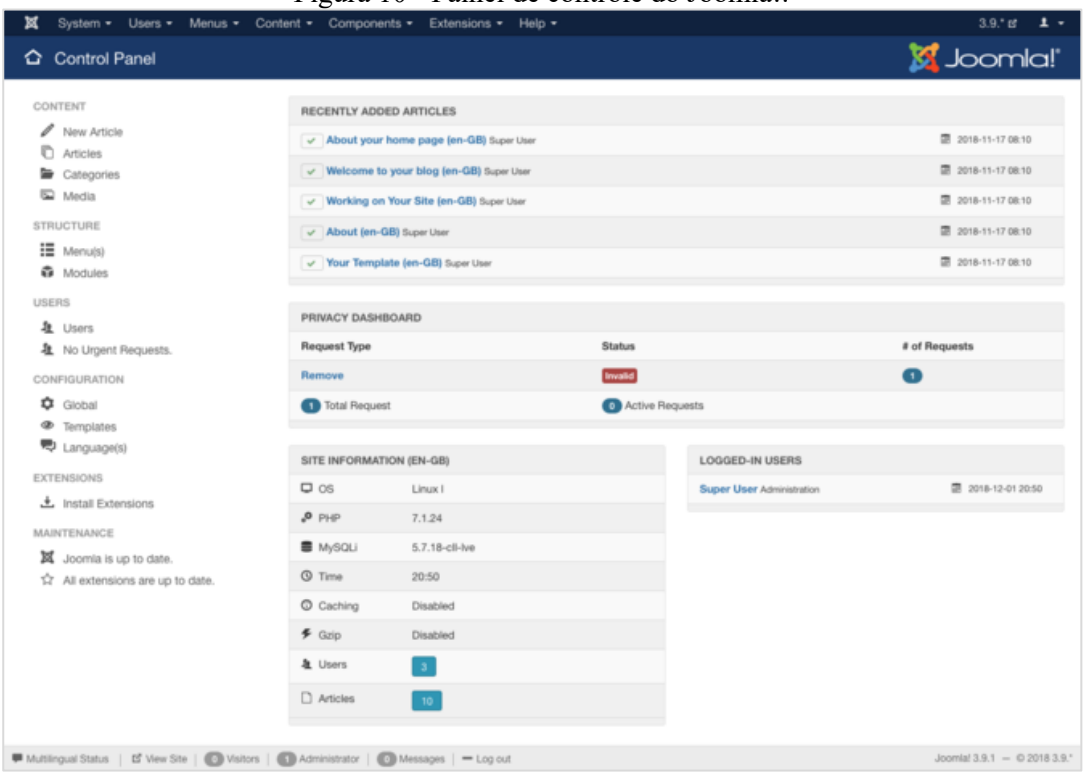
Figura 9 - Dashboard do WordPress.



Fonte: O autor.

Harwani (2015), explica que o Joomla! é um SGC baseado em PHP que ajuda no desenvolvimento de Websites dinâmicos e funcionais, seja blogs ou Websites profissionais como lojas de e-commerce. De acordo com o autor, além de possuir uma interface de fácil uso, o Joomla! fornece funcionalidades para publicar conteúdo em seu Web site, moderar e responder a comentários, registrar usuários em seu site e permitir acesso ao seu conteúdo. A Figura 10 ilustra o painel de controle do Joomla!.

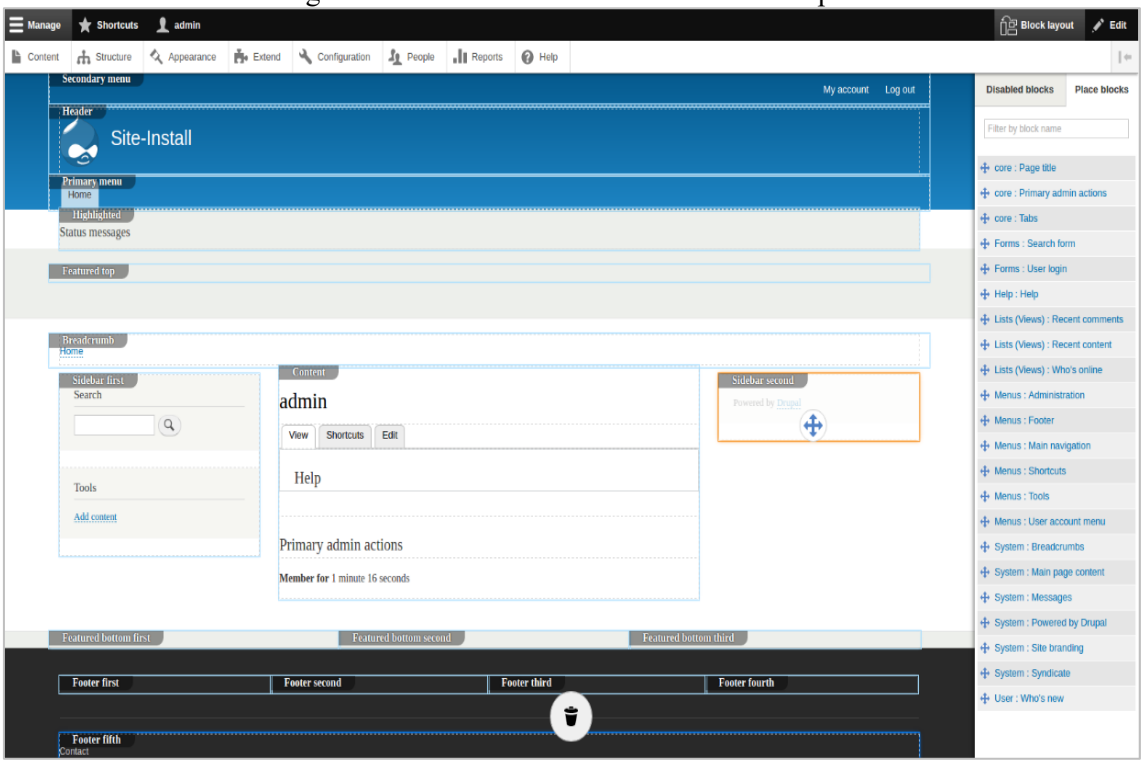
Figura 10 - Painel de controle do Joomla!.



Fonte: O autor.

Sipos (2020), menciona que o Drupal é um poderoso sistema de gerenciamento de conteúdo baseado na Web, e que é usado para construir desde sites mais simples, até aplicações complexas. De acordo com o autor, o Drupal é escrito na linguagem de programação PHP e usa a biblioteca PHP Data Objects (PDO) que permite aos desenvolvedores oferecer suporte a vários bancos de dados, incluindo MySQL, PostgreSQL, SQLite e MariaDB. A Figura 11 ilustra a interface de administrador do Drupal.

Figura 11 - Interface de administrador do Drupal.



Fonte: O autor.

Embora esses SGC possuam algumas similaridades em relação a sua estrutura e ferramentas básicas, Kudaisi (2017) explica que eles diferem em funcionalidades e serviços. Nesse contexto, o autor indica que os seguintes pontos devem ser observados ao estabelecer um sistema de gerenciamento de conteúdo para uma aplicação Web:

- Segurança: é preciso que o SGC forneça atualizações e melhorias constantes de versão e dos incrementos de segurança, pois isso ajuda a dificultar as tentativas do invasor de aprender sobre a versão e procurar por vulnerabilidades;
- Custo de execução: há custos incorridos quando um SGC está em funcionamento. O cálculo de custos deve ser feito antes do site ser desenvolvido e esses custos podem ser reduzidos dependendo de como a execução do site é planejada.
- Custo de hospedagem: existem duas opções disponíveis para hospedar um site que usa SGC, a primeira é terceirizar o serviço por meio de uma empresa de hospedagem e a segunda é hospedar o site em um computador pessoal ou servidor dedicado;
- Custo para empregar pessoal de tecnologia da informação (TI): conforme a empresa cresce, surge a necessidade de obter um site cada vez mais dinâmico e robusto, o que requer atualizações e melhorias. Com isso, se torna indispensável a contratação de pessoal de TI para ajudar a solucionar problemas técnicos;

- Custo de atualização: à medida que uma aplicação Web aumenta seus arquivos de conteúdo, mais espaço no servidor será necessário para o carregamento adequado da aplicação. Outrossim, sistemas operacionais, softwares de segurança e quaisquer outros que permitam que o servidor funcione adequadamente também precisam ser atualizados e melhorados;
- Eficiência do SGC: eficiência é o ponto de apoio que equilibra custos e serviços. O SGC deve resistir ao teste do tempo e se revigorar para uma nova demanda. A eficiência cresce com o aumento de usuários e a demanda por novos serviços. Assim, a eficiência será determinada se o SGC for capaz de atender a novas demandas.

3 TRABALHOS RELACIONADOS

Neste capítulo são apresentados alguns trabalhos relacionados a abordagem proposta. Inicialmente é abordado o artigo “Live Versioning of Web Applications through Refactoring” de Grigera et al. (2018). Em seguida, é apresentado a patente de Rudek et al. (2020), intitulada “Distributed Version Control for Tracking Changes in Web Applications”. Por fim, é posto o trabalho de Lunyov (2019), chamado “Detecting Changes in Web Applications”. Embora cada um desses trabalhos forneça como resultado uma ferramenta específica, é importante esclarecer que estão alinhados a este trabalho no sentido de propor soluções para adequar o controle de versão a natureza das aplicações Web.

3.1 Live Versioning of Web Applications through Refactoring

Em seu trabalho, Grigera et al. (2018) mostram uma ferramenta capaz de criar versões alternativas de uma aplicação Web com menor esforço através de um processo conhecido como Client-Side Web Refactoring (CSWR). Conforme expõe os autores, sua proposta é dedicada a geração de versões para usabilidade, visível apenas para gerentes de aplicação. Basicamente, a ferramenta é definida a partir de um serviço denominado Kobold, o qual possibilita aplicar o processo CSWR para corrigir possíveis sintomas de código ruim.

Grigera et al. (2018), explicam que a criação de versões de uma aplicação Web sem a modificação do código fonte permite que os desenvolvedores possam avaliar alternativas para usabilidade com menor esforço sem comprometer a versão de produção, e ainda aproveitar a mesma infraestrutura. Segundo os autores, isso é muito importante ao considerar um cenário em que os requisitos mudam constantemente e as interfaces de usuário precisam acompanhar essa evolução.

A conclusão do trabalho indica que, embora a ferramenta proposta seja útil para gerar diferentes versões de aplicações Web em execução com pouco esforço e ainda permitir o uso dos resultados para realizar testes direcionados a melhoria de usabilidade da aplicação, há uma limitação. De acordo com os autores, tarefas relacionadas a testes com efeito permanente não poderiam ser feitas (p. ex. completar um processo de check-out). No entanto, nesses casos a configuração de um ambiente ainda permite que a ferramenta de controle de versão crie quantas versões de teste sejam necessárias em uma única infraestrutura.

3.2 Distributed Version Control for Tracking Changes in Web Applications

Rudek et al. (2020), descrevem em sua patente uma invenção que, de modo geral, está relacionada a aplicações Web, e mais particularmente a um SCV distribuído que é configurado para rastrear mudanças por usuários (desenvolvedores) em aplicações Web.

Aspectos da invenção incluem implementar um SCV para aplicações Web. Para esse propósito, os autores explicam o funcionamento de um sistema que inclui um dispositivo de usuário configurado para acessar um servidor que, por sua vez, armazena uma aplicação Web que é acessível por um outro dispositivo de usuário para fazer mudanças que sejam armazenadas em um histórico de versões no repositório original.

Nesse cenário, Rudek et al. (2020), ressaltam que o dispositivo do usuário é configurado para obter acesso ao histórico de versões da aplicação Web no repositório original e armazenar as mudanças feitas por um outro dispositivo de usuário no histórico de versões da aplicação Web em um primeiro repositório local do dispositivo.

3.3 Detecting Changes in Web Applications

Lunyov (2019), estabelece um trabalho sobre a identificação e classificação de mudanças em aplicações baseadas na Web, a partir da comparação de duas versões de código de uma página Web, adquiridas em diferentes períodos. De acordo com o autor, o resultado da pesquisa é uma ferramenta semiautomática, desenvolvida em Python.

A ferramenta compara duas versões de código da página Web para encontrar mudanças e classificá-las. Segundo o autor, o resultado da execução dessa ferramenta é um arquivo de relatório que contém estatísticas sobre a execução geral do algoritmo e informações agrupadas por tipo sobre as mudanças detectadas entre duas versões do código da página Web.

Lunyov (2019), conclui que a análise dos relatórios mostrou que a ferramenta implementada fornece resultados confiáveis e aloca todos os tipos de mudanças possíveis nos códigos de páginas Web que são reconhecidas por análises estatísticas.

4 AJUSTANDO CONTROLE DE VERSÃO À NATUREZA DAS APLICAÇÕES WEB

Este capítulo descreve a abordagem inicialmente proposta para adequar o controle de versão à natureza das aplicações Web. Esta abordagem não rejeita os princípios, práticas e ferramentas do controle de versão. Em vez disso, inclui um conjunto de estratégias que moldam o controle de versão conforme as necessidades das aplicações Web. A primeira seção mostra como essas estratégias são estabelecidas. Nas seções seguintes essas estratégias são delineadas.

