



Software Architecture Conformance and Recovery

Marco Túlio Valente

mtov@dcc.ufmg.br

Ricardo Terra

terra@dcc.ufmg.br

II Congresso Brasileiro de Software: Teoria e Prática, 2011

- 1 Introdução
- 2 Sistema Motivador
- 3 DSM
- 4 SCQL
- 5 RM
- 6 ACL
- 7 Outras Técnicas
- 8 Considerações Finais

Introdução₅

Introdução – O que é arquitetura?

- Arquitetura é um termo que admite múltiplas definições
- Existem duas definições comumente encontradas:
 - uma separação de alto nível do sistema em suas partes
 - decisões que são difíceis de modificar
- Assim, definiremos como:
“Um conjunto de decisões de projeto que tem impacto em cada aspecto da construção e evolução de sistemas. Isso inclui como sistemas são estruturados em componentes e restrições sobre como tais componentes devem interagir.”

Introdução – Qual o papel de um arquiteto de software?

- Muitos acreditam se tratar de um desenvolvedor sênior, contudo conhecimento técnico é só uma de suas habilidades
- Um bom arquiteto de software deve:
 - Limitar as escolhas durante o desenvolvimento:
 - escolher um padrão de como desenvolver aplicações
 - definir/criar um *framework* para ser utilizado na aplicação
 - Indicar pontos potenciais de reutilização:
 - possuir uma visão abrangente do sistema e de seu contexto
 - adotar um *design* de componentização
 - ter conhecimento de outras aplicações na empresa

- Dentre suas atribuições, a necessidade de considerar a aplicação por um ângulo de visão mais abrangente contempla:
 - Quebrar a complexidade do desenvolvimento de aplicações em pedaços menores e mais gerenciáveis
 - Definir as funções de cada componente
 - Definir as interações e dependências entre os componentes
 - Comunicar esses pontos aos desenvolvedores

Arquitetura Concreta

- Também conhecida como Arquitetura Implementada
- É a arquitetura que está representada no código fonte

Arquitetura Planejada

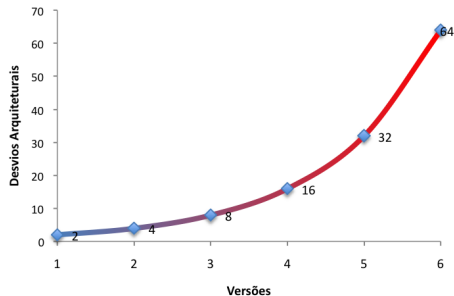
- Também conhecida como Arquitetura Documentada
- É a arquitetura definida nos modelos e documentos arquiteturais do sistema, conforme definições do arquiteto

- Apesar de sua inquestionável importância, a arquitetura documentada de um sistema – se disponível – geralmente não reflete a sua implementação atual
 - Arquitetura Planejada \neq Arquitetura Concreta
- Isso indica que existem decisões implementadas no código fonte que violam a arquitetura planejada
 - A isso denomina-se **desvio arquitetural**
- Desvios arquiteturais são comuns
 - Devido ao desconhecimento por parte dos desenvolvedores, requisitos conflitantes, dificuldades técnicas etc
 - Geralmente, não são capturados e resolvidos
 - Levando ao fenômeno conhecido como **erosão arquitetural**

- Por exemplo, suponha um sistema organizado estritamente em camadas
 - Módulos: M_n, M_{n-1}, \dots, M_0
 - Comunicação: M_i utiliza serviços de M_{i-1}



- Erosão arquitetural tende a crescer com o tempo



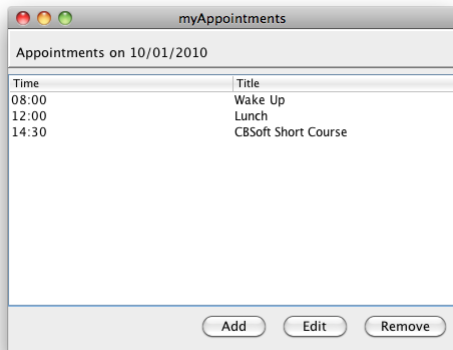
- Erosão arquitetural indica que o sistema está se degenerando
- Isso faz com que os benefícios proporcionados por um bom projeto arquitetural sejam anulados:
 - Manutenibilidade
 - Reusabilidade
 - Escalabilidade
 - Portabilidade
 - etc

