PROPOSTA DE MELHORIA PARA GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE EM UMA PEQUENA EMPRESA

TIAGO UEMURA

GISLAINE CAMILA LAPASINI LEAL

Resumo

As mudanças em um software ocorrem por diversos fatores como implantação de novas funcionalidades, correção de bugs, adaptação a novas interfaces e tecnologias, entre outros. A falta de controle sobre as mudanças gera falta de informações e pode ocasionar em retrabalho e lentidão de desenvolvimento pela equipe. O objetivo geral deste trabalho é apresentar uma proposta de melhoria para gerência de configuração de software em uma empresa de desenvolvimento de software situada no estado do Paraná, para tornar os processos de desenvolvimento mais confiáveis e estáveis, mantendo a agilidade na entrega ao cliente e capacidade de atender a novas oportunidades. Planejar os processos no mercado atual significa conter custos e aumentar o potencial de escalar a empresa a nível de grande porte. Este trabalho faz a análise do processo de desenvolvimento, observando os itens que compõem o software para poder priorizar aqueles que precisam de um controle mais rígido de alteração, versionamento, histórico e entregas.

Palavras-chave: *Engenharia de Software; Qualidade de Software, Gerência de Configuração de Sofware.*

1. Introdução

No campo da engenharia de software, pesquisadores e profissionais da área produziram um extenso conhecimento e práticas de desenvolvimento de software ao longo dos anos que conduziram as empresas a terem sucesso em seus produtos e processos. Diversos modelos de desenvolvimento foram elaborados conhecidos como Métodos Clássicos de Desenvolvimento (p.e. Modelo em Cascata, Modelos em Espiral e processos iterativos) que focam no planejamento e na elaboração de documentação do software. Porém, a evolução do mercado e o impacto gerado nas necessidades de desenvolvimento gerou demanda para novas metodologias de desenvolvimento, onde em projetos de desenvolvimento de software altamente dinâmico com potencial de escalar, tanto em recursos quanto em performance, deve-se focar em atender as demandas atuais em um tempo razoável, mesmo que tenha que sacrificar burocracia para correr riscos e ter retrabalhos além de perder qualidade no produto, decorrente desta situação surge os Métodos Ágeis de Desenvolvimento (DIAS, 2005).

As metodologias ágeis vêm ganhando mais espaço no mercado profissional, ainda que com pouca comprovação técnica de sua eficácia, sua aplicabilidade tem se manifestado principalmente no que defende a metodologia: o foco na satisfação do cliente por meio de entregas rápidas e contínuas, equipes pequenas com alto grau de comunicação, alto nível de alterações de requisitos, foco na excelência e qualidade de projeto que aprimoram a agilidade. (DIAS, 2005)

Porém, mesmo adotando metodologias de gestão ágeis, alguns requisitos de processos de softwares não devem ser deixados de lados afim de evitar desperdícios e riscos, e deixar passar oportunidades de negócio. Neste sentido, este trabalho procura realizar um estudo aplicado à uma empresa de desenvolvimento de software, afim de encontrar quais processos, já estabelecidos na literatura do campo de Engenharia de Software, possuem valor indispensáveis para sustentação de projetos considerados complexos, de alta incerteza, alterações frequentes nos processos e imprevisíveis, de forma a conseguir adotar medidas rápidas de implantação e escalar as oportunidades do mercado.

Segundo o SWEBOK (2014), as áreas de conhecimento da Engenharia de Software são: requisitos de software, projeto de software, construção de software, teste de software, manutenção de software, gerência de configuração de software, gerência de engenharia de software, processos de engenharia de software, ferramentas e métodos de engenharia de software e qualidade de software. Cada uma das áreas contribui para organização, produtividade e qualidade do processo de desenvolvimento de software.

A empresa desenvolve os conceitos da metodologia ágil, principalmente no foco com o a comunicação aberta entre a equipe, a divisão de tarefas por meio do Scrum, a aplicação do *Extreme Programming* entre outros. A formalidade em documentação e desenvolvimento é requerida em muitas etapas, porém pela falta de experiência dos membros e pela extrema sobrecarga, estes processos são deixados de lado para serem feitos mais à frente. O problema é que a solução desenvolvida está em crescente ascensão no mercado e a manutenção tem se tornado cada vez mais caótico. Além disso, a expansão da equipe para aumentar a capacidade do projeto tem se tornado difícil devido à falta de formalidade de documentação do software onde os membros novos tem dificuldade em entender os processos, códigos e estruturas.

Os processos podem variar de acordo com as características de cada projeto e para melhor aplicar os conceitos, neste trabalho será feito um estudo da literatura sobre o tema e um levantamento do valor que a empresa enxerga em seu produto, desde esclarecer a estratégia da

empresa até mapear os processos técnicos envolvidos, desta forma, é possível definir com maior precisão o valor de cada prática e adotar aquelas que são de interesse da empresa.

2. Revisão da literatura

2.1 Qualidade de Software (QS)

Frente às novas tecnologias e complexidades na produção de software da época, a Crise do Software (década de 70) evidenciou as dificuldades do desenvolvimento de software para lidar com o rápido crescimento da demanda e manter os prazos, orçamentos e qualidade do produto. A partir de então, a engenharia de software se preocupou em desenvolver melhores processos, modelos e metodologias em prol de uma qualidade satisfatória de Software. Foram criadas, diversas subáreas para problemas específicos, entre elas a Qualidade de Software (PRESSMAN, 2016; SOUZA, 2009; GIBBS, 1994; DIJKSTRA, 1972).

Qualidade de Software (QS) é um termo que sua percepção varia de acordo com a visão dos atores envolvidos com o software. Segundo a *Institute of Electrical and Electronics Engineers (IEEE)* em seu guia SWEBOK (2014), Qualidade de Software (QS) pode ser entendido como o grau de conformidade no qual um sistema, componente ou processo atende os requisitos, necessidades e expectativas do cliente/usuário. De forma geral, as especificações do produto são orientadas conforme os interesses do cliente, porém existem os requisitos próprios da empresa que também estão na especificação do projeto como por exemplo a manutenibilidade e eficiência, e segundo Sommerville (2011), a qualidade nos processos de desenvolvimento influenciam na capacidade de atender os requisitos dos clientes. Nesta concepção, Marques (2008) avalia que as medidas de qualidade de software provêm dos atributos do produto como também das etapas de produção, e ambos devem buscar a melhoria contínua.

Processo e produto de qualidade o aumentam produtividade e confiabilidade, melhoram a precisão nas estimativas de projeto, reduzem defeitos no produto, custo de desenvolvimento e manutenção, tempo de atender o mercado, promovem maior índice de satisfação do cliente/usuário, entre outros benefícios (PRESSMAN, 2016; SOMMERVILLE, 2011, MAGALHÃES 2008; SOUZA e MONTEIRO, 2009; ROCHA, 2001).