4.1 Definição das estratégias

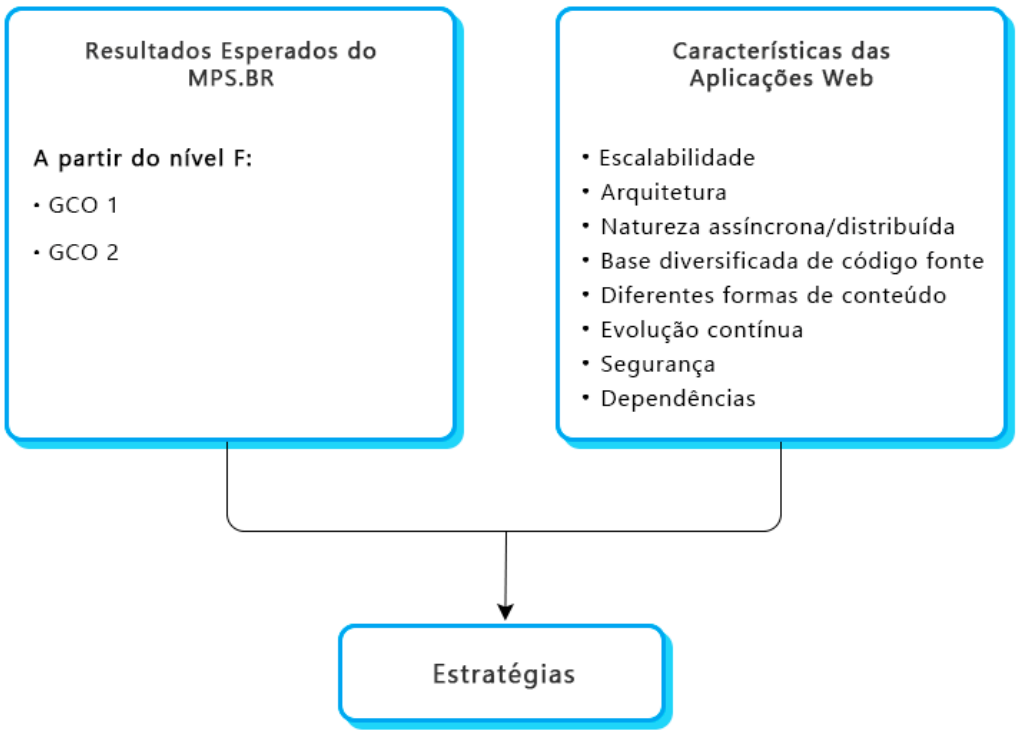
Conforme mencionado, este trabalho inclui estratégias para adequar o controle de versão à natureza das aplicações Web. Como ponto de partida, esta seção explica como as estratégias são estabelecidas para compor a abordagem proposta.

Segundo Dart Apud Pressman e Maxim (2016), para adequar o GC e o controle de versão a projetos de aplicações Web, alguns elementos devem ser considerados:

1. Conteúdo: Aplicações Web tipicamente envolvem uma ampla variedade de conteúdo (p. ex. texto, gráficos, arquivos de áudio e vídeo, formulários, tabelas, scripts e muitos outros). Nesse sentido, é essencial estabelecer mecanismos de controle apropriados;
2. Pessoas: Devido a uma porcentagem significante do desenvolvimento de aplicações Web continuar a ser conduzido de uma maneira ad hoc (isto é, para uma finalidade específica), qualquer pessoa envolvida na aplicação Web pode criar conteúdo. Muitos criadores de conteúdo não possuem noções sobre o controle de versão. Como resultado, a aplicação cresce e muda de forma descontrolada;
3. Escalabilidade: À medida que o tamanho e a complexidade da aplicação Web crescem, pequenas mudanças podem ter efeitos não intencionais de longo alcance problemáticos. Portanto, o rigor dos mecanismos de controle de versão deve ser diretamente proporcional a escala da aplicação;
4. Políticas: As seguintes perguntas ajudam uma equipe de desenvolvimento a compreender as políticas associadas a engenharia Web: Quem garante que os processos de controle de qualidade foram aplicados antes da informação ser publicada no Website? Quem é responsável pela realização de mudanças? Quem assume o custo da mudança?

De fato, manter o controle sobre elementos como conteúdo, pessoas, escalabilidade e políticas é algo imprescindível. Para isso, é necessário definir estratégias capazes de alinhar o controle de versão à natureza das aplicações Web. Neste trabalho, essa definição é feita com base no padrão MPS.BR e nas características que formam a natureza das aplicações Web. A Figura 12 ilustra esse procedimento.

Figura 12 - Definição das estratégias para o controle de versão.



Fonte: O autor.

O modelo MPS.BR determina que o propósito do gerenciamento de configuração é estabelecer e manter a integridade dos produtos de trabalho e disponibilizá-los a todos os envolvidos. Para isso, o modelo MPS.BR define cinco resultados esperados no nível de maturidade F:

- GCO 1 (a partir do nível F): Itens de configuração são identificados e seus níveis de controle são estabelecidos.
- GCO 2 (a partir do nível F): Um sistema para gerencia de configuração e controle de mudanças é estabelecido, mantido atualizado e utilizado.
- GCO 3 (a partir do nível F): Baselines são estabelecidas considerando entregáveis e liberações aos interessados.

- GCO 4 (a partir do nível F): Registros de itens de configuração e de modificações realizadas nestes itens são estabelecidos, mantidos atualizados e utilizados.
- GCO 5 (a partir do nível F): Auditorias de configuração são executadas para avaliar as baselines e o conteúdo do sistema de gerenciamento de configuração.

Por que o modelo MPS.BR? O modelo MPS.BR traz significado para o processo de gerenciamento de configuração e suas atividades no desenvolvimento e manutenção de software, pois define com clareza os resultados esperados para esse processo. Logo, as aplicações Web podem obter maior efetividade do controle de versão ao estabelecer estratégias que possam estar alinhadas com os resultados esperados do modelo MPS.BR e com as características que formam a sua natureza única.

Por que o uso de estratégias? Paez (2018) explica entre os diversos tópicos abordados pelo GC a utilização de estratégias para o controle de versão é um dos pontos chave. Nesse sentido, o trabalho estabelece diferentes estratégias para o controle de versão.

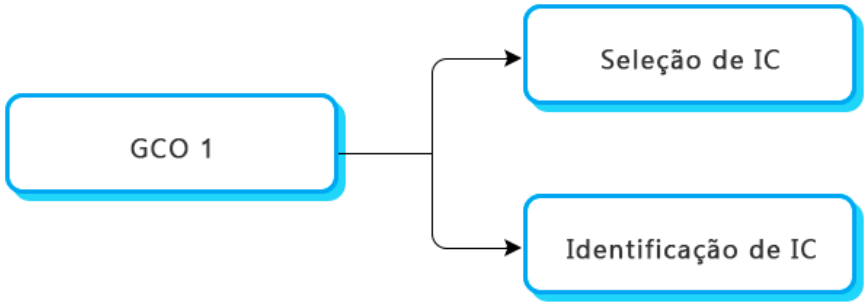
É importante dizer que a definição das estratégias é feita apenas com base nos resultados esperados GCO 1 e GCO 2 do modelo MPS.BR e nas características das aplicações Web apresentadas na Figura 12. Nessa composição, as principais características das aplicações Web incluem escalabilidade, arquitetura, natureza assíncrona e distribuída, base diversificada de código fonte, diferentes formas de conteúdo, evolução contínua, segurança e dependências.

Por que incluir apenas os resultados esperados GCO1 e GCO 2? O nível de maturidade F do modelo MPS.BR abrange o gerenciamento de configuração e todas as suas atividades. No entanto, este trabalho é direcionado apenas para o controle de versão e aplicações Web. Portanto, definir estratégias para cada um dos resultados esperados do modelo MPS.BR significa estabelecer estratégias para as demais atividades do gerenciamento de configuração e esta não é a finalidade do trabalho. Dito isso, as seções seguintes descrevem as estratégias.

4.2 Estratégias para o GCO 1

Esta seção define estratégias alinhadas ao resultado esperado GCO 1 do modelo MPS.BR para adequar o controle de versão a natureza das aplicações Web. Conforme mencionado, o resultado esperado GCO 1 determina que os itens de configuração devem ser identificados e seus níveis de controle estabelecidos. Para identificar os itens de configuração é necessário que seja feito previamente a seleção dos itens. Em outras palavras, é preciso determinar quais itens de configuração devem estar sobre controle de versão e, por conseguinte, identificar esses itens. Logo, as estratégias para esse resultado esperado envolvem dois aspectos do controle de versão, conforme ilustra a Figura 13.

Figura 13 - Aspectos do controle de versão para o resultado esperado GCO 1.



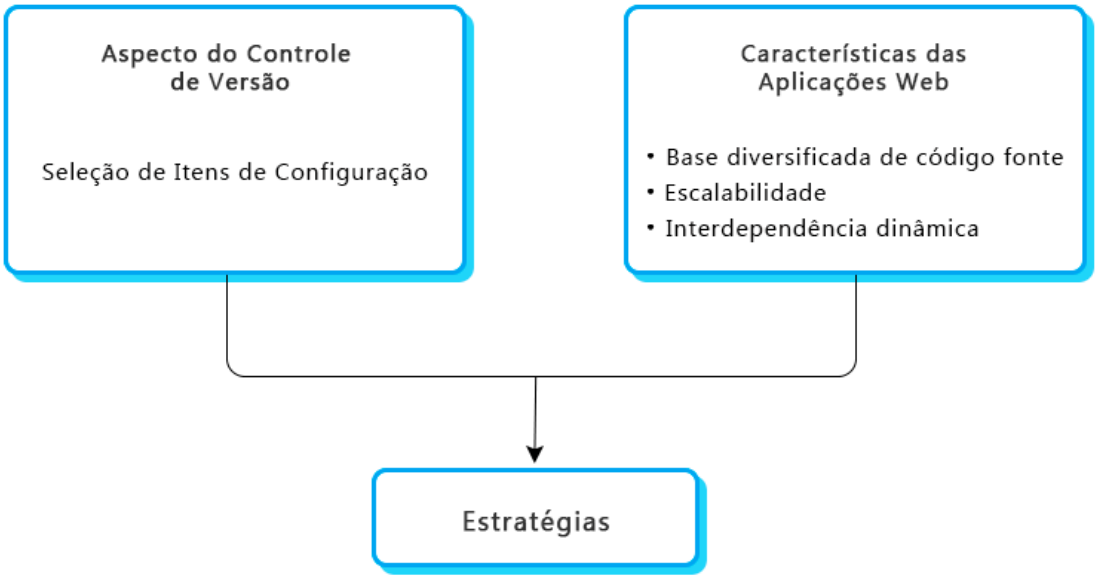
Fonte: O autor.

Como mostra a Figura 13, o resultado esperado GCO 1 está relacionado a seleção de itens de configuração e a identificação de itens de configuração. Para cada um desses aspectos do controle de versão é definido um conjunto de estratégias. As seções seguintes descrevem essas estratégias.

4.2.1 Estratégias para a Seleção de Itens de Configuração

Esta seção define estratégias para a seleção de itens de configuração em projetos de aplicações Web. Essas estratégias são formadas a partir da seleção de itens de configuração e de características das aplicações Web como a base diversificada de código fonte, escalabilidade e interdependência dinâmica entre itens de configuração. A Figura 14 mostra a origem das estratégias.

Figura 14 - Origem das estratégias para a seleção de itens de configuração.



Fonte: O autor.

Dito isso, temos as seguintes estratégias:

A. Reconhecer quais são os tipos de itens de configuração em aplicações Web;

B. Avaliar a importância do item para o desenvolvimento posterior da aplicação Web;

C. Estabelecer a quantidade de itens de configuração;

D. Verificar o relacionamento entre os itens de configuração.

As seções seguintes descrevem cada uma dessas estratégias.

A. Reconhecer quais são os tipos de Itens de Configuração em Aplicações Web

Em um projeto de aplicação Web é imprescindível reconhecer quais são tipos de itens de configuração. Esse passo é útil para determinar se o item de configuração deve estar sobre controle de versão. Aplicações baseadas na Web possuem uma natureza multilíngue e diferentes tipos de código fonte, além de módulos e bibliotecas que podem ser necessários para o funcionamento da aplicação. Diante dessa base heterogênea de itens de configuração é fundamental saber quais tipos devem receber prioridade.

Em geral, aplicações Web incluem documentos de hipertexto, folhas de estilo e scripts, mas existem muitos outros tipos de arquivos que podem fazer parte dessas aplicações, incluindo módulos que adicionam funcionalidades como, por exemplo, estabelecer a comunicação entre banco de dados. Dessa forma, é importante observar as tecnologias a serem usadas no projeto, pois, a partir disso, arquivos de diferentes tipos são criados e mantidos.

Para reconhecer os itens de configuração em aplicações Web é importante analisar as seguintes perguntas:

- Quais são os itens de configuração que compõe a aplicação Web?
- Qual é o tipo de cada item de configuração (p. ex. documento de hipertexto, folha de estilo, documento de banco de dados, etc.)?
- Qual é o seu formato (p. ex. js, CSS, SCSS, XML, HTML)?

B. Avaliar a Importância do Item para o Desenvolvimento posterior da Aplicação

É fundamental verificar a importância do item de configuração para o desenvolvimento posterior de uma aplicação Web. Em outras palavras, é preciso avaliar se o item é, de fato, necessário para o andamento futuro dessa aplicação. Se o item de configuração é essencial para lançamentos futuros, seu acompanhamento é necessário e, portanto, deve estar sobre a guarda do controle de versão.

Como as aplicações Web possuem uma natureza assíncrona e distribuída é pertinente observar de forma estratégica o potencial impacto de scripts que possam provocar falhas no desenvolvimento da aplicação devido a alterações em seu código. Daí a necessidade de manter tais itens sobre o controle de versão.

