Universidade de Brasília
Projeto Métodos de Programação

Checklist do Projeto

Alunos: Daniel Moraes Da Silva - 18/0112333
David Goncalves Mendes - 19/0056967
Kayran Vieira de Oliveira - 19/0031891
Disciplina:Metodos de Programação

Maio
2021

Resumo

Neste documento iremos descrever as regras e padrões que utilizamos para projetos da disciplina. Aqui dispoõe de conceitos importante para melhor aproveitamento de tempo e formas de evitar "bugs"e bons custumes para programas eficientes, claros e otimizados que foram empregados no decorrer do projeto.

1 Como fazer a especificação dos requisitos

1.1 Entendimento da aera de aplicação

É importante deixar claro o contexto o qual o software será inserido e sua funcionalidade. Uma vez que os niveis de abstrações e principalmente de aplicação varia de cada empresa. Ou seja cada empresa tem sua estruta de aplicação.

1.2 Análise do levantamento de requisitos

As motivações e resultados aqui tem extrema relevância,pois na elaboração da especificação dos requisitos a prioridade é definir os meios de troca entre um analista e o desenvolvedor.

1.3 Lidar de forma sistematica com domínios desconhecidos

É importante definir as ferramentas que serão utilizadas em um determinado ambiente, para melhor utilização do programador. Por exemplo: Um software de processamento de imagens de um satélite que utiliza um formato de imagem específico e não qualquer formato "jpeg"por exemplo.

1.4 Definição de papéis

Definir funções e objetivos para cada membro de uma equipe.

1.5 Boa conduta do responsável pelos requisitos

Geralmente escolhe-se um membro para consultar o cliente e coletar as informações primárias para o desenvolvimento.É importante que esse membro tenha boas condutas em reuiniões e esteja sempre atento para melhor coletar os detalhes.

1.6 Elaboração formal da documentação

Aqui utiliza-se alguns modelos para melhor oficialização do projeto. Na documentação explica-se desde o fluxo dos dados e abordagens e outras séries de especificações.

1.7 Definir prioridades

Assim como no iten anterior, ter discernimento e definir o que dever ser feito primeiro são passos crucial para um bom desenvolvimento. Dividir em etapas, selecionando quais etapas devem ser resolvidas primeiro, também é uma boa conduta.

1.8 Ser claro

A crareza em todos os quesitos no desenvolvimento de software é um ponto chave e de extrema importância em uma processo de desenvolvimento. Ser claro no que será feito, no que está sendo feito e no que já foi feito são umas das principais fases de desenvolvimento.

1.9 Brainstorming

Dentro de umas das técnicas empregadas em workshops, o brainstorm (ou da tradução livre "chuvas de idéias") é uma maneira bastante prática de agregar novas ideias e discutilas. Pondendo em muitas ocasiões alavancar o processo de desenvolvimento dos projetos.

1.10 vesões de testes

A fase de testes é uma das etapas mais relevantes no desenvolvimento. Testar o programa ajuda em diversas forma,como verificar erros, bugs, e até estruturação de idéias equivocadas.

1.11 Questionários

Um ramo importante para se definir as necessidades de um código assim buscando diretamente as informações mais precisas com os usuários. Se o usuário do programa em questão não está satisfeito, pode-se enterromper o desenvolvimento para corrigir aplicações incoerente.

1.12 Avaliação de cenários dinâmicos

Olhar para o memsmo problema mas com pespectivas diferentes é fundamental na adaptabilidade e avaliação de varias outras formas de uso do software.

1.13 Pesquisa direta

Levantar opniões sobre como as pessoas lidaram com a idéia do codigo, e como suas aplicações diferem uma das poutra.

1.14 Requisitos não-funcionais

É de suma importância definir com stakeholders ou usuários um apanhado de requisitos do programa.

2 Como fazer o design do software

2.1 Interpretação de protótipos

Aqui é elaborado o ponto de controle primario do software

2.2 Criação do melhor modelo conceitual

Parte teórica do projeto

2.3 Definir as estruturas de dados

Definição de estruturas

2.4 Definir os algoritmos

definir os algoritmos que serão utilizados é importante para a equipe de programadores para melhor elaborar os tipos de operação que serão implementados.

2.5 Definir relações entre os recursos utilizados

Esclarecer como diferentes funções e como diferentes registros interagem entre si.Pode-se efinir uma hierarquia se necessário. Por exemplo, em um programa de uma Pilha, definimos que as funções dependem diretamente uma struct que guarda os atributos necessários para armazenar o obejto de interesse.

2.6 Elaboração da Interface

É importante definir como o usuário se relacionará com o resultado final. Para isso é necessário elaborar esse conceito a partir do perfil de quem utiliza o programa.

