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

asd

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

O objetivo desse projeto era aplicar os conhecimentos adquiridos na disciplina de Engenharia de Software II, e para isso, em sua na etapa inicial formaram-se as seguintes equipes Luiz, Tiago e Gilberto pertencendo à equipe 1 e os membros Leonardo, Marília e Maicon à equipe 2. Ficou acordado entre ambas que seria utilizada a documentação do software “Pastelaria LM” elaborada na disciplina de Engenharia de Software I e que seria utilizada uma metodologia genérica para o processo de desenvolvimento, conforme citam os autores Pressman e Maxim (2011), adaptada ao nosso trabalho conforme tabela 1.

Tabela 1 – Metodologia Genérica Adaptada

|  |  |
| --- | --- |
| **ETAPA** | **DESCRIAÇÃO** |
| 1. Comunicação | Desenvolvimento e correção da documentação conforme requisitos de um sistema real para uma pastelaria. |
| 1. Planejamento | Planejamento do projeto, Software gerenciador de versão. |
| 1. Modelagem | Utilização de telas do software existente e documentação com os requisitos, para a modelagem da arquitetura do software. |
| 1. Construção | Desenvolvimento do código fonte |
| 1. Entrega | Entrega do projeto |

Tabela 1 - Fonte: Adaptado de Pressman e Maxim (2011).

Após as correções, deu-se início a etapa de planejamento do projeto, em reunião presencial as equipes decidiram utilizar o aplicativo GitHub para gerenciamento das versões dos documentos. Também ficou definido o aplicativo Team Viewer para realização de futuras reuniões virtuais, onde seria possível o compartilhamento de tela de um membro host para os demais integrantes contribuindo para um desenvolvimento do código em conjunto com todos os membros e uma maior flexibilidade de horários para as reuniões.

Com isso deu-se inicio à etapa de Modelagem que contém um modelo conceitual que engloba a definição da arquitetura utilizada no software, definição das classes, e demais componentes conforme citado por Pressman e Maxim (2011). Para a modelagem, foi utilizado o software Astah onde foram desenvolvidos os diagramas de classes e pacotes por ambas as equipes, conforme revisões da documentação original sendo esses diagramas adicionados à documentação posteriormente. O padrão Modelo-Visão-Controlador (MVC), baseado em componentes organizados em camadas conforme descrito por Sommerville (2011), foi optado pelos grupos para ser utilizado no projeto. Nessa etapa as equipes encontraram dificuldade pois divergiam algumas opiniões quanto a arquitetura das classes que deveria ser adotada no software, surgindo algumas dúvidas quanto à modelagem dos requisitos utilizando o modelo MVC devido a abstração criada pelo pacote DAO e sua adaptabilidade.

A quarta etapa consistia na implementação, onde todos os membros participaram do desenvolvimento da base do código que era comum à duas equipes, através de reuniões presenciais e virtuais, e no final foi necessário direcionar as tarefas específicas de desenvolvimento para as equipes de acordo com a tabela 2. Também ficou definido que algumas funcionalidades não seriam implementadas como o gerenciamento de clientes, realizados pelos atendentes e o gerenciamento de usuários, realizados pelos administradores. Todos os membros participaram do desenvolvimento da base do código que era comum à duas equipes, através de reuniões presenciais e virtuais, e no final foi necessário direcionar as tarefas específicas de desenvolvimento para as equipes de acordo com a tabela 2.

Tabela 2 - Divisão das tarefas de acordo com as equipes

|  |  |  |
| --- | --- | --- |
| **COMPONENTE** | **CLASSE** | **EQUIPE** |
| INTERFACE | INTERFACE TEXTO | AMBAS |
| INTERFACE ADMINISTRADOR | EQUIPE 1 |
| INTERFACE ATENDENTE | EQUIPE 2 |
| MODELO | CLIENTE | AMBAS |
| PEDIDO |
| PRODUTO |
| TIPOPEDIDO |
| CONTROLADOR | CONTROLADOR | EQUIPE 1 |
| DAO | PRODUDTODAO E PRODUTODAOIMPLEMENTS | EQUIPE 1 |
| PEDIDODAO E PEDIDODAOIMPLEMENTS | EQUIPE 2 |

Tabela 2 – Fonte: Elaborada pelos Autores (2018).

Então o trabalho de nossa equipe, equipe 1, foi reestruturado para desenvolvimento do código responsável pelas atividades da interface do administrador, que envolve métodos de gerenciamento de estoque necessários para administrar o fluxo de produtos. Também foi necessário desenvolver e realizar modificações nas classes do pacote “DAO” e Controlador.

Nessa etapa surgiram algumas dificuldades em relação ao uso do Git que não teve a sua funcionalidade bem compreendida pelos membros e surgiram problemas de sobrescrita de arquivos da documentação e do próprio código causando erros, confusões dos membros e retrabalho, sendo necessário voltar a estaca zero reiniciando essa etapa várias vezes e também a criação de um novo repositório de arquivos para sanar alguns problemas relacionados à detecção de versionamento do código. O histórico do novo repositório encontra-se disponível na plataforma GitHub através do seguinte link < https://github.com/Tiagoumr/Projeto-pastelaria>. Outros problemas reportados foram em relação à abstração das interfaces DAO e suas implementações através do DAOImplements, mais especificamente, dúvidas de como e onde ficariam armazenados os atributos dessas classes e suas interações com os demais pacotes do código.

A quinta etapa de entrega será realizada conforme apresentação presencial do projeto para a professora da disciplina no dia 10/12/2018.

Durante a etapa de Construção que consistia no desenvolvimento do código fonte foram realizados testes de verificação através da execução e depuração do programa na sequencia conforme fora implementada, identificando assim falhas na modelagem que exigiram alterações na mesma causando assim uma mudança na estrutura do código fonte, gerando um enorme retrabalho para ambas equipes que se encontravam em um período muito conturbado devido ao período de final de semestre na universidade.

Como conclusão final a equipe conseguiu verificar através da aplicação prática de todos os ciclos de um projeto de desenvolvimento de software que foram estudadas nas disciplinas de Engenharia de Software I e II ministradas no ano de 2018, criando assim uma maior noção e entendimento do vínculo das diversas atividades que estão contidas nas disciplinas, desenvolvendo um pensamento crítico que pode ser aplicado em diversas áreas do curso de Engenharia de Produção.

1. Bibliografia

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

[2] PRESSMAN, Roger S. Engenharia de Software: uma abordagem profissional. 7. ed. São Paulo: Pearson Makron Books, 2011.