Desse modo, para avaliar a importância do item de configuração em uma aplicação Web é importante responder as seguintes perguntas:

- O Item de configuração é necessário para o desenvolvimento posterior da aplicação Web?
- O Item de configuração tem potencial para provocar falhas no desenvolvimento da aplicação Web?

C. Estabelecer a Quantidade de Itens de Configuração

A medida em que mais itens de configuração são adicionados a um projeto de aplicação Web, mais difícil se torna a manutenção dessa aplicação. É notável que aplicações dessa natureza, possuam necessidades atreladas a escalabilidade e desempenho. Nesse sentido, os itens de configuração mantidos pelo controle de versão passam por revisões e processos para manter de forma adequada a rastreabilidade desses artefatos. No entanto, essas revisões podem custar tempo e sobrecarga de esforços, já que muitos itens de configuração podem fazer parte de uma aplicação Web.

Com base nisso, definir a quantidade de itens de configuração a serem postos sobre o controle de versão é uma estratégia a considerar. Para isso, avaliar critérios como escalabilidade pode ser um mecanismo útil.

Posto isso, para estabelecer a quantidade de itens de configuração em aplicações Web é importante responder as seguintes perguntas:

- Quantos itens de configuração devem passar por revisões e processos formais?
- Quais são os custos para realizar essas revisões e processos formais (p. ex. tempo, pessoas, etc.)?
- A quantidade de itens de configuração estabelecida pode afetar o desenvolvimento, teste e manutenção da aplicação Web?

D. Verificar o Relacionamento Entre os Itens de Configuração

É importante observar o relacionamento entre os itens de configuração antes colocá-los sobre a guarda do controle de versão. Conforme dito, aplicações Web possuem uma ampla variedade de código fonte e uma natureza assíncrona e distribuída que tornam o seu desenvolvimento e manutenção uma tarefa difícil.

Dessa forma, é preciso entender como um item de configuração está atrelado ao funcionamento de outros itens e verificar suas dependências. Esse relacionamento é um ponto chave, uma vez que aplicações Web envolvem itens de configuração que são responsáveis por manter o equilíbrio entre o lado servidor e o lado cliente.

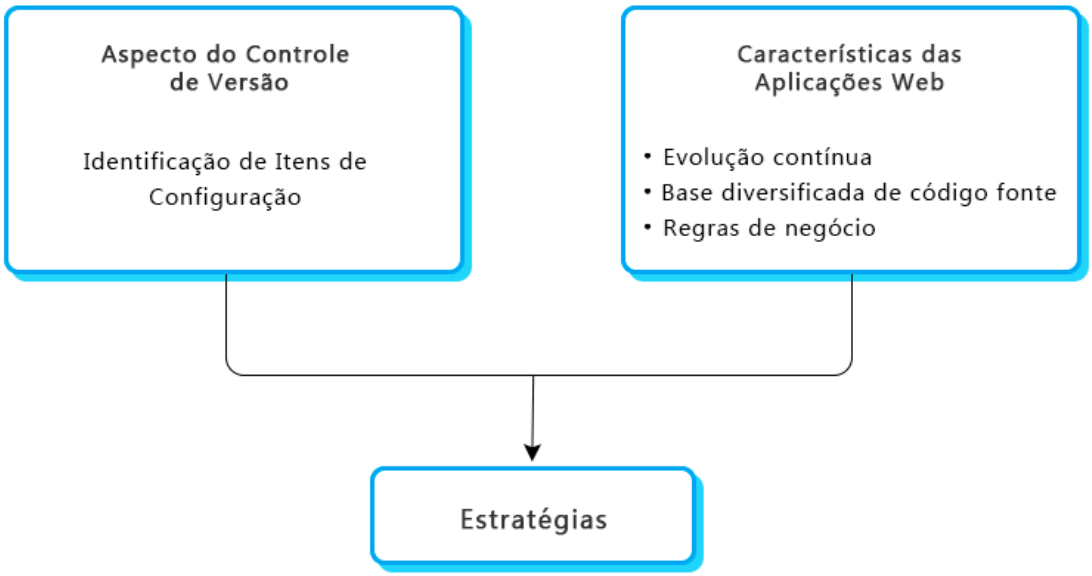
Assim, para verificar o relacionamento entre os itens de configuração de uma aplicação Web, é importante responder as seguintes perguntas:

- Qual é a relação de dependência entre os itens de configuração?
- A modificação de um item de configuração possui qual impacto para outros itens de configuração?

4.2.2 Estratégias para a Identificação de Itens de Configuração

Esta seção faz o delineamento de estratégias para a identificação de itens de configuração em aplicações Web. Essas estratégias ponderam a convenção de nomenclatura desses itens considerando características como evolução contínua, base diversificada de código fonte e regras de negócio. Para ilustrar essa composição, a Figura 15 mostra a origem das estratégias.

Figura 15 - Origem das estratégias para a identificação de itens de configuração.



Fonte: O autor.

Com base nisso, temos as seguintes estratégias:

A. Estabelecer um código único para cada item;

B. Estimar a quantidade de versões e lançamentos;

C. Por em evidência o tipo do item de configuração;

D. Verificar as regras de negócio;

E. Observar a convenção de nomenclatura das ferramentas.

Vale dizer que não existe um modelo único de nomenclatura para os itens de configuração. Em outras palavras, a convenção de nomenclatura muda conforme características como escalabilidade e regras de negócios. Logo, é preciso entender como diferentes modelos de nomenclatura podem ser utilizados e adaptados para estar em conformidade com a natureza das aplicações Web.

Portanto, a finalidade dessas estratégias não é criar uma regra específica de nomenclatura para itens de configuração. Em vez disso, a ideia é fornecer um direcionamento para refinar a identificação dos itens de configuração de acordo com as características da aplicação Web. As seções seguintes descrevem essas estratégias.

A. Estabelecer um Código Único para Cada Item de Configuração

Cada item de configuração deve possuir um código de identificação único. Embora seja algo básico para qualquer identificador, é essencial definir as posições que representam o item de configuração em uma aplicação Web. Essas aplicações podem incluir uma grande

quantidade de itens de configuração. Com base nisso, é importante definir uma convenção de nomenclatura capaz de representar a quantidade estimada de itens da aplicação Web. Por exemplo, é possível definir três posições PPP de modo que cada posição possa incluir dígitos de 0 a 9. Logo, temos 10^3, isto é, podemos representar 1.000 itens de configuração.

Nesse cenário, cada item de configuração é representado por um código de três posições. Assim, podemos identificar um dos muitos itens de configuração de uma aplicação Web como 035. Vale dizer que essas três posições são apenas uma parte do identificador e outras posições devem ser incluídas, especialmente para definir a versão e os lançamentos.

Dito isso, para estabelecer um código único para cada item de configuração em uma aplicação Web é importante responder as seguintes perguntas:

- Cada item de configuração possui um código formado por um conjunto de posições que o diferenciam dos demais itens?
- A convenção de nomenclatura definida é capaz de representar a quantidade estimada de itens da aplicação Web?

B. Estimar a Quantidade de Versões e Lançamentos

À medida que uma aplicação Web cresce a nível de funcionalidades e recursos, maior é quantidade de mudanças no seu código fonte. Nesse sentido, aplicações Web estão em evolução contínua e essa característica deve ser ponderada ao definir uma convenção de nomenclatura para os itens de configuração. Logo, é preciso obter uma noção da possível quantidade de versões de itens de configuração e dos lançamentos que a aplicação Web pode alcançar. Essa é uma estratégia útil para evitar possíveis problemas como a escolha de uma convenção de nomenclatura incapaz de representar a quantidade de versões e lançamentos de uma aplicação Web. Posto isso, é necessário definir uma quantidade razoável de posições. Daí a necessidade de estimar o número de versões e lançamentos de acordo com a evolução contínua de tais aplicações.

Cada posição pode ser puramente numérica, isto é, de 0 a 9. Além disso, é possível incluir níveis de versão, isto é, podemos criar uma posição que representa uma versão principal, outra posição para uma versão secundária, e assim por diante. A quantidade de posições do identificador pode variar, mas é preciso ter um consenso a partir da estimativa de versões e lançamentos para uma aplicação Web. Por exemplo, um identificador com 4 posições VVVV (onde V representa o número de versão do item) é capaz de representar 10.000 versões para o item de configuração. Também é importante definir as posições que representam os

lançamentos da aplicação Web por meio de um raciocínio similar, como RRR (onde R, significa Release, isto é, o número de lançamentos). Nesse cenário, é útil estimar tanto a quantidade de versões que um item de configuração pode assumir, quanto a quantidade de lançamentos que a aplicação Web pode alcançar. Esse direcionamento pode ser obtido através do potencial de escalabilidade da aplicação.

Sabendo que a quantidade de itens de configuração que compõe uma aplicação Web pode variar, é interessante traçar estimativas para o número de posições de versão e lançamento, tais como:

- VV.RR (duas posições para V e duas para R): Para dígitos de 0 a 9, temos 10^2, isto é, 100 versões para cada item de configuração e 100 lançamentos. O que pode suprir aplicações com baixa escalabilidade.
- VVV.RRR (três posições para V e três para R): Para dígitos de 0 a 9 temos 10^3, isto é, 1000 versões e 1000 lançamentos. O que é suficiente para suprir projetos de software em geral.
- VVVV.RRR (quatro posições para V e três para R): Para dígitos de 0 a 9 temos 10^4, isto é, 10.000 versões e 1.000 lançamentos. Embora seja uma quantidade de posições que forneça um grande número de versões, vale notar que em determinados projetos de aplicações Web muitas funcionalidades e mudanças podem ser implementadas. Dessa forma, em cenários como esse, pode ser razoável a definição de quatro posições para V. Em relação aos lançamentos, como há processos formais a serem seguidos para a garantia de qualidade, geralmente não ocorre uma grande quantidade lançamentos ao ano. Assim, três posições para R é, em geral, mais do que suficiente para grande parte dos sistemas de software, inclusive aplicações baseadas na Web.

A partir disso, para estimar a quantidade de versões e lançamentos dos itens de configuração que formam uma aplicação baseada na Web é importante responder as seguintes perguntas:

- Com base em critérios de escalabilidade e evolução contínua, quantas posições são necessárias para definir a quantidade de versões e lançamentos dos itens de configuração da aplicação Web?
- Há muitos processos formais a serem seguidos para a garantia de qualidade que possam influenciar o controle sobre as versões e os lançamentos?

C. Por em Evidência o Tipo do Item de Configuração

Por em evidência o tipo do item de configuração é algo que pode facilitar a sua identificação, já que as aplicações baseadas na Web possuem uma composição diversificada de código fonte, módulos de bibliotecas e arquivos. Levando em conta essa característica é útil incluir posições para indicar o tipo do item de configuração. Por exemplo, considerando posições puramente numéricas de 0 a 9, podemos separar duas posições TT (onde T é o tipo de item) para representar até 100 tipos de itens de configuração diferentes. Mesmo para aplicações baseadas na Web, duas posições é o suficiente, uma vez que essas posições representam o tipo e não o formato do item de configuração. Para ficar mais claro, podemos identificar os itens a partir do modelo:

PPP.VVVV.RRR.TT

Neste ponto a finalidade é mostrar alguns exemplos para a utilização das posições TT. Logo, as demais posições são apenas ilustrativas. Dito isso, através do modelo acima podemos indicar documentos de script para a realização de testes como 12 e com isso facilitar a sua identificação no projeto, isto é, PPP.VVVV.RRR.12. Nesse cenário de exemplos, podemos ainda definir o código 19 para representar documentos que realizam a descrição do banco de dados, isto é, PPP.VVVV.RRR.19. O código para as posições TT muda conforme o tipo do item de configuração. Nesse sentido, associar cada tipo de item a um código de duas posições é uma solução a ser considerada.

Posto isso, para colocar em evidência o tipo dos itens de configuração que formam aplicações Web é importante responder as seguintes perguntas:

- Há quantos tipos de itens de configuração?
- Há posições para representar cada tipo de item de configuração?

Vale dizer que a primeira pergunta se refere apenas a quantos tipos de itens de configuração a aplicação Web possui e não quais são esses tipos. Definir quais são os tipos dos itens de configuração é objeto de estudo delineado na seleção de itens de configuração.

D. Verificar as Regras de Negócio

Outra estratégia a considerar na identificação de itens de configuração é o acompanhamento das regras de negócio da aplicação Web. Essas regras podem incluir elementos como idiomas, recursos, etc.

Por exemplo, imagine uma aplicação Web na qual seus investidores lançam uma campanha para obter mais usuários ativos. A campanha consiste em conceder moedas digitais que podem ser convertidas em dinheiro para usuários que compartilharem com outros usuários seu código único gerado pela aplicação. Como esse plano de negócios requer um alto investimento e processos de análise, somente alguns países podem receber essa atualização. Nesse cenário, é possível criar um identificador que além das posições convencionais possua duas posições específicas para representar o país no qual determinadas funcionalidades podem estar disponíveis. Com base nesse evento, podemos incluir as posições CC (onde C significa Country) no identificador do item.

Considerando apenas caracteres alfabéticos, temos 26^2, isto é, 676, o que é mais do que o suficiente para indicar os países do globo. Logo, podemos representar o Brasil no evento como BR e o Japão como JP. Dessa forma, fica mais fácil identificar itens de configuração que correspondam as regras de negócio em determinados países. Para esclarecer ainda mais podemos usar o modelo PPP.VVVV.RRR.TT.CC. Nesse cenário, os itens de configuração com funcionalidades para o Brasil podem ser indicados como PPP.VVVV.RRR.TT.BR. Se trocarmos as demais posições por dígitos temos algo como 035.0134.033.23.BR.

