Padrões, Frameworks e Arquitecturas

Ademar Aguiar
www.fe.up.pt/~aaguiar
ademar.aguiar@fe.up.pt

Universidade do Porto
Faculdade de Engenharia
FEUP

Objectivos

- Dar uma visão geral sobre as principais abordagens para a reutilização de software, ao nível de desenho.
- Apresentar as noções principais sobre padrões de software e as suas origens
- Definir o conceito de frameworks aplicacionais e a sua relação com os padrões de software e componentes de software

Introdução

- A evolução dos sistemas computacionais não pára ...
  - A capacidade dos sistemas computacionais tem evoluído muito rapidamente (processamento/$ = 2x/ano; transmissão/$ = 3x/ano).
  - A conectividade entre sistemas é um factor crucial de sucesso.
  - O software é um elemento estratégico para o sucesso de muitas organizações.
  - O software é indispensável e está cada vez mais enraizado no nosso quotidiano.

Introdução ...

- Os desafios postos à engenharia de software (ES) são grandes ...
  - Os sistemas de software são cada vez mais sofisticados e complexos.
  - Os níveis de qualidade exigidos aos sistemas desenvolvidos são elevados, sobretudo devido ao enorme impacto que têm no nosso dia-a-dia.
  - Existe uma procura insaciável de software.
- A produtividade de desenvolvimento tem de melhorar
  - Há que saber responder RÁPIDO e com QUALIDADE!
  - A solução passa pela REUTILIZAÇÃO!

Reutilização de Software

- A importância da reutilização tem aumentado bastante
  - Surgiu como uma grande promessa da tecnologia de objectos.
  - Permite aumentos efectivos de produtividade de desenvolvimento.
  - A reutilização nem sempre se consegue aos níveis desejados.
- A experiência existente em termos de ES é vasta
  - É grande a experiência acumulada na resolução de problemas típicos de desenvolvimento
  - Esta experiência estende-se desde as técnicas de programação até às metodologias de análise, teste e controlo de qualidade.

Algumas formas de reutilização ...

- Código-fonte
  - Reutilização de código em bibliotecas de classes, bibliotecas de procedimentos, ou no limite, copy-paste-modify de código existente.
- Componentes de Software
  - Unidades físicas de software que podem ser distribuídas de forma independente e que implementam a funcionalidade especificada numa interface pública bem definida.
  - Exemplos: CORBA (OMG), JavaBeans (Sun) e ActiveX (Microsoft).
- Modelos e Documentos
  - Reutilização de modelos e documentos previamente definidos para serem utilizados em diversas actividades de desenvolvimento: convenções de programação, documentos de análise e projecto.

Algumas formas de reutilização ...

- Frameworks
  - Colecções de classes que conjuntamente constituem aplicações semi-prontas (? 80%) específicas a um domínio de problema.
  - Exemplos: JFC, SanFrancisco, (MFC), MacApp, ET++, Qt/Kde...
- Padrões
  - Soluções genéricas para problemas recorrentes num contexto.
  - Permitem reutilizar não código, mas as ideias e raciocínios que estão na base da escrita de código.
  - Constituem uma forma de reutilização a muito alto-nível, capaz de veicular o conhecimento de especialistas a não-especialistas.
- Componentes de domínio
  - Componentes de grande dimensão que englobam a funcionalidade específica de um determinado domínio.

Padrões, Frameworks e Arquitecturas

- Padrões de Desenho
  - São **micro-arquitecturas** que descrevem mecanismos abstractos de interacção entre diferentes objectos cooperantes por forma a resolver um problema recorrente.
- Frameworks
  - Definem uma **macro-arquitectura** que interliga diversos mecanismos de interacção entre objectos participantes e a infraestrutura que suporta a sua integração.
  - São conjuntos de classes cooperantes, concretas e abstractas, que foram desenhadas para implementar funcionalidade específica a um determinado conceito.
  - As frameworks possuem partes inalteráveis (congelada) e partes que podem ser configuradas pelo utilizador, através de mecanismos de herança (*white-box*), composição (*black-box*), ou ambas (*gray-box*).

