Métodos de Programação – Trabalho Final
Laudo de Checklist

a) Como fazer o levantamento de requisitos

1) Nós devemos nos atentar ao que o cliente quer receber (que produtos);
Verificado
2) Devemos saber quais serão os benefícios gerados em estar desenvolvendo este projeto, o que motivou o início deste projeto;
Verificado
3) Devemos documentar o histórico do projeto, se é um projeto existente que irá sofrer melhorias;
Verificado, projeto documentado usando git e trello
4) Devemos nos preocupar com as áreas impactadas, quais áreas que utilizarão a solução proposta;
Verificado
5) Este projeto proposto será utilizado para uma solução interna ou externa, se algum parceiro irá utilizar desta solução;
Não relevante
6) Definir o escopo é a parte mais importante e nele deve conter os limites de entrega do projeto, o conjunto detalhado das suas funcionalidades e as características da demanda;
Verificado, entregue especificação ao professor
7) Devemos nos preocupar com os itens que ficarão fora do escopo que poderá impactar o projeto em pauta, mas que não está previsto pelo cliente como item da demanda;
Verificado, mudanças externas foram documentadas
8) Devemos saber qual é a necessidade de desenvolvimento e qual a solução a ser adotada;
Verificado
9) Se existir parametrização para o requerimento do projeto, estes devem ser informados;
Verificado
10) Devemos nos preocupar com as informações técnicas do projeto, isto é, caso exista troca de arquivo com alguma entidade externa, descrever as informações que deverão constar para esta entidade;
Verificado, não existem agentes externos
11) Importante saber quais são os requerimentos não-funcionais, que são:
Informações sobre Hadware / Software - existe alguma necessidade de hardware especial, é necessário algum terminal, leitora de código de barras e etc;
Informações sobre Interface com Usuário - saber se existe alguma interface com o usuário fora do padrão da empresa solicitante
Informações sobre Integração - verficar se existe integração com outros sistemas, se existir qual é a tecnologia de transferência, este projeto depende de informações provenientes de outros sistemas, existe integração com parceiros, fornecedores, clientes e etc;
Verificado, projeto não leva em conta requisitos de hardware e sistema
12) Devemos saber se existe arquivos que serão gerados a patir do sistema a ser desenvolvido, qual é o seu formato e se este arquivo será transmitido para alguma área ou aplicativo e se existir é necessário saber de que forma isso irá ocorrer;
Verificado, é gerado um arquivo de save de acordo com o especificado
13) É importante saber se o seu projeto necessitará de relatórios, se for necessário é

importante levantar quais as informações que deverá constar no relatório e qual é a periodicidade do mesmo;
Verificado, informações de cada membro foram compartilhadas pelo trello e em reuniões presenciais
14) Importante é pegar os pontos de contatos, pois são eles que tirarão as dúvidas que surgirão enquanto for construída a especificação funcional;
Verificado
15) Cada requisito deve ser expresso de maneira separada, ou seja, sua definição não está dispersa nem ambígua com outros requisitos
Verificado, requisitos enviados

b) Como fazer a especificação dos requisitos

1) Cada requisito está descrito com clareza e sem ambiguidade?
Verificado
2) É consistente? Ou seja, nenhum dos requisitos do documento, tomado individualmente, está em conflito com qualquer outro requisito do mesmo documento;
Verificado
3) Existem requisitos implícitos?
Verificado, itens que teriam maior nível de complexidade do que especificado ou funcionalidades extra estavam já previstos
4) Existem requisitos contendo um nível inadequado de detalhes?
Verificado
5) As restrições e dependências foram claramente escritas?
Verificado
6) Os casos de uso e protótipos definem todas as informações a serem apresentadas aos usuários?
Verificado
7) Os casos de uso descrevem as respostas do sistema ao usuário devido às condições de erro?
Verificado
8) Todas as necessidades que precisam ser consideradas foram tratadas pelos requisitos?
Verificado
9)Os requisitos estão priorizados e foi estabelecida uma ordem de precedência para implementação?
Não verificado, a implementação foi organizada ao longo do projeto
10)Os requisitos poderão ser objetivamente verificados no sistema após sua implementação?
Verificado
11)Os requisitos são apresentados numa estrutura organizada e coerente, contendo índices (tabela de conteúdo) e referências cruzadas?
Verificado
12)Os requisitos contêm redundâncias não controladas, ou seja, a mesma informação em mais de um local sem a devida referência?
Verificado
13)Os requisitos possuem sua fonte de referência (documentos, pessoas ou outros artefatos) claramente identificadas e acessíveis?
Verificado
1
4)Os requisitos são identificados de forma unívoca de modo a facilitar referências

futuras aos mesmos?
Verificado
15)Cada requisito está livre de erros de ortografia e de gramática?
Verificado
16) É completo? Ou seja, contem toda e apenas a informação necessária para que o sistema correspondente seja produzido;
Verificado
17) É modificável? Ou seja, modificações possam ser agregadas aos documentos de forma fácil, completa e consistente, com relação à sua estrutura e estilo;
Verificado