Em outras palavras, quando itens de configuração incluem funcionalidades atreladas as regras de negócio, é preciso definir soluções que estejam alinhadas às regras envolvidas. De fato, essas regras mudam conforme o tipo da aplicação Web, isto é, há e-commerces, plataformas de stream, mídias sociais, etc., e conforme as necessidades do projeto e as tomadas de decisão dos stakeholders é possível refinar o identificador dos itens de configuração para trazer um significado ainda maior para a convenção de nomenclatura a ser adotada no projeto.

Dito isso, para verificar as regras de negócio em uma aplicação Web é importante responder as seguintes perguntas:

- Quais são as funcionalidades relacionadas as regras de negócio da aplicação Web?
- Quais itens de configuração possuem funcionalidades atreladas as regras de negócio da aplicação Web?
- A convenção de nomenclatura é capaz de representar as regras de negócio da aplicação Web?

E. Observar a Convenção de Nomenclatura das Ferramentas

Há diversas ferramentas disponíveis para o controle de versão e cada uma delas pode incluir uma convenção de nomenclatura para os itens de configuração. Com base nisso, é

importante observar como essas ferramentas realizam a identificação de itens de configuração. Em outras palavras, é preciso verificar a possibilidade de adaptar o modelo de nomenclatura que a ferramenta fornece. Essa estratégia é útil, pois a identificação de itens de configuração em aplicações baseadas na Web pode mudar conforme as diversas características que formam a natureza dessas aplicações.

Logo, é importante ler a documentação da ferramenta e realizar a sua instalação para testes a fim de verificar a forma como a ferramenta lida com a nomenclatura dos itens de configuração. Através desse procedimento é possível avaliar a flexibilidade da ferramenta em relação a regras de nomenclatura de itens de configuração.

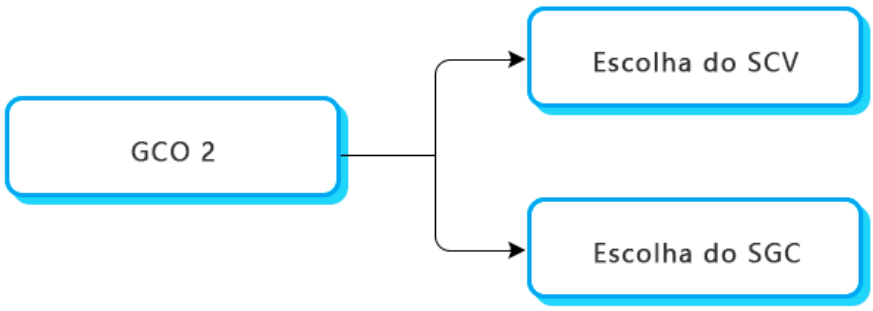
Dito isso, para observar a convenção de nomenclatura das ferramentas, é importante responder as seguintes perguntas:

- A documentação da ferramenta possui algum esclarecimento sobre a convenção de nomenclatura dos itens de configuração?
- A ferramenta foi instalada para a realização de testes de convenção de nomenclatura dos itens de configuração?
- A ferramenta instalada possui flexibilidade em relação a convenção de nomenclatura de itens de configuração?

4.3 Estratégias para o GCO 2

Esta seção define estratégias alinhadas ao resultado esperado GCO 2 do modelo MPS.BR para adequar o controle de versão a natureza das aplicações Web. Conforme mencionado, o resultado esperado GCO 2 determina que um sistema para gerencia de configuração e controle de mudanças deve ser estabelecido, mantido atualizado e utilizado. Conforme ilustra a Figura 16, as estratégias para esse resultado esperado envolvem dois aspectos do controle de versão.

Figura 16 - Aspectos do controle de versão para o resultado esperado GCO 2.



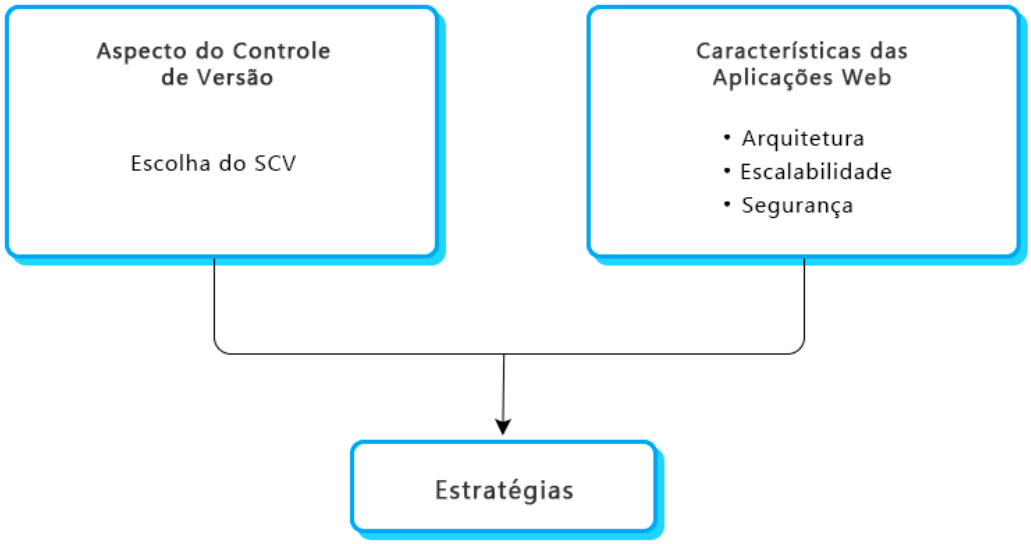
Fonte: O autor.

Como mostra a Figura 16, o resultado esperado GCO 2 está relacionado a escolha do Sistema de Controle de Versão e a escolha do Sistema de Gerenciamento de Conteúdo. Para cada um desses aspectos do controle de versão é definido um conjunto de estratégias alinhadas ao resultado esperado GCO 2. As seções seguintes descrevem essas estratégias.

4.3.1 Estratégias para a Escolha do SCV

Esta seção define estratégias para a escolha do SCV com base em determinadas características das aplicações Web. Essas características incluem arquitetura do repositório, escalabilidade e segurança. Diante disso, a Figura 17 ilustra a origem das estratégias.

Figura 17 - Origem das estratégias para a escolha do SCV.



Fonte: O autor.

Posto isso, temos as seguintes estratégias:

A. Estabelecer o modelo de arquitetura;

B. Observar os mecanismos de segurança;

C. Reconhecer as técnicas de controle de versão.

As seções seguintes descrevem essas estratégias.

A. Estabelecer o Modelo de Arquitetura

Em projetos de aplicações Web, a escolha da arquitetura do repositório é um fator importante. Em outras palavras, o funcionamento do sistema de controle de versão muda conforme o modelo da arquitetura, isto é, centralizada ou distribuída. Logo, observar o tipo de arquitetura é uma estratégia útil para a escolha do SCV. Para aplicações baseadas na Web, um SCV distribuído possui vantagens em relação a um SCV centralizado. Dentre essas vantagens

podemos mencionar confiabilidade, flexibilidade da arquitetura e desenvolvimento geograficamente distribuído.

A confiabilidade dos SCV distribuídos é algo a considerar. Na arquitetura distribuída cada desenvolvedor possui um clone do repositório em seu workspace e não apenas de alguns itens de configuração. No modelo distribuído, cada desenvolvedor também possui um repositório privado. Após modificar os itens de configuração em seu workspace, o desenvolvedor pode atualizar essas mudanças no seu repositório privado. Quando houver atualizações disponíveis no repositório privado dos desenvolvedores, o gerente de repositórios pode decidir sobre a integração dessas atualizações.

Nesse cenário, um SCV distribuído fornece um modelo de arquitetura que ajuda a evitar problemas como a perda parcial ou completa do repositório do projeto devido a erros no servidor que mantém os itens de configuração. Sabendo que aplicações Web possuem uma natureza dinâmica e podem envolver uma grande quantidade de itens de configuração, é essencial que o relacionamento entre esses itens não seja afetado por falhas no repositório do projeto. Em outras palavras, a perda de alguns itens de configuração pode comprometer outros itens do repositório e, dessa forma, colocar em risco todo o projeto. Portanto, um SCV distribuído traz maior confiabilidade ao manter os itens de configuração de uma aplicação Web, já que funciona com base em um modelo de clonagem do repositório e não em um servidor centralizado que recebe constantemente procedimentos de check-in e check-out.

A flexibilidade da arquitetura para o desenvolvimento de aplicações Web é uma característica a ser observada. No modelo distribuído, não é necessária uma conexão à rede para que os desenvolvedores realizem mudanças nos itens de configuração do projeto. Por outro lado, ao considerar uma arquitetura centralizada e a escalabilidade de uma aplicação Web, muitos procedimentos de check-in e check-out podem sobrecarregar o servidor remoto que mantém os itens de configuração do projeto e, por conseguinte, provocar falhas no controle de versão. Para contornar isso, o modelo distribuído permite que cada desenvolvedor possa trabalhar de forma independente e sem a necessidade de acesso constante a um servidor centralizado para fazer suas modificações. Dessa forma, o modelo distribuído traz flexibilidade e fornece uma estrutura útil para o desenvolvimento de aplicações baseadas na Web.

O desenvolvimento geograficamente distribuído é uma forma útil para conduzir projetos de aplicações baseadas na Web. Não raramente, membros de uma equipe de desenvolvimento podem estar em lugares diferentes, seja cidades, estados ou mesmo países. Nesse sentido, um SCV distribuído facilita o processo de desenvolvimento geograficamente disperso, pois cada desenvolvedor possui um clone do repositório do projeto e pode modificar os itens de

configuração em seu próprio workspace sem a necessidade de estar conectado a uma rede centralizada. Por exemplo, há projetos de aplicações baseadas na Web como e-commerces que podem requerer a contribuição de muitos desenvolvedores para implementar diversas funcionalidades dentro de um curto prazo. Caso não haja desenvolvedores disponíveis localmente, é possível usufruir dos recursos que um SCV distribuído fornece para incluir no projeto desenvolvedores que estejam geograficamente distribuídos.

Posto isso, para estabelecer o modelo de arquitetura do SCV em projetos de aplicações Web é importante responder as seguintes perguntas:

- Com base em critérios de confiabilidade e flexibilidade o modelo de arquitetura do SCV é capaz de atender as necessidades da aplicação Web?
- O modelo de arquitetura do SCV fornece mecanismos para o desenvolvimento geograficamente distribuído?
- Quais são as limitações do modelo de arquitetura do SCV?

B. Observar os Mecanismos de Segurança

A segurança é um critério que requer atenção especial ao estabelecer a escolha de um SCV para projetos de aplicações Web. É necessário reconhecer quais são os mecanismos de segurança disponíveis pela ferramenta, isto é, os módulos de bibliotecas, APIs e extensões capazes de fornecer soluções úteis para a segurança dos itens de configuração.

Por exemplo, uma aplicação Web para fins de inscrição e acompanhamento em concursos públicos possui diversos usuários com suas respectivas senhas e outras informações sensíveis. Constantemente, os nomes dos itens de configuração mudam, os locais mudam, as senhas mudam, novas informações surgem e outras são removidas. Nesses casos, não é recomendado que itens de configuração dessa natureza sejam mantidos diretamente sobre o controle de versão sem camadas de segurança. Portanto, é essencial que o SCV forneça mecanismos para garantir a segurança desses itens de configuração e seja capaz de evitar vulnerabilidades.

Os pontos a seguir descrevem brevemente o modelo de segurança das seguintes ferramentas de controle de versão com base na sua documentação:

- Apache Subversion (SVN): Fornece uma estrutura e uma equipe específica para receber relatórios sobre vulnerabilidades de segurança. Nesse caso, ao invés de escrever uma grande quantidade de código fonte proprietário para estabelecer mecanismos de segurança, o Apache Subversion é “ensinado” a interoperar com

bibliotecas e protocolos de segurança fornecidos por especialistas. Conforme expõe a documentação da ferramenta, a finalidade não é fingir o desenvolvimento proprietário de funcionalidades para criptografia e segurança, e sim fornecer uma estrutura com bibliotecas e APIs de terceiros com profundo conhecimento no espaço da segurança;

- IBM Rational Synergy: Inclui mecanismos para garantir segurança durante seu processo de instalação e permitir a comunicação segura entre várias aplicações. Por razões de segurança, somente o administrador pode iniciar e interromper os serviços. Outrossim, o IBM Rational Synergy permite personalizar suas configurações de segurança e configurar funções de usuário e acesso. Essas operações de gerenciamento de usuário incluem segurança de leitura do banco de dados, configurações de segurança para objetos, relacionamentos definidos pelo usuário e problemas de segurança e visibilidade.
- Azure DevOps Server: Usa vários conceitos de segurança para garantir que somente aqueles que de fato devem ter acesso a recursos, funções e dados tenham as devidas permissões. Logo, o Azure DevOps Server controla o seu acesso por meio da autenticação de credenciais de segurança e da autorização dos direitos de conta para acessar um recurso ou uma função. Em outras palavras, o modelo de segurança do Azure DevOps Server é baseado na compreensão dos tipos de conta, métodos de autenticação e autorização e politicas de segurança. Outro mecanismo do Azure DevOps é o log de secrets que na medida do possível realiza a depuração de informações sensíveis, mas não é capaz de filtrar todas as formas pelas quais os secrets podem ser vazados.
- Mercurial: A linha de comando do Mercurial usa um modelo de segurança específico. Nesse modelo, um usuário que pode executar um comando do Mercurial tem permissão para fazer qualquer coisa que o sistema operacional permitir, incluindo a execução de outros comandos. De acordo com a sua documentação, para servir repositórios com acesso limitado de modo seguro, o Mercurial conta com mecanismos de segurança embutidos no sistema operacional ou servidor Web em que está sendo executado. Basicamente, esse modelo não expõe o Mercurial diretamente à Internet, já que o executa por meio de um script de segurança chamado Wrapper ou por trás de um servidor HTTP como o Apache;

- BitKeeper: O modelo de segurança do BitKeeper possui mecanismos para evitar compilações quebradas ou estados indeterminados. De acordo com a documentação da ferramenta, todo acesso ao sistema de arquivos de dados controlados por revisão inclui uma soma de verificação para garantir a integridade dos dados. Além disso, todos os dados são gravados por meio de uma codificação redundante para permitir que falhas comuns de sistema de arquivos e hardware sejam detectadas e corrigidas. Por meio da coleta de dados, o BitKeeper permite análises e auditoria aprofundadas. Em outras palavras, o BitKeeper permite saber quem fez todas as alterações na base de origem, quando foram feitas e de que computador.
- Git: Possui extensões para fornecer camadas de segurança e aprimorar limitações relacionadas ao armazenamento de senhas e chaves de acesso. Uma dessas extensões é o git-secret que possui funcionalidades para realizar a criptografia de itens de configuração e os armazenar no repositório Git.

