UNIVERSIDADE ESTADUAL DE MARINGÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

ENGENHARIA DE SOFTWARE II

|  |
| --- |
| **RELATÓRIO DE PROJETO**  **“Sistema Pastelaria LM”**  Gilberto Leal Santos 99999  Luiz Gustavo Riberio 68651  Tiago Uemura 94108  **2018** |

MARINGÁ

Sumário

[1. Introdução 4](#_Toc530331981)

[2. Revisão Literária 4](#_Toc530331982)

[2.1. Gerenciamento de Configuração de Software 4](#_Toc530331983)

[2.1.1. Gerenciamento de Mudanças (GM) 4](#_Toc530331984)

[2.1.2. Gerenciamento de Versões (GV) 5](#_Toc530331985)

[3. Resultados 5](#_Toc530331986)

[4. Conclusão 5](#_Toc530331987)

[5. Bibliografia 6](#_Toc530331988)

**TO DO =======================================================================**

**COD**

**05/12/20218 13:30**

**Controlador.cadastrarProduto(): retirei a função de add dele e coloquei a responsabilidade para ProdutoDao.adicionarProduto() ------desta forma não utiliza o controlador.buscarProduto()**

**Construtor de Pedido deve ser sem parâmetros, para ser criado vazio e colocado os produtos em sua lista e depois as outras informações. Controlador.criarPedido(): criei um sem parâmetro e comentei que estava com parâmetros. pedidoDao.criarPedido() e Implements mesma coisa**

**// Pedido pd = new Pedido(); linha 61 InterfaceAtendente**

**Não compete a interface criar, coloquei no controlador.criarPedido() que chama o DAO (talvez o controlador mesmo possa criar), então na interface atendente chama o método do controlador.**

**\*\* ainda não feito**

**\*\*Criar método public void menuTipoPedido(): mostra os enums de tipo pedido e pede para selecionar um.**

**\*\*: Colocar novoPedido() no controlador**

**Produto produto = new Produto();**

**interfaceAtendente.menuTipoPedido(); //criar instancia da interface**

**case 1, 2,**

**3: while etc**

**Controlador.incluirPedido(): pega o produto p e setQuantidade, depois pede para Dao colocar o produto em pedido pd**

**fecharPedido(): vai incluir as outras informações no pedido. calcula o valor total em controlador**

**imprimirPedido(): vai imprimir o pedido pelo toString da classe Pedido;**

**Por que não listar os produtos já cadastrados e pedir pro usuário selecionar entre 1 a n os produtos? (utilizando case): para evitar erro no pedido, o user pode apertar o número errado. É melhor escrever o nome do produto.**

TO DO

DOING

DONE

Dito pelo pdf da Juliana

*Dito por Gil, Luiz ou Tiago*

**3) O que deve estar contido no relatório do trabalho:**

a) Identificação da equipe.

b) Descrição dos passos necessários para a execução do projeto.

c) Pesquisa sobre gerenciamento de configuração, citando fontes científicas e confiáveis, apresentando no final do relatório as referências bibliográficas conforme o padrão ABNT.

1. *Gerenciamento de mudanças*
2. *Gerenciamento de versões*
3. *Construção do sistema*
4. *Gerenciamento de release*

d) Com base na pesquisa, deve ser apresentada uma descrição do histórico de gerenciamento de configuração realizada durante o desenvolvimento do trabalho prático.

e) Pesquisa sobre técnicas e estratégias de teste de software, citando fontes científicas e confiáveis, apresentando no final do relatório as referências bibliográficas conforme o padrão ABNT.

f) Com base na pesquisa, deve ser apresentada uma forma de implementação de teste de software para o projeto desenvolvido.

g) Conclusão:

• Destaque o que foi implementado;

• Descreva como as tarefas foram atribuídas aos membros da equipe;

• Explique como as tarefas foram gerenciadas;

• Discuta sobre a utilidade dos testes de software e como foram implementados na prática.

• Descreva as dificuldades encontradas.