Diversas organizações criaram normas e padrões para garantir a qualidade de software (GQS) no processo de desenvolvimento de software, como por exemplo, a *International Organization for Standardization* (ISO) produziu a norma ISO/IEC 9126-1[1] que define modelos

de avaliação da qualidade de software e sistemas, a ISO/IEC 12207 relacionada ao ciclo de vida de software, a ISO/IEC 15504 aos processos de desenvolvimento , *Institute of Electrical and Electronics Engineers* (IEEE) , a *Software Engineering Institute* (SEI) elaborou o CMMI – *Capability Maturity Model Integration*, entre outros. As diversas normas abordam vários elementos de garantia da qualidade de software (GQS), Pressman (2016) sintetiza alguns como: padrões, revisões e auditorias, testes, coleta e análise de erros/defeitos, gerenciamento de mudanças, educação, gerência de fornecedores, administração da segurança, proteção e administração de riscos.

Assim como outros modelos, o CMMI (*Capability Maturity Model Integration*), definido pela SEI, definiu um modelo, dentre vários outros processos de software, de Gerenciamento de Configuração de Software (GCS) que contribui para a qualidade do projeto de software pelo suporte às atividades gerenciais e técnicas; No entanto, ainda existe a dificuldade das em adotar o processo de GCS de qualidade, pois exige investimentos em recursos financeiros e humanos, escassos principalmente para pequenas organizações.

2.2 Gerência de Configuração de Software

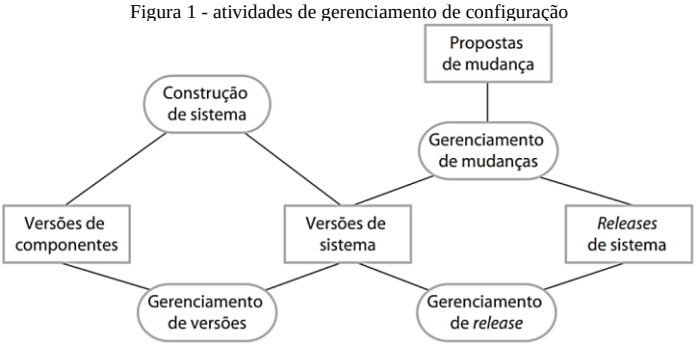
A indústria aeroespacial norte-americana dos anos 50 controlavam suas modificações em documentos de produção de aviões de guerra e naves espaciais, foi onde surgiu a Gerência de Configuração (GC) e posteriormente, nos anos 60 e 70, a GC passou a abranger artefatos de software, indo além dos artefatos de hardware já estabelecidos e desencadeando o surgimento da Gerência de Configuração de Software (GCS) (HASS, 2003; ESTUBLIER et al., 2005).

A Gerência de Configuração de Software (GCS) define procedimentos administrativos e técnicos para identificar e controlar os itens entregáveis (Itens de Configuração IC) de um software além de gerenciar o processo de mudanças, armazenamento e entregas das versões destes itens (WEBER et al, 2001). Segundo Pressman (2016), o gerenciamento de configuração de software é um conjunto de atividades de controle e rastreamento de itens desde o início do desenvolvimento até a retirada do software de operação.

Segundo Sommerville (2011) os sistemas mudam por diversos motivos, como por exemplo novas tecnologias de hardware, novas plataformas de sistemas, inovação pelo mercado concorrente, requisições dos usuários entre outros. As mudanças nos itens de software geram novas versões de um sistema que devem ser mantidas e gerenciadas. O autor divide quatro atividades que envolvem a realização do GSC conforme as elipses da

Figura 1:

Figura 1 - atividades de gerenciamento de configuração



Fonte: sommerville (2011)

I. Gerenciamento de Mudanças: O registro das alterações realizadas nos itens de configuração do software permite controle sobre o desenvolvimento. As informações de registro podem ser: as funcionalidades adicionadas, removidas ou modificadas, os bugs corrigidos, pendências de correção, data, versão entre outros. Há também as políticas de controle de acesso que gerencia as autorizações para acessar e modificar um objeto de configuração particular. O controle de sincronização são as políticas para de alterações paralelas que asseguram a não sobreposição de um arquivo à outro.

II. Gerenciamento de versões: É o processo de acompanhamento de diferentes versões de componentes de software ou itens de configuração e os sistemas em que esses componentes são usados. Ele também garante a concorrência de mudanças feitas por diferentes desenvolvedores sem interferência umas nas outras por meio do gerenciamento de *codelines* e *baselines*. Uma *codeline* é uma sequência de versões de código-fonte de um item de configuração. Uma *baseline* é a definição do sistema formado por diversos itens de configuração. Uma *baseline* é a definição do sistema formado por diversos itens de configuração. Diferentes *baselines* usam versões diferentes dos componentes de cada *codeline*. A mainline é uma sequência de versões de sistema geradas por uma *baseline* original. Um identificador permite definir quais componentes pertencem a uma versão da *baseline*.

Excluído: ¶

III. Construção do sistema: é o processo da criação de um sistema completo, interligando os componentes. Pode haver três plataformas de sistemas envolvidos neste processo: o sistema de desenvolvimento (Compiladores, editores de código-fonte, etc) o servidor de construção (usado para construir versões definitivas e executáveis) e o ambiente-alvo (plataforma de execução). Uma ferramenta de construção do sistema pode fornecer a

Geração de script de construção (arquivo de configuração), a integração de gerenciamento de versões, a recompilação mínima, criação de sistemas executáveis, automação de testes, emissão de relatórios e geração de documentação.

IV. Gerenciamento de releases: A entrega do todo ou a parte do software é chamado de um Release. Gerenciar os *releases* significa manter informações sobre da versão liberada para os diferentes clientes cada um com uma versão específica do software. Existem dois tipos de releases, os principais que são para entregas com novas funcionalidades, e os secundários que entregam correções de bugs e falhas apontado pelos clientes (Sommerville, 2011). Segundo o autor, as atualizações do software podem ser feitas pelo cliente quando o mesmo desejar ou automaticamente pela empresa fornecedora do software, afim de entregar consistência no software adquirido.

2.2.1 Gerenciamento de Qualidade

Segundo Weber, Rocha e Nascimento (2001) o processo de garantia da qualidade verifica se o produto atende os padrões e requisitos dos clientes, de forma independente as questões de desenvolvimento, sem sofrer influências que possam afetar o julgamento. Os autores destacam que esta área definem padrões e processos de garantia do produto e processo de desenvolvimento durante todo o ciclo de vida do projeto. Além da técnica definida é importante desenvolver a cultura nos responsáveis pelo desenvolvimento para se comprometerem em alcançar um alto nível de qualidade de produto" (Sommerville, 2011). Dessa forma, é necessário que as equipes assumam a responsabilidade pelo seu trabalho, independente da fase de desenvolvimento, iniciando com o levantamento de requisitos, passando pela codificação, até os testes e entrega do sistema.

