THE DEVELOPER'S CONFERENCE

Trilha – Design de Código

Alessandra Monteiro Martins

Especialista em Governança de TI pela Universidade Católica de Brasília,
Licenciada em Informática pela Universidade do Estado do Amazonas,
Certificações ITIL, COBIT, ISO27002,CTFL, KMPI, Scrum Master, CLF

THE DEVELOPER'S CONFERENCE

FemmeIT
O JEITO FEMININO DE FALAR DE TI

Code Smell e Métricas para Qualidade de Código

Agenda

THE DEVELOPER'S CONFERENCE

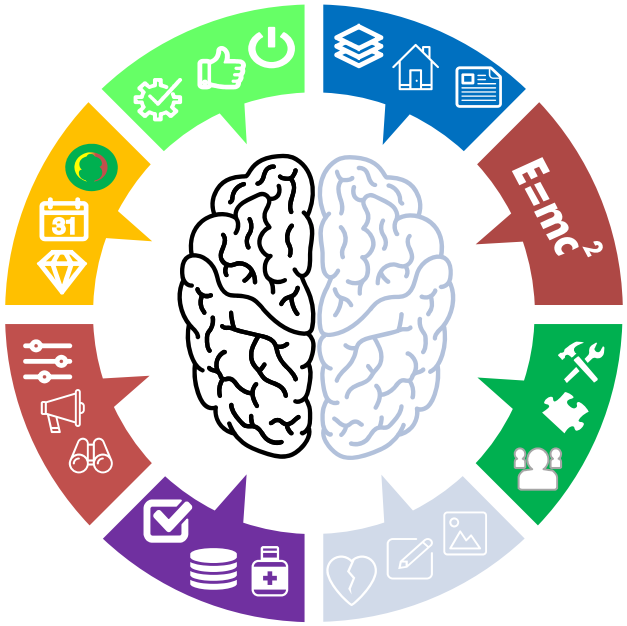
- Engenharia de Software– Contexto e Conceitos
  - *Paradigma de Orientação a Objetos*
  - *Linguagens*
  - **Design Patterns**
    - *Manutenabilidade*
    - *Sustentabilidade*
  - **Code Smell**
    - *Plug-ins & Ferramenta*
    - *Segurança*
      - **Métricas de Qualidade de Código**
        - *Papéis*

Knowledge Area (KA) SWEBOK

THE DEVELOPER'S CONFERENCE

Conceitos - Áreas de Conhecimento da Engenharia de Software

- REQUISITOS DE SOFTWARE
- DESIGN DE SOFTWARE
- CONSTRUÇÃO DE SOFTWARE
- TESTE DE SOFTWARE
- MANUTENÇÃO DE SOFTWARE
- MODELOS E MÉTODOS DE ENGENHARIA DE SOFTWARE
- QUALIDADE DE SOFTWARE



- ENGENHARIA DE SOFWARE ECONÔMICA

- GERENCIAMENTO DE CONFIGURAÇÃO DE SOFTWARE
- GERENCIAMENTO DE EENGENHARIA DE SOFTWARE
- PROCESSO DE ENGENHARIA DE SOFTWARE
- PRÁTICA PROFISSIONAL DE ENGENHARIA DE SOFTWARE
- FUNDAMENTOS DE COMPUTAÇÃO
- FUNDAMENTOS MATEMÁTICOS
- FUNDAMENTOS DE ENGENHARIA

