UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

CONCURSO PÚBLICO 01/2022 UDESC

PADRÃO DE RESPOSTA DAS QUESTÕES CONSTANTES NA PROVA ESCRITA

Área de Conhecimento: Ciências Exatas e da Terra/ Ciência da Computação

Modelo Referencial:

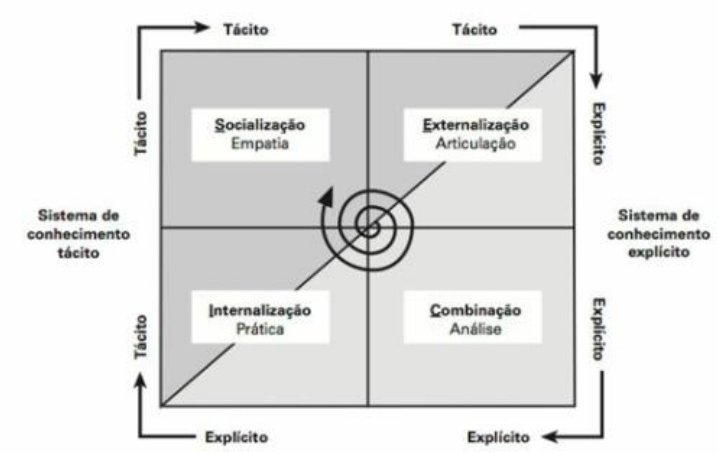
Questão 01 (valor 2,0)

De acordo com Takeuchi e Nonaka (2008), a criação do conhecimento consiste em quatro processos de conversão. Cite-os e explique-os.

RESPOSTA:

A criação do conhecimento consiste em quatro processos de conversão:

- SOCIALIZAÇÃO: é o processo que cria conhecimento tácito a partir de conhecimento tácito através da partilha de experiências. O compartilhar da cultura organizacional não expresso em palavras é um exemplo disso
- EXTERNALIZAÇÃO: é o processo que converte o conhecimento tácito de um indivíduo em conhecimento explícito, usando palavras e códigos. Em uma organização, a externalização é geralmente conduzida através do diálogo.
- COMBINAÇÃO: é o processo que cria conhecimento explícito novo, categorizando e combinando o conhecimento explícito externalizado. As peças e os produtos podem ser considerados como a incorporação do conhecimento explícito. Fazer produtos a partir de combinações de peças pode ser considerado um processo de combinação
- INTERNALIZAÇÃO: é o processo que incorpora o conhecimento explícito externalizado de volta ao conhecimento tácito do indivíduo. O conhecimento tácito criado a partir da internalização é o que é sentido quando os indivíduos colocam manuais em prática e usam os produtos. As características do novo pensamento sobre o conhecimento da marca são comparadas com aquelas do pensamento convencional refletidas no Modelo de Valor da Marca Baseado no Consumidor.



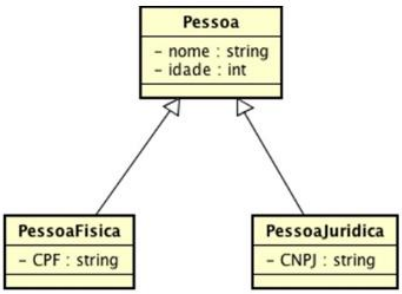
(FONTE: Gestão do Conhecimento. TAKEUCHI, H.; NONAKA, I, 2008. Págs. 254-255)

Questão 02 (valor 2,0):

O conceito de herança é um dos conceitos básicos do Paradigma da Orientação a Objetos. Considerando isso, exemplifique o uso de herança com diagramas UML e com um código escrito em uma linguagem de programação orientada a objetos.