O SWEBOK (2014), divide a Gerência de Configuração de Software (GCS) em seis subáreas:

I. Gerência do Processo de GCS: destaca os planos iniciais e a contextualização do ambiente em que se aplicará o GSC (Organização, recursos, estratégias e tempo)

II. Identificação de itens a serem controlados: estabelece esquemas de identificação, versões, instrumentos e técnicas.

III. Controle de Configuração de Software: Abrange o processo para determinar quais as mudanças a fazer, a autoridade para aprovar as mudanças, o apoio para a

Excluído: <#>**Gerenciamento de Mudanças**¶
<#>O registro das alterações realizadas nos itens de configuração do software permitem controle sobre o desenvolvimento. As informações de registro podem ser: as funcionalidades adicionadas, removidas ou modificadas, os bugs corrigidos, pendências de correção, data, versão entre outros (Pressmann, 2016).¶
<#>Gerenciam*ento de releases*A entrega do todo ou a parte do software é chamado de um Release. Gerenciar as releases significa manter informações sobre da versão liberada para os diferentes clientes cada um com uma versão específica do software. Existem dois tipos de releases, os principais que são para entregas com novas funcionalidades, e os secundários que entregam correções de bugs e falhas apontado pelos clientes (Sommerville, 2011). Segundo o autor, as atualizações do software podem ser feitas pelo cliente quando o mesmo desejar ou automáticamente pela empresa fornecedora do software, afim de entregar consistencia no software adquirido. ¶

Formatado: Fonte: Itálico

Comentado [u1]: Colocar o texto nos itens acima.. definir o que é construção do sistema.. e gerenciamento de versões..

Excluído: ¶

Excluído: ¶
¶
¶
¶
¶

implementação dessas mudanças, e o conceito de desvios formais de requisitos do projeto, bem como a renúncia a eles.

IV. Registros de Estado de Configuração de Software: define informação sobre o estado de configuração do software e forma de reportar este estado as mudanças, validações, desvios e aprovações.

V. Auditoria de Configuração de Software: composto pela Auditoria de Configuração Funcional, Auditoria de Configuração Física e Auditorias de Processo a partir de uma base de referência do software.

VI. Gerência de Liberação e Entrega de Software: elaboração de versões de software e gerenciamento de liberação de software.

3. MÉTODO DE PESQUISA

O Estudo se deu em uma empresa de desenvolvimento de software, por meio da participação do ambiente de estudo, onde se desenvolve uma solução de análise de dados voltada a planejamento orçamentário de empresas que possuem um sistema de controle de movimentação financeira.

Segundo Gerhardt (2009), a pesquisa científica tem o objetivo de resolver um problema por meio da interpretação de exames minuciosos e procedimentos científicos, sistemático e intensivo. Para atingir os objetivos do trabalho, além da pesquisa bibliográfica, realizou-se uma Pesquisa Aplicada Qualitativa e Estudo de Caso acerca da aplicação das atividades da engenharia de software mais especificamente na disciplina de gerenciamento de configuração de software.

Excluído: de modelos clássicos alinhados a métodos ágeis de desenvolvimento…

3.1 Pesquisa Aplicada

Excluído: Primeiramente, o

Segundo Gerhardt (2009), a pesquisa aplicada objetiva gerar conhecimentos para aplicação prática, dirigidos a problemas específicos de verdades e interesses locais. Utiliza teorias, conhecimentos e técnicas acumuladas de comunidades de pesquisa para problemas cotidianos.

3.2 Pesquisa Qualitativa

Segundo Gerdhart (2009) a pesquisa qualitativa não se preocupa com representatividade numérica e sim com aprofundamento da compreensão de um grupo social, de uma organização, etc. Os métodos qualitativos buscam exprimir o que convém ser feito, sem se submeter à prova

de fatos, pois os dados analisados são não-métricos (suscitados e de interação) e se valem de diferentes abordagens. As características deste método envolve objetivação do fenômeno, hierarquização das ações de descrever, compreender e explicar para precisar as relações entre um fenômeno global e o local, observância das diferenças entre o mundo social e o mundo natural; aceitação do caráter interativo entre os objetivos iniciais, orientações teóricas dados empíricos, resultados os mais fidedignos possíveis, diversos modelos de pequisa.

3.3 Estudo de Caso

Segundo Fonsesa (2002), o estudo de caso é um estudo de um programa, instituição, sistema educativo, pessoa ou unidade social para conhecer uma situação que se supõe ser única em muitos aspectos para descobrir o que há de mais essencial e característico. O estudo de caso pode decorrer de acordo com uma perspectiva interpretativa, que procura compreender como é o mundo do ponto de vista dos participantes, ou uma perspectiva pragmática, que visa simplesmente apresentar uma perspectiva global, tanto quanto possível completa e coerente, do objeto de estudo do ponto de vista do investigado.

Portanto, para realizar a pesquisa aplicada, definiu-se algumas macro etapas para dar consistência ao trabalho e delimitar o escopo, que são:

I- Por meio de entrevista com o gerente da área, levantar as informações sobre plano estratégico da empresa

II- Por meio de observações do pesquisador, mapear os processos de gerenciamento de configuração de software da empresa, a iniciar pela técnica de visualização de processos SIPOC (Suppliers (fornecedores), Inputs (insumos), Process (processo), Outputs (produtos obtidos na saída) e Customers (consumidores) (ANDRADE et al 2012). Em seguida mapear os ambientes específicos do objeto de estudo separando suas características de trabalho de acordo com o relato dos colaboradores.

III- Identificar e relacionar possíveis propostas de trabalho em gerenciamento de configuração de software de acordo com as características levantadas.

IV- Criar cenários de propostas onde possa ser discutido cada cenário entre e a equipe de acordo com a realidade de trabalho da empresa.

Excluído: Portanto, para poder ampliar o campo de conhecimento na área, realizou-se um estudo bibliográfico com diversos autores para analisar os pontos convergentes e divergentes de cada um, e assim poder fazer uma análise crítica com maior qualidade no objeto de estudo.¶

Excluído: s

Os processos podem variar de acordo com as características de cada projeto, portanto conhecer o valor que a empresa enxerga em seu produto e esclarecer a estratégia da empresa auxilia na definição de valor de cada prática e adotar aquelas que são de interesse da empresa.

4. RESULTADOS

4.1 Caracterização da Empresa

O estudo é realizado em uma empresa de tecnologia de Maringá que atua também em Londrina, Curitiba e Rio de Janeiro. No mercado desde 2005, a companhia oferece serviços de tecnologia nas áreas de planejamento estratégico, projeto, venda, pós-venda, suporte e operação. A empresa comercializa e implementa soluções IBM das linhas de hardware e software, desenvolvendo projetos para vários segmentos de mercado. É *Premier Business Partner* da IBM e atende clientes como GVT, Pioneer e Super Muffato. Atualmente, emprega 25 pessoas em Maringá.

Apesar de ter cases de sucesso durante a empreitada, sua infraestrutura ainda é carente em muitos setores. Este estudo focou-se no setor de desenvolvimento de software, mais especificamente em um dos projetos iniciado em 2013 que já passou por diversas equipes e hoje possui um quadro de 9 desenvolvedores multidisciplinares, porém todos com pouca experiência em desenvolvimento de software. O projeto consiste em uma solução criada dentro do software *Planning Analytics*, uma ferramenta da IBM desenvolvida com o Cognos TM1, que é uma plataforma de análise multidimensional de dados utilizando a tecnologia OLAP (*On-Line Analytical Processing*). A solução permite a criação integrada do planejamento orçamentário, planejamento estratégico, cenários preditivos e gestão de estoque, tudo por meio da integração com o ERP, BI ou planilhas da empresa. Todas estas funcionalidades são entregues em uma plataforma online para o cliente, de forma que não é preciso instalação de nenhum componente para sua utilização, ou seja, é um *Software as a Service* (SAAS).