Engenharia de Software: Contexto e Conceitos

THE DEVELOPER'S CONFERENCE

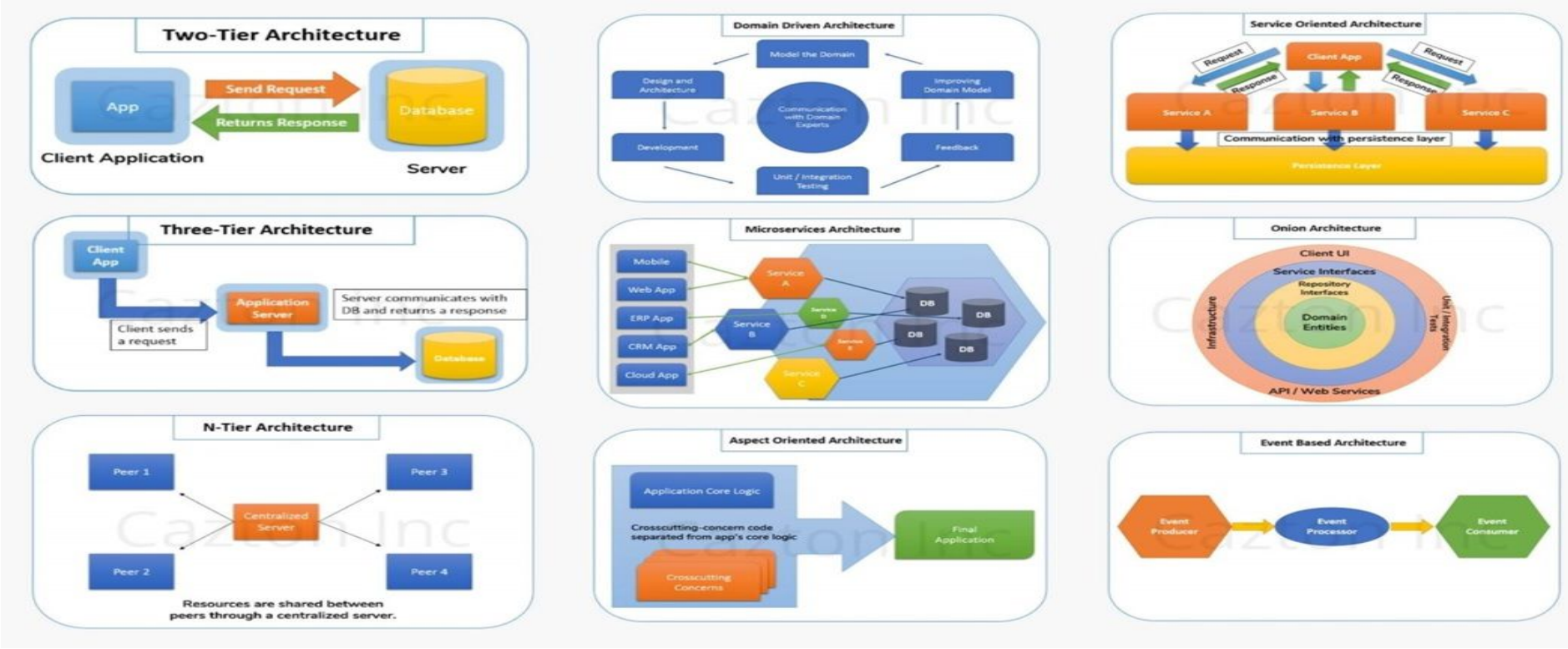
Engenharia de Software: Requisitos

THE DEVELOPER'S CONFERENCE

- Qualidade dos Requisitos: Características de Qualidade
- Correto – Coerência do Enunciado
- Preciso – UML | Revisões e Inspeções
- Completo – Definições de resposta para todas as entradas| Termos e entidades|
- Consistente – Conflito lógico ou temporal entre ações
- Priorizado – MOSCOW – Essencial, Desejável, Opcional
- Verificável – Processo finito, custo compensador, mostrar conformidade do produto
- Modificável – organização coerente | ausência de redundância
- Rastreável – Para Trás e para Frente

Qualidade de Software: Desenho - Arquitetura

THE DEVELOPER'S CONFERENCE



- Arquitetura de Software define instruções de estrutura, diretriz e liderança

Paradigma de Orientação a Objetos

THE DEVELOPER'S CONFERENCE

- Modelo de Objetos ->
- – Objeto
- – Mensagem
- – Método
- Classe
- Atributo
- Protocolo

- Classificação
- Herança
- Encapsulamento/Ocultamento de Informação
- Polimorfismo

Objeto

=

Atributos (Dados)

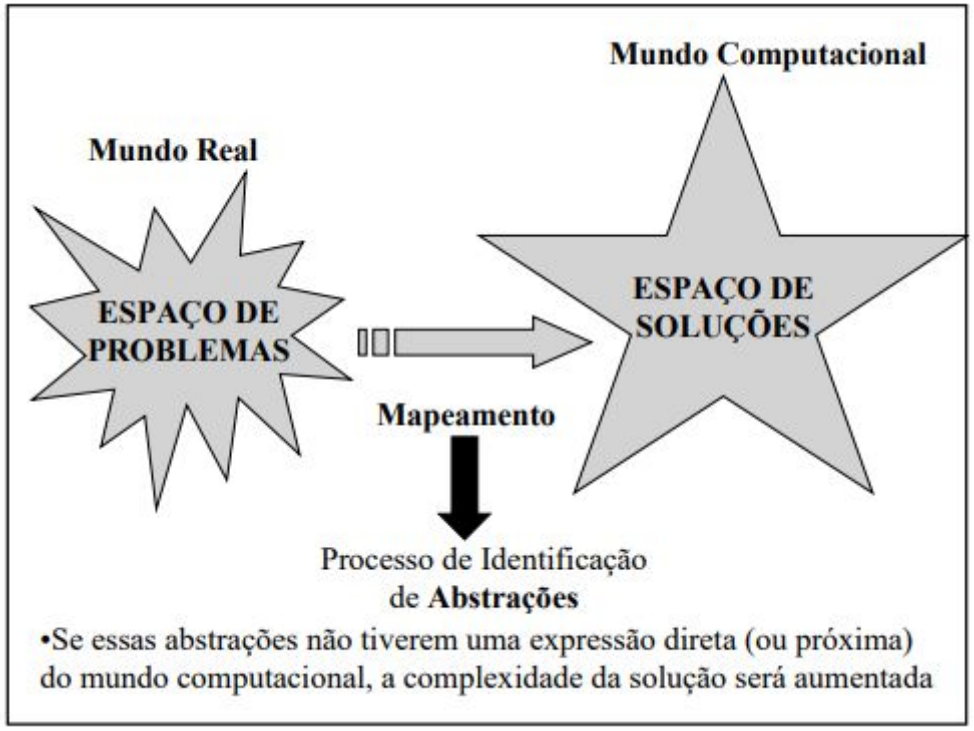
+

Métodos (Funcionalidade)

+

Encapsulamento

Paradigma de Orientação a Objetos



- 1 - Abstração:
- Identidade
- Propriedade
- Métodos

Paradigma de Orientação a Objetos

THE DEVELOPER'S CONFERENCE

- 2 - Encapsulamento:

- Elemento que adiciona segurança à aplicação em uma programação orientada a objetos pelo fato de esconder as propriedades, criando uma espécie de caixa preta.