RESPOSTA:

Usando UML e Python




```
class Pessoa:
    __nome = None
    __idade = None
    def __init__(self, nome, idade):
        self.__nome = nome
        self.__idade = idade
    def setnome(self, nome):
        self.__nome = nome
    def setidade(self, idade):
        self.__idade = idade
    def getnome(self):
        return self.__nome
    def getidade(self):
        return self.__idade
```

```
class PessoaFisica(Pessoa):
    __CPF = None
    def __init__(self, CPF, nome, idade):
        super().__init__(nome, idade)
        self.__CPF = CPF
    def getCPF(self):
        return self.__CPF
    def setCPF(self, CPF):
        self.__CPF = CPF

class PessoaJuridica(Pessoa):
    __CNPJ = None
    def __init__(self, CNPJ, nome, idade):
        super().__init__(nome, idade)
        self.__CNPJ = CNPJ
    def getCNPJ(self):
        return self.__CNPJ
    def setCNPJ(self, CNPJ):
        self.__CNPJ = CNPJ
```

Questão 03 (valor 2,0):

Em relação a modelagem de software orientada a objetos, defina cada um dos conceitos abaixo e apresente um ou mais diagramas UML identificando tais conceitos no(s) diagrama(s):

- classe
- atributos
- operações
- generalização
- associação

- restrição
- multiplicidade
- notas

RESPOSTA:

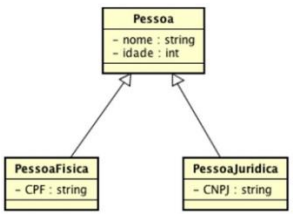
(FONTE: Utilizando UML e Padrões - Craig Larman – Cap. 3)

- CLASSE: Em UML, uma classe representa um objeto ou um conjunto de objetos que compartilham uma estrutura e um comportamento comuns. As classes, ou instâncias de classes, são elementos de modelo comuns nos diagramas UML. Uma classe identifica os atributos, operações, relacionamentos e semântica que as instâncias, ou objetos, da classe possuem. Cada objeto que instancia uma classe possui seus próprios valores de atributos. O nome de uma classe, que pode ser derivado do vocabulário do sistema que está sendo modelado, reflete sua função ou uma de suas funções no sistema. Em UML uma classe é representada como um retângulo com três compartimentos: A divisão superior exibe o nome da classe; A divisão do meio exibe uma lista de atributos; e, A divisão inferior exibe uma lista de operações.

- ATRIBUTOS: um atributo representa uma definição de dados para uma instância de um classificador. Um atributo descreve um intervalo de valores para a definição de dados. Um classificador pode ter qualquer quantidade de atributos ou então nenhum.

- OPERAÇÕES: uma operação solicita um serviço que um classificador ou uma instância de uma classe é chamado a executar. Operações são contidas por classes e interfaces. Um classificador pode ter qualquer quantidade de operações ou então nenhuma.

- GENERALIZAÇÃO: Uma generalização é um relacionamento que podemos chamar de "é um tipo de", onde objetos gerais (no exemplo abaixo, Pessoa) se relacionam com objetos mais específicos do mesmo tipo (PessoaFisica, PessoaJuridica). No exemplo, pode-se dizer que Pessoas Físicas e Pessoas Jurídicas são um tipo de Pessoa. Uma generalização define uma herança, tal que uma classe refina, isto é, especializa detalhes sobre a classe mais geral. A classe generalizada é freqüentemente chamada de superclasse, e a classe especializada subclasse. Todos os atributos e operações da classe generalizada que tem visibilidade pública e protegida, estão disponíveis para a subclasse. Uma generalização tem um triângulo apontando para a superclasse.



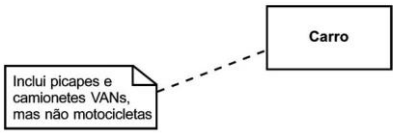
- ASSOCIAÇÃO: Uma associação define que os objetos de uma classe são conectados a objetos de outra classe. Sem essa associação nenhuma mensagem pode passar entre objetos da classe em tempo de execução. Existe uma associação entre duas classes se

uma instância de uma classe deve conhecer sobre a existência da outra de modo a realizar seu trabalho. No diagrama UML, uma associação é uma linha conectando duas classes. Podemos definir a direcionalidade de associação colocando setas abertas nas extremidades da linha. Com isso pode-se definir como é feita a navegação entre as classes. Quando não se coloca setas a navegação é definida como bidirecional.