Padrões

Estruturas Repetitivas

- Os sistemas complexos contêm estruturas repetitivas
  - Nos sistemas de elevada complexidade, bem sucedidos, podem-se identificar estruturas que se repetem em variadas combinações.
- A identificação destas estruturas auxilia a sua compreensão
  - Criam um nível de abstracção intermédio entre o nível mais baixo dos elementos que o constituem e o nível global do sistema.
  - Da análise de diversos destes sistemas, podem-se identificar semelhanças entre muitas destas estruturas repetitivas.
- Como se descrevem estas estruturas repetitivas ?
  - Os "Manuais de Engenharia" são uma forma utilizada para documentar, comunicar e veicular o conhecimento prático existente sobre este tipo de estruturas nos ramos de engenharia tradicionais.

As Dificuldades em Reutilizar

- 1º Necessita de Esforço Extra
  - É necessário tempo e esforço adicional para criar software reutilizável. Não pode ser visto como um fim, mas um meio ...
- 2º Exige Boas Abstracções
  - Exige uma boa abstracção sobre os detalhes não-essenciais.
  - Obriga à identificação das melhores abstracções, o que não é fácil.
- 3º Requer Qualidades Difíceis de Alcançar
  - Flexibilidade, modularidade e "elegância" são qualidades que facilitam a reutilização mas nem sempre se atingem facilmente.
- 4º Necessita de Documentação de Qualidade
  - O potencial de reutilização de um pedaço de software só consegue ser plenamente rentabilizado se fôr fácil comunicar o seu valor aos seus potenciais reutilizadores e orientá-los na sua utilização.

As Origens dos Padrões

Patterns: origens na Arquitectura

- Cristopher Alexander
  - A noção tem as suas origens no trabalho de Cristopher Alexander.
  - Durante 10 anos, Alexander recolheu e documentou soluções genéricas para problemas recorrentes no domínio da arquitectura.
  - O objectivo inicial foi o de habilitar não-especialistas a projectar as suas próprias casas e comunidades.
- O livro "A Pattern Language" [Alexander77]
  - Apresenta os padrões de Alexander, i.e., descrições textuais de soluções para problemas recorrentes na criação de cidades e vilas, urbanizações e edifícios.

> *"Each pattern describes a problem that occur over and over again in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice" [Alexander77, p. x]*

Patterns: origens na Arquitectura

- "the quality without a name" (TQWAN)
  - O objectivo de Alexander neste trabalho de pesquisa de padrões, consistiu em identificar o que distingue uma construção com 'qualidade' de uma outra.
  - Segundo Alexander, as agradáveis sensações que uma construção nos transmite de liberdade, conforto, harmonia e vida são tudo reflexos da presença ou não desta multifacetada 'qualidade'.
  - A linguagem de padrões que documentou em livro auxilia na detecção dos elementos que contribuem para o aparecimento ou ausência desta 'qualidade', referida por TQWAN.
- Definição
  - Os padrões são uma *forma textual* de descrição de uma *solução genérica* para um *problema recorrente* num determinado *contexto*.

Padrões de Alexander

- 253 padrões: apresentados do global para o particular
  - Em "A Pattern Language" são descritos 253 padrões, interrelacionados, que variam no nível de detalhe, sendo a sua apresentação iniciada pelos padrões de nível mais global e seguindo depois para os de nível mais particular.
  - Alguns exemplos destes 253 padrões:
    - 1. Independent Regions
    - 2. The Distribution of Towns
    - 16. Web of Public Transportation
    - 83. Master and Apprentices
    - 134. Zen View
    - 251. Different Chairs
    - 253. Things from your life

Forma dos Padrões de Alexander

Nome	• Uma frase ou nome descritivo identificativo do padrão.
Exemplo	• Fotografias, desenhos e descrições que ilustram a aplicação dos padrões.
Contexto	• As circunstâncias nas quais o padrão pode ser aplicado. Explica a razão da existência do padrão e a sua generalidade.
Problema	• Descreve as forças relevantes e restrições e como interactuam entre si.
Solução	• Relações e regras dinâmicas que descrevem como construir artefactos em concordância com o padrão. • São referidos padrões similares e variantes, bem como padrões de níveis superior e inferior relevantes para a concepção da solução proposta.

"83. Master and Apprentices"

- Problema

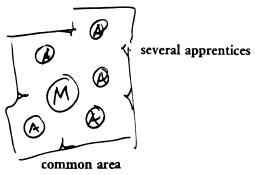
  "The fundamental learning situation is one in which a person learns by helping someone who really knows what he is doing."