Métodos getters	Métodos setters

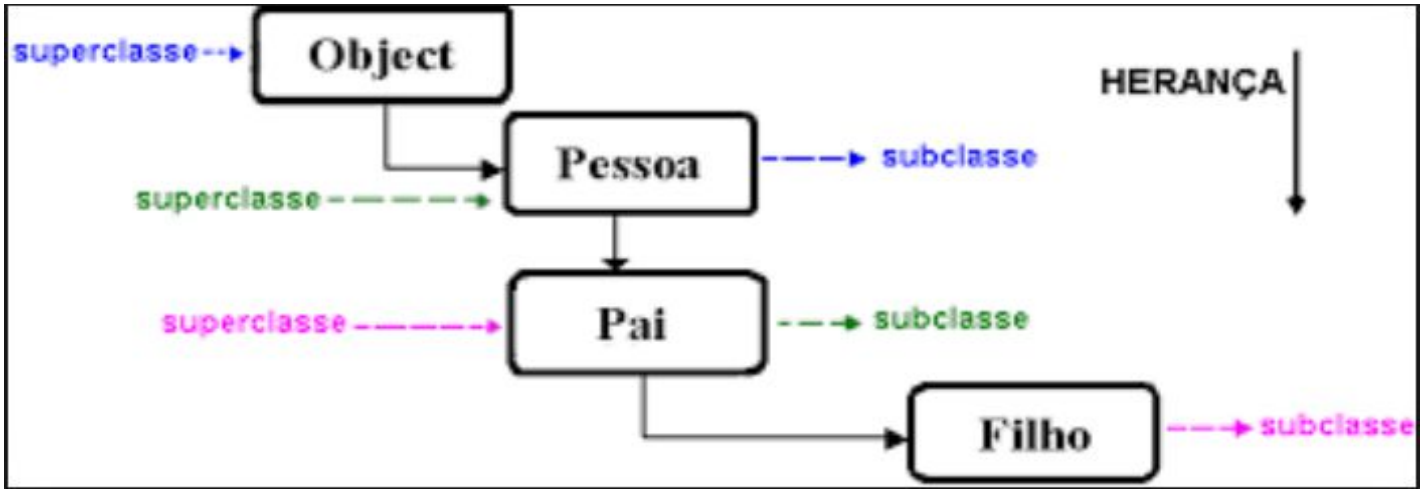
public String getNome() {
return nome;
}
 	
public double getSalario() {
return salario;
}
	public void setNome(String nome) {
	this.nome = nome;
	}
	
	public void setSalario(double salario) {
	this.salario = salario;
	}

Paradigma de Orientação a Objetos

THE DEVELOPER'S CONFERENCE

- 3 - Herança:

- O reuso de código é uma das grandes vantagens da programação orientada a objetos. Muito disso se dá por uma questão que é conhecida como *herança*. Essa característica otimiza a produção da aplicação em tempo e linhas de código.

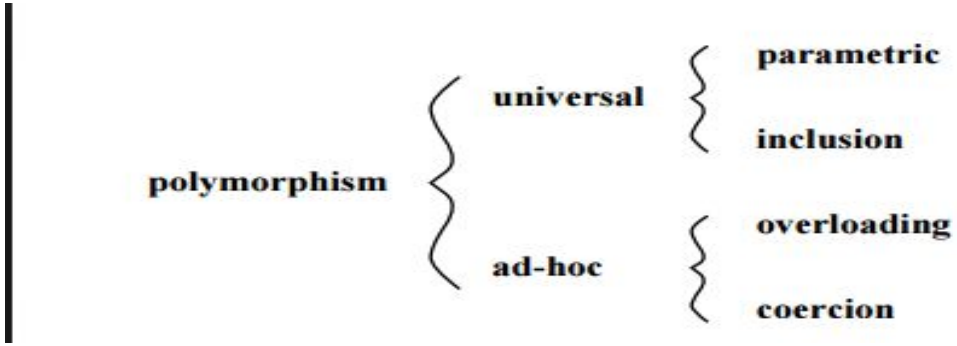


Paradigma de Orientação a Objetos

THE DEVELOPER'S CONFERENCE

- 4 - Polimorfismo:

- Consiste na alteração do funcionamento interno de um método herdado de um objeto pai.
- Princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe. De acordo com a literatura, existem quatro tipos de polimorfismo que uma linguagem de programação pode ter (nem toda linguagem orientada a objeto tem suporte para os quatro tipos de polimorfismo):



Linguagens

- C++, C#, VB.NET, Java, Object Pascal, Objective-C, Python, SuperCollider, Ruby e Smalltalk são exemplos de linguagens de programação orientadas a objetos.
- ActionScript, ColdFusion, Javascript, PHP (a partir da versão 4.0), Perl (a partir da versão 5), Visual Basic (a partir da versão 4), VimL (ou Vim script) são exemplos de linguagens de programação com suporte a orientação a objetos.

- Outro ponto é o fato de que as linguagens de programação implementam o polimorfismo de maneiras diferentes, por exemplo: O C#, por exemplo, faz uso de método virtuais (com a palavra-chave *virtual*) que podem ser reimplementados (com a palavra-chave *override*) nas classes filhas. Já em Java, apenas o atributo "@Override" é necessário.

- Cada linguagem irá implementar esses princípios de uma forma, mas essencialmente é a mesma coisa
- A herança, é outro exemplo, que pode trazer variações mais bruscas, como a presença de herança múltipla. Além disso, o encapsulamento também é feito de maneiras distintas nas diversas linguagens, embora os *getters* e *setters* sejam praticamente onipresentes.

Design Patterns - Manutenabilidade

THE DEVELOPER'S CONFERENCE

Padrão que descreve um conjunto de diretrizes a serem seguidas na implementação de software, envolvendo as atividades de desenho detalhado e codificação. O principal objetivo dessa diretriz é facilitar a manutenção do Código, promovendo uma legibilidade e desestimulando as construções propensas a erros, principais referências [Humphrey95], [Hyde00], [McConnell04], [Ambler00] e [Vermeulen+00].

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Recomendações genéricas em geral aplicáveis a muitas linguagens de programação usadas na prática. Essas diretrizes deve ser personalizadas para cada linguagem específica, levando em consideração os recursos que são suportados por cada uma, ainda que esteja sendo utilizada uma linguagem orientada a objetos como padrão para implementação.