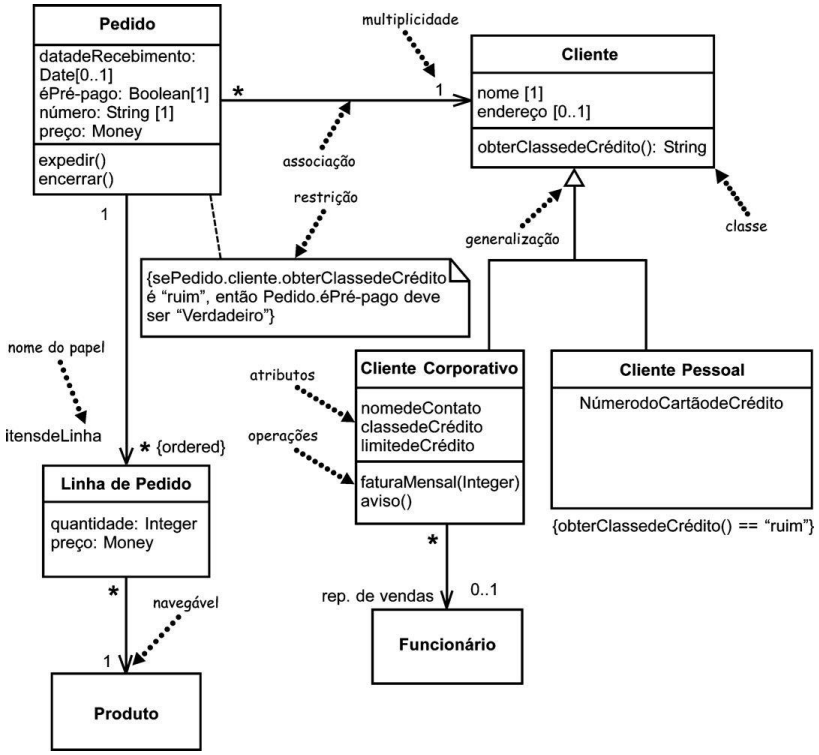
- RESTRIÇÃO: Nos modelos UML, uma restrição é um mecanismo de extensão que permite refinar a semântica de um elemento de modelo UML. Uma restrição refina um elemento de modelo expressando uma condição ou uma restrição com a qual o elemento de modelo deve estar em conformidade. A única regra é que você as coloque entre chaves ({}). Você pode utilizar linguagem natural, uma linguagem de programação ou a linguagem formal de restrições de objetos de UML.

- MULTIPLICIDADE: A multiplicidade nada mais é do que os limites inferior e superior da quantidade de objetos aos quais outro objeto está associado. Esses limites podem receber os valores: apenas um; zero ou muitos; um ou muitos; zero ou um; ou até um intervalo específico.

- NOTAS: Notas são comentários nos diagramas. As notas podem ser isoladas ou vinculadas, com uma linha tracejada, aos elementos que estão sendo comentados. Elas podem aparecer em qualquer tipo de diagrama.



A figura abaixo ilustra os conceitos citados acima



Questão 04 (valor 2,0):

David Hooker propôs sete princípios que se concentram na prática da engenharia de software como um todo. Cite e comente cada um destes princípios.