Logo, ao observar os mecanismos de segurança do SCV a ser usado em projetos de aplicações Web é importante responder as seguintes perguntas:

- A ferramenta fornece mecanismos de segurança (p. ex. criptografia e outras camadas de segurança)?
- A ferramenta permite a integração de extensões, bibliotecas e APIs para estabelecer soluções de segurança?
- Qual é o modelo de segurança da ferramenta?
- Quais são as limitações de segurança da ferramenta?

C. Reconhecer as Técnicas de Controle de Versão

Reconhecer quais são as técnicas de controle de versão que a ferramenta utiliza é algo essencial para o projeto. Conforme o nível de escalabilidade cresce, aplicações baseadas na Web podem incluir uma grande quantidade de itens de configuração e, com isso, aumentar a frequência de mudanças. Por essa razão, é necessário que o sistema de controle de versão forneça técnicas que sejam capazes de conduzir a rastreabilidade desses itens, criar ramificações e mesclar mudanças de forma eficiente.

Conforme mencionado, existem técnicas de controle de versão como deltas e snapshots. Para aplicações Web, o controle de versões com snapshots é uma técnica viável. Posto de forma simples, essa técnica consiste em referenciar links para itens de configuração. Com o uso de snapshots, se os itens de configuração não receberem mudanças ao criar uma nova versão do

projeto, o sistema de controle de versão não armazena o arquivo novamente, pois é criado apenas um link para o item que já está armazenado em uma versão anterior.

Desse modo, a técnica de snapshots contribui para a economia de armazenamento, pois não cria cópias desnecessárias de itens de configuração. Para exemplificar, o Git é uma ferramenta eficiente para realizar o controle de versões por meio de snapshots e pode trazer as contribuições mencionadas para o desenvolvimento de aplicações Web, especialmente ao levar em consideração a escalabilidade e a frequência de mudanças nessas aplicações.

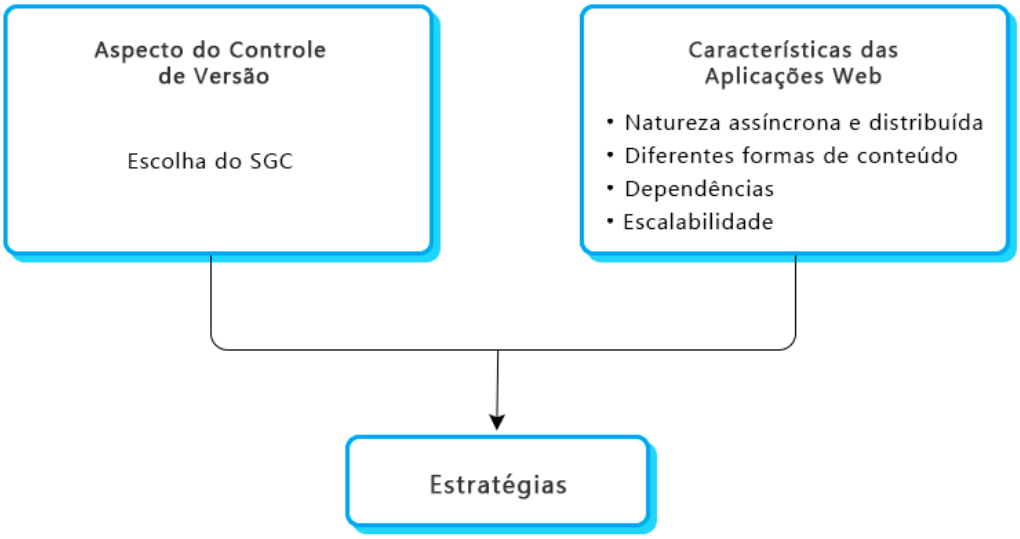
Nesse sentido, ao reconhecer as técnicas de controle de versão da ferramenta a ser usada em projetos de aplicacões Web é importante responder as seguintes perguntas:

- Qual é a técnica que o SCV usa para estabelecer a rastreabilidade, criar ramificações e mesclar mudanças em itens de configuração (deltas ou snapshots)?
- Considerando a escalabilidade e a frequência de mudanças em aplicações Web, a técnica do SCV contribui para a economia de armazenamento?

4.3.2 Estratégias para a escolha do SGC

Esta seção define estratégias para a escolha do sistema de gerenciamento de conteúdo com base em determinadas características das aplicações Web. Essas características incluem a sua natureza assíncrona e distribuída, diferentes formas de conteúdo, dependências e escalabilidade. A Figura 18 ilustra a origem das estratégias.

Figura 18 - Origem das estratégias para escolha do SGC.



Fonte: O autor.

A partir disso, temos as seguintes estratégias:

A. Verificar a flexibilidade do controle de versão;
B. Observar os mecanismos de gerenciamento de dependências;
C. Reconhecer a eficiência da ferramenta com base na escalabilidade.

As seções seguintes fazem um delineamento sobre essas estratégias.

A. Verificar a Flexibilidade do Controle de Versão

A flexibilidade de um sistema de gerenciamento de conteúdo é um fator essencial, especialmente para aplicações baseadas na Web. Devido à natureza assíncrona e distribuída dessas aplicações, é necessário flexibilidade para manter o controle de versão sobre as diversas formas de conteúdo da Web.

Nesse caso, flexibilidade significa estabelecer controles capazes de manter a rastreabilidade do conteúdo e suas versões, sabendo que as aplicações Web possuem um comportamento assíncrono e distribuído que diferencia a forma como esse conteúdo é criado, modificado e acessado.

Dessa forma, um sistema de gerenciamento de conteúdo deve fornecer mecanismos para acessar diferentes versões de modo flexível, isto é, como um histórico que permite retornar a versões anteriores do conteúdo e que possa evitar falhas relacionadas a característica assíncrona e distribuída dessas aplicações.

Logo, para verificar a flexibilidade do controle de versão em um sistema de gerenciamento de conteúdo é importante responder as seguintes perguntas:

- Há mecanismos para o funcionamento adequado do controle de versão considerando a natureza distribuída das aplicações Web?
- Há controles efetivos para criar e acessar versões de conteúdo considerando o comportamento assíncrono das aplicações Web?

B. Observar os Mecanismos de Gerenciamento de Dependências

Conforme já mencionado, aplicações Web possuem várias formas de conteúdo (p. ex. imagens, gráficos, texto, áudio, vídeo, etc.). Essas formas de conteúdo podem estar relacionadas entre si e criar dependências nas páginas de uma aplicação Web. Nesse caso, a modificação de um determinado conteúdo pode afetar outros. Por exemplo, imagine que uma imagem de um produto em uma plataforma de e-commerce seja modificada. Nesse exemplo, há conteúdo na

página que possui relações de dependência com essa imagem, como a descrição do produto e outras informações na forma de texto. Logo, caso não haja mecanismos efetivos para gerenciar as dependências do conteúdo relacionado a essa imagem, a experiência de navegação pode ser prejudicada.

Nesse sentido, é fundamental que um sistema de gerenciamento de conteúdo forneça mecanismos para lidar com as dependências entre as várias formas de conteúdo das aplicações Web. É importante ressaltar que aplicações dessa natureza são dinâmicas, isto é, as modificações ocorrem de forma constante. Para o conteúdo dessas aplicações não é diferente, pois a dependência entre as diferentes formas de conteúdo também é dinâmica e isso pode afetar o controle de versões.

Nesse contexto, para reconhecer mecanismos capazes de controlar adequadamente as dependências de conteúdo das aplicações Web é importante responder as seguintes perguntas:

- Há mecanismos para gerenciar as dependências entre as várias formas de conteúdo das aplicações Web?
- Há mecanismos efetivos para o controle de versão em meio a natureza dinâmica do conteúdo?

C. Reconhecer a Eficiência da Ferramenta com Base na Escalabilidade

A escalabilidade é uma característica determinante para aplicações baseadas na Web. Não raro, aplicações dessa natureza podem atingir um grande volume de conteúdo em um curto espaço de tempo e isso pode criar problemas ao estabelecer o gerenciamento de conteúdo.

Nesse sentido, ao passo que a demanda por escalabilidade aumenta é necessário que o sistema de gerenciamento de conteúdo seja capaz de manter o controle do conteúdo dessas aplicações de forma efetiva. No entanto, manter esse ponto de equilíbrio é uma tarefa difícil, pois há fatores relacionados a custos e serviços necessários para gerenciar o conteúdo e suas versões. Esses custos podem incluir serviços de armazenamento, manutenção, profissionais especializados, ferramentas, etc. Portanto, é importante ponderar esses custos com base nas demandas relacionadas a escalabilidade da aplicação.

Dito isso, para reconhecer a eficiência do sistema de gerenciamento de conteúdo em relação a escalabilidade, é importante responder as seguintes perguntas:

- À medida que a demanda aumenta é possível manter adequadamente o controle de versão para o conteúdo das aplicações Web?

- Fatores relacionados a custos e serviços necessários para a execução da ferramenta prejudicam o gerenciamento do conteúdo e suas versões?

5 RESULTADOS E DISCUSSÃO

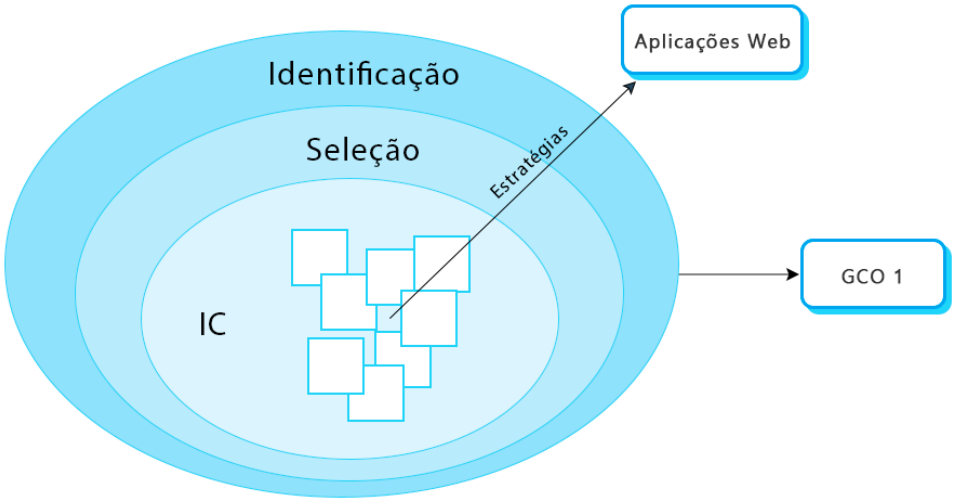
Este capítulo estabelece a discussão sobre os resultados pertinentes ao trabalho. Para isso, é feito uma análise qualitativa sobre as estratégias definidas a partir dos resultados esperados do modelo de software MPS.BR e das características que formam a natureza das aplicações Web. A primeira seção do capítulo discute as estratégias definidas para o resultado esperado GCO 1 e a segunda seção discute as estratégias definidas para o resultado esperado GCO 2. Essas seções possuem subseções para que a discussão das estratégias seja feita conforme o aspecto do controle de versão.

5.1 Resultado Esperado GCO 1

Esta seção discute as estratégias definidas para o resultado esperado GCO 1. Como sabemos, o resultado esperado GCO 1 determina que os itens de configuração devem ser identificados e seus níveis de controle estabelecidos. Primeiramente, é essencial esclarecer como a abordagem é definida para atingir o resultado esperado GCO 1.

Basicamente, o resultado esperado GCO 1 está relacionado a um aspecto do controle de versão conhecido como identificação de itens de configuração. Para identificar os itens de configuração é preciso definir previamente quais itens devem fazer parte do controle de versão. Em outras palavras, é necessário realizar a seleção de itens de configuração e, só então, a identificação desses itens. Como mostra a Figura 19, podemos visualizar esse processo como um arranjo de camadas concêntricas. Nesse processo, os itens de configuração estão dispostos ao centro e as camadas seguintes representam os aspectos do controle de versão mencionados.

Figura 19 – Composição da abordagem para o resultado esperado GCO 1.