- **Recuperação Arquitetural** consiste de um conjunto de métodos para extração de informações arquiteturais a partir de representações de baixo nível de um sistema de software, como o código fonte
- **Conformação Arquitetural** consiste no processo de verificar se uma representação de baixo nível de um sistema de software – como o código fonte ou algo similar – está em conformidade com sua arquitetura planejada

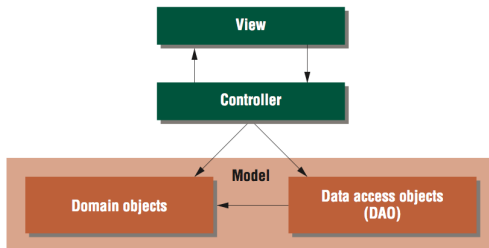
- **Padrão de Projeto** é uma solução para um problema recorrente no desenvolvimento de sistemas. Consiste em uma descrição ou modelo de como resolver um problema. Normalmente, mostra relacionamentos e interações entre classes e objetos
 - Exemplo: *Abstract Factory, Facade, Singleton* etc
- **Padrão Arquitetural** tem um escopo mais amplo que um padrão de projeto. Eles são mais amplos, geralmente descrevendo um padrão global seguido por todo o sistema
 - Exemplo: *MVC, Layers, Pipe* etc

Sistema Motivador

myAppointments é um sistema simples de gerenciamento de informações pessoais implementado exclusivamente para avaliação das soluções de recuperação e conformação arquitetural que serão tratadas neste mini-curso



- Padrão arquitetural MVC



- Divisão clara entre objetos da Visão e do Modelo
- Visão está associada a componentes GUI (*Frames, Buttons, TextField* etc)
- Objetos do Modelo são completamente independentes de qualquer *framework* para construção de interfaces gráficas
- Interações entre o Modelo e a Visão são mediadas por objetos da camada de Controle
- Modelo inclui:
 - Objetos de Domínio (*Domain Objects*) que representam entidades, como Compromissos
 - Objetos de Acesso a Dados (*Data Access Objects* ou DAOs) que encapsulam o *framework* de persistência subjacente

Sistema Motivador – Restrições Arquiteturais

- RA1** Somente a camada de Visão deve utilizar AWT e Swing
- RA2** Somente objetos DAO da camada de Modelo devem depender de serviços SQL
- RA3** A camada de Visão somente depende dela mesma, das APIs AWT e Swing, da camada de Controle e de classes utilitárias. Isto é, não acessam diretamente o Modelo
- RA4** Objetos de Domínio não devem depender de DAOs nem de qualquer objeto das camadas de Visão ou de Controle
- RA5** Classes DAO somente devem depender de Objetos de Domínio, de classes utilitárias e de serviços SQL
- RA6** Classes do pacote util não devem depender de nenhuma classe específica do sistema

- Esse sistema nos guiará no entendimento de algumas técnicas existentes para conformação e recuperação arquitetural
 - Para cada técnica, faremos recuperação de modelos e conformação arquitetural, sempre levando em consideração as RAs descritas para o sistema
- Assim, antes de estudarmos essas técnicas, vamos “bisbilhotar” o código fonte desse sistema

DSM

- DSMs (*Dependency Structure Matrixes*) são matrizes de adjacência utilizadas para representar dependências entre módulos de um sistema
- LDM (*Lattix Dependency Manager*), uma ferramenta para conformação e gerenciamento arquitetural baseada no conceito de DSMs
- LDM também suporta o conceito de regras de projeto (*design rules*), que podem ser utilizadas para definir dependências que violam a arquitetura planejada de um sistema

- Uma DSM é uma matriz quadrada cujas linhas e colunas denotam classes ou agrupamento de classes
- Um **x** na linha referente à classe **A** e na coluna referente à classe **B** denota que a classe **B** depende da classe **A**, isto é, existem referências explícitas em **B** para elementos sintáticos de **A**. Uma outra possibilidade é representar na célula (**A**, **B**) o número de referências que **B** contém para **A**

		1	2	3
Class A	1	.	X	
Class B	2		.	
Class C	3			.