  [Alexander77, p. 413]

- Solução

  "Arrange the work in every workgroup, industry, and office, in such a way that work and learning go forward hand in hand. Treat every peace of work as an opportunity for learning. To this end, organize work around a tradition of masters and apprentices: and support this form of social organization with a division of the workspace into spacial clusters - one for each master and his apprentices - where they can work and meet together."

  [Alexander77, p. 1159]



"251. Different Chairs"

- Problema

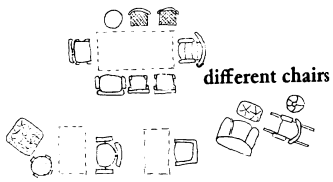
  "People are different sizes: they sit in different ways. And yet there is a tendency in modern times to make all chairs alike."

  [Alexander77, p. 1158]

- Solução

  "Never furnish any place with chairs that are identically the same. Choose a variety of different chairs, some big, some small, some softer than others, some rockers, some very old, some new, with arms, without arms, some wicker, some wood, some cloth."

  [Alexander77, p. 1159]



Padrões de Software

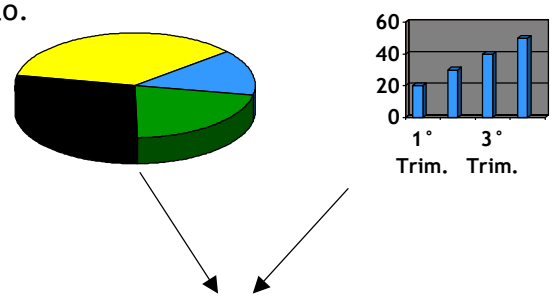
Padrões de Software

- Existem problemas e soluções recorrentes também em Engenharia de Software e a vários níveis:
  - Arquitectura
  - Análise
  - Desenho
  - Construção
- "GoF book": 1º livro sobre Padrões de Software
  - Foi apresentado na OOPSLA'94, uma co-autoria de um grupo de quatro indivíduos conhecido pelo *Gang of Four* (GoF).
  - Documenta 23 padrões que descrevem soluções reconhecidamente boas para problemas recorrentes de desenho OO [Gamma95].

Exemplo: "*Observer*"

- Pretende-se visualizar um conjunto de dados (modelo) através de duas formas distintas (vistas) que estejam sempre coerentes com o modelo.

Vistas (*observers*)



Modelo (*subject*)

Observer

- Nome: *Observer* **[Buschmann96]**
- Contexto
  - Um objecto utiliza informação fornecida por outro objecto.
- Problema
  - A alteração de informação interna a um objecto pode introduzir inconsistências em eventuais objectos que com ele cooperam.
  - Para manter a consistência, é necessário estabelecer um mecanismo de troca de informação entre eles.
- Forças
  - O objecto fornecedor de informação não pode depender dos detalhes internos dos objectos que com ele cooperam.
  - Os objectos observadores que dependem da informação do objecto fornecedor podem não ser todos conhecidos *a priori*.
  - Os objectos observadores podem reagir de forma diferente a uma mesma alteração de informação do objecto fornecedor.

Observer ...

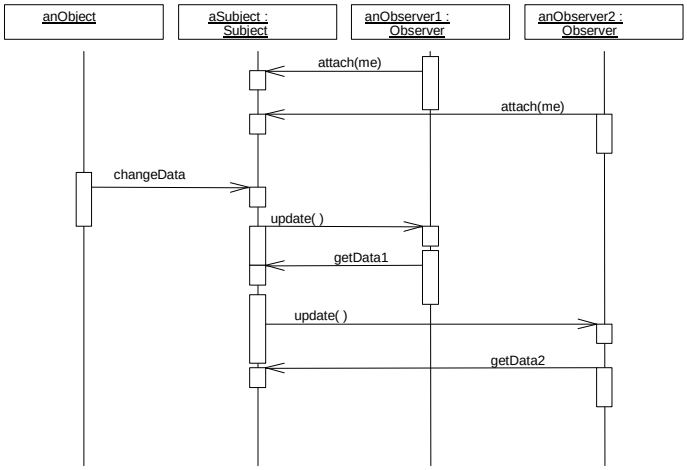
- Solução:
  - Implementar um mecanismo de propagação de alterações entre o objecto fornecedor de informação - ***subject*** - e os objectos que dela dependem - os ***observers***.
  - Os *observers* podem ser registados dinamicamente com este mecanismo.
  - Sempre que o *subject* altera a sua informação, inicia o mecanismo de propagação de alterações para repor a consistência com todos os *observers* registados.
  - As alterações podem ser propagadas invocando uma função comum a todos os *observers*.