2.7 Waterfall development

ou Modelo Sequencial Linear: pode ser considerado como o mais tradicional entre os modelos disponíveis. Nele, cada etapa impacta e é impactada no resultado. Essa metodologia ainda é bastante usada em projetos grandes e complexos, como na implantação de um sistema de gestão empresarial

2.8 Desenvolvimento Ágil de Software

como o SCRUM,DSDM, Extreme Programing (XP), que apostam também na metodologia ágil de desenvolvimento a partir de Sprints e entregas.

2.9 Desenvolvimento Rápido de Aplicação

Introdução de circulos de desenvolvimentos mais curtos e repetitivos.

2.10 Desenvolvimento interativo e incremental

Modelo classico
Serve como um bom complemento para o modelo de cascata.

2.11 Prototipação

Criação de modelos – ou protótipos – que são utilizados para fins de ilustração e desenvolvimento;

2.12 Programar e Arrumar

Consiste em fazer alterações continuamente no software até que seja aceito e entrege. Fases do desenvolvimento

2.13 Análise de requisitos

Em um projeto, os pré-requisitos podem ser elencados e definidos de diferentes formas. Esses critérios nortearão o projeto, por isso, é uma das principais etapas de qualquer desenvolvimento de software.
A análise de requisitos passa por três etapas: elicitação (interação com os usuários finais);
análise;
registros.

2.14 Especificação

Aqui são definidas as funcionalidades do software e suas aplicações, assim como os módulos e formas de entrada e saída de dados.

2.15 Arquitetura de Software

Nessa fase são definidas todas as interfaces, integrações e camadas que o software terá. Também é determinada qual será a forma do sistema garantir que os pré-requisitos de software serão atendidos. Ela deve prever a escalabilidade do projeto e o tratamento de futuros requisitos que possam surgir.

2.16 Implementação

É nessa etapa que todas as especificações são transformadas em código fonte executável.

2.17 Testes

a fase de testes identifica e elimina os erros de software que forem surgindo. Os "bugs" aparecem justamente nesta etapa e é importante corrigi-los.

2.18 Documentação

Nessa etapa, todo o arcabouço utilizado para o desenvolvimento do projeto é incluido, para assim ter uma melhor qualidade da execução bem como produção, análise ou manutenção.

2.19 Suporte e treinamento de software

Capacitar a equipe para usar o software adequadamente

2.20 Manutenibilidade

Ouvir as solicitações e dúvidas dos usuários

2.21 Curva de produtividade

Uma forma de analisar os beneficios tanto dos funcionarios quanto do cliente.

3 Como fazer o código

3.1 Domínio da linguagem

Considerar o nivel de abstração de um codigo é uma analise fundamental para a escrita de um código bem estruturado e livre de problemas de correções a longo prazo.

3.2 Reutilização de códigos que já foram testados

Reuitilizar um código pode otimizar tempo de produção e facilitar na correção de erros e evitar bugs. Vale ressaltar que tais codigos, para servir como basa de criação de um novo codigo, é importante validar que o codigo utilizado já possou por uma tomada de testes.

3.3 Programação Defenciva

Garantir a funcionalidade do software em consições extremas e imprevisiveis.

3.4 Programação Ofenciva

Visa especificar erros externos ao programador, ou seja, minimizar erros de entradas gerados pelo usuario.

3.5 Secure Programming

São práticas defencivas que visa otimizar o nivel de funcionalidade do código. Um cenário que descreve bem essa abordagem é executar o programa da pior forma possível.Gerando assim o maior número de bugs possíveis para assim porder trabalhar e analisar por cima desses bugs.

3.6 Idealizar um projeto em Open Source

Dependendo da aplicação, idealizar uma pespectiva de software livre pode ser extremamente bénefico para um excelente resultado final.Em casos onde a aplicação final é algo em comunidade, a qual lida de forma direta com a comunidade de usuários, uma abordagem open source pode ser bastante benéfica para os usuários e desenvolvedores.

3.7 Especificar corretamente suas variáveis

A especificação de variavel é tem suma importancia na construção de um codigo legivel e entendivel para outros desenvolvedores que iram estudar o codigo ou até mesmo trabalhar em cima do código. Agrupar bem as variáveis e escolher nomes abreviaveis de simples leitura facita tanto no desenvolvimento quanto na documentação final.

3.8 Nomear funções

O nome de uma função deve ser clara e sucinta. Sua explicação deve está em seu propio nome, facilitando assim o entendimento do codigo.

3.9 Não alocar memória excessivamente

A alocação de memória execiva pode gerar danos na execução do programa e dependendo dos caso, pode até causar perda de memoria e funcionalidade do hardware.Para isso, é sempre importante fazer alocações de forma conciente e coerente com a proposta do codigo.