- DSMs foram inicialmente propostas por Baldwin e Clark para demonstrar a importância de princípios de projeto modular na indústria de hardware
 - Após isso, Sullivan *et al.* demonstraram que o conceito de DSMs também pode ser utilizado no projeto de software
- Neste mini-curso, serão utilizadas DSMs geradas pela ferramenta LDM^a (*Lattix Dependency Manager*) 6.0.5
- LDM é uma ferramenta de conformação e visualização arquitetural que utiliza DSMs para representar e gerenciar dependências inter-classes em sistemas OO

^aFerramenta disponível em: <http://www.lattix.com>

- LDM possui dois objetivos principais:
 - revelar padrões arquiteturais
 - detectar dependências que indiquem violações arquiteturais
- Para esse propósito, LDM automaticamente extrai a DSM do código fonte de sistemas existentes utilizando técnicas de análise estática

- Para auxiliar os arquitetos a descobrir e raciocinar sobre estilos arquiteturais, LDM implementa um algoritmo de reordenação (ou particionamento)
- Esse algoritmo decide a ordem de apresentação das linhas em uma DSM, iniciando pelos pacotes que proveem menos serviços e finalizando com os pacotes que são mais utilizados pelos outros pacotes
- Esse algoritmo também agrupa pacotes que são mutualmente dependentes

- LDM inclui uma linguagem simples para declarar regras de projeto que devem ser seguidas pela implementação do sistema
- Regras de projeto possuem duas formas:
 - **A can-use B** **A cannot-use B**
indicando que classes do conjunto A podem (ou não) depender das classes do conjunto B
- Violações em regras de projeto são visualmente exibidas na própria DSM extraída, com o objetivo de alertar sobre possíveis erosões arquiteturais

\$root		view	controller	AbstractAgendaDAO	AgendaDAO	DAOCommand	Appointment	util
		1	2	3	4	5	6	7
myAppointments	+ view	1	.	11				
	+ controller	2	4	.		4		
	- []							
	- []							
	- []							
	- []							
	- []							
	util	7	1	4	2		1	.

Figura: DSM do myAppointments

SCQL

- SCQL (*Source Code Query Language*) contempla linguagens que realizam consultas a nível de código fonte
- .QL é uma linguagem de consulta em código fonte que provê suporte a uma ampla gama de tarefas de desenvolvimento de software
 - Tais como verificação de convenções de código, procura por erros, cálculo de métricas, detecção de oportunidades de refatoração etc
- Apesar de .QL automatizar várias tarefas de desenvolvimento, este mini-curso concentra-se na utilização da linguagem para gerar visualizações da arquitetura e para detectar desvios arquiteturais

- .QL é inspirada na linguagem SQL, o que torna sua sintaxe familiar para a maioria dos desenvolvedores
- .QL inclui outras características que aumentam o seu poder de expressão para a consulta em código fonte
 - *engine* Datalog, consultas recursivas em hierarquia de herança, chamada de métodos etc

- Conceitos de orientação a objetos – tais como classes e herança – podem ser usados para estender a linguagem com novos predicados e construir bibliotecas
- Para melhorar seu desempenho e escalabilidade, .QL utiliza um SGBD relacional para armazenar relações entre elementos do código fonte
 - Assim, consultas .QL são primeiramente traduzidas para Datalog (otimizadas) e, em seguida, traduzidas para SQL

- SemmleCode .QL^a é um *plug-in* para a IDE Eclipse que permite a execução de consultas .QL sobre sistemas Java
- Foi utilizado o Semmlecode Professional Edition, versão 1.0
- Basicamente, a ferramenta inclui um editor de consultas cuja apresentação de seus resultados pode ser vista em forma de árvores, tabelas, gráficos, grafos e *warnings* reportados pelo ambiente de desenvolvimento
- Além disso, conta com a definição de repositórios de consultas para armazenar, por exemplo, consultas realizadas frequentemente

^aFerramenta disponível em: <http://semmle.com>

Primeiramente, será demonstrada uma consulta que retorna as dependências entre os pacotes do sistema `myAppointments` e os pacotes `AWT`, `Swing` e `SQL` da API de Java:

```
1 from RefType r1 , RefType r2
2 where
3     r1.fromSource() and depends(r1 , r2) and
4     (r2.fromSource() or isSwingApi(r2) or isSqlApi(r2))
5 select r1.getPackage() , r2.getPackage()
```

Predicados utilizado:

```
1 predicate isSwingApi(RefType r) {  
2     r.getPackage().getName().matches("java.awt")      or  
3     r.getPackage().getName().matches("java.awt.%")    or  
4     r.getPackage().getName().matches("javax.swing") or  
5     r.getPackage().getName().matches("javax.swing.%")  
6 }  
7  
8 predicate isSqlApi(RefType r) {  
9     r.getPackage().getName().matches("java.sql") or  
10    r.getPackage().getName().matches("java.sql.%")  
11 }
```