O desafio para equipe atual é transformar a solução criada em um produto escalável, confiável na questão de disponibilidade do sistema. e para isto é preciso otimizar, padronizar e documentar os processos e produto para que a replicabilidade seja atinginda de forma rápida e fácil, além de reduzir a manutenção específica de cada cliente.

Excluído: e para melhor aplicar os conceitos, neste trabalho será feito um estudo da literatura sobre o tema e um levantamento do

Excluído: , desde

Excluído: até mapear os processos técnicos envolvidos, desta forma, é possível definir com maior precisão o

Excluído: .

Comentado [u2]: ALINHAR AS ETAPAS DA PESQUISA

Excluído: Diagnóstico dos problemas em relação a gerencia de configuração¶

Formatado: Fonte: Itálico

Formatado: Fonte: Itálico

Formatado: Fonte: Itálico

Formatado: Fonte: Itálico

4.2 Diagnóstico

4.2.1 Mapeamento dos macroprocessos

Para melhor compreensão do ambiente de produção e seus problemas, os processos foram levantados, in loco, de forma holística para então aprofundar o detalhamento, portanto o primeiro passo foi a criação de um SIPOC (*Suppliers*, *Input*, *Process*, *Output*, *Customer*) que apresenta os principais Macroprocessos envolvidos na produção e manutenção do software, além dos stakeholders impactados (fornecedores e clientes) e as entradas e saídas (Figura 2).

Figura 2 - SIPOC do processo de desenvolvimento

Supplier	Input	Process	Output	Customer
•Desenvolvedor •Analista de Negócio •Gerente de projeto •Empresa Cliente •Usuário final	•Servidores de ambientes •Aplicação Tm1 •Itens de configuração •Solicitação de Requisito •Sistema de Registro de Chamados	•Criar demanda de desenvolvimento •Desenvolver demanda em instância base •Registrar modificações em chamado •Replicar em ambiente DEV para cada instância de cliente •Replicar em ambiente de produção na intância do cliente	•Instâncias atualizadas •Repositório de registro de alterações atualizado	•Desenvolvedor •Analista de Negócio •Gerente de projeto •Empresa Cliente •Usuário final

Fonte: do autor

A ferramenta de desenvolvimento o *Cognos TM1 Architect* possui dois compiladores com propósitos diferentes e cada um deles geram os arquivos do código fonte no diretório da instância. O primeiro compilador é o "*Turbo Integrator Process*" (*TI Process*) é considerado um *ETL* (*Extract Transform Load*), um motor de processamento e integração de dados utilizado para construir um *Data Warehouse*, ou seja, pode extrair os dados de um sistema origem, transformar os dados conforme necessidade e armazenar em outro sistema. Neste compilador, se cria rotinas que automatizam diversos processos dentro da solução. Outro compilador é o editor de regras dos cubos, onde se cria fórmulas em células específicas dentro de um cubo para automatizar o cálculo em tempo real, assim, pode-se criar relacionamento entre os objetos como cubos, dimensões, atributos etc, cada cubo possui seu arquivo de regra. Com estes dois compiladores é que se cria toda a estrutura de cada instância (servidor).

Comentado [u3]: Como foi feito? Observação in loco, entrevista informal..

Excluído: ¶

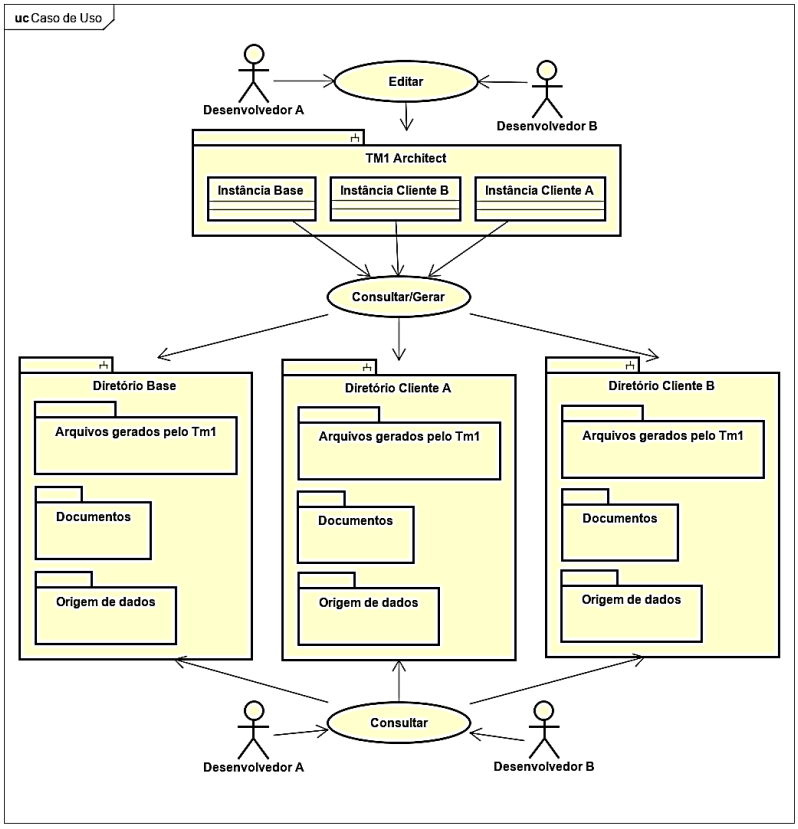
Formatado: Fonte: Itálico

Formatado: Normal, Justificado

Formatado: Português (Brasil)

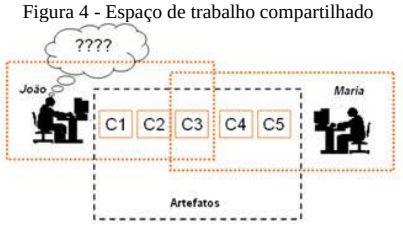
As instâncias dos clientes são criadas através de uma instância base que se mantêm atualizada com todas as funcionalidades desenvolvidas, cada uma gera seu próprio conjunto de itens de configuração que são editados pelos desenvolvedores diretamente na aplicação *TM1*, porém, também podem ser consultados diretamente pelos arquivos do diretório. A Figura 3 ilustra o relacionamento dos desenvolvedores, sistema e os diretórios. Como os desenvolvedores editam simultaneamente a mesma aplicação, ocorre conflitos de trabalho quando uma edição é sobreposta a outra, a Figura 4 ilustra a situação.

Figura 3 - Caso de Uso do sistema e desenvolvedores



Fonte: do autor

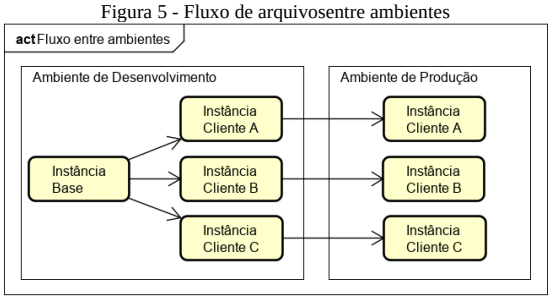
Figura 4 - Espaço de trabalho compartilhado



Fonte: https://www.devmedia.com.br/gerencia-de-configuracao-de-software/9145

O desenvolvimento é feito primeiramente em um ambiente de desenvolvimento compartilhado, chamado de "*Dev*", para que as alterações possam ser testadas sem risco de perder os dados do cliente. O ambiente de produção fica exlusivo para utilização por usuários finais e somente as alterações validadas são atualizadas, dependendo da situação é preciso agendar um tempo de parada do sistema do cliente para realizar a atualização. Este fluxo de arquivos entre os ambiente é ilustrado na Figura 5.