Observer: estrutura e exemplo


```
class Modelo {
   // colecção de vistas registadas
   private Vector vistas;
   // registar novos observers
   public attach (Vista vista) { vistas.add(vista); }
   // anular registo de observers existentes
   public detach(Vista vista) { vistas.remove(vista); }

   public propagateChange() {
      Iterator iterator = vistas.iterator();
      while(iterator.hasNext())
         iterator.next().update();
   }
}
```

```
interface Vista {
   public void update();
}

public class VistaTipoPie implement Vista {
   ...
   public void update() { … }
}

public class VistaTipoBarras implement Vista {
   ...
   public void update() { … }
}
```

Observer: colaborações



Padrões de Software ...

- Uma forma de documentação ...
  - Os padrões podem muito simplesmente ser vistos como uma forma de documentação.
- Uma partilha de experiência ...
  - Os autores documentam, sob a forma de padrões (problema, contexto, solução), as boas soluções que observaram terem sucedido em vários projectos.
- Uma garantia de valor e utilidade ...
  - Os especialistas ao ler estes padrões, reconhecem-nos e validam o seu valor e utilidade, tipicamente em conferências "*PLoP".
- Uma forma mais rápida de aprender na prática ...
  - Estes conhecimentos e experiências podem depois de publicados serem utilizados por principiantes (mas com um sentido critico).

Padrões de Software ...

- Uma Definição:

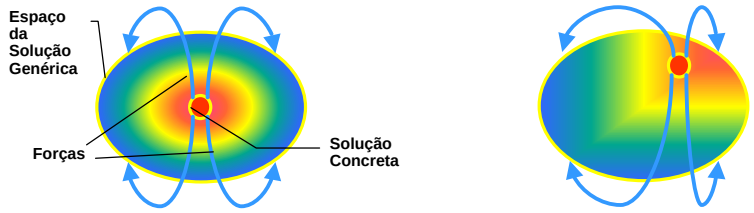
  "Um padrão de software descreve um problema recorrente num contexto específico e uma solução genérica comprovadamente boa para o resolver.

  A solução descreve os componentes que a constituem, as suas responsabilidades, as associações e os mecanismos de colaboração entre eles" [Buschmann96]

- Alguns exemplos bem populares:
  - *Observer* (Model-View-Controller)
  - *Iterator*: para abstrair a navegação em estruturas de dados ...
  - *Broker*: para estruturar sistemas distribuídos ...
  - *Proxy*: para controlar acessos, melhorar eficiência ...

Projecto com Padrões

- Os padrões fornecem soluções de projecto em abstracto
  - Os padrões de software têm de ser adaptados e instanciados ao contexto específico em que vão ser aplicados.
  - Neste processo é comum o recurso a outros padrões relacionados.
- Facilitam a avaliação dos compromissos envolvidos
  - Auxiliam a tomar decisões de projecto e de implementação.



Forma dos Padrões de Software

- As formas utilizadas variam de autor para autor, mas normalmente possuem as seguintes partes:
  - Nome: sugestivo do que o padrão representa
  - Descrição breve: (*pattlet*) útil para pesquisa e indexação
  - Contexto: enquadramento das situações onde o problema ocorre
  - Problema: descrição e enumeração das forças em conflito
  - Solução: descrição abstracta da estrutura e principais colaborações
  - Consequências: positivas e negativas
  - Exemplo: de instanciação do padrão
  - Padrões relacionados: com importância para a solução
  - Utilizações conhecidas: pelo menos três

Benefícios dos Padrões

- Veiculam sabedoria entre quem desenvolve software
  - Os padrões de software permitem documentar e transmitir soluções comprovadas para a resolução de problemas típicos de desenvolvimento de software.
- Vocabulário Comum
  - Os padrões oferecem um vocabulário e um entendimento comum sobre desenhos de software.
- Aumentam a "largura de banda"
  - Os nomes dos padrões, se bem escolhidos, podem ser utilizados como abreviaturas para estruturas de desenho complexas, que aumentam assim a "*largura de banda*" entre quem desenvolve software.