3.10 Muitas variáveis auxiliares criadas

Normalmente não existe a necessidade de se criar um "contador" para cada loop "for" que existir.A criação de muitas variáveis desnecessárias resultará em um código incoerente e mal formulado, de difícil leitura e certamente mal otimizado.

3.11 Identação

A identenção fornece uma estruturação logica e clara para melhor leitura do codigo e organização de ideias a serem analisadas. Para algumas linguagens a identação pode ser um fator determinante para o funcionamento decente do código.

3.12 Avaliar o sinal das variáveis

Unsigned para casos onde não há necessidade das variaveis serem negativas

3.13 Avaliar o tipo das variáveis

Diferentes tipos de variaveis ocupam diferentes tamanhos na memoria. declarar uma variável como “float” e não como “double” é melhor para se otimizar o código.

3.14 Evitar repetições com loops

Muitas repetições podem ser adaptadas em um loop.

3.15 Valores que não mudam ao longo da execução

Pode-se mudar valores para variáveis constantes, porém, se evitando o uso de variavel global.E se determinada se demostrou inutilizavel essa deva ser deletada do codigo.

3.16 Evitar redundância ao explicar funções

Redundância em comentários atrapalha na escrita, acrescenta o numero de linhas e atrapalha no tempo de escrita de um código.

3.17 Manter os comentários simples e diretos

Ser simples e claro é de suma importancia em qualquer codigo e documnetação.

3.18 Separar “blocos” de código usando comentários

Utiliza-se sempre com cautela sem poluir ou desorganizar o codigo e sua identação.

3.19 Comentar o que for importante

Tudo que for relevante em um codigo e que não for auto explicativo, dever ser comentado de forma clara e objetiva.

3.20 Idéia de ”Design by Contract”

Planejamento de software pré definido visando resultados e hierarquia

3.21 Evitar explicar cada variável criada

‘ A explicação de variáveis tem sua importância em um codigo. Mas nem sempre, é necessário explicar algumas variaveis, exites variaveis que são auto explicativa.

4 Como testar o código

4.1 TESTCHECK

Inspecionamento de artefatos de teste de software

4.2 Gcov

Ferramenta usada para análisar a execução efetiva de um código.

4.3 Teste de entradas supostamente inválidas

Tëstes de intradas falsas e invalidas para o codigo, é uma forma de teste e validação.

4.4 Uso de funções que medem o tempo de execução do programa

Está sempre atualizado do tempo de depuração de funções espécíficas ajuda a otimizar tempo, codigo e melhora na performace do codigo.

4.5 Avaliação de variáveis

Checagem dinamica de variaveis durante a execução.

4.6 Execução em outra máquina

A execução em outra maquina, pra muitos dos caso pode ser algo vantajoso para validação de dependências do codigo.

4.7 Condições extremas

Executar o programa implantando testes de condições extremas, pode ajudar na detecção de fálias futuras ou de longo praso no projeto.

4.8 Testar memória

A memória é o compente físico mais importante na interação do software e hardware. Por isso , expor o programa a testes de memória ajudar a entender bem o funcionamento do codigo e da maquina.

4.9 Testar possíveis falhas de segurança

Aferir o nivel de segurançã de um programa, por meio de testes forçados é de suma importância para o bem funcionamento do código e correção de vunerabilidade.

4.10 princípios SOLID

A palavra SOLID é um acróstico onde cada letra significa a sigla de um princípio: SRP, OCP, LSP, ISP e DIP

4.11 Project Warnings

Elimine os alertas do projeto.

5 Como depurar o código

5.1 Ferramenta de depuração nativa do Visual Studio

Facilita na praticidade ao longo do desenvolvimento do codigo.

5.2 Instruções README

Um arquivo readme sempre é útil para esclarecer e instruir o usuario a depurar o codigo.

5.3 Vizualizar Variaveis

É sempre bom acompanhar as alterações das variáveis no decorrer da execução, pois assim tem-sem melhor segurança de que o que foi determinado, de fato está acontecendo.

5.4 Execução Linha a linha

Em algumas IDE esse recurso, muitas vezes chamado de "Next Step", pode facilitar no entendimento do comportamento do codigo e na detecção de erros.

5.5 Parameters

A adição de parametros no momento da execução tem importancia na depuração ou otimização.

5.6 Debug

Executar o codigo por completo.

6 Bibliografia

- Material de aula, Métodos de Programação
- “Técnicas para levantamento de requisitos”, devmedia.com.br
- https://privatebin.info/
- ”Secure Programming for Linux and Unix HOWTO” by David A.Wheeler