Figura 5 - Fluxo de arquivosentre ambientes



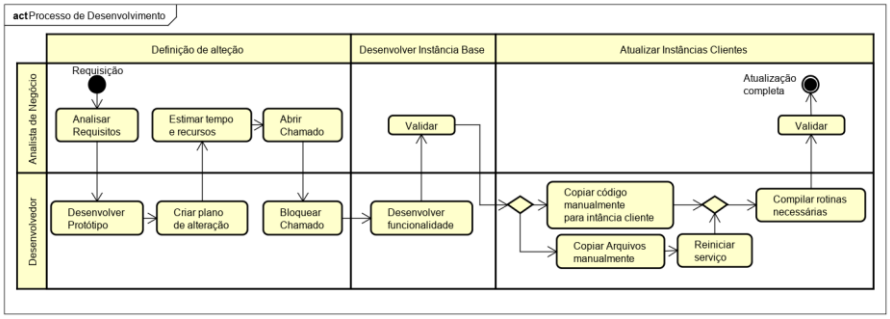
Fonte: do autor

Atualmente existe uma idéia de processo de desenvolvimento de software que se tenta aplicar, porém não é uma prática oficial consolidada, como as alterações são pequenas, não envolve muitos custos e riscos para o projeto. De forma mais comum, as solicitações vem dos clientes usuários e para ter uma resposta de cronograma de implantação desta solicitação, o time de desenvolvimento cria um protótipo (quando necessário) de como ficaria a funcionalidade, em seguida, através do sistema de gestão de serviços OTRS é criado um "chamado" ao time de desenvolvimento para formalizar a requisição, e assim o desenvolvimento propriamente dito é iniciado na instância base, assim que finalizado este trabalho precisa ser copiado para as instâncias dos clientes.

Apesar de cada instância possuir seus próprios objetos, a estrutura e a maioria das funcionalidades são comum entre estas instâncias. Quando uma alteração é feita no código-fonte da instância base (seja por correção ou nova funcionalidade) é preciso alterar o mesmo

código de todas as intâncias dos clientes, porém um complicador é a arquitetura da ferramenta de desenvolvimento (*TM1 Architect*) que não permite atualizar a configuração do sistema por apenas substituir o arquivo do código fonte, é preciso reiniciar a aplicação p/ que as alterações sejam compiladas, o que pode demorar até 20 minutos e deixa o serviço indisponível todo este tempo, outra forma de atualização é manualmente dentro dos compiladores, que ao compilar geram os arquivos do código fonte, porém sua interface tem diversos entraves que dificultam o trabalho do desenvolvedor, como por exemplo configurações que exigem seleção do mouse, não é possível alterar fonte nem o tamanho do texto do editor entre outros. Este fluxo está representado na Figura 6. Em seguida, todo este trabalho ainda é feito no ambiente de produção novamente, com um fator agravante de que os dados dos clientes correm grande risco de serem perdidos, pois os objetos podem perder a referência se caso uma alteração for feita de forma errada.

Figura 6 - Fluxo de desenvolvimento



Fonte: do autor

Devido a estas constatações, os desenvolvedores passam boa parte do tempo realizando "*monkey job*", ou seja, trabalhos repetitivos sem necessidade do potencial humano de criar e inovar. Além disso, nestas etapas manuais há alto risco de erros humanos gerando retrabalho, desperdício de tempo e recursos, bugs inesperados e portanto, clientes insatisfeitos.

Conforme muitas alterações são realizadas nos itens de configuração, é comum perder o controle de qual atualização ainda não foi realizada em alguma instância, desperdiçando tempo p/ analizar os arquivos fontes, ou às vezes até deixando de lado algumas atualizações, o que prejudica o desenvolvimento de novas atualizações, pois algumas delas possui dependencias de outras. Neste cenário, a qualidade do produto, processo e serviço de software é prejudicada.

Formatado: Português (Brasil)

De acordo com o levantando acima, os problemas identificados são essencialmente o que ocorre na maioria dos projetos de software, onde em busca de desenvolver atualizações de software, as equipes trabalham paralelamente no mesmo projeto, módulo ou até mesmo item de configuração para suprir demandas de tempo e qualidade, e sem um controle adequado, as alterações não são documentadas e se perde a possibilidade de planejar eficazmente qualquer futura implementação, comprometendo a credibilidade da empresa com os clientes e fornecedores.

Portanto, neste modelo o que se considera de maior deficiente para alcançar a escalabilidade desejada é a falta de controle de versões dos itens de configuração e a falta de gerenciamento de mudanças, pois nenhuma política de alteração foi definida e nenhuma alteração é controlada, estas responsabilidades fica a encargo dos próprios desenvolvedores utilizarem o bom senso e atenção para realizar as atividade, o que compromete a segurança dos dados e aumenta o tempo de trabalho, já que cada atividade é feita minuciosamente, além de retrabalhos e o tempo buscando informações em lugares não definidos.

Formatado: Italiano (Itália)

4.3 Proposta

As quatros atividades do Gerenciamento de Configuração de Software serão consideradas para propor cenários de soluções adequados para realidade da empresa, estas atividades são: gerenciamento de mudanças, gerenciamento de versões, construção de sistema, e gerencimento de release. Tendo uma visão sistêmica de todo o processo de gerenciamento de configuração, primeiramente escolheceu-se ferramenta de Gerenciamento de Versão e Mudanças que auxilia o controle deste processo, atualmente, existe uma variedade grande de ferramentas para este propósito, cada uma com diferentes funcionalidades e níveis de especialidades, alguns exemplos são: CVS, Subversion, IBM Rational ClearCase, Github, Microsoft Visual Source Safe, Bugzilla, Jira,Trac entre outros. Depois de definido o sistema de suporte, é feito a definição dos itens de configuração que serão gerenciados pelo sistema dentro de um repositório, portanto é definido um referencial dos itens de configurações até o momento, e que a partir deste ponto, toda alteração será controlada, avaliada e controlada. Em seguida estabelece-se uma política de versionamento dos projetos de forma a possibilitar a evolução dos itens de configuração conforme novas atualização são realizadas. Foi proposto uma política de trabalho colaborativo para evitar concorrência de arquivos no trabalho simultâneo e por fim, elabora-se um cenário onde não há necessidade de atualização manual das instâncias de clientes.