Problemas dos Padrões

- Promovem a reutilização de código? Não!
  - Os padrões não originam a reutilização directa de código.
- Simplicidade decepcionante
  - Apesar da sua enorme utilidade, os padrões podem ser considerados extremamente simples.
- Exigem estudo prévio
  - A integração de padrões no processo de desenvolvimento de software é uma actividade não-desprezável, e que exige o estudo de uma quantidade significativa de padrões.
- Validação pela experiência
  - Os padrões são validados pela experiência (bom ou mau?)
  - Não se consegue provar que a solução descrita no padrão é boa, mas apenas ter uma garantia de confiança sobre ela.

Principais Tipos de Padrões

- "Design Patterns"
  - Os primeiros a merecer atenção alargada
  - Incidem sobre soluções para problemas de projecto
  - Popularizados pelo [Gamma95] e [Buschmann96]
- "Architectural Patterns"
  - Fornecem orientações em como estruturar sistemas e subsistemas de software com o intuito de lhes conferir determinadas propriedades
- "Idioms"
  - Padrões de um nível de abstracção mais baixo por serem específicos a uma determinada linguagem de programação
  - Descrevem como implementar aspectos particulares recorrendo às capacidades específicas de uma linguagem

Exemplos de Catálogos de Padrões

- "Analysis Patterns"
  - Fornecem soluções para problemas de domínios específicos [Martin Fowler]
- "Anti-Patterns"
  - Descrevem soluções a não usar e erros comuns a não cometer
- "Meta-Patterns"
  - "Padrões" de padrões independentes do domínio do problema
- "Organizational Patterns"
  - Padrões que incidem sobre os aspectos organizacionais do desenvolvimento de software

Áreas de Aplicação

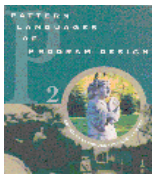
- Muito diversas ...
  - Organizacionais
  - Engenharia de software
  - Ensino OO
  - Sistemas adaptáveis
  - Sistemas distribuídos
  - Sistemas interactivos
  - Sistemas persistentes
  - Interfaces gráficas
  - Sistemas de Telecomunicações (TelePLoP) ...
  - Desenvolvimento de frameworks e de componentes
  - Escrita de Padrões
  - ...

Livros sobre Padrões

"GoF book"
[Gamma95]

"POSA"
[Buschmann96]

"POSA2"
[Buschmann00]



"PLOPD 1,2,3,4

Web

- Patterns Home Page
  - http://hillside.net/patterns/
- Cetus Links
  - http://www.cetus-links.org/oo_patterns.html
- Wiki Wiki Web
  - http://c2.com/cgi-bin/wiki

Frameworks
ou infraestruturas aplicacionais

Frameworks

- Definição
  - Um ***framework*** é um (sub)sistema de software semi-pronto desenhado com o intuito de ser facilmente configurado.
- Útil para a resolução de familias de problemas relacionados
  - Uma *framework* codifica a resolução de uma família de problemas e fornece os blocos fundamentais para a sua implementação.
- Reutilização de código e desenho
  - As *frameworks* possibilitam a reutilização de desenho e de código.
- Padrões em *Frameworks* = Máximo benefício de reutilização
  - Padrões e *frameworks* são conceitos bastante sinérgicos mas não subordinados um ao outro.

Frameworks ...

- Princípio de Hollywood: *"Don't call us, we call you!"*
  - Comparativamente às bibliotecas de classes, as *frameworks* caracterizam-se por uma **inversão de controlo de fluxo**
  - É a *framework* que comanda a resposta do sistema aos eventos externos, invocando operações definidas pelo programador
  - O *main()* está na *framework* e não no código do programador.
- Objectivo das Frameworks
  - Elevada produtividade
  - Tempos de desenvolvimento mais reduzidos
  - Menos erros (bugs)
  - Conjuntos de aplicações mais homogéneos (Palm SDK, Mac, JDK)
- Alguns exemplos de frameworks populares
  - MacApp, NEXTSTEP, MFC, JFC, SanFrancisco