RESPOSTA:

(FONTE: PRESSMAN, R.; MAXIM, B. Engenharia de Software: Uma abordagem profissional. 9 ed. Porto Alegre: Grupo A, 2021. Item 1.4.2)

Primeiro princípio: a razão de existir

Um sistema de software existe por um motivo: agregar valor para seus usuários. Todas as decisões devem ser tomadas com esse princípio em mente. Antes de especificar um requisito de um sistema, antes de indicar alguma parte da funcionalidade de um sistema, antes de determinar as plataformas de hardware ou os processos de desenvolvimento, pergunte a si mesmo: "Isso realmente agrega valor real ao sistema?". Se a resposta for "não", não o faça. Todos os demais princípios se apoiam neste primeiro.

Segundo princípio: KISS (Keep It Simple, Stupid!, ou seja: não complique!)

Existem muitos fatores a considerar em qualquer trabalho de projeto. Todo projeto deve ser o mais simples possível, mas não simplista. Este princípio contribui para um sistema mais fácil de compreender e manter. Isso não significa que características, até mesmo as internas, devam ser descartadas em nome da simplicidade. De fato, os projetos mais elegantes normalmente são os mais simples. Simples também não significa "gambiarra". Na verdade, muitas vezes são necessárias muitas reflexões e trabalho em várias iterações para simplificar. A contrapartida é um software mais fácil de manter e menos propenso a erros.

Terceiro princípio: mantenha a visão

Uma visão clara é essencial para o sucesso de um projeto de software. Sem uma integridade conceitual, corre-se o risco de transformar o projeto em uma colcha de retalhos de projetos incompatíveis, unidos por parafusos inadequados. Comprometer a visão arquitetural de um sistema de software debilita e até poderá destruir sistemas bem projetados. Ter um arquiteto responsável e capaz de manter a visão clara e de reforçar a adequação ajuda a assegurar o êxito de um projeto.

Quarto princípio: o que um produz, outros consomem

Sempre especifique, projete, documente e implemente ciente de que mais alguém terá de entender o que você está fazendo. O público para qualquer produto de desenvolvimento de software é potencialmente grande. Especifique tendo como objetivo os usuários. Projete tendo em mente os implementadores. Codifique se preocupando com aqueles que deverão manter e ampliar o sistema. Alguém terá de depurar o código que você escreveu, e isso o torna um usuário de seu código. Facilitando o trabalho de todas essas pessoas, você agrega maior valor ao sistema.

Quinto princípio: esteja aberto para o futuro

Nos ambientes computacionais de hoje, em que as especificações mudam de um instante para outro e as plataformas de hardware se tornam rapidamente obsoletas, a vida de um software, em geral, é medida em meses em vez de em anos. Contudo, os verdadeiros sistemas de software com “qualidade industrial” devem durar muito mais. Para serem bem-sucedidos nisso, esses sistemas precisam estar prontos para se adaptar a essas e outras mudanças. Sistemas que obtêm sucesso são aqueles que foram projetados dessa forma desde seu princípio. Jamais faça projetos limitados. Sempre pergunte “e se” e prepare-se para todas as respostas possíveis, criando sistemas que resolvam o problema geral, não apenas o específico.

Sexto princípio: planeje com antecedência, visando a reutilização

A reutilização economiza tempo e esforço. Alcançar um alto grau de reutilização é indiscutivelmente a meta mais difícil de ser atingida ao se desenvolver um sistema de software. A reutilização de código e projetos tem sido proclamada como uma grande vantagem do uso de tecnologias orientadas a objetos. Contudo, o retorno desse investimento não é automático. Planejar com antecedência para a reutilização reduz o custo e aumenta o valor tanto dos componentes reutilizáveis quanto dos sistemas aos quais eles serão incorporados.

Sétimo princípio: pense!

Este último princípio é, provavelmente, o mais menosprezado. Pensar bem e de forma clara antes de agir quase sempre produz melhores resultados. Quando se analisa alguma coisa, provavelmente ela sairá correta. Ganha-se também conhecimento de como fazer correto novamente. Se você realmente analisar algo e mesmo assim o fizer da forma errada, isso se tornará uma valiosa experiência. Um efeito colateral da análise é aprender a reconhecer quando não se sabe algo, e até que ponto poderá buscar o conhecimento. Quando a análise clara faz parte de um sistema, seu valor aflora. Aplicar os seis primeiros princípios exige intensa reflexão, para a qual as recompensas em potencial são enormes.