4.3.1 Sistema de gerenciamento de configuração

A começar pela escolha do sistema de suporte, considerou-se que a empresa ainda não possui maturidade no processo de gerenciamento de configuração, não há incentivo financeiro para gastos com tais processos, não há recurso necessário para disponibilizar um especialista para implantação do gerenciamento de configuração. Neste sentido, buscou-se uma ferramenta o*pensouce* para controle de versões que pudesse ser implementada pelos próprios colaboradores. Entre as mais populares estão o Subversion, Mercurial e Github, sendo que o último já é conhecido por alguns integrantes da equipe e demonstrou ter uma interface mais amigável e intuitiva que facilita no processo inicial de aprendizado, além disso, o Github se mostra mais completo em suas funcionalidades em relação aos concorrentes, porém isto não atrapalha no aprendizado por ser mais complexo, pois permite iniciar com as funcionalidades mais básicas e que mais a frente possa ser implementado por completo. O Github é um serviço web que oferece o controle de versão de desenvolvimento através do sistema Git e este possui intregração com o TM1 que a própria IBM disponibilizou em versões mais recentes da ferramenta, permitindo a atualização de servidores de forma direta. Alguns dos recursos relevantes do Github são:

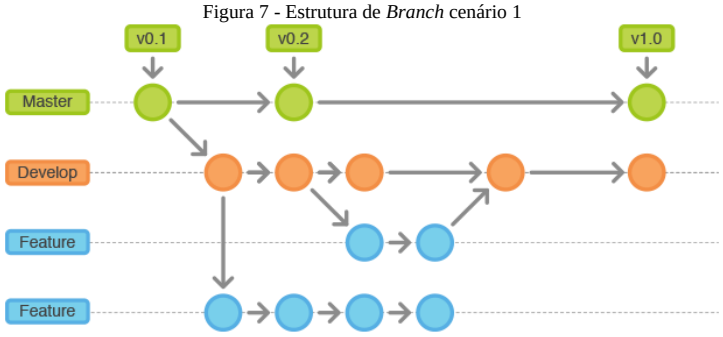
- Repositório na nuvem: permite a colaboração compartilhada de itens de configuração organizados pelos diretórios versionados.
- *Commits*: histórico de edição dos arquivos com detalhes de modificação e um ponto de restauração de uma determinada versão do projeto.
- *Pull/push*: função de copiar o diretório para uma estação de trabalho privada e de devolver ao diretório compartilhado depois de realizar a alteração.
- *Branchs*: ramificações do diretório principal (*master*) que podem ser editadas isoladamente e ser mesclado (*merge*) à *master* conforme necessidade.
- *Pull Requests*: meio de controlar o fluxo de atualização da *Branch master*, onde um administrador do projeto aceita ou não os pedidos de atualização
- *Issues*: painel para postar/controlar bugs e sugestões por status aberto, fechado ou então por *tags* que identificam diferentes propósitos.
- *Projects*: painel de ações kanban que centraliza as informações do projeto centralizadas.
- *Insights*: gráficos de contribuição de cada colaborador, acessos ao projeto, frequência de *Commits*, cópias do projeto (*forks*), evolução de dependências de *Commits* do projeto (*network*), entre outros.

- Editor online: Editor de texto com funções apropriadas para programação, disponível para arquivos do projeto diretamente pelo site.

4.3.2 Política de versionamento

Em um primeiro cenário de processo de desenvolvimento não é garantido que as instâncias dos clientes serão iguais a instância base, há muitas especificidades de cada um para se implantar um modelo genérico. Neste cenário, propõe-se que cada instância seja um repositório separado, cada uma com a *Branch* de desenvolvimento (*develop*) e outras *Branch* de recursos adicionados por cada desenvolvedor ou grupo (*feature*), a *Branch master* seria designada para o release da versão que funciona em produção, portanto cada instância teria a estrutura de *Branch* da Figura 7.

Figura 7 - Estrutura de *Branch* cenário 1



Fonte: https://leanpub.com/git-flow/read#leanpub-auto-end-user-discovers-a-bug

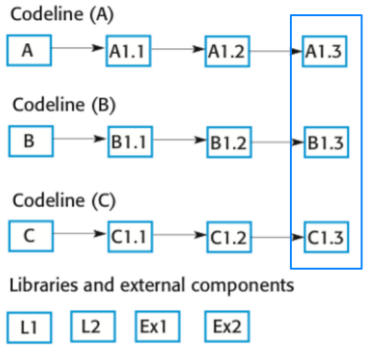
Além de cada *Commit* ter seu título de alteração e descrição, propõe-se incluir uma *tag* quando se realiza o *fork* para *Branch develop*, ou seja, as *features* são implementadas por diversos colaboradores e desenvolvidas paralelamente, assim que uma estiver completa é enviada para *develop* com um número incremental de versão, seguindo o modelo *Semantic Versioning* (semver.org), onde se utiliza uma estrutura de 3 partes:

1. *Major*: quando se faz uma alteração incompatível com a versão anterior
2. *Minor*: quando acrescenta funcionalidades compatível com a versão anterior
3. *Patch*: quando se faz uma correção de bug compatível com a versão anterior

Portanto, a *Branch feature* teria o título do *Commit* como, por exemplo, “função copiar cenário módulo Vendas”, e uma breve descrição do que poderia se tratar esta função. Já a *tag* da *Branch develop* seria, por exemplo, “1.2.0”. Também define-se que a *Branch master* seja sempre a última atualização da *develop*, mantendo assim a mesma *tag*.

A *baseline* do projeto é sempre a última versão dos itens de configuração, portanto esta abordagem de *Branch* por projeto não afetaria o controle de itens de configurações, visto que cada *feature* implementada estará dentro de um *Commit*, assim, quando se deseja saber a evolução dos itens de configuração basta olhar o *Commit* relacionado a ele. No entanto, um controle mais minucioso poderia ser alcançado se cada item de configuração fosse versionado separadamente, controlando as *codelines* e *baselines* conforme a Figura 8

Figura 8 - Representação da *Codeline* e *Baseline* dos itens de configuração



Fonte: Pressman (2016) modificado pelo autor

4.3.3 Política de manutenção colaborativa

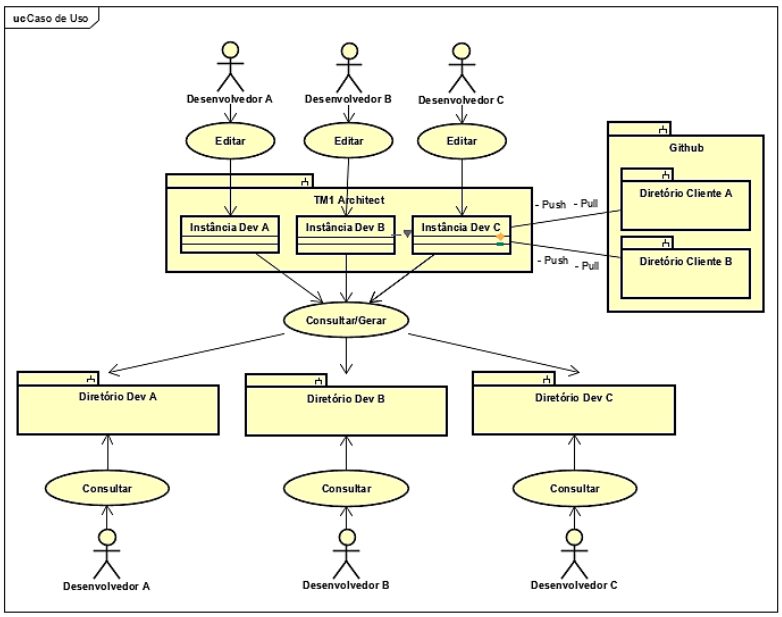
O gerenciamento de versão já resolve o conflito de atualização simultânea do projeto, pois permite criar *Branchs* separadas para cada *feature* desenvolvida (geralmente realizado por um colaborador), no entanto, atualmente a base de desenvolvimento "privada" é realizada no mesmo servidor e na mesma instância por todos os usuários, neste sentido, ainda há risco de perda de trabalho caso dois usuários estejam trabalhando no mesmo item de configuração, o que não é raro de acontecer. Como levantado no diagnóstico, a ferramenta utilizada trabalha carregando todo os dados em memória RAM para ter maior desempenho, é recomendação da própria IBM manter a ferramenta em um servidor de trabalho robusto com requisições de desempenho acima da média de computadores pessoais, justificando o trabalho no mesmo servidor.