c) Como fazer o design do software

1)Os requisitos documentados do sistema devem ser usados como base para selecionar uma metodologia de design;
Verificado
2)A estrutura do software é identificada usando uma metodologia de design documentada;
Verificado, a metodologia foi descrita
3) O design das interfaces de usuário são elaboradas consultando o dono do sistema a ser desenvolvido;
Verificado, já que os desenvolvedores são dos donos do projeto
4) O design dá suporte tanto ao produto quanto aos objetivos do projeto?
Verificado
5) O design é viável das perspectivas tecnológica, de custo e de prazo?
Verificado
6) Os riscos relacionados a escolha do design foram levados em consideração e existem medidas para mitigá-los?
Verificado
7) O design escolhido é viável para uma futura expansão do projeto?
Verificado
8) O design possui integridade conceitual? (o design como todo faz sentido)
Verificado
9) O design possui técnicas padronizadas e evita procedimentos de difícil compreensão?
Verificado
10) O design é compacto? (utiliza apenas o necessário para seu funcionamento pleno)
Verificado
11) Estâo especificados todo o hardware que irá rodar o design?
Não verificado, não leva em consideração o hardware
12) O design suporta escalabilidade?
Verificado, com mudanças
13) Será fácil migrar o design para outro ambiente se necessário?
Verificado
14) O design é complexo demais? Caso seja, deve-se procurar outras opções mais viáveis
Verificado
15) O design é capaz de desenvolver componentes reutilizáveis se necessario?
Verificado

d) Como fazer o código

1) Existe vetore dimensionados a partir de uma constante?
Int vetor[13]
deveria ser
int vetor[meses+1]2) Uma variável é constante?
Int dias_da_semana = 7;
deveria ser
const unsigned int dias_da_semana = 7;
Verificado
3) Uma variável nunca se torna negativa?
Int tempo;
deveria ser
unsigned int tempo;
Verificado
4) O programa deve usar float ou double?
Float valor_em_dinheiro
deveria ser
unsigned long valor_em_dinheiro;
Verificado
5) O argumento sizeof está sendo usado de maneira correta?
sizeof(ptr) poderia ser sizeof(*ptr)
Verificado
6) É alocado memória suficiente para as variáveis?
Int* v; // quer um vetor de 5 inteiros
v = malloc(sizeof(4* int);
deveria ser
v = malloc(sizeof(5* int);
Verificado
7) Quando desreferenciado um ponteiro pode apontar NULL?
Verificado
8) Quando vai ser copiado um valor de um ponteiro para outro, está levando em consideração se o segundo ponteiro deve apontar o endereço do primeiro ou apontar o item que o primeiro está ligado?
ptr2 = ptr poderia ser ptr2 = ptr->item;
Verificado
9) Quando utiliza-se condicionais, os parentesis são colocados de maneira correta?
if (a = function() == 0)
should be
if ((a = function()) == 0)
Verificado
10) Em condicionais, é utilizado uma igualdade em valores de ponto flutuante?if (someVar == 0.1)
deveria ser
if (someVar < ou > ou =<> 0.1)
Verificado
11) Existem condicionais que sempre vão dar verdadeiro?
if(unsigned int var > 0)
Verificado
12) Existe algum switch que não é terminado com um break;?
switch (variável)
{
case constante1:
Instruções;

case constante2:
Instruções;
default
Instruções;
}
deveria ser
switch (variável)
{
case constante1:
Instruções;
break;
case constante2:
Instruções;
break;
default
Instruções;
}
Verificado
13) O retorno de uma função está contido em um tipo muito pequeno?
funcao_raiz_quadrada(float a)
int retorno = sqrt(a);
return retorno;deveria ser
funcao_raiz_quadrada(float a)
float retorno = sqrt(a);
return retorno;
Verificado
14) O código usa variáveis globais?
Devem ser evitadas, a menos que se tenha total certeza de que não irão conflitar em nenhuma parte do programa
Verificado, as variáveis globais foram planejadas para que só afetassem partes específicas do código
15) O código é feito utilizando ferramentas de debug? Ex: gdb, cppcheck, cpplint, valgrind;
Verificado
e) Como comentar o código e escrever o código com “design by contract” com assertivas de entrada, saída, invariantes e como comentários de argumentação do código

1) O fornecedor deve prover um determinado produto(obrigação) e fica entendido que o cliente tenha pago por isto (benefício);
Não se aplica
2) O cliente deve pagar o valor (obrigação) e é garantido a obter o produto (benefício);
Não se aplica
3) Ambas partes satisfazem suas obrigações, através de leis e regulamentações, aplicadas a todos os contratos;
Não se aplica
4) Toda função deve ser comentada acerca de sua funcionalidade e para esclarecer variáveis ou operações não explícitas no código em si;
Verificado
5) É recomendado comentar ao final de laços de repetição, casos condicionais ex.:

