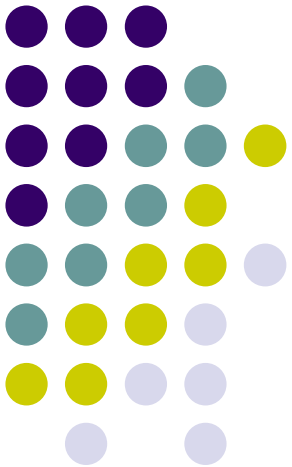
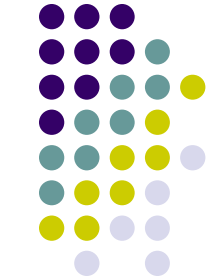
Maus Cheiros

Prof. Alberto Costa Neto
DComp/UFS
alberto@ufs.br



Cheiros do Código

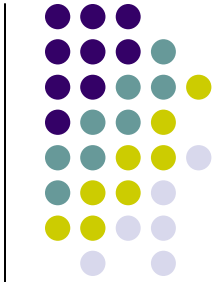


- If it stinks, change it…….Grandma Beck

- Existem certas coisas que produzem maus cheiros e existem certas refatorações que desodorizam o código

- Dicas para encontrar problemas no código e diretrizes para eliminar

Código Duplicado

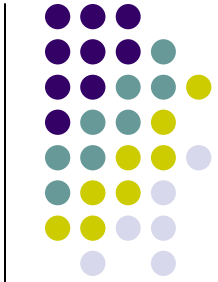


- “Faça tudo exatamente uma vez”
- Código duplicado torna o sistema difícil de entender
- Código duplicado é difícil de manter
  - Toda mudança deve ser duplicada
  - Quem faz a manutenção deve saber disto

Consertando Código Duplicado

- Coloque métodos idênticos numa superclasse comum
- Coloque o método mais geral encima
- Coloque o método num componente comum (ex. Strategy)
- (Ver também: Métodos Longos)

Métodos Longos

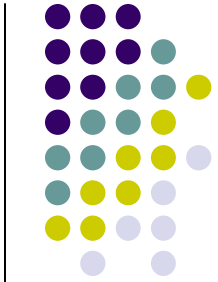


- Nenhuma métrica será sempre correta (LOC, ...) para determinar se um método é longo
  - Use o bom senso
- O método é a menor unidade de *overriding*
- Comandos dentro de um método devem estar no mesmo nível de abstração

Consertando Métodos Longos

- Extraia peças como métodos pequenos!
  - Se um método inteiro é longo e de baixo nível, encontre a seqüência de passos de mais alto nível
  - Comentários no meio de um método freqüentemente apontam bons lugares para extrair
- Peças menores podem ser reusadas freqüentemente

Classes Longas

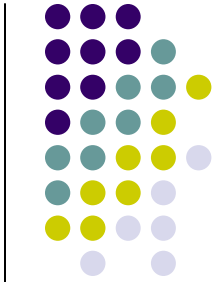


- Novamente, nenhuma métrica é suficiente
- Muitos métodos
- Muitas variáveis de instância
- Procure por conjuntos não relacionados de métodos e variáveis de instância

Consertando Classes Longas

- Crie composições de classes menores
- Encontre sub-componentes lógicas da classe original e crie classes que os representem
- Mova métodos e variáveis de instância para as novas componentes

Variáveis de Instância usadas às vezes

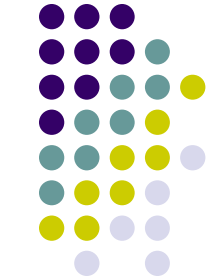


- Se alguma instância a usa e outras não, crie subclasses
- Se é usada somente durante uma certa operação, considere um objeto operador

Parâmetros co-ocorrentes

- Frequentemente disfarçam uma abstração latente
  - (ex., Point)
- Com freqüência, uma vez que os objetos existem, comportamento pode ser adicionado naturalmente