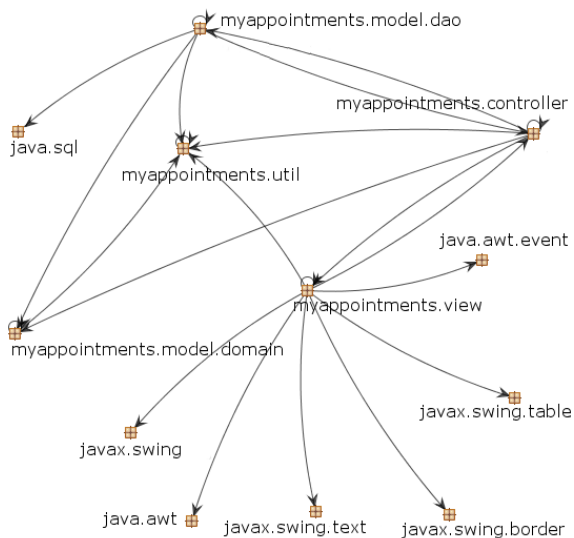
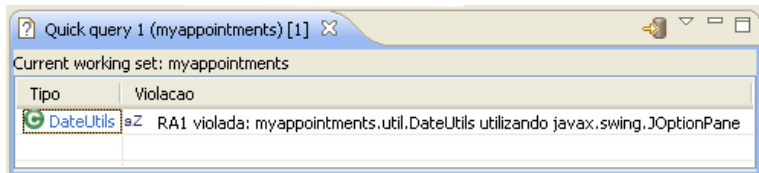


Figura: Grafo do `myAppointments`

Utilizando classes, métodos e predicados de .QL, foram definidas consultas para detectar violações das restrições arquiteturais definidas para o sistema `myAppointments`. Por exemplo, a seguinte consulta verifica se a restrição arquitetural RA1 é seguida:

```
1 from RefType r1 , RefType r2
2 where
3     r1.fromSource()
4     and not(r1.getPackage().getName().matches("myappointments.view"))
5     and depends(r1,r2) and isSwingApi(r2)
6 select r1 as Tipo ,
7     "RA1 violada: " + r1.getQualifiedName()
8         + " utilizando " + r2.getQualifiedName() as Violacao
```



Quick query 1 (myappointments) [1] X

Current working set: myappointments


Tipo	Violação
 DateUtils	RA1 violada: myappointments.util.DateUtils utilizando javax.swing.JOptionPane

Figura: Exibição de uma violação

De maneira similar, a restrição RA3 é definida como a seguir:

```

1  from RefType view , RefType ref
2  where
3      view.getPackage().getName().matches("myappointments.view")
4      and ref.fromSource()
5      and not (ref.getPackage().getName().matches("myappointments.view"))
6      and not isController(ref)
7      and not isUtil(ref)
8      and depends(view , ref)
9  select view as Tipo_Visao ,
10      "RA3 violada: " + view.getQualifiedName()
11          + " utilizando " + ref.getQualifiedName() as Violacao
    
```

Nessa consulta são utilizados os predicados `isController` e `isUtil`, definidos como a seguir:

```
1 predicate isController(RefType ref) {  
2     ref.getASupertype*().hasQualifiedName  
3     ("myappointments.controller", "IController")  
4 }  
5 predicate isUtil(RefType ref) {  
6     ref.getPackage().getName().matches("myappointments.util")  
7 }
```

RM

- Técnicas baseadas em Modelos de Reflexão (*Reflexion Models* ou RM) comparam um modelo arquitetural (isto é, a arquitetura planejada de um sistema) com o modelo de código fonte (isto é, a arquitetura concreta desse sistema)
- O resultado, chamado modelo de reflexão, destaca relações convergentes, divergentes e ausentes entre os dois modelos

- SAVE (*Software Architecture Visualization and Evaluation*) é uma ferramenta de conformação arquitetural centrada no conceito de modelo de reflexão de software proposto por Murphy *et al.*
 - Desenvolvida pelo Instituto Fraunhofer IESE

- Segundo essa abordagem, arquitetos devem primeiramente definir um modelo de alto nível que represente a arquitetura planejada de um sistema
 - Esse modelo inclui os principais componentes do sistema e as relações entre eles (invocações, instanciações, herança etc)
- Arquitetos também devem definir um mapeamento entre o modelo de código fonte (arquitetura concreta) e a arquitetura planejada