Para garantir o trabalho paralelo é preciso que haja um ambiente de trabalho privado onde as alterações possam ser realizadas isoladamente, para isto, propõe-se a utilização de instâncias de desenvolvimento para cada desenvolvedor. Até o momento, esta forma de trabalho não foi utilizada devido ao esforço em manter o ambiente de cada instância atualizada, pois além da falta de controle de versões, onde não se sabia quais itens de configurações deveriam

ser atualizado e quais seriam as alterações destes itens, ainda que os arquivos fontes fossem atualizados, era necessário a reinicialização do serviço desta instâncias para compilar as mudanças.

No entanto, conforme o a ferramenta Github grava os pontos de história do desenvolvimento, é possível rastrear as alterações de forma rápida e fácil. Além disso, a integração da tecnologia git com a ferramenta permite o *check-out* de uma *Branch* diretamente para o diretório privado da instância de forma que não necessite da reinicialização do serviço, pois por meio de API são gerados arquivos Json que convertem as alterações em tempo de execução da ferramenta, agilizando assim o processo de atualização. Desta forma, os custos associados a criação de uma instância por desenvolvedor são minimizados ficando apenas o consumo maior de memória RAM do servidor, porém é justificável pela otimização de trabalhos concorrentes e velocidade de processamento, visto que haveria apenas um usuário utilizando os recursos daquela instância. A Figura 9 destaca a forma de trabalho descrita acima:

Figura 9 - Ambiente de trabalho privado



Fonte: do autor

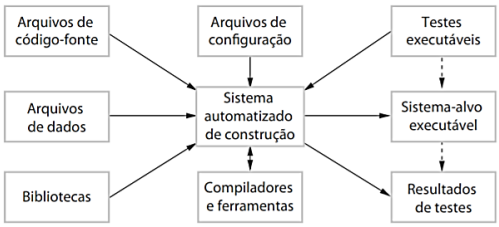
Neste cenário, como não há concorrência de artefatos (pois a *Branch develop* fica no repositório em nuvem) os desenvolvedores podem realizar o *check-out* do projeto em seu repositório privado simultaneamente e realizar as alterações necessárias, porém no momento de

realizar o check-in pode ter ocorrido mudanças no mesmo item de configuração, neste caso o desenvolvedor que realizar o *Commit* por último deve resolver o conflito analisando o código e incluído as alterações nos lugares apropriados. Esta política é conhecida como Copia-Modifica-Resolve.

4.3.4 Política de construção de sistema

A construção de sistemas envolve a montagem de uma grande quantidade de informações sobre o software e seu ambiente operacional, como o código fonte, bibliotecas externas fornecidas, arquivos de dados e arquivos de configuração que definem a instalação-alvo (Figura 10).

Figura 10 - Construção de sistemas



Fonte: Sommerville (2011)

Atualmente, a construção de cada instância precisa ser gerenciada manualmente, etapa por etapa, pois as diferentes definições dos clientes fazem com que alguns scripsts ou objetos precisem ser modificados. No entanto, estas especificidades devem estar mapeadas de forma que as os scripts e objetos sejam parametrizáveis, garantindo a uniformização dos itens de configuração entre as instâncias, assim, para qualquer atualização seria possível sincronizar as instâncias de forma automática, eliminando o processo de copiar manualmente códigos entre instâncias.

Neste cenário, o controle de versão eliminaria as *Branchs* de cada instância, e estas seriam atualizadas pela *Branch develop* ou *master*. Uma maior otimização seria alcançanda pela integração contínua entre os sistemas, isto é, construir uma automatização de atualização das instâncias baseadas na branchs master. Isto é possível através de scripts programados dentro de ferramentas de construção de sistemas, como por exemplo: CloudBees, AWS CodePipeline, Bitbucket Pipelines, Jenkin e outros.

5. CONCLUSÃO

O essencial para encontrar as melhores práticas e políticas para os processos de desenvolvimento de software foi a realização de um mapeamento dos processos as-is, levantando o fluxo de informação e a relação dos recursos como ambientes e pessoas por meio de um conjunto de metodologias de mapeamento como diagramas UML (*Unified Modeling language*) de caso de uso, diagramas de classe, e mapeamento BPMN (*Business Process Model and Notation*). Enxergar a situação atual de forma ampla e por diversas visões permite encontrar soluções de forma modular, considerando possíveis cenários para cada módulo sem se preocupar com as combinações que seriam geradas se fossem uma solução que integrasse todos os problemas de uma vez.

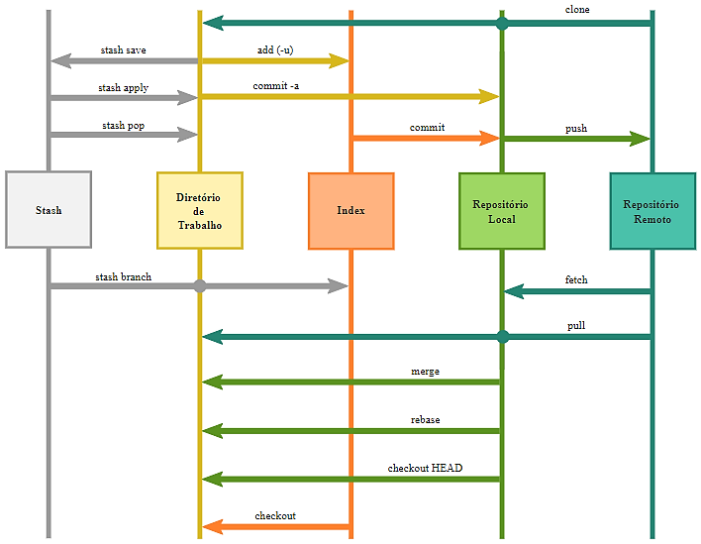
Após um primeiro ciclo do projeto, alguns aprimoramentos poderiam ser realizados, como o *git flow* que pode se tornar mais robusto para atender a demanda da empresa, conforme a Figura 11, onde as *branch* de *Hotfix* e *Release* poderiam ser incluídas antes de levar uma alteração para *master*

Figura 11 - *Git flow* com *hotfix* e *release*

Fonte: https://leanpub.com/git-flow/read#leanpub-auto-end-user-discovers-a-bug

Além disso, os *git states* também poderiam ser reavaliados e incrementados conforme necessidade da empresa, onde mais ambientes de trabalho seriam incluídos antes de realizar um push para o repositório em nuvem, conforme a Figura 12 sugere:

Figura 12 - *Git States*



Fonte: https://leanpub.com/git-flow/read#leanpub-auto-end-user-discovers-a-bug

Existem diversas formas de se adaptar uma política de versionamento, cabe aos participantes dos processos encontrar os modelos que melhor se encaixem a realidade de trabalho, considerando a criticidade da aplicação, a quantidade de desenvolvimento e de desenvolvedores, a complexidade de relacionamentos dos itens de configuração. O ciclo PDCA pode ser utilizado para incrementar a política conforme a necessidade é sentida pela equipe de projetos e conforme se ganha maturidade no processo.

