PROVA I

P: Imagine que você está participando de uma reunião de projeto como o arquiteto de software sênior e um de seus colegas não entende claramente padrões de projeto e faz a seguinte pergunta: "Existe relação entre os padrões GRASP e o GOF?" O que você responderia para o colega?

R: Pelo fato dos padrões GRASP consistirem na resolução de problemas comuns no desenvolvimento, algumas pessoas não consideram essas práticas como “padrões de projeto” exatamente e consideram mais como uma filosofia de design. Além disso, alguns dos padrões GOF implementam soluções que utilizam conceitos discutidos pelos padrões GRASP.

P: Criar um código de qualidade durante toda a fase de desenvolvimento é, sem dúvidas, a missão de qualquer desenvolvedor que realmente se importa com o produto final do software. O uso de boas práticas de programação tem por finalidade justamente reduzir a complexidade do código, o acoplamento entre classes, separar responsabilidades e definir muito bem as relações entre elas, como forma de melhorar a qualidade interna do código-fonte. Um dos princípios que nos ajuda neste sentido são os princípios SOLID. O acrônimo SOLID foi introduzido por Michael Feathers, após observar que os cinco princípios poderiam se encaixar nesta palavra. São eles: Single Responsibility, Open Closed, Liskov Substitution, Interface Segregation e Dependency Inversion.

Escolha um desses princípios e descreva a) O que é? b) Vantagens. c) Desvantagens. d) Exemplo.

R: Interface Segregation Principle

- O princípio de segregação de interface afirma que nenhum cliente deve ser forçado a depender de métodos que não vai usar. Dividindo interfaces que são muito grandes em interfaces menores e que sejam mais específicas, de modo que os clientes só precisam saber dos métodos que interessam a eles.
- A vantagem de utilizar esse princípio é que para realizar a manutenção de um sistema ou implementar modificações se torna mais fácil e menos trabalhoso. Além de diminuir o acoplamento de classes, que de contrário seria muito trabalhoso implementar qualquer alteração no sistema sem que houvesse diversas alterações adicionais.

- O problema em aplicar esse princípio é que devido a separar interfaces grandes em menores, a quantidade de interfaces e arquivos pode se tornar muito maior resultando em mais verbosidade e uma olhada rápida no código base torna-se mais difícil
- Um exemplo famoso foi justamente como se originou esse princípio quando Robert C. Martin estava prestando uma consulta para a empresa Xerox e nessa empresa havia sido criado um sistema para uma impressora multitarefa, mas com o passar do tempo realizar modificações foi se tornando cada vez mais e mais difícil.
  O problema era que havia apenas uma classe responsável por todas as funções da impressora, de forma que um método para grampear teria acesso a todos os métodos da impressora, mesmo sem haver necessidade de sabê-los.
  A solução sugerida foi justamente dividir essa interface grande em várias interfaces menores a fim de que cada função necessitasse apenas de métodos relativos ao funcionamento da mesma.