Tema	Regra
Dados	Nunca Redefinir um tipo predefinido do ambiente
	Usar Nome de tipos Funcionalmente Orientados
	Tipos Simples – Evitar Números Mágicos
	Tipos Complexos – Usar arranjos apenas para acesso sequencial Evitar os apontadores onde for possível, Isolar os apontadores,

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Variáveis	Declarar sempre todas as variáveis
	As declarações devem ser acompanhadas de comentários descritivos
	Iniciar todas as variáveis em sua declaração
	Minimizar o Escopo das Variáveis
	Usar cada Variável com uma única finalidade
	Minimizar o Uso de Variáveis Globais

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Nomes	Todos os nomes devem descrever de forma completa e precisa a entidade que representam
	Declaração – todos os identificadores devem representar palavras ou frases do idioma utilizado
	Nomes Compostos – Separar as palavras constituintes do nomes longos
	Abreviações devem ser evitadas se a linguagem permitir
	Pronúncia – Os nomes devem ser pronunciáveis
	Diferenciação – Os Nomes devem ser diferenciáveis pelos seus primeiros caracteres

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Nomes	Notação Húngara – Evitar o uso de notação Húngara e de outras notações que denotem informação de baixo nível
	Confusão de Nomes – Evitar a ocorrência de Identificadores com nomes que possam ser confundidos
	Nomes Predefinidos – Evitar o uso de nomes predefinidos no ambiente de desenvolvimento, a não ser que estejam sendo redefinidos: Constantes, Tipos Enumerados, Variáveis Temporárias, Booleanas, Variáveis de Status e Índices de Malha

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Estruturas de Controle	**Sequência Obrigatória – Quando existe uma sequência correta de execução das instruções, ela deve ser facilmente reconhecível pela leitura do Código**
	Sequência de Leitura – O código deve ser organizado de forma que possa ser lido de cima para baixo: minimizar o alcance da variáveis
	Agrupamento – Instruções correlatas devem ser agrupadas, e os grupos deve ser separados por linhas em branco e linhas de comentário

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Estruturas de Seleção	**IF Simples – escrever o primeiro caso normal, e depois as exceções**
	Cadeias de IF – Não utilizar o *Else* final de uma cadeia de IF para tratar um caso restante; reserva-lo para tratar condições de exceção.
	Seleções Múltiplas - Ao utiliza seleções múltiplas (case/swithc) classificar os casos por ordem alfabética/numérica ou por frequência de ocorrência, mantendo entretanto juntos os casos cujo tratamento usa o mesmo código
Estruturas de Iteração	**Tipos de Malhas – Usar sempre o tipo de malha mais adequado a lógica da iteração: No Início – While No Final – Do{ <instruções>} While {<condição>} No Meio – (While (true) { <instruções>; IF {<condição>} break; <instruções>}**

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Estruturas de Iteração	**Malhas– Malhas tipo *For* devem ser usadas quando se sabe que a malha deverá ser executada um número específico de vezes**
	Malhas Eternas – Usar um formato padronizado e facilmente reconhecível para malhas eternas.
	Pontos de Saída – A condição de saída da malha deve ser facilmente identificável
	Deve- se evitar malhas vazias -> Quando um único objeto da malha é produzido por um efeito colateral do teste de terminação, mover o efeitos colateral para dentro da malha, ou pelo menos colocar um comentário dentro do corpo da mesma.
	Funções das Malhas – Cada malha deve executar apenas uma função, máximo 3 níveis de aninhamento, curta o bastante para caber uma tela

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Estruturas Especiais de Controle	**Instruções *GOTO* – Se a linguagem permitir as instruções *goto*, elas só devem ser usadas em casos extremos, em que o código ficaria mais ilegível sem o *goto* do que com ele**
	Instruções *Return* – Só usar o *Return* para melhorar a legibilidade
	Recursão – Só usar a recursão quando puder provar que ela tem limite
	Aspectos Gerais – Expressões de Testes – Simplificar as expressões de teste usadas para controle – Usar tabelas de decisão, usar parênteses, usar funções booleanas – comparações false devem ser implícitas (!pronto)
	Testes Numéricos – Organizar os testes numéricos na direção dos pontos de uma reta
	Aninhamento de Testes – Minimizar os níveis de aninhamento

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Layout (Leiaute)	**O Layout deve refletir a estrutura lógica do programa, o espaço em branco é a principal ferramenta de layout**
	Espaçamento em Expressões – Operadores Unitários – Evitar espaços entre operadores unitários e seus operandos: -a , !x , i++
	Operadores binários – Deve haver pelo menos um espaço de cada lado de um operador binário
	Operadores de seleção devem ser adjacentes a seus operandos: vetor [i], objeto.mesa
	Separadores de itens devem seguir imediatamente o objeto anterior, e devem ser seguidos por pelo menos um espaço
	Símbolos parentéticos (( ), [ ]) não devem ter espaços depois do símbolo de abertura e antes do símbolo de fechamento

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Layout (Leiaute)	**Endentação – Tamanho – A endentação das estruturas de controle deve ocupar de 2 a 4 espaços**
	Tabulações - Se a endentação for feita com tabulações, indicar o fato em um comentário colocado no início do programa, que indica o número de espaço por tabulações
	A endentação das chaves de estruturas de controle deve ser consistente
	Alinhamento de declarações – Declarações de Dados deve ser colocados uma por linha, com alinhamento do tipo
	Linhas em Branco – Espaços entre instruções – Instruções correlatas devem ser agrupadas, sem linhas em branco ou endentação desnecessária
	Espaço para Comentários – Pelo menos uma linha em branco deve separar um comentário do código precedente