Se todo engenheiro de software e toda a equipe de software simplesmente seguissem os sete princípios de Hooker, muitas das dificuldades enfrentadas no desenvolvimento de sistemas complexos baseados em computador seriam eliminadas.

Questão 05 (valor 2,0):

Em relação ao SCRUM responda:

a) O que é e qual a relação com o manifesto ágil?

b) Cite e explique os principais papéis, eventos e artefatos.

c) Cite e comente três vantagens e três desvantagens deste método?

RESPOSTA:

PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de software. Grupo A, 2021. E-book. ISBN 9786558040118. Página 41 - 51

a) *Scrum* (o nome provém de uma atividade que ocorre durante a partida de *rugby*) é um método de desenvolvimento ágil de software bastante popular concebido por Jeff Sutherland e sua equipe de desenvolvimento no início dos anos 1990. Ela foi criada principalmente para projetos de desenvolvimento de software, cujo objetivo é disponibilizar novos recursos de software a cada 2 a 4 semanas. Os princípios do *Scrum* são coerentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades metodológicas: requisitos, análise, projeto, evolução e entrega. Em cada atividade metodológica, ocorrem tarefas realizadas em um período (janela de tempo) chamado de *sprint*. O trabalho realizado dentro de um *sprint* (o número de *sprints* necessários para cada atividade metodológica varia dependendo do tamanho e da complexidade do produto) é adaptado ao problema em questão e definido, e muitas vezes modificado em tempo real, pela equipe *Scrum*.

b) - papéis: *Scrum Master*, *Product Owner* e a Equipe de Desenvolvimento *Scrum*;

- Eventos: planejamento, *stand-up* diário, revisão de *sprint* e retrospectiva de *sprint*;

- Artefatos: lista de pendências de *sprint*, lista de pendências do produto, gráfico de burndown, log, etc;

c) Prós do Scrum

- O product owner define as prioridades.
- A equipe se responsabiliza pela tomada de decisões.
- A documentação é leve.
- Apoia atualizações frequentes.

Contras do Scrum

- É difícil controlar o custo das mudanças.
- Pode não ser apropriado para equipes de grande porte.
- Exige membros de equipe especializados.

Assinaturas do documento

Código para verificação: **79FM0V3S**

Este documento foi assinado digitalmente pelos seguintes signatários nas datas indicadas:

ALVARO ROGERIO CANTIERI (CPF: 963.XXX.029-XX) em 07/11/2022 às 11:26:53
Emitido por: "AC Final do Governo Federal do Brasil v1", emitido em 07/11/2022 - 11:22:06 e válido até 07/11/2023 - 11:22:06.
(Assinatura Gov.br)

NILSON RIBEIRO MODRO (CPF: 988.XXX.239-XX) em 07/11/2022 às 11:31:06
Emitido por: "SGP-e", emitido em 30/03/2018 - 12:41:45 e válido até 30/03/2118 - 12:41:45.
(Assinatura do sistema)

LUIZ CLAUDIO DALMOLIN (CPF: 380.XXX.000-XX) em 07/11/2022 às 11:39:53
Emitido por: "SGP-e", emitido em 30/03/2018 - 12:39:25 e válido até 30/03/2118 - 12:39:25.
(Assinatura do sistema)

Para verificar a autenticidade desta cópia, acesse o link https://portal.sgpe.sea.sc.gov.br/portal-externo/conferencia-documento/VURFU0NfMTIwMjJfMDAwNDk3ODJfNDk4NTFfMjAyMl83OUZNMFYzUw== ou o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo **UDESC 00049782/2022** e o código **79FM0V3S** ou aponte a câmera para o QR Code presente nesta página para realizar a conferência.