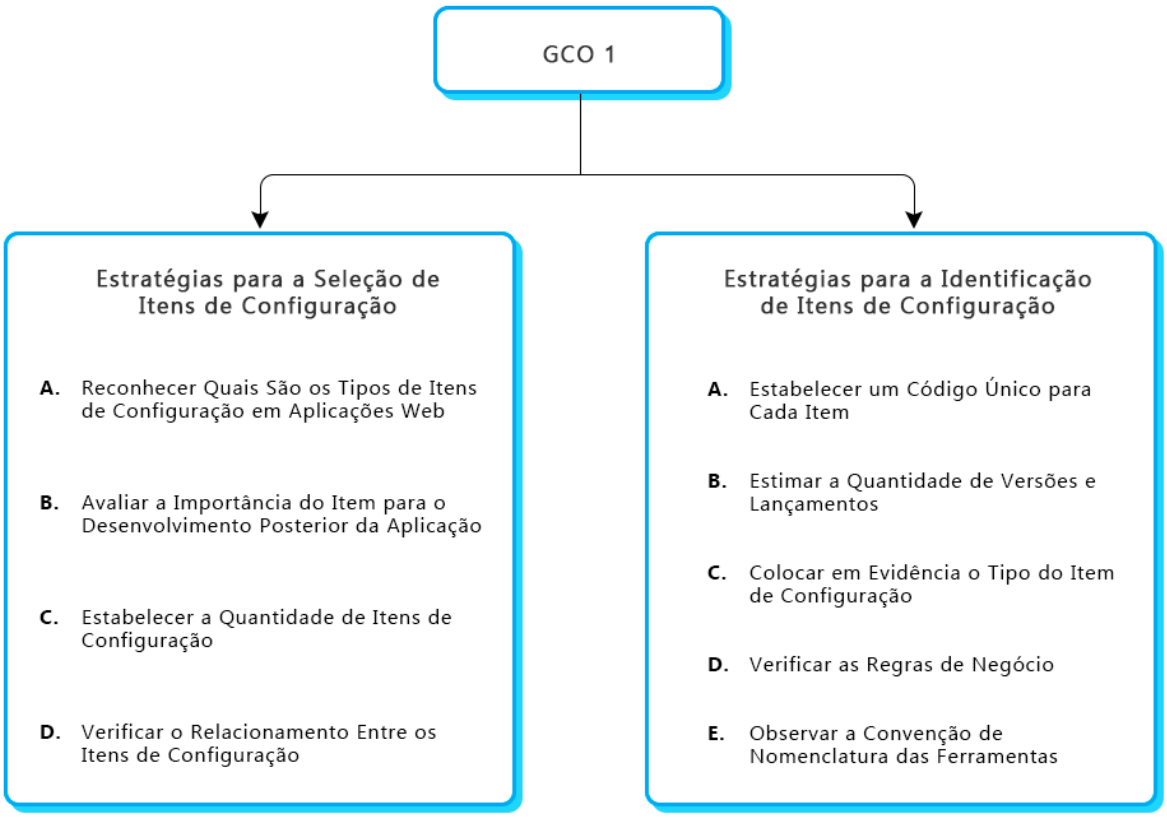


Fonte: O autor.

De fato, seguir os critérios estabelecidos pelo resultado esperado GCO 1 é uma forma de validação para as estratégias que incluem a seleção e identificação de itens de configuração. Porém, não basta apenas a utilização das estratégias gerais do controle de versão, é necessário ir além e adaptar essas estratégias para obter maior efetividade ao lidar com itens de configuração em projetos de aplicações Web. Diante disso, estratégias para a seleção e identificação de itens de configuração são adaptadas de acordo com a natureza das aplicações Web. Logo, a Figura 19 ilustra como os itens de configuração fluem de uma camada para a outra de modo que cada aspecto do controle de versão inclui estratégias refinadas à natureza das aplicações Web.

Nesse cenário, fica claro que a definição das estratégias é um ponto chave. A partir disso, a Figura 20 resume as estratégias alinhadas ao resultado esperado GCO 1.

Figura 20 - Resumo das estratégias para o resultado esperado GCO 1.



Fonte: O autor.

Essas estratégias também determinam os níveis de controle para os itens de configuração, pois definem quais itens devem fazer parte do controle de versão e estabelecem meios para que itens de configuração sejam unicamente identificados, isto é, as estratégias são delineadas de acordo com o tipo do item e as regras de negócio, além de incluir níveis para as versões e lançamentos.

Neste ponto, é necessário levantar os prós e contras. Embora a abordagem esteja alinhada com o resultado esperado GCO 1 e a natureza das aplicações Web, é muito importante entender a nível prático o efeito dessas estratégias. Afinal, “em teoria não há diferença entre teoria e prática, mas, na prática existe.” (Snepscheut Apud Pressam e Maxim, 2016, p. 106).

Nesse sentido, é interessante analisar o uso dessas estratégias por meio de ferramentas do controle de versão. Embora a abordagem ainda não possua essa forma de validação, é um importante caminho a ser traçado. Dito isso, as seções seguintes discutem as estratégias alinhadas ao resultado esperado GCO 1.

5.1.1 Seleção de Itens de Configuração

A seleção de itens de configuração consiste em definir quais itens de configuração devem estar sobre o controle de versão. Para adequar a seleção de itens de configuração a natureza das aplicações Web incluímos as seguintes estratégias:

A. Reconhecer quais são os tipos de itens de configuração em aplicações Web:
    - Quais são os itens de configuração que compõe a aplicação Web?
    - Qual é o tipo de cada item de configuração (p. ex. documento de hipertexto, folha de estilo, documento de banco de dados, etc.)?
    - Qual é o seu formato (p. ex. js, CSS, SCSS, XML, HTML)?

B. Avaliar a importância do item para o desenvolvimento posterior da aplicação:
    - O Item de configuração é necessário para o desenvolvimento posterior da aplicação Web?
    - O Item de configuração tem potencial para provocar falhas no desenvolvimento da aplicação Web?

C. Estabelecer a quantidade de itens de configuração:
    - Quantos itens de configuração devem passar por revisões e processos formais?
    - Quais são os custos para realizar revisões e processos formais (p. ex. tempo, esforços, etc.)?
    - A quantidade de itens de configuração estabelecida pode afetar o desenvolvimento, teste e manutenção da aplicação Web?

D. Verificar o relacionamento entre os itens de configuração:
    - Qual é a relação de dependência entre os itens de configuração?

- A modificação de determinado item de configuração possui qual impacto para outros itens de configuração?

As estratégias definidas para a seleção de itens de configuração reforçam o resultado esperado GCO 1 do modelo MPS.BR como passo inicial. Dito isso, essas estratégias são adaptadas de acordo com a base diversificada de código fonte, escalabilidade e interdependência dinâmica entre os itens de configuração de uma aplicação web.

Com base nas estratégias estabelecidas, é fundamental reconhecer quais são os tipos de itens de configuração em aplicações Web. Para isso, a abordagem sugere verificar qual é o tipo de cada item de configuração e qual é o seu formato.

Um dos principais critérios para definir quais itens devem estar sobre o controle de versão é o seu potencial de contribuição para o desenvolvimento do projeto. Nesse caso, é imprescindível reconhecer a utilidade do item de configuração. Para isso, a abordagem sugere verificar se o item de configuração é de fato necessário para o desenvolvimento posterior da aplicação Web.

Também há outros critérios que podem ajudar na seleção de itens de configuração, dentre os quais é possível mencionar o impacto no cronograma de desenvolvimento e implementação do projeto. Em outras palavras, isso significa que é preciso entender o impacto que um item de configuração pode causar no desenvolvimento e manutenção de uma aplicação Web. Nesse ponto, a abordagem sugere verificar se o item de configuração tem potencial para provocar falhas no desenvolvimento da aplicação Web.

Em relação a esses critérios, ambos se complementam. Em outras palavras, o potencial do artefato para o desenvolvimento posterior da aplicação e o impacto no cronograma do projeto estão relacionados, pois aplicações Web envolvem não apenas um cronograma inicial, mas também lançamentos futuros.

Além disso, é importante observar outros critérios, como o número de mudanças previstas e a complexidade e dimensões do item de configuração. Critérios como esses são determinantes, pois aplicações Web possuem uma ampla diversidade de itens de configuração, além de uma natureza multilíngue. Nesse contexto, é essencial definir não apenas quais os itens de configuração devem estar sobre o controle de versão, mas também levar em consideração a quantidade de itens a ser definida, pois esses itens são postos sobre rastreabilidade e correção de falhas, o que consome tempo e esforços. Nesse sentido, a abordagem sugere estabelecer a quantidade de itens de configuração que devem passar por revisões e processos formais,

determinar quais são os custos para realizar essas revisões e processos formais e verificar se a quantidade de itens de configuração estabelecida pode afetar o desempenho da aplicação Web.

O relacionamento entre os itens de configuração é ponto chave. De fato, é preciso entender a relação de dependência entre os itens de configuração de uma aplicação Web. Logo, a abordagem sugere como estratégia observar se a modificação de determinado item de configuração possui algum impacto em outros itens de configuração.

5.1.2 Identificação de Itens de Configuração

A identificação de itens de configuração está relacionada a convenção de nomenclatura usada para identificar versões de itens de configuração em projetos de aplicações Web. Para adequar a identificação de itens de configuração a natureza das aplicações Web, temos as seguintes estratégias:

A. Estabelecer um código único para cada item;
   - Cada item de configuração possui um código formado por um conjunto de posições que o diferenciam dos demais itens?
   - A convenção de nomenclatura definida é capaz de representar a quantidade estimada de itens da aplicação Web?

B. Estimar a quantidade de versões e lançamentos;
   - Com base em critérios de escalabilidade e evolução contínua, quantas posições são necessárias para definir a quantidade de versões e lançamentos dos itens de configuração da aplicação Web?
   - Há muitos processos formais a serem seguidos para a garantia de qualidade que possam influenciar as versões e os lançamentos?

C. Colocar em evidência o tipo do item de configuração;
   - Há quantos tipos de itens de configuração?
   - Há posições para representar cada tipo de item de configuração?

D. Verificar as regras de negócio;
   - Quais são as funcionalidades relacionadas as regras de negócio da aplicação Web?
   - Quais itens de configuração possuem funcionalidades atreladas as regras de negócio da aplicação Web?

- A convenção de nomenclatura é capaz de representar as regras de negócio da aplicação Web?

E. Observar a convenção de nomenclatura das ferramentas.
   - A documentação da ferramenta possui algum esclarecimento sobre a convenção de nomenclatura dos itens de configuração?
   - A ferramenta foi instalada para a realização de testes de convenção de nomenclatura dos itens de configuração?
   - A ferramenta instalada possui flexibilidade em relação a convenção de nomenclatura dos itens de configuração?

As estratégias definidas para a identificação de itens de configuração estão alinhadas ao resultado esperado GCO 1 do modelo MPS.BR e são adaptadas de acordo com características específicas das aplicações web, incluindo evolução contínua, base diversificada de código fonte e regras de negócio.

A primeira estratégia sugere que cada item de configuração deve possuir um código formado por um conjunto de posições que o diferenciam dos demais itens de configuração e sugere verificar se a convenção de nomenclatura escolhida é capaz de representar a quantidade estimada de itens da aplicação Web. Nesse caso, uma solução de nomenclatura é o uso de rótulos e números de versão para identificar de forma única cada item de configuração.

No entanto, há sistemas de controle de versão que não atribuem significado ao identificador de um item de configuração. Nesse sentido, é fundamental atribuir significado a cada identificador, seja em relação aos incrementos de versão ou as regras de negócio da aplicação Web, pois a finalidade dessa atribuição é facilitar a identificação dos itens de configuração. Dessa forma, diferentes modelos de nomenclatura podem ser definidos e adaptados conforme o projeto. Para isso, a abordagem estabelece estratégias para agregar significado aos identificadores dos itens de configuração.

Aplicações Web possuem características especiais e isso inclui observar a evolução contínua dessas aplicações ao criar um modelo de nomenclatura. Logo, a abordagem sugere estimar a quantidade de versões e lançamentos dos itens de configuração. Dito isso, com base em características como escalabilidade e evolução contínua é essencial definir quantas posições são necessárias para as versões e lançamentos dos itens de configuração de uma aplicação Web. Também é importante observar se há muitos processos formais para a garantia de qualidade, uma vez que esses processos podem influenciar as versões e os lançamentos.

As regras de negócio são determinantes ao estabelecer um modelo de nomenclatura. Nesse contexto, a abordagem sugere verificar quais itens de configuração estão relacionados com as regras de negócio da aplicação Web e observar se a convenção de nomenclatura escolhida é capaz de representar as regras de negócio dessa aplicação.

Além desses pontos, é importante verificar as convenções de nomenclatura que o sistema de controle de versão possui para evitar problemas ao definir um modelo específico de nomenclatura para os itens de configuração. Sabendo disso, a abordagem sugere como estratégia verificar se a documentação da ferramenta possui algum esclarecimento sobre a convenção de nomenclatura dos itens de configuração. Além disso, é importante realizar a instalação da ferramenta para testes de convenção de nomenclatura.

Diante disso, as estratégias definidas para a seleção e identificação de itens de configuração seguem os critérios estabelecidos pelo resultado esperado GCO 1, pois além de identificar os itens de configuração e estabelecer níveis de controle, são formadas a partir de características que formam a natureza única das aplicações Web.