============================================================

1. Introdução

Descrição dos passos necessários para a execução do projeto.

1. Revisão Literária
   1. Gerenciamento de Configuração de Software

Sistemas de softwares sofrem alterações a partir do início do seu desenvolvimento, durante seu uso, passam por novas versões, e assim segue até o dia que não seja mais utilizado ou aproveitável. Essas mudanças geram grande conflitos caso não sejam bem gerenciadas, exemplos são: alterações nos requisitos organizacionais, reparos de bugs, alterações do ambiente, seja software ou hardware.

Segundo Sommerville (2011), o Gerenciamento de Configuração de Software (GCS) é o conjunto de políticas, processos e ferramentas que é utilizado no gerenciamento de mudanças de sistemas de software.

O autor define quatro atividades do GCS:

1. Gerenciamento de mudanças
2. Gerenciamento de versões
3. Construção do sistema
4. Gerenciamento de releases

Entende-se que efetuar essas quatros atividades auxilia a uma equipe ou o gerente do projeto a ter o controle de quais mudanças foram feitas em quais versões de cada componente de software, mesmo que estejam sendo desenvolvidas em paralelo com vários desenvolvedores em lugares remotos. O GCS melhora o fluxo de trabalho, minimiza o retrabalho pois a informação está acessível a todos e diminui o risco de perda de dados.

A seguir apresenta-se uma breve definição de cada atividade dentro do GSC, segundo Sommerville (2011).

* + 1. Gerenciamento de Mudanças (GM)

O objetivo desta atividade é garantir a evolução do sistema seja um processo controlado, por alterações prioritárias e efetivas, no momento certo e do jeito certo.

O GM implica em analisar os custos e benefícios de cada alteração proposta, aprovar o desenvolvimento daquelas que valem a pena no momento e acompanhar os resultados das alterações.

Este processo se inicia pela requisição de mudança (CR, Change Request) no sistema por algum cliente. Geralmente, utiliza-se um formulário para realizar a requisição (CRF, Change Request Form), assim as informações da CR podem ser compartilhadas entre todos envolvidos, e à medida que é processada pode-se incluir registros sobre as decisões em cada etapa. O CRF pode conter, além da mudança, recomendações, estimativas de custos, datas de início, aprovação, conclusão etc.

A aprovação da CR deve considerar alguns fatores antes de ser passado para a equipe de desenvolvimento como, por exemplo, as consequências de não fazer a mudança, os benefícios, os custos, o ciclo de release (entregas de versões ao cliente), etc.

Quando a equipe de desenvolvimento altera um componente de software, é necessário registrar esta mudança em um histórico de derivação, este pode estar no cabeçalho do código fonte em forma de comentário, contendo a referência ao CR, autor da modificação, data, razão etc.

Muitos processos são realizados no tratamento de CR e por isso algumas ferramentas são desenvolvidas para automatizar desde a etapa de proposta inicial até a aprovação da mudança.

* + 1. Gerenciamento de Versões (GV)

Os itens de configuração de um software podem ter várias versões independentes devido a necessidade de alterações. Cada item tem seu conjunto de versões chamados de *codelines*. O sistema de software pode ter várias configurações distintas e independentes, combinando as versões de cada item de configuração. Este conjunto de componentes definido para o sistema é chamado de *baseline*, que podem ser especificadas de acordo com uma referência a cada componente e sua versão, por exemplo, X.1.2.

O GV é o processo que gerencia os *codelines* e *baselines*, ou seja, identificar, armazenar, controlar o acesso as versões de componentes. O GV garante que desenvolvedores possam alterar versões dos componentes sem interferir umas às outras, permite recriar versões de sistemas completos para outro cliente, e com auxílio de ferramentas específicas podem automatizar alguns processos como a identificação de versão e release, gerenciamento de armazenamento, registro de histórico de mudanças, desenvolvimento independente, suporte a projetos etc.