Uma ferramenta baseada em modelo de reflexão, como SAVE, classifica relações entre componentes como:

- Convergente: quando uma relação prescrita no modelo de alto nível é seguida pelo código fonte
- Divergente: quando uma relação não prescrita no modelo de alto nível existe no código fonte
- Ausente: quando uma relação prescrita no modelo de alto nível não existe no código fonte

- 1 Modelo de Alto Nível
São definidos os componentes de alto nível e as comunicações entre eles de acordo com a arquitetura planejada do sistema
- 2 Modelo de Código Fonte
É extraído um modelo a partir do código fonte do sistema
- 3 Mapeamento
Tarefa na qual os arquitetos devem manualmente associar cada componente de alto nível aos seus componentes correspondentes no modelo de código fonte
- 4 Modelo de Reflexão
A ferramenta compara os dois modelos e destaca as relações ausentes e divergentes

RM – SAVE – Prática – Screen shot

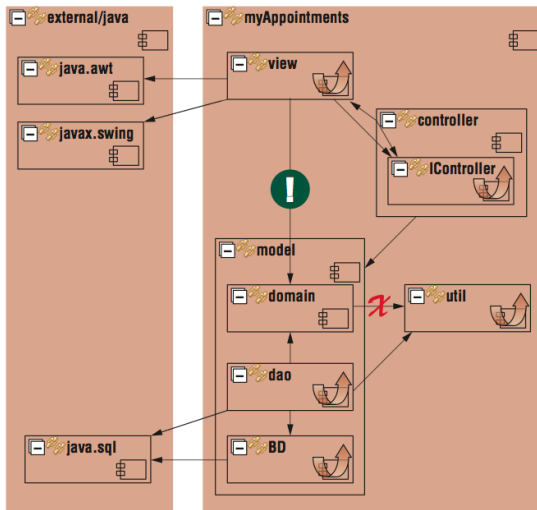
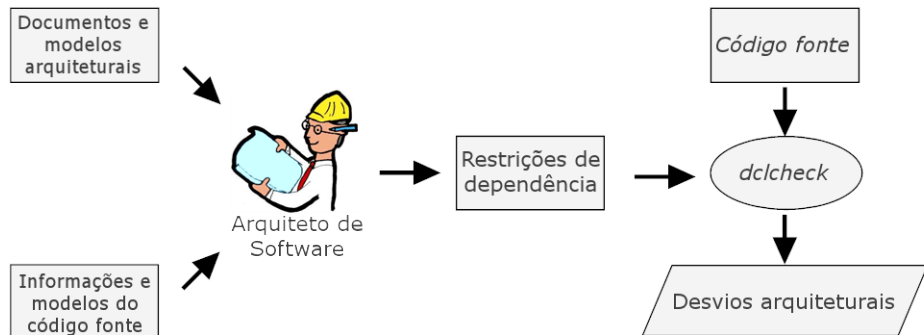


Figura: Modelo de Reflexão Computado para o **myAppointments**

ACL

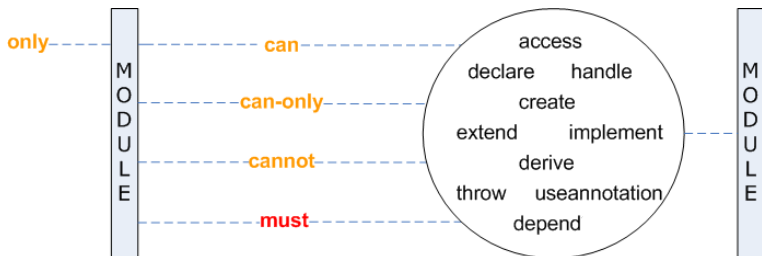
- ACL (*Architectural Constraint Language*) contempla soluções que garantem a conformação arquitetural de um sistema por meio de um conjunto de restrições
- Linguagem DCL (*Dependency Constraint Language*) visa restringir o espectro de dependências aceitáveis e inaceitáveis em um sistema
- O objetivo principal é impedir a erosão arquitetural, isto é, que a arquitetura concreta (aquela presente no código fonte) viole a arquitetura planejada de um sistema
 - Por exemplo, violações de camada, não utilização de padrões, má uso de frameworks etc



- Linguagem de domínio específico, declarativa e estaticamente verificável
- Permite a definição de restrições estruturais entre módulos
- Princípio de funcionamento:
 - Definem-se os módulos
 - Definem-se as restrições entre eles