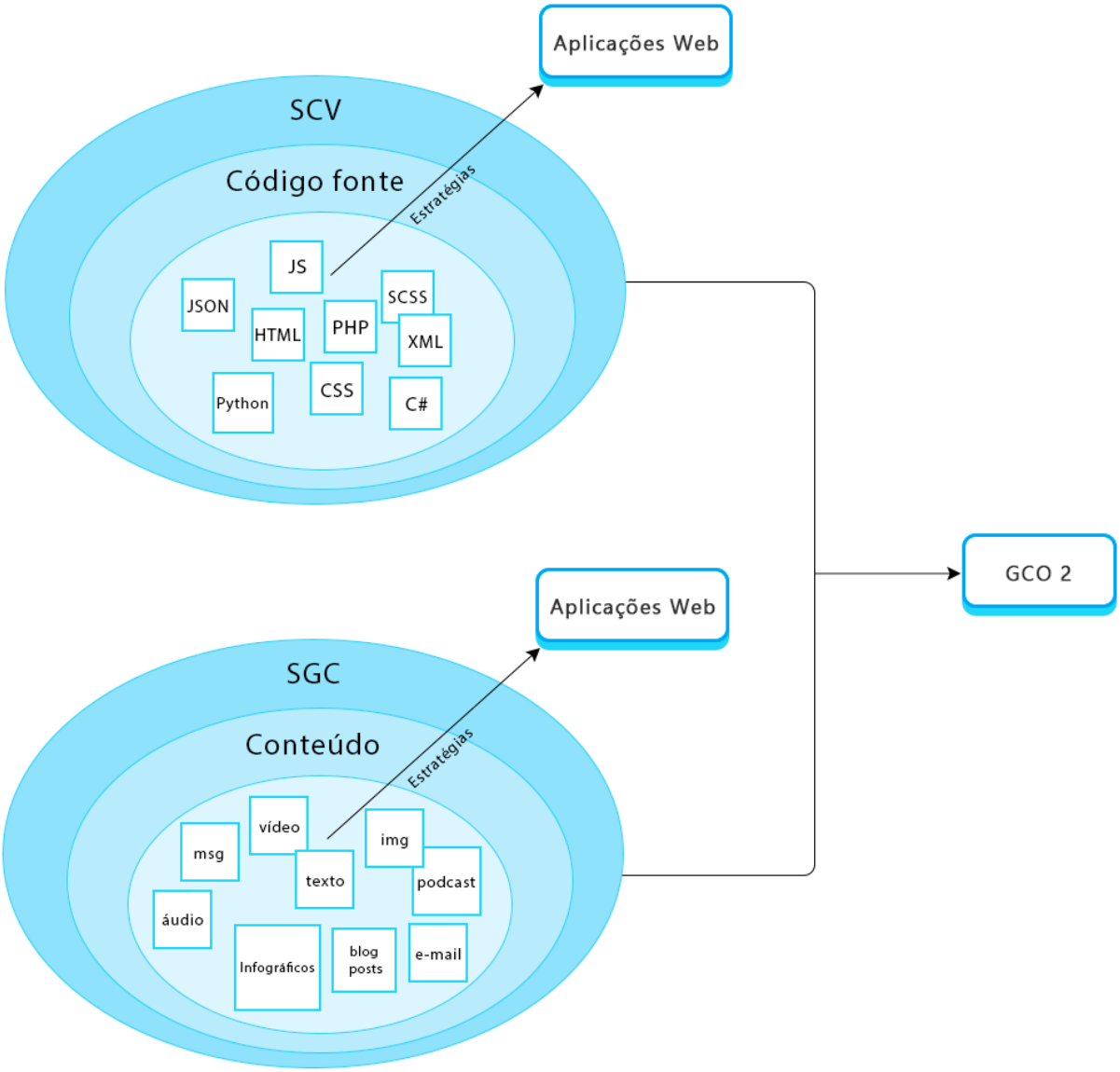
5.2 Resultado Esperado GCO 2

Esta seção discute as estratégias definidas para o resultado esperado GCO 2. Como sabemos, o resultado esperado GCO 2 determina que um sistema para gerencia de configuração e controle de mudanças deve ser estabelecido, mantido atualizado e utilizado. Neste ponto, é essencial esclarecer o significado da abordagem em relação ao resultado esperado GCO 2.

Posto de forma simples, o resultado esperado GCO 2 está relacionado a escolha do SCV e do SGC. Esses aspectos do controle de versão formam uma base estratégica para atingir o resultado esperado GCO 2, pois aplicações Web possuem não apenas código fonte, mas também diversas formas de conteúdo. Nesse sentido, o SCV mantém o controle sobre o código fonte e o SGC gerencia o conteúdo da aplicação Web, conforme ilustra a Figura 21.

É importante dizer que seguir os critérios estabelecidos pelo resultado esperado GCO 2 é uma forma de validação para as estratégias que incluem a escolha do SCV e SGC. No entanto, não basta apenas a utilização das estratégias gerais do controle de versão, é necessário ir além e adaptar essas estratégias para obter maior efetividade ao lidar com itens de configuração em projetos de aplicações Web. Logo, estratégias para a escolha do SCV e SGC são adaptadas de acordo com a natureza das aplicações Web.

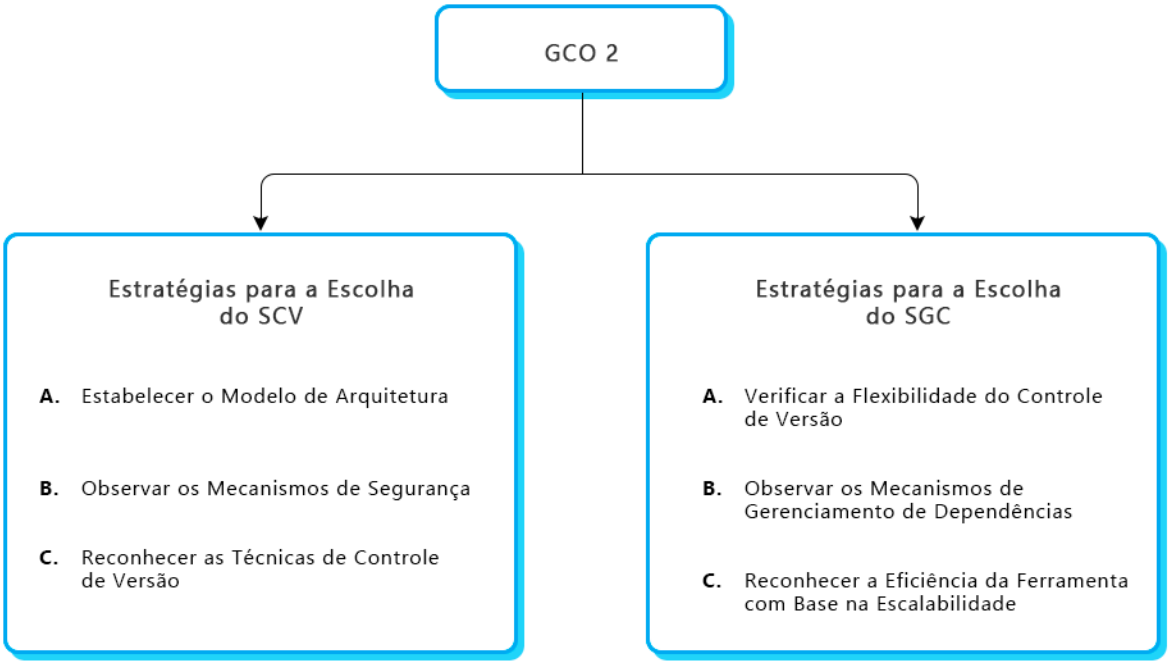
Figura 21 - Composição da abordagem para o resultado esperado GCO 2.



Fonte: O autor.

Posto isso, a Figura 22 resume as estratégias delineadas para o resultado esperado GCO 2.

Figura 22 - Resumo das estratégias para o resultado esperado GCO 2.



Fonte: O autor.

Embora a abordagem esteja alinhada com o resultado esperado GCO 2 e a natureza das aplicações Web, é necessário discutir outro ponto de vista – o efeito dessas estratégias a nível prático. Em outras palavras, é interessante analisar o uso dessas estratégias por meio de ferramentas do controle de versão em um projeto real. Embora a abordagem ainda não possua essa forma de validação, é um importante caminho a ser traçado para trabalhos futuros.

5.2.1 Escolha do SCV

A escolha do SCV consiste em estabelecer qual ferramenta será utilizada e mantida atualizada para manter os itens de configuração de aplicações Web. Para adequar a escolha do SCV a natureza das aplicações Web temos as seguintes estratégias:

A. Estabelecer o modelo de arquitetura;
   - Com base em critérios de confiabilidade e flexibilidade, o modelo de arquitetura do SCV é capaz de atender as necessidades da aplicação Web?
   - O modelo de arquitetura do SCV fornece mecanismos para o desenvolvimento geograficamente distribuído?
   - Quais são as limitações do modelo de arquitetura do SCV?

B. Observar os mecanismos de segurança;
   - Há mecanismos de segurança disponíveis pela ferramenta (p. ex. criptografia e outras camadas de segurança)?
   - A ferramenta permite a integração de extensões, bibliotecas e APIs para estabelecer soluções de segurança?
   - Qual é o modelo de segurança da ferramenta?
   - Quais são as limitações de segurança da ferramenta?

C. Reconhecer as técnicas de controle de versão.
   - Qual é a técnica que o SCV usa para estabelecer a rastreabilidade, criar ramificações e mesclar mudanças em itens de configuração (deltas ou snapshots)?
   - Considerando a escalabilidade e a frequência de mudanças em aplicações Web, a técnica do SCV contribui para a economia de armazenamento?

As estratégias definidas para a escolha do SCV seguem os critérios estabelecidos pelo resultado esperado GCO 2 do modelo MPS.BR e são adaptadas de acordo com determinadas características das aplicações web, incluindo arquitetura do repositório, escalabilidade e segurança.

A arquitetura do repositório é um critério determinante para a escolha do SCV, pois essas ferramentas diferem em seu nível flexibilidade. Logo, a abordagem sugere entender quais são as limitações do modelo de arquitetura do SCV e se o modelo de arquitetura é capaz de atender as necessidades do projeto com base na sua flexibilidade.

Além disso, as aplicações Web possuem itens de configuração que requerem determinados cuidados ao serem postos sobre o controle de versão como, por exemplo, senhas e chaves de acesso. Por essa razão, a abordagem sugere observar se a ferramenta fornece funções de criptografia e outras camadas de segurança. Da mesma forma, é essencial reconhecer as limitações do modelo de segurança da ferramenta e se é permitido a integração de extensões, bibliotecas e APIs para estabelecer soluções de segurança.

À medida que uma aplicação Web evolui, muitas versões podem existir simultaneamente. Nesse cenário, o SCV se torna uma ferramenta essencial para gerenciar funções de controle como check-in, check-out e mesclagens de código, além de manter diferentes versões da aplicação Web. Muitos sistemas de controle de versão estão disponíveis e diferem em relação ao método para criar versões. Sabendo disso, a abordagem sugere reconhecer qual é a técnica que o SCV usa para estabelecer a rastreabilidade, criar ramificações

e mesclar mudanças em itens de configuração. Outrossim, a abordagem sugere identificar se a técnica do SCV contribui para a economia de armazenamento ao considerar características como escalabilidade e a frequência de mudanças em aplicações Web.

5.2.2 Escolha do SGC

A escolha do SGC consiste em estabelecer qual ferramenta será utilizada e mantida atualizada para gerenciar o conteúdo de uma aplicação Web. Para estabelecer a escolha do SGC temos as seguintes estratégias:

A. Verificar a flexibilidade do controle de versão no SGC;
   - Há mecanismos para o funcionamento adequado do controle de versão considerando a natureza distribuída das aplicações Web?
   - Há controles efetivos para criar e acessar versões de conteúdo considerando o comportamento assíncrono das aplicações Web?

B. Observar os mecanismos de gerenciamento de dependências;
   - Há mecanismos para gerenciar as dependências entre as várias formas de conteúdo das aplicações Web?
   - Há mecanismos efetivos para o controle de versão em meio a natureza dinâmica do conteúdo?

C. Reconhecer a eficiência da ferramenta com base na escalabilidade.
   - À medida que a demanda aumenta é possível manter adequadamente o controle de versões para o conteúdo das aplicações Web?
   - Fatores relacionados a custos e serviços necessários para a execução da ferramenta prejudicam o gerenciamento do conteúdo e suas versões?

As estratégias definidas para a escolha do SGC são determinantes para adequar o controle de versão a natureza das aplicações Web. Estas estratégias estão alinhadas ao resultado esperado GCO 2 do modelo MPS.BR e são adaptadas de acordo com algumas características das aplicações web, incluindo natureza assíncrona e distribuída, diferentes formas de conteúdo, dependências e escalabilidade.

Como sabemos, aplicações Web possuem conteúdo a ser gerenciado. Logo, ferramentas específicas devem ser usadas para estabelecer o gerenciamento de conteúdo nessas aplicações. Em um SGC, a flexibilidade é um critério essencial. Desse modo, abordagem sugere verificar a flexibilidade do controle versão no SGC. Em outras palavras, isso significa verificar se a

ferramenta possui mecanismos para o funcionamento adequado do controle de versão considerando a natureza distribuída e assíncrona das aplicações Web.

Ao estabelecer a escolha do SGC é importante observar os mecanismos de gerenciamento de dependências. Nesse contexto, a abordagem sugere verificar se o SGC possui mecanismos efetivos para gerenciar as dependências entre as várias formas de conteúdo das aplicações Web e se é possível realizar o controle de versão em meio a natureza dinâmica do conteúdo e suas dependências.