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Layout (Leiaute)	**Tamanho das Linhas – Tamanho Máximo – linhas de código não devem exceder mais de 80 caracteres**
	Quebra de Linhas – Quando a instrução não cabe em uma linha de 80 caracteres, parti-la em duas ou mais linhas nos pontos de menor impacto sobre a legibilidade
	Lista de Parâmetros – Se uma lista de parâmetros for longa demais, de tal modo que a respectiva definição ou chamada de função não caiba em uma linha, cada linha de continuação de ser alinhada e começada por um parâmetro

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Comentários	Qualidade dos Comentários: evite abreviações, manter próximo ao código que comentam, evitar comentários de pouca importância, comentar enquanto codifica, comentar blocos de instruções
	Casos Especiais – Unidades de Programa - os prólogos de unidades devem descrever as entradas, saídas e hipóteses sobre essa unidade, e não o seu conteúdo
	Seções de Programa – Seções mais importantes do programa devem ser precedidas de um comentário do bloco
	Declarações de Dados – Em declaração de dados, usar os comentários para descrever aspectos da variável que não são descritos por seu nome

Design Patterns – Diretrizes Genéricas

THE DEVELOPER'S CONFERENCE

Tema	Regra
Comentários	Comentários de Linha – Usar com Cuidado os comentários de linha
	Manutenção – Todos os comentários devem estar sempre atualizados
	Código inacabado : código não-funcional; código parcialmente funcional; código suspeito ; código que precisa de aperfeiçoamento
	Referências a outros documentos – Toda referência a outros documentos, embutida em comentários, deve ser marcada de forma padronizada

Design Patterns

THE DEVELOPER'S CONFERENCE

THE 23 GANG OF FOUR DESIGN PATTERNS

C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

Design Patterns: PHP

THE DEVELOPER'S CONFERENCE

Cada padrão descreve um problema que ocorre repetidas vezes e, em seguida, descreve o núcleo da solução para esse problema, de tal forma que você pode usar essa solução um milhão de vezes, sem fazê-lo da mesma maneira duas vezes.

Propósito	Design Pattern	Aspectos
Criacional	**Abstract Factory**	Famílias de Objetos de produto
	Builder	Como um Objeto composto é criado
	Factory Method	Subclasse do Objeto que é instanciado
	Prototype	Classe do Objeto que é instanciado
	Singleton	O único exemplo de uma Classe

Design Patterns

THE DEVELOPER'S CONFERENCE

Propósito	Design Pattern	Aspectos
Estrutural	**Composite**	Estrutura and Composição de um Objeto
	Decorator	Responsabilidade de um Objeto sem subclasse
	Facade	Interface para um subsistema
	Flyweight	Custo de Armazenamento dos Objetos
	Proxy	Como um Objeto é acessado, sua localização
	Adapter	Interface para um Objeto
	Bridge	Implementação de um Objeto

Design Patterns

THE DEVELOPER'S CONFERENCE

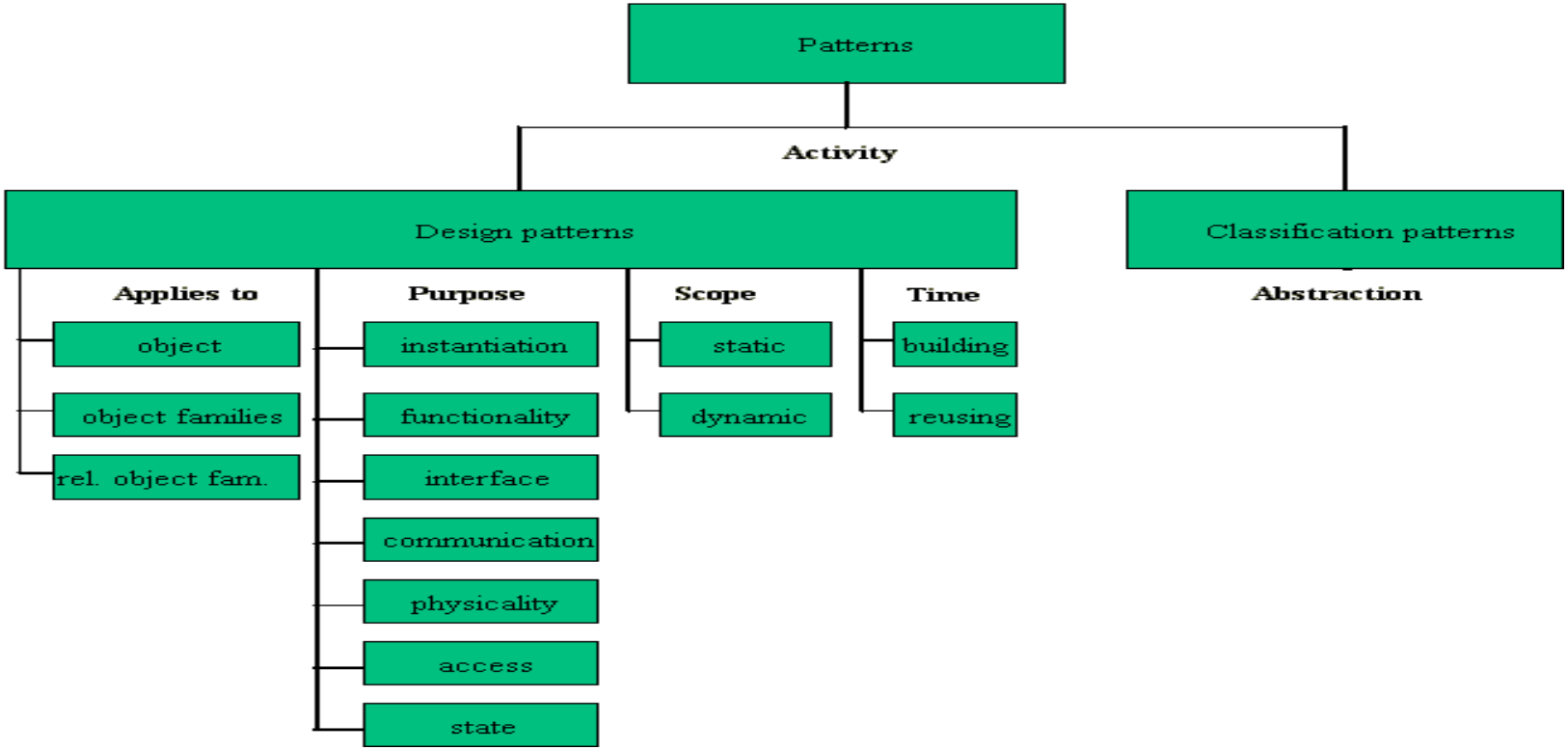
Propósito	Design Pattern	Aspectos
Comportamental	**Chain of Responsibility**	Objeto que pode attender uma solicitação
	Command	Quando e como uma solicitação é atendida
	Interpreter	Gramática e Interpretação de uma língua
	Iterator	Como os elementos de agregados são acessados, percorridos
	Mediator	Como e quais Objetos interagem uns com os outros
	Memento	Qual informação privada é armazenada fora de um objeto, e quando
	Observer	Número de objetos que dependem de outro objeto; Como os objetos dependentes permanecem atualizados