Frameworks e Arquitecturas

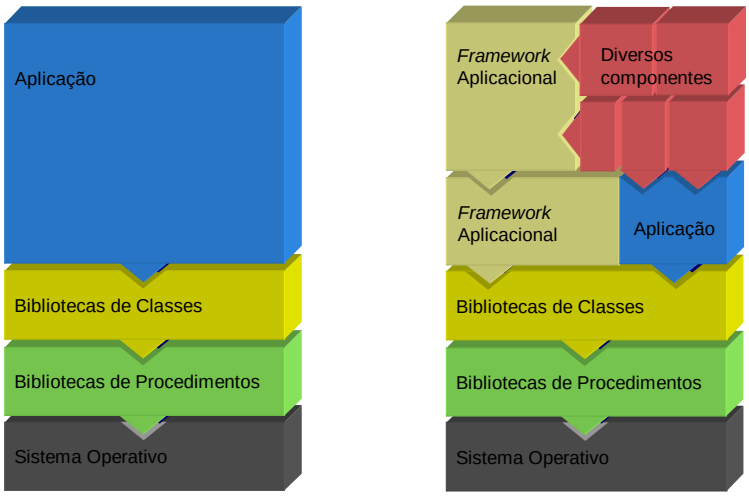
- Arquitectura de Interfaces
  - Conjunto de interfaces que determina toda a estrutura e comportamento
  - Prescreve uma arquitectura para um determinado domínio de problema ou algum aspecto em particular
  - "Abstract Design"
- Arquitectura de Implementação
  - Conjunto de classes concretas que implementam a arquitectura de interfaces
  - Fornecem classes imediatamente utilizáveis e classes semi-prontas facilmente configuráveis
  - "Concrete but Extensible Design"

Tipos de *Frameworks*

- White-Box
  - Fornece superclasses abstractas para serem especializadas
  - Clientes estendem a framework utilizando primeiramente herança
  - Clientes compõem objectos para aplicar a framework
  - Exemplos de javax.swing.text: Document e AbstractDocument
- Black-Box
  - Fornece classes imediatamente utilizáveis
  - Clientes usam composição de objectos para aplicar a framework
  - Exemplo: Document, PlainDocument e DefaultStyledDocument
- Gray-Box
  - Alia características White-Box e Black-Box
  - A maioria das frameworks são deste tipo
  - Exemplo: Swing!

Configuração de *Frameworks*

- Por Herança
  - Utiliza-se o mecanismo de herança
  - A framework fornece um conjunto de pontos de extensão através de classes abstractas
  - Exemplo: javax.swing.text.AbstractDocument
- Por Utilização/Associações
  - Os clientes criam objectos de classes da framework ou de classes de uma sua extensão
  - Para tal, as frameworks normalmente possuem especificações de colaborações com papeis livres
  - Exemplo: atributo *Document document* em javax.swing.text.JTextComponent



Aplicação OO convencional **Aplicação OO com *frameworks***

Bibliografia

- [Alexander77] C. Alexander and S. Ishikawa and M. Silverstein, A Pattern Language, Oxford University Press, 1977.
- [Alexander79] C. Alexander, A Timeless Way of Building, Oxford University Press, 1979.
- [Gamma95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
- [Buschmann96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern Oriented Software Architecture - a System of Patterns, John Wiley and Sons, 1996.
- [Buschmann99] F. Buschmann, Building Software with Patterns, EuroPLoP'99 Proceedings.
- [Cope95] J. O. Coplien and D. C. Schmidt, Pattern Languages of Program Design, Addison-Wesley, 1995.
- [Vlissides96] J. M. Vlissides, J. O. Coplien, and N. L. Kerth, Pattern Languages of Program Design 2, Addison-Wesley, 1996.
- [Martin97] R. C. Martin, D. Riehle, and F. Buschmann, Pattern Languages of Program Design 3, Addison-Wesley, 1997.
- [Harrison99] N. Harrison, B. Foote, H. Rohnert, Pattern Languages of Program Design 4, Addison-Wesley, 2000.
- [Beck96] K. Beck, Smalltalk Best Practice Patterns, Prentice Hall, 1996.
- [Beck&Gamma] "JUnit Cook's Tour", www.junit.org