Com efeito, aplicações Web envolvem uma ampla diversidade e quantidade de conteúdo. Portanto, para gerenciar esse conteúdo é necessário funções de controle efetivas, pois quanto maior a quantidade de conteúdo, mais difícil é manter essas aplicações. A partir dessa premissa, a abordagem sugere reconhecer a eficiência da ferramenta com base na escalabilidade. Em outras palavras, é preciso determinar se à medida que a demanda aumenta é possível manter adequadamente o controle de versão para o conteúdo de uma aplicação web. Outrossim, é fundamental identificar se fatores relacionados a custos e serviços para a execução da ferramenta prejudicam o gerenciamento de conteúdo dessas aplicações.

Embora haja muitos sistemas de sistemas de gerenciamento de conteúdo disponíveis (p. ex. WordPress, Joomla! e Drupal) é necessário observar de forma estratégica as diferenças entre as funcionalidades e serviços de cada SGC para definir a melhor solução para uma aplicação Web. O gerenciamento de conteúdo é uma forma refinada de controle versão para o conteúdo de aplicações Web. Logo, incluir o gerenciamento de conteúdo é um passo essencial para adequar o controle de versão a natureza das aplicações Web.

Mas afinal, qual é a relação entre SCV e SGC? Basicamente, o código fonte da aplicação é mantido por um SCV e o conteúdo é mantido por um SGC. Segundo Barker (2016), quase sempre o conteúdo muda de forma mais frequente que o código fonte da aplicação, ou seja, é possível publicar e modificar o conteúdo muitas vezes por dia, mas o ajuste do código fonte, é feito somente em alguns meses. Laporte e April (2018), reforçam esse entendimento quando esclarecem que os itens de configuração estão sujeitos a um processo de revisão formal que envolve a rastreabilidade, correção e verificação de falhas.

Com base nisso, não basta apenas o uso de sistemas de controle de versão em projetos de aplicações Web, pois além do código fonte existe o conteúdo da aplicação Web. Esse conteúdo envolve fatores como diversidade, quantidade e frequência de mudanças, os quais podem dificultar ainda mais os esforços para manter uma aplicação Web. Daí a o sentido para o uso complementar entre SCV e SGC.

6 CONCLUSÃO

Este trabalho delineou estratégias para adequar o controle de versão a natureza das aplicações Web. Com efeito, essas aplicações possuem características únicas e envolvem uma ampla diversidade de itens de configuração e conteúdo, além de fatores como quantidade e frequência de mudanças para esses itens. Dito isso, embora o controle de versão seja uma atividade imprescindível para o desenvolvimento e manutenção de sistemas de software, fica claro que as estratégias gerais do controle de versão precisam ser adaptadas para estar em conformidade com a natureza das aplicações Web.

Diante disso, este trabalho concentrou esforços para estabelecer estratégias alinhadas aos resultados esperados GCO 1 e GCO 2 do modelo MPS.BR para adequar o controle de versão à natureza das aplicações Web. Cada um desses resultados esperados está relacionado a determinados aspectos do controle de versão. Nessa composição, o resultado esperado GCO 1 está relacionado a seleção e a identificação de itens de configuração. Por sua vez, o resultado esperado GCO 2 está relacionado a escolha do SCV e do SGC.

Em um projeto de software, os itens de configuração precisam ser identificados, conforme determina o resultado esperado GCO 1. Para isso, é necessário definir de forma estratégica quais itens de configuração devem fazer parte do controle de versão e adotar convenções de nomenclatura para que cada versão de item de configuração possa identificada de forma única e com significado para o modelo de nomenclatura de acordo com as características dessas aplicações. Nesse ponto, é possível concluir que as estratégias definidas para a seleção e identificação de itens de configuração se complementam para adequar o controle de versão à natureza das aplicações Web.

Para fornecer funções de controle capazes de gerenciar efetivamente os itens de configuração dessas aplicações é necessário a utilização de ferramentas específicas conforme determina o resultado esperado GCO 2. Embora o uso de sistemas de controle de versão seja essencial para manter o código fonte e outros itens de configuração, as aplicações Web podem envolver uma ampla diversidade e quantidade de conteúdo. Portanto, é necessário agregar soluções como o uso de sistemas de gerenciamento de conteúdo. Nesse ponto, é possível concluir que as estratégias definidas para a escolha do SCV e do SGC se complementam para adequar o controle de versão à natureza das aplicações Web.

Vale dizer que a nível de trabalhos futuros é possível delinear o efeito dessas estratégias através do uso de ferramentas do controle de versão em um projeto real para aplicações Web. Além disso, é interessante abordar outros aspectos do controle de versão que possuam relação

com aplicações Web como, por exemplo, o modo pelo qual as metodologias utilizadas para o desenvolvimento de aplicações Web, como métodos ágeis e entrega contínua podem influenciar o controle de versão.

Em suma, este trabalho busca trazer contribuições para desenvolvedores e demais profissionais da área sobre a forma como o controle de versão é utilizado em projetos de aplicações Web. A mudança é parte do desenvolvimento dessas aplicações, seja em relação ao código fonte ou ao conteúdo, e cada versão de um item de configuração implementa uma nova mudança. Logo, muitas versões podem surgir ao mesmo tempo e criar confusões no desenvolvimento dessas aplicações, que além de dinâmicas, possuem uma base heterogênea de código e outras particularidades já mencionadas. Nesse sentido, é essencial estabelecer não apenas de forma genérica o controle de versão, mas adaptar as estratégias dessa atividade para estar de acordo com as características das aplicações Web.

REFERÊNCIAS

ADAM, Khleel Nasraldeen Alnor; KÁROLY, Nehéz. Comparison of version control system tools. **Multidiszciplináris Tudományok**, v. 10, n. 3, p. 61-69, 2020.

ANICHE, Maurício F. Detection strategies of smells in Web software development. In: **2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. IEEE, 2015. p. 598-601.

ARORA, Tarun; SHIGIHALLI, Utkarsh. **Azure DevOps Server 2019 CookBook**. Packt Publishing, 2019.

ASSOCIAÇÃO PARA PROMOÇÃO DA EXCELÊNCIA DO SOFTWARE BRASILEIRO – SOFTEX. **MPS.BR – Guia Geral MPS de Software:2021**, janeiro 2021. Disponível em: www.softex.br.

Azure DevOps: About security, authentication and authorization. **Microsoft**, 2021. Disponível em: <https://docs.microsoft.com/pt-br/azure/devops/organizations/security/about-security-identity?view=azure-devops>. Acesso em: 28 de agosto de 2021.

BARKER, Deane. **Web content management: Systems, features, and best practices**. O'Reilly Media, Inc., 2016.

BOTELHO, Joacy M.; CRUZ, Vilma Aparecida G. **Metodologia Científica**. São Paulo: Pearson Education do Brasil, 2013.

BUKHARI, Syed Shafique Ali et al. Improving Requirement Engineering Process for Web Application Development. In: **2018 12th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)**. IEEE, 2018. p. 1-5.

CHACON, Scott; STRAUB, Ben. **Pro git**. Second Edition. Apress, 2014.

DAWEI, Xiao et al. Web Application Automatic Testing Solution. In: **2016 3rd International Conference on Information Science and Control Engineering (ICISCE)**. IEEE, 2016. p. 1183-1187.

DEL SOLE, Alessandro. **Beginning Visual Studio for Mac: Build Cross-Platform Apps with Xamarin and. NET Core**. Apress, 2017.

DEEPA, N. et al. An analysis on Version Control Systems. In: **2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)**. IEEE, 2020. p. 1-9.

GALIN, Daniel. **Software quality: concepts and practice**. John Wiley & Sons, 2018.

GEORGE, Nigel. **Beginning django CMS**. Apress, 2015.

GRIGERA, Julián et al. Live versioning of Web applications through refactoring. In: **Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering**. 2018. p. 872-875.

GUPTA, Shivangi; DHIR, Saru. Issues, Challenges and Estimation Process for Secure Web Application Development. In: **2016 Second International Conference on Computational Intelligence & Communication Technology (CICT)**. IEEE, 2016. p. 219-222.

HIRAMA, Kechi. **Engenharia de software: qualidade e produtividade com tecnologia**. Rio de Janeiro: Elsevier, 2012.

HORCAS, Jose-Miguel et al. Integrating the common variability language with multilanguage annotations for Web engineering. In: **Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1**. 2018. p. 196-207.

HORSMAN, Graeme. Web Content Management Systems: an analysis of forensic investigatory challenges. **Journal of forensic sciences**, v. 63, n. 5, p. 1392-1400, 2018.

How to Handle Security Issues. **Subversion**, 2018. Disponível em: <https://subversion.apache.org/docs/community-guide/issues.html#security>. Acesso em: 26 de agosto de 2021.

JIRAPANTHONG, Waraporn. Requirements traceability on Web applications. In: **2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)**. IEEE, 2015. p. 18-23.

JYANI, Jai Prakash; BARWAR, N. C. A Collaborative Versioning Framework for Model-Based Version Control Systems. In: **Proceedings of International Conference on Communication and Computational Technologies**. Springer, Singapore, 2019. p. 623-639.

KHAN, Khansa et al. A Meta-model For Software Project Change and Configuration Management. In: **Proceedings of the 9th International Conference on Information Communication and Management**. 2019. p. 12-16.

KIATPHAO, Parichat; SUWANNASART, Taratip. Version Control on Database Schema and Test Cases from Functional Requirements' Input Changes. In: **IMECS**. 2017.

KUDAISI, Mobolaji. **Web Applications Content Management System**. 2017.

LAPORTE, Claude Y.; APRIL, Alain. **Software quality assurance**. John Wiley & Sons, 2018.

LAVERDIÈRE, Marc-André; MERLO, Ettore. Detection of protection-impacting changes during software evolution. In: **2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)**. IEEE, 2018. p. 434-444.

LI, Shu; TSUKIJI, Hayato; TAKANO, Kosuke. Analysis of software developer activity on a distributed version control system. In: **2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)**. IEEE, 2016. p. 701-707.

LUNYOV, Phillip. **Detecting changes in Web applications**. Dissertation, 2020.

MAGANA, Alex; MULI, Joseph. **Version control with Git and GitHub**. Packt Publishing, 2018.

MALEKI, Nasrin Ghasempour; RAMSIN, Raman. Agile Web development methodologies: a survey and evaluation. In: **International Conference on Software Engineering Research, Management and Applications**. Springer, Cham, 2017. p. 1-25.

MESBAH, Ali. Software analysis for the Web: Achievements and prospects. In: **2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)**. IEEE, 2016. p. 91-103.

NEELAKANTAM, Srushtika; PANT, Tanay. **Learning Web-based virtual reality: build and deploy Web-based virtual reality technology**. Apress, 2017.

O'REGAN, Gerard. **Concise guide to software testing**. Springer International Publishing, 2019.

PAEZ, Nicolás. Versioning Strategy for DevOps Implementations. In: **2018 Congreso Argentino de Ciencias de La Informática y Desarrollos de Investigación (CACIDI)**. IEEE, 2018. p. 1-6.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Software engineering: a practitioner's approach**. 8th ed. New York: McGraw-Hill Education, 2015.

RIO, Américo; E ABREU, Fernando Brito. Analyzing Web applications quality evolution. In: **2017 12th Iberian Conference on Information Systems and Technologies (CISTI)**. IEEE, 2017. p. 1-4.

RUDEK, Krzysztof et al. **Distributed version control for tracking changes in Web applications**. U.S. Patent Application n. 16/209,309, 4 jun. 2020.

Security considerations for IBM Rational Synergy. **IBM**, 2021. Disponível em: <https://www.ibm.com/docs/en/rational-synergy/7.2.1?topic=overview-security-considerations>. Acesso em: 27 de agosto de 2021.

SIPOS, Daniel. **Drupal 9 Module Development: Get up and running with building powerful Drupal modules and applications**. Packt Publishing Ltd, 2020.

SOMMERVILLE, Ian. **Software Engineering**. 10th ed. Boston: Pearson, 2016.

TSITOARA, Mariot. **Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer**. Berkeley: Apress, 2020.

TSUI, Frank F.; KARAM, Orlando; BERNAL, Barbara. **Essentials of software engineering**. Jones & Bartlett Learning, 2018.

Uma ferramenta bash para armazenar seus dados privados dentro de um repositório git. **Git-secret**, 2021. Disponível em: < https://git-secret.io>. Acesso em: 30 de agosto de 2021.

UZUNBAYIR, Serhat; KURTEL, Kaan. A Review of Source Code Management Tools for Continuous Software Development. In: **2018 3rd International Conference on Computer Science and Engineering (UBMK)**. IEEE, 2018. p. 414-419.

VEMULA, Rami. **Real-Time Web Application Development: With ASP. NET Core, SignalR, Docker, and Azure**. Apress, 2017.

HARWANI, Bintu. **Foundations of Joomla!**. Apress, 2015.

WAZLAWICK, Raul S. **Engenharia de software: conceitos e práticas**. Rio de Janeiro: Elsevier, 2013.

Why use BitKeeper when there are lots of great alternatives?. **BitKeeper**, 2021. Disponível em: <https://www.bitkeeper.org/why.html>. Acesso em: 29 de agosto de 2021.

WILLIAMS, Brad; TADLOCK, Justin; JACOBY, John James. **Professional WordPress Plugin Development**. John Wiley & Sons, 2020.

XU, Xin et al. Enforcing Access Control in Distributed Version Control Systems. In: **2019 IEEE International Conference on Multimedia and Expo (ICME)**. IEEE, 2019. p. 772-777.

YADAV, D. et al. Vulnerabilities and Security of Web Applications. In: **2018 4th International Conference on Computing Communication and Automation (ICCCA), India, prosinac**. 2018.

ZOLKIFLI, Nazatul Nurlisa; NGAH, Amir; DERAMAN, Aziz. Version Control System: A Review. **Procedia Computer Science**, v. 135, p. 408-415, 2018.