PDS

O que é Design de Software?

Preenche a lacuna que existe entre saber o que é preciso (requisitos do software) e a construção do código (fase de construção).

Requirements -> Design -> Construction

O processo de design pode-ser tornar mais sistemático e previsível através da aplicação de métodos, técnicas e padrões aplicados de acordo com princípios e heurísticas.

Programas mal desenhados são difíceis de entender e modificar. Quanto maior é o programa, mais acentuadas são as consequências de um mau desenho.

Complexidades essenciais são inerentes ao problema, Complexidade acidental são artefactos da solução.

Total de complexidade de um programa = Essenciais + Acidentais.

O objectivo principal do design de software é controlar a complexidade.

Métodos para controlar a complexidade:

- Modularity – subdivide the solution into smaller easier to manage components. (divide and conquer)
- Abstraction – use abstractions to suppress details in places where they are unnecessary.
- Information Hiding – hide details and complexity behind simple interfaces
- Inheritance – general components may be reused to define more specific elements.
- Composition – reuse of other components to build a new solution

É não determinístico - Não há 2 designers ou designs que produzem o mesmo resultado.
Heurístico -
Emergente - o design final evolui através de experiência e feedback.

Processo:

- Entender o problema
- Construir um modelo de solução
- Procurar soluções já existentes

- Considerar construir protótipos
- Documentar e rever o design
- Iterar a solução (refactor)

Minimal complexity – Keep it simple. Maybe you don't need high levels of generality.
Loose coupling – minimize dependencies between modules
Ease of maintenance – Your code will be read more often then it is written.
Extensibility – Design for today but with an eye toward the future. Note, this characteristic can be in conflict with "minimize complexity". Engineering is about balancing conflicting objectives.
Reusability – reuse is a hallmark of a mature engineering discipline
Portability – works or can easily be made to work in other environments
High fan-in on a few utility-type modules and low-to-medium fan-out on all modules. High fan-out is typically associated with high complexity.
Leanness – when in doubt, leave it out. The cost of adding another line of code is much more than the few minutes it takes to type.
Stratification – Layered. Even if the whole system doesn't follow the layered architecture style, individual components can.
Standard techniques – sometimes it's good to be a conformist! Boring is good.
Production code is not the place to try out experimental techniques.

GRASP

Creator

Quem cria uma instância do objecto A?

Atribuir a classe B a inicialização se um ou mais for verdade:

- B agrega A (numa coleção)
- B regista A
- B usa frequentemente A
- B tem os dados para inicializar A

Promove o baixo acoplamento ao responsabilizar a criação de objectos às classes que o referenciam. Ligar um objeto ao seu criador quando:

- Agrega
- Contém
- Regista
- Tem os dados de inicialização

No entanto este padrão por requerer uma complexidade significativa.

Information Expert

Como atribuir responsabilidades a objectos?

É uma abordagem genérica que visa atribuir a responsabilidade de fazer ou conhecer algo ao "especialista na informação" — a classe que possui a informação necessária para cumprir tal responsabilidade.

Atribuímos a responsabilidade à classe que tem informação para a concretizar.

Exemplo: Board tem squares. O getSquare e da responsabilidade do board.

Pode ser necessário espalhar a informação pelas várias classes, cada uma “expert” no seu próprio dominio.

Facilita o encapsulamento da informação

- As classes usam a informação que possuem para concretizar tarefas - classes muito coesivas
- Código mais fácil de entender
- Promove o baixo acoplamento

No entanto pode tornar uma classe demasiado complexa. Exemplo guardar informacao numa base de dados.

Low Coupling Pattern

Reduzir o maior número de dependências possíveis entre objectos. Deste modo as mudanças não causam tanto impacto. Promove também a reusabilidade.
A solução é atribuir responsabilidades para que o acoplamento se mantenha baixo. Evitar que uma classe tenha que saber sobre várias outras.

O acoplamento mede o quão forte uma classe depende, está ligada ou necessita de objectos de outras classes.
Classes com acoplamento alto sofrem com mudanças das classes relacionadas, são mais difíceis de manter e entender e de reutilizar.

Acoplamento mais elevado não é problema em casos em que as classes são estáveis
Exemplo: libraries e classes bem testadas.

High Cohesion Pattern

A coesão mede o quao forte e focadas são as responsabilidades relacionadas de um elemento.
Como manter as classes focadas e controladas? Atribuir responsabilidade para que a coesão se matenha alta.

Torna o código mais legível e facilita a manutenção. Complementa o baixo acoplamento. No entanto, às vezes é necessário criar objectos servidor menos coerentes.

Controller

Um controlador é um objeto que não é de interface GUI responsável pelo tratamento de eventos do sistema. Um controlador define métodos para as operações do sistema

PATTERNS

Factory Method Pattern

No método de Fabrica, o objectivo é criar objetos sem expor a lógica da criação para o cliente e referir ao novo objeto criado através de uma interface comum
Em palavras simples, se tivermos uma super class e n-sub classes, baseado na data providenciada, queremos retornar o objeto de uma das subclasses. O princípio basico deste padrão é que em “run time” obtemos um objecto de um tipo baseado no parâmetro passado.

É fácil de implementar, a aplicação de cliente não precisa de mudar drasticamente

- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Defining a "virtual" constructor.
- The `new` operator considered harmful.

passar construtor para privado e criado um static method publico.

Abstract factory

É uma super-fábrica que cria outras fábricas. É também chamado de Fábrica de fábricas. O objectivo é criar uma interface para criar famílias relacionados a objetos sem especificar concretamente a sua classe.

Builder

O objetivo é separar a construção de um objecto complexo da sua representação de modo a que o mesmo processo de construção possa criar diferentes representações.

O objeco é construído passo a passo. Esta construção é independente de outros objectos. Dev

Singleton

O objectivo do singleton é assegurar que uma classe tem uma, e só uma instância e providenciar um ponto de acesso global à mesma.

Pode ser encapsulada durante a inicialização ou inicializada na primeira utilização (lazy initiation)