Para permitir o trabalho simultâneo em um mesmo componente do sistema, sistemas de GV utiliza-se um repositório público e espaço de trabalho privado para realizar o *check*-*out* de componente do repositório público para trabalho no espaço privado, e quando concluir as alterações faz o *check-in* do componente para o repositório. Desta forma, quando uma pessoa tentar realizar o *check*-*out* de um componente já realizado por outra, o sistema deve avisar o usuário.

Com o desenvolvimento independente assegurado pode-se surgir ramificações dos *codelines* que são diferentes continuações de uma versão anterior em comum. As ramificações criadas podem se fundir em algum momento para criar uma nova versão.

* + 1. Construção de sistemas

É o processo de ligar os componentes de sistemas criados separadamente para criar um sistema completo e executável. Podem ter três plataformas de sistemas envolvidos e cada um está sujeito a erros em sua construção

1. O sistema de desenvolvimento (compiladores, editores código-fonte etc.)
2. Servidor de construção (construção de versões definitivas e executáveis)
3. Ambiente alvo (plataforma de execução)

O sistema gerenciador de versões pode atuar tanto no ambiente de desenvolvimento ou no servidor de construção. Dependendo do ambiente alvo, pode ser instalado um ambiente de simulação no próprio ambiente de desenvolvimento.

A construção do sistema do sistema pode ser automatizada por scripts de configuração para agilizar e realizar verificações consistentes antes de iniciar a construção.

* + 1. Gerenciamento de Releases

O *release* é a versão entregue ao cliente. Normalmente há um release principal e os menores que fazem reparos ou pequenas alterações no release principal. É importante que cada release esteja documentado, pois pode-se rastrear e utilizá-los para outros clientes ou para futuras implementações. Para tanto, é necessário documentar cada componente de código fonte que foram utilizados para criar o código executável.

Releases podem conter arquivos de configuração para instalação, arquivos de dados, programas de instalação e outros.

* 1. Pesquisa sobre técnicas e estratégias de teste de software

O teste de software busca institucionalizar práticas de gestão de projetos e desenvolvimento de produtos de software, com o objetivo de localizar os problemas - e não garantir a sua inexistência. A partir de uma perspectiva sociotécnica e dos modelos teóricos SET (gerencial) e ST-TS (processual) desenvolvidos nesta pesquisa para um melhor entendimento e orientação das atividades de teste, entrevistas em profundidade com dez especialistas em qualidade e teste de software permitiram a identificação e validação de categorias analíticas que mediam a relação entre fatores desses dois modelos, disto derivando-se o modelo organizacional VAST para auxiliar gestores, desenvolvedores e clientes em projetos de software para sistemas de informações empresariais (TOSSETO, 2008).

Uma forma de abordar a qualidade dá-se por meio do teste de software (TS), que define atividades de verificação e validação com o objetivo de encontrar problemas. A importância da integração da atividade de TS ao desenvolvimento fica evidente quando esta "tem sido considerada cada vez mais como fator essencial para a obtenção de sistemas de software de qualidade" (Herbert, 1999, p. 17). Tal importância é reforçada pelo fato de que menores serão o custo e a dificuldade das correções se mais próximos de sua origem os erros forem detectados (Da Rocha et al., 2001). Não obstante, o TS foi tradicionalmente visto como atividade complementar (Beizer, 1990; Hetzel, 1987).

Segundo Pressman (1995, p. 724), qualidade de software é a "conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo software profissionalmente desenvolvido". Para a produção de software com qualidade superior, é essencial haver um processo formal e padronizado de desenvolvimento. Também não é possível alcançar qualidade na ausência de uma garantia de qualidade de software (GQS) que ocorra paralelamente ao desenvolvimento e que verifique e valide suas etapas. Para a implantação de processos de qualidade, normas e padrões como ISO/IEC-12207 e os diversos modelos CMM (de maturidade de competências em processo de software) podem ser empregados (Bartié, 2002).