Considerar um ambiente de desenvolvimento compartilhado entre a equipe pode ser uma solução viável atualmente, devido à baixa quantidade de conflitos encontrados até hoje, porém à medida que o software evolui, maior complexidade é inserida no sistema e, portanto, são criados maiores relacionamentos entre os itens de configuração, aumentando a possibilidade de um alterações concorrentes.

A construção de sistemas automatizados precisa ser construída após a maturidade do processo de versionamento, pois é a partir da *branch master* que se criaria *scripts* de construção para as instâncias em produção. A ferramenta Jenkins aparenta ter uma maior popularidade na comunidade de desenvolvedores, por ser de fácil integração com ferramentas de versionamento como o Git e por ser uma ferramenta *opensouce*.

A implantação da ferramenta de controle de versão Github atende amplamente as necessidades de gerenciamento de configuração de software da empresa, visto que manter o histórico de alterações com informações relevantes para que os desenvolvedores possam facilmente identificar os impactos, relacionamentos, e outro atributos das alterações, contribui para um processo otimizado de evolução do software. As políticas podem ser incrementadas conforme o crescimento da demanda e da equipe, ou criticidade entendida pelos colaboradores.

6. REFERÊNCIAS

ANDRADE, G.E.V., MARRA, B.A.M., LEAL, F., MELLO, C.H.P.: Análise Da Aplicação Conjunta Das Técnicas Sipoc, Fluxograma E Fta Em Uma Empresa De Médio Porte, ENEGEP 2012), Disponível em: http://www.abepro.org.br/biblioteca/ENEGEP2012_TN_WIC_157_920_20681.pdf, acessado em: 20/01/2020

CORDEIRO, A. G., FREITAS, A. L. P.: "O cenário atual da qualidade de software". In: SIMPÓSIO DE ENGENHARIA DE PRODUÇÃO. 2008. Disponível em: https://www.researchgate.net/profile/Andre_Luis_Freitas/publication/267623024_O_cenario_atual_da_qualidade_de_software/links/545395830cf2cf51647c1b44/O-cenario-atual-da-qualidade-de-software.pdf. Acessado: 15/06/2019

DAHL, O. J.; DIJSKTRA, E. W.; HOARE, C. A. R.: Structured Programming. Academic Press, Londres, Inglaterra, 1972. Disponível em: http://seriouscomputerist.atariverse.com/media/pdf/book/Structured%20Programming.pdf. Acessado: 24/06/2019

DIAS, M. V. B.: Um novo enfoque para o gerenciamento de projetos de desenvolvimento de software. Universidade de São Paulo, Programa de pós-graduação em administração, São Paulo – SP, 2005. Disponível em https://teses.usp.br/teses/disponiveis/12/12139/tde-03012006-122134/pt-br.php. Acessado em 15/07/2019.

ESTUBLIER, J., LEBLANG, D., VAN DER HOEK, A., et al. (2005).: Impact of software engineering research on the practice of software configuration management. ACM Transactions on Software Engineering and Methodology, Vol. 14, No. 4, October 2005, Pages 1–48. Disponível em: https://www.researchgate.net/publication/220403673_Impact_of_software_engineering_research_on_the_practice_of_software_configuration_management/citation/download, acessado em 20/08/2019.

FRANÇA, F. A., MEDEIROS, G. C.: Gerência De Configuração De Software: Análise Prática De Sistemas De Controle De Versão – Universidade Do Sul De Santa Catarina, Palhoça SC, 2016. Disponível em https://riuni.unisul.br/handle/12345/2007. Acessado em 08/07/2019

GERHARDT, T.E., SILVEIRA, D.T.: Métodos de pesquisa, Universidade Aberta do Brasil – UAB/UFRGS Curso de Graduação Tecnológica – Planejamento e Gestão para o Desenvolvimento Rural da SEAD/UFRGS. Porto Alegre: Editora da UFRGS, 2009. Disponível em http://www.ufrgs.br/cursopgdr/downloadsSerie/derad005.pdf, acessado em 18/01/2020.

GIBBS, W. Wayt. Software's Chronic Crisis. 1994. Scientific American. Disponível em: https://www.researchgate.net/publication/247573088_Software's_Chronic_Crisis/download. Acessado: 15/06/2019.

Git Flow Workflow https://leanpub.com/git-flow

HASS, A. M. J.: Configuration Management Principles and Practices Boston, MA, Pearson. Education, Inc., 2003. Disponível em: https://pdfs.semanticscholar.org/9df4/c66fb78b3346a615ccf6c24bf6e42e561dfe.pdf, acessado em 20/08/2019

Formatado: Inglês (Estados Unidos)

Formatado: Inglês (Estados Unidos)

MAGALHÃES, A. L. C. C.: A Importância do Controle da Qualidade na Melhoria de Processos de Software. In II Workshop de Empresas (W6-MPS. Br), Campinas. 2008. Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/wamps/2008/012.pdf. Acessado: 15/06/2019.

MARQUES, B. A., SILVA, M. C. M.: Qualidade de Software: Uma Análise a partir dos Critérios da Norma ISO 9126. XXXII Encontro da Associação Nacional de Pós-Graduação e Pesquisa em Administração (ANPAD), Rio de Janeiro RJ, 2008. Disponível em http://www.anpad.org.br/admin/pdf/ADI-A1410.pdf. Acessado: 15/06/2019

PRESSMAN, R. S.; MAXIM, B. R. Engenharia de Software – Uma Abordagem Profissional, 8ª edição. Porto Alegre: AMGH, 2016.

ROCHA, A. R., MALDONADO, J. C. WEBER, K. C.: "Qualidade de Software: teoria e prática". São Paulo: Prentice Hall, 2001.

SOMMERVILLE, I. – Engenharia de Software – 9. ed. – São Paulo: Pearson Prentice Hall, 2011.

SOUZA, A. C. C. L., MONTEIRO, R. F.: gestão da qualidade de software: Garantia da Qualidade Total. Universidade Fumec, Belo Horizonte, 2009. Disponível em http://professores.dcc.ufla.br/~terra/publications_ files/students/2009_fumec_souza_e_monteiro.pdf. Acessado: 15/06/2019

SWEBOK - Guide to the Software Engineering Body of Knowledge, 2014 Version 3.0. project of the IEEE Computer Society Professional Practices Committee. Disponível em: <https://www.computer.org/education/bodies-of-knowledge/software-engineering/v3>. Acesso em: 01/05/2019.

WEBER, K. C., ROCHA, A. R, NASCIMENTO, C. J. do.: Qualidade e produtividade em software (4a ed.), 2001: São Paulo: Makron Books.