Consertando Parâmetros co-ocorrentes



- Crie um objeto para agrupar todos os parâmetros que co-ocorrem
- Passe este objeto como parâmetro
- Procure métodos que devem estar no novo objeto

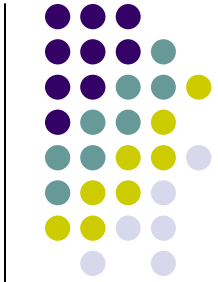
Inveja de Dados

- Sintoma de métodos no lugar errado
- Um método sempre está acessando valores de outra classe
- Parece estar mais interessado em outra classe mais do que na classe onde habita

Consertando Inveja de Dados

- Use *Move Method* para mover o método na classe que ele está usualmente trabalhando
- Se está interessado em múltiplas classes, coloque o método nas classes onde ele acessa mais valores, ou utilize *Extract Method* em conjunção com *Move Method*

Condicionais Aninhados



- Switchs espalhados em vários métodos/classes
- Comportamentos diferentes de acordo com o tipo (em vários métodos)
- Sintoma de métodos no lugar errado

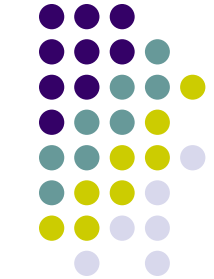
Consertando Condicionais Aninhados

- Ao invés de Switchs, permita que *dynamic dispatch* faça o “switch” (explore polimorfismo)
- Se o código condicional envolve testes por tipos, ponha o método nesta classe
  - instanceOf, switch por tipo enumerado, …
- Se o código condicional envolve isEmpty, isNull, etc., considere o padrão *Null Object*
- Novos casos não requererão mudanças no código existente (A Meta Final)

Intimidade Inadequada

- Classes tornam-se íntimas demais e gastam muito tempo sondando as partes privadas das outras
- Classes fortemente acopladas … você não pode mudar uma sem mudar a outra
- Herança em excesso pode levar a intimidade demais

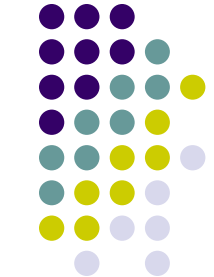
Consertando Intimidade Inadequada



- Use *Move Method* e *Move Field* para separar as partes para reduzir intimidade
- Se classes têm interesses comuns, use *Extract Class* para colocar a parte comum num lugar seguro, mais reusável
- Se a intimidade se dá por herança, substitua herança por delegação

O resultado final é acoplamento mais baixo

Alteração Divergente



- Uma classe é frequentemente alterada de diferentes maneiras por diferentes razões
  - Implementa 2 ou + conceitos simultaneamente (persistência e negócio)
- Provavelmente é melhor dois objetos do que um

- Use *Extract Class*

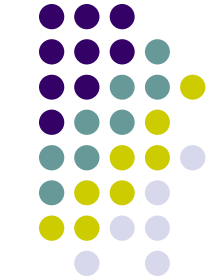
Devemos estruturar o software para tornar as alterações mais fáceis (software deve ser soft)

Cirurgia com Rifle

- Cada vez que executa uma mudança, tem que fazer pequenas alterações em muitas classes

- Use *Move Method* e *Move Field* para colocar todas as alterações numa única classe

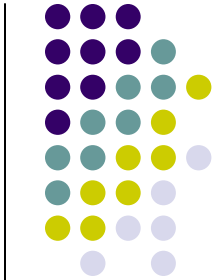
Hierarquias Paralelas



- Cada vez que você faz uma subclasse de uma classe, você tem que fazer uma subclasse de outra classe em outra hierarquia
- Se usou técnicas de boa nomeação, você pode reconhecer isto porque o prefixo (ou sufixo) dos nomes das classes serão os mesmos em ambas hierarquias

- *Move Method* e *Move Field* podem ajudar a eliminar uma hierarquia

Comentários



- Comentários não são ruins, porém …

- Se encontrar métodos longos com trechos onde o código é comentado, use *Extract Method*