Design Patterns

THE DEVELOPER'S CONFERENCE

Propósito	Design Pattern	Aspectos
Comportamental	**State**	Estado de um objeto
	Strategy	Um Algoritmo
	Template Method	Steps de um Algoritmo
	Visitor	operações que podem ser aplicadas ao (s) objeto (s) sem alterar sua (s) classe (s)

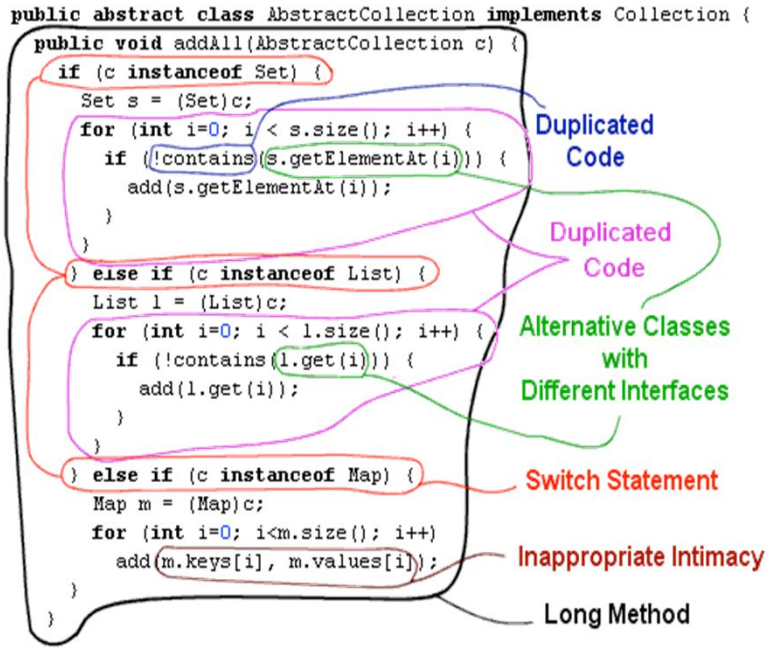
Design Patterns



Code Smell

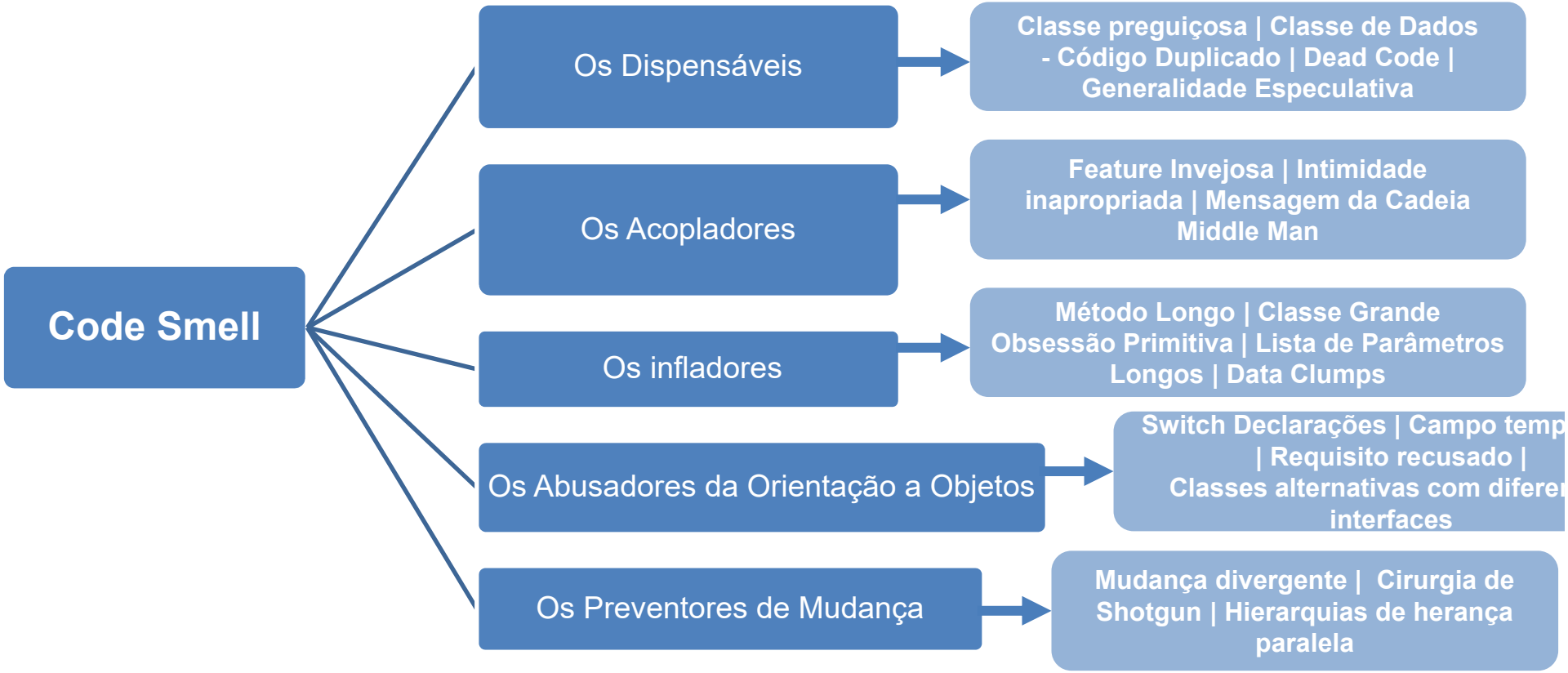
é qualquer característica no código-fonte de um programa que possivelmente indica um problema mais profundo. *Determinar o que é e não é um cheiro de código é subjetivo e varia de acordo com a linguagem, o desenvolvedor e a metodologia de desenvolvimento.*