Segundo Nguyen et alli (2013), teste de software é uma atividade importante no desenvolvimento de software e projetos de manutenção, garantindo a qualidade, aplicabilidade e utilidade de produtos de software. Na verdade, nenhum software pode ser liberado sem uma quantidade razoável de testes envolvidos. Para atingir a qualidade aceitável, as equipes de projeto de software dedicam parte substancial do esforço de desenvolvimento total para a realização teste. De acordo com relatórios da indústria, teste de software consome cerca de 10 a 25% do esforço total do projeto, e em alguns projetos este número pode chegar a 50%.

Para Zhou et alli (2013), o teste de software, definido como a execução sistemática do software com o objetivo de revelar falhas, é uma importante fase para validar a correção do software. As atividades de teste de software são compostas da definição dos casos de teste e validação do comportamento da execução. Em geral, a execução dos casos de teste é limitada, uma vez que validar todos os caminhos de execução tende a ser inviável em termos de prazo e custo. Assim, a qualidade dos casos de teste afeta diretamente a qualidade do software, e um dos atributos dos casos de teste de qualidade é a capacidade de detecção de falhas ainda por desvendar.

Segundo Patel et alli (2001), gerar estimativa eficaz de um projeto de software é uma das atividades mais importantes e desafiadoras. Apenas com uma estimativa precisa e confiável é possível concluir um projeto no prazo. Estimativas desempenham um papel vital em todas as fases do ciclo de vida de desenvolvimento de software.

1. Resultados

Foi definido que a construção do sistema LM seria dividida em duas equipes de desenvolvimento, e que, portanto, necessitaria de uma gestão de configuração de software para sincronizar as alterações. Para desenvolvimento do código-fonte, a equipe trabalhou com o kit de desenvolvimento Java (JDK) disponibilizado pela Oracle que contém o compilador, bibliotecas e a máquina virtual Java. A equipe utilizou o Ambiente de desenvolvimento integrado (*IDE*, *Integrated Development Environment)* Netbeans que contém editor de código fonte, compilador java, depurador e outas ferramentas que auxiliam no desenvolvimento.

Os arquivos de configuração iniciais, criados automaticamente pela IDE, foram armazenadas em um repositório em nuvem próprio para versionamento de código fonte e trabalho colaborativo, o GitHub. Esta solução utiliza o sistema Git de versionamento e faz a integração entre das alterações de diferentes contribuintes do projeto. Cada desenvolvedor realizou um “clone” do repositório do GitHub em sua máquina local para desenvolver os módulos separadamente. Assim que uma alteração significativa era realizada, realiza-se o *“commit”* de todas as alterações, e desta forma o sistema do GitHub faz o versionamento dos arquivos alterados automaticamente. Para cada commit, descreveu-se as alterações realizadas.

Muitas alterações foram simultâneas em um mesmo arquivo, e quando eram levadas ao repositório comum, podia-se escolher o momento de incluir essas alterações e ainda resolver os conflitos da combinação.

No início, a configuração do repositório foi realizada em uma pasta configurada para sincronizar com a nuvem do OneDrive da Microsoft, o que levou a uma má estabilidade do uso do sistema de versionamento e também do trabalho colaborativo simultâneo. Quando uma alteração era feita em uma máquina local e salvo na IDE, o sistema OneDrive sincronizava automaticamente as alterações na outra máquina o que não era desejável. Além disso, o GitHub identificava as alterações feitas por qualquer uma das máquinas já que estavam ligadas pela nuvem OneDrive.

Depois de entendido a dificuldade de se trabalhar desta forma, cada usuário criou um novo repositório local do GitHub, clonando-se o projeto original e assim o fluxo de trabalho se desenvolveu de uma melhor forma.

1. Conclusão

Destaque o que foi implementado;

• Descreva como as tarefas foram atribuídas aos membros da equipe;

• Explique como as tarefas foram gerenciadas;

• Discuta sobre a utilidade dos testes de software e como foram implementados na prática.

• Descreva as dificuldades encontradas.

1. Bibliografia

[1] SOMMERVILLE, I. Engenharia de Software. 9 ed. São Paulo: Pearson Addison Wesley, 2011. .