for(){
} // end for()if(){
} // end if()
Não verificado
6) Os comentários devem estar de acordo com a especificação do padrão do código ex.: no padrao verificado pelo cpplint o comentário deve ser feito dando 2 espaços do último caractere do código
para os comentários serem reconhecidos pelo doxygen é necessário um tipos de escrita específicos /*!*/, /// …
Verificado
7) Aguardar uma certa condição para garantir que uma informação no módulo de clientes o chame: a pré-condição da rotina — uma obrigação para o cliente, e o benefício do fornecedor (a rotina em si), eliminando casos de executar por outros meios;
Não se aplica
8) Garantir uma certa condição de saída: A pós-condição da rotina — uma obrigação do fornecedor, e obviamente um benefício (o maior benefício de chamar a rotina) para o cliente;
Não se aplica
9) Manter uma determinada propriedade, assumida na entrada e garantida na saída: uma Classe Constante;
Não se aplica

f) Como testar o código

1) Foram definidos os itens, requisitos e funcionalidades que serão testados?
Verificado
2) Os requisitos de teste foram determinados com base nas prioridades do projeto?
Verificado
3) Foram definidos o escopo e a abrangência dos testes?
Verificado
4) O critério de teste escolhido agrega qualidade ao projeto?
Verificado
5) A abordagem de teste está clara e atende os requisitos de qualidade?
Verificado
6) O ambiente de teste foi contemplado?
Verificado
7) Os testes complementares foram definidos?
Verificado
8) Foram consideradas às restrições ambientais e técnicas do ambiente do sistema?
Não se aplica
9) Os casos especiais são tratados?
Verificado
10) O ambiente foi especificado de acordo com as necessidades?
Verificado
11) Foram verificadas as restrições de máquina?
Não se aplica
12) Precisa ser instalado algum componente?
Verificado, não se aplica ao contexto
13) A massa de dados para teste foi gerada?
Verificado
14) A massa de dados contempla todas as funcionalidades e estruturas internas do

módulo?
Verificado
15) A massa de dados para teste contempla todos os casos de teste?
Verificado
16) A massa de dados testa valores de fronteira, valores nulos, valores negativos, valores de tipos diferentes e os caminhos independentes do código?
Verificado
17) Foi testado a consistência dos dados de entrada?
Verificado
18) Foram testados campos obrigatórios, digitos verificadores, datas válidas, domínios de tabela, valores de fronteira, valores nulos, valores padrão (default) e de tipos diferentes?
Verificado
19) Foram testados a necessidade de caixa alta/baixa e a dependência de valores comoutros atri butos?
Verificado
20) As expressões e operações lógicas foram revisadas?
Verificado
21) O código está de acordo com os padrões da empresa?
Verificado
22) Os módulos codificam o que está especificado no projeto?
Verificado
23) O tratamento de erros e exceções foi incluído no código?
Verificado
24) As mensagens do programa estão claras?
Verificado
25) Todos os dados foram atualizados corretamente?
Verificado
26) Todas as operações de atualização permitidas foram testadas?
Verificado
27) Foi verificada a situação de duplicidade de dados?
Verificado
28) Foi verificada a solicitação de atualização/exclusão de um dado inexistente?
Verificado
29) O relatório foi testado com mais de uma página?
Verificado
30) Foram testadas todas as quebras do relatório?
Verificado
31) Os totais do relatório estão corretos?
Verificado
32) Foram feitas simulações para verificar interação com outros sistemas e bases de dados?
Não necessário
33) Todas as possibilidades de execução da estrutura da aplicação foram consideradas e simuladas?
Verificado

g) Como depurar o código

1) O código foi depurado com ferramentas externas de teste e checagem de erros? ex.: valgrind, gdb, cppcheck;
Verificado

2) Foi-se levado em consideração todos os erros e warnings apresentados na compilação do projeto? ex.: (segmentation fault, main exited -1);
Verificado
3) Deve-se isolar o bug em um projeto separado para que se possa testar maneiras de resolver o problema;
Verificado
4) Usando um programa de depuração deve-se aplicar checkpoints onde se suspeita ser a origem do bug;
Verificado
5) Rode o programa aos poucos para garantir que não passe nenhum detalhe que possa comprometer a depuração de partes mais adiante;
Verificado
6) Sua solução apresentada resolveu por inteiro o problema em questão? Se sim avance o código, se não procure outras alternativas de abordar a questão;
Verificado
7) Para evitar bugs no código, as vezes pode ser recomendado a análise de uma opinião externa aos programadores que estão liderando o projeto, para evitar a “síndrome da ideia fixa” e talvez unir novas perspectivas e conhecimentos a cerca do assunto;
Verificado
8) Certifique-se que todos os requisitos da seção “d” foram cumpridos. (é possível que o problema seja originário de uma má prática no desenvolvimento do código);
Verificado
9) Rode o projeto em diversos dispositivos, a fim de garantir que não haja problemas de compatibilidade e que alguma funcionalidade não funcione perfeitamente em todos os ambientes possíveis;
Verificado
10) Procure áreas do código que podem ser otimizados e que podem ser eliminados más práticas de programação que podem ser danosas ao projeto;
Verificado
11) Verifique se todas as entradas são constantes e dentro do esperado na especificação do código;
Verificado
12) Verifique que as saídas estão de acordo e no formato padronizado;
Verificado