O termo foi popularizado por Kent Beck na WardsWiki no final da década de 1990. O uso do termo aumentou depois que ele foi apresentado no livro Refatoração: Melhorando o Design do Código Existente, de Martin Fowler.



Code Smell

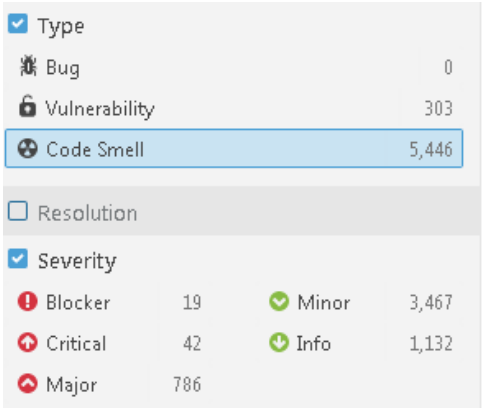
THE DEVELOPER'S CONFERENCE

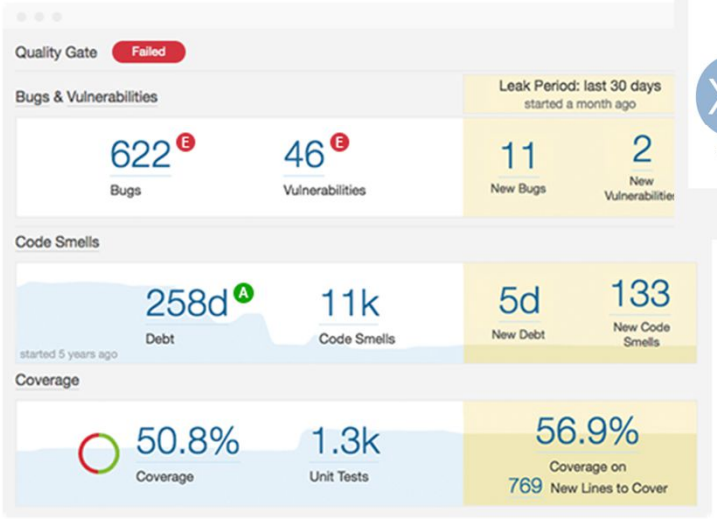


Code Smell: Plug-ins e Ferramentas

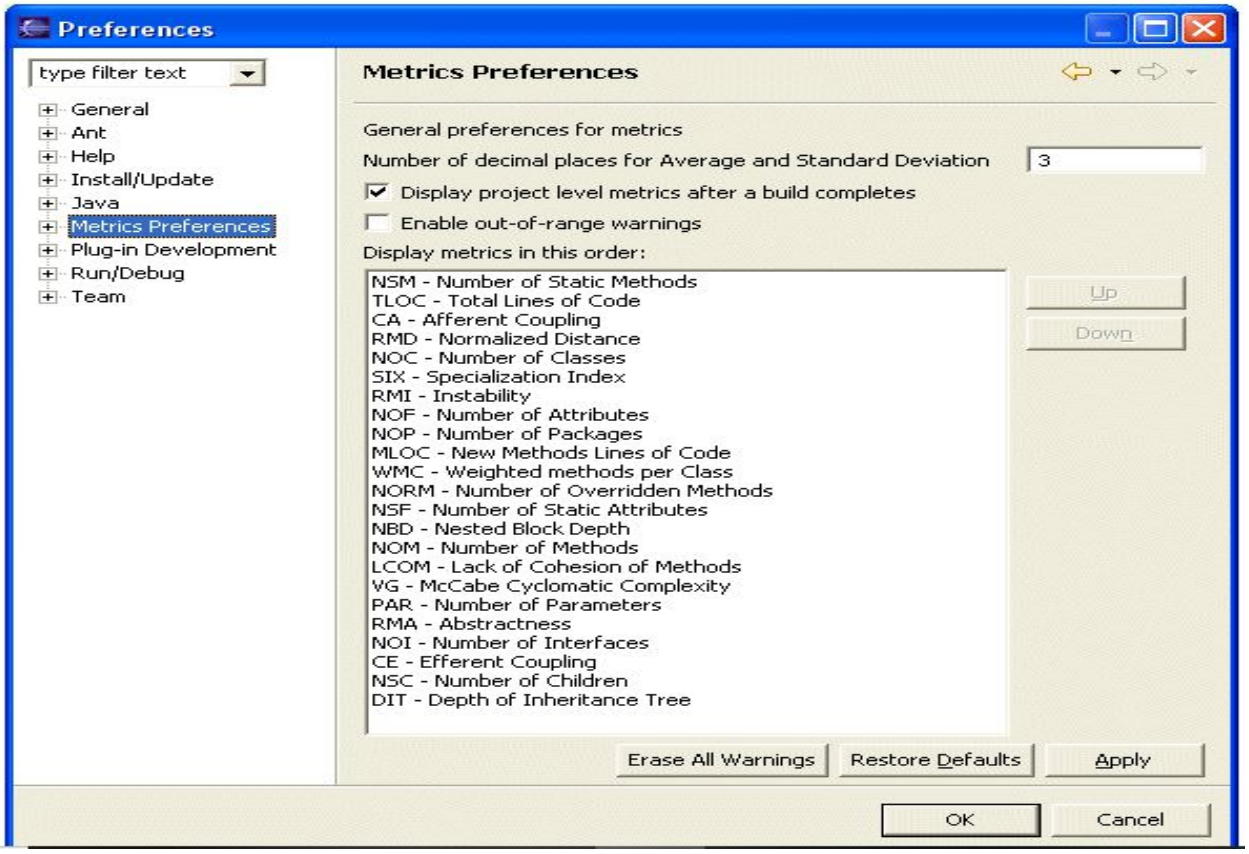
THE DEVELOPER'S CONFERENCE

- SonaQube | Cloud | Lint | Code Analyzer
- biteGarden
- VeraCode
- Fortify
- Coverage
- Eclemma





Code Smell: Plug-ins e Ferramentas



THE DEVELOPER'S CONFERENCE

Metrics

Code Smell: Segurança

THE DEVELOPER'S CONFERENCE

Checklist Desenvolvimento Seguro de Software

1. Validação de entrada - 16
2. Codificação de saída - 6
3. Autenticação e Gerenciamento de Senhas - 34
4. Gerenciamento de Sessão - 18
5. Controle de Acesso - 23
6. Práticas Criptográficas - 6
7. Tratamento de erros e registro - 35
8. Segurança de Comunicação - 8
9. Configuração do Sistema - 15
10. Segurança do banco de dados - 13
11. Gerenciamento de arquivos - 14
12. Gerenciamento de Memória - 9
13. Práticas Gerais de Codificação -12

Total 214

Métricas de Qualidade de Código

THE DEVELOPER'S CONFERENCE

Descrição	Valores de Referência
Taxa (%) Comentários no Código	Entre 10% e 30%
Taxa (%) Cobertura de Testes Unitários	A partir de 40%
Package Tangle Index	+ Próximo de 0
Lcom4 (Falta de Coesão)	+ Próximo de 1
Complexidade de Classes (Ciclomática)	1 a 7 (quanto menor melhor)
Linhas Duplicadas	+ Próximo de 0
Linhas Úteis por Classe	1500

$$LOCM = 1 - \frac{\sum_{f \in F} |M_f|}{|M| \times |F|}$$

M = static and instance methods in
F = instance fields in the class,
M_f = methods accessing field f, and
$|S|$ = cardinality of set S.

Qualidade de Código ≠ Qualidade de Software

Papéis: DEV

THE DEVELOPER'S CONFERENCE

DEV

PREOCUPA-SE

Codificação, Desenvolvimento Construção de Software, Documentação da aplicação, revisão em pares, TDD

ENVOLVE-SE

Testes Unitários e de Integração, cumprir os patterns de desenvolvimento especificados

QA

PREOCUPA-SE

Gestão de Testes, Modelagem de Testes - BDD, Documentação dos cenários, Scripts de automação, Estratégia de Testes Manuais, Exploratórios, Funcionais e Não Funcionais.

ENVOLVE-SE

Teste de Sistema, Aceitação, Regressão, Smoke, coleta de métricas de qualidade de software e de código

Referências:

- FILHO, Wilson de Pádua – "Engenharia de Software: Fundamentos Métodos e Padrões – LTC -3º Edição , 2009, Rio de Janeiro https://pt.wikipedia.org/wiki/Arquitetura_de_dados
- http://www.codingthearchitecture.com/presentations/sa2008-why-software-projects-fail/
- https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264
- https://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objetos
- https://respostas.guj.com.br/38321-poo-orientacao-a-objetos
- http://www.lcad.icmc.usp.br/~jbatista/sce537/mat/Inicio.pdf
- https://www.script-tutorials.com/design-patterns-in-php/
- https://people.cs.umu.se/jubo/ExJobbs/MK/patterns.htm
- https://www.xmind.net/m/6uY7/
- https://www.xmind.net/embed/bFMM/
- https://en.wikipedia.org/wiki/Code_smell
- https://slideplayer.com/slide/13032012/
- https://pt.wikipedia.org/wiki/Complexidade_ciclom%C3%A1tica
- http://metrics.sourceforge.net/

Femme it
O JEITO FEMININO DE FALAR DE TI

THE DEVELOPER'S CONFERENCE

Adolceegabbana

in/alessandramonteiromartins/

monteiromartins@bol.com.br

@Ale_TI

THE DEVELOPER'S CONFERENCE