- Módulos: conjunto de classes
 - pacotes, subtipos, expressões regulares etc
- Exemplos:
 - 1 **module** View: `org.foo.view.*`
 - 2 **module** Remote: `java.rmi.UnicastRemoteObject+`
 - 3 **module** Frame: `"org.foo.(a-zA-Z0-9/.)*Frame"`

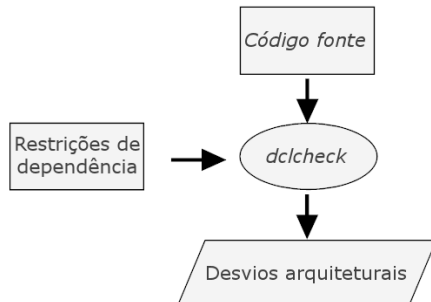
- Restrições para capturar **divergências** e **ausências**:



- Exemplos:

- 1 **only** Factory **can-create** Products
- 2 Util **can-only-depend** Util , \$java
- 3 View **cannot-handle** Model
- 4 Products **must-implement** java.io.Serializable

- Ferramenta ***dc1check***^a
 - Verifica se o código fonte respeita restrições DCL
 - *Plug-in* para a IDE Eclipse



^aFerramenta disponível em: <http://dcc.ufmg.br/~terra/dcl>

ACL – DCL – Prática – Screen shot

The screenshot shows the Eclipse IDE with the following components:

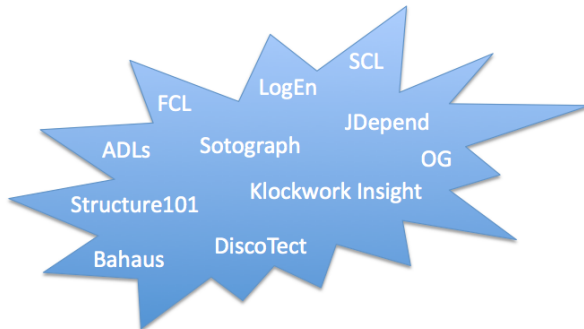
- Package Explorer:** Shows the project structure with packages `myappointments` and `myappointments.view`. The `AppointmentView.java` file is selected.
- Main Editor:** Displays the `AppointmentView.java` file. The `displaySavingConfirmation` method is visible, which calls `app.getTitle()`.
- Dialog Box:** A message box titled `dclcheck` states: "[ACCESS] 'myappointments.view.AppointmentView\$9' contains the method 'run' that invokes the method 'getTitle' of an object of 'myappointments.model.domain.Appointment'".
- dcl Architectural Drifts View:** A table listing detected architectural drifts.

Project Name	Dependency Constraint	Class Name	Location
myappointments	only visao can-depend javaAWT.Swing	myappointments.util.DateUtils	line 113
myappointments	visao cannot-handle modelo	myappointments.view.AppointmentView	
myappointments	visao cannot-handle modelo	myappointments.view.AppointmentView\$9	
myappointments	visao cannot-handle modelo	myappointments.view.AppointmentView\$9	line 133

Figura: Violações detectadas para o **myAppointments**

Outras Técnicas

Existem uma série de outras soluções que lidam com conformação e recuperação arquitetural



Structural Constraint Language (SCL)

- Linguagem lógica de primeira ordem que permite expressar intenções arquiteturais e de projeto
- Restrições definidas sobre a estrutura estática de sistemas orientados a objetos
- Especificações SCL consistem em uma sequência de declarações e fórmulas lógicas

LogEn

- Linguagem lógica de domínio específico
- Permite expressar dependências estruturais entre grupos lógicos de elementos do código fonte, chamados *ensembles*

ADLs

- Alternativa para conformação arquitetural por construção
- ADLs não se aplicam em sistemas existentes, pois são extensões de linguagens

- **Structure101**: definição de modelos em termos de camadas e dependências aceitáveis entre componentes
- **Bahaus**: RM com decomposição hierárquica
- **Sotograph**: permite realizar consultas de conformação sobre dependências do código fonte que ficam armazenadas em um repositório
- **Klocwork Insight**: provê suporte a visualização arquitetural em forma de grafos
- **JDepend**: gera métricas que podem ser utilizadas para medir e controlar o processo de erosão arquitetural

DiscoTect

Utiliza observações em tempo de execução de sistemas para construir uma visão arquitetural do sistema

Considerações Finais

O curso apresentou quatro soluções distintas para Recuperação e Conformação Arquitetural:

- **DSM**: usando como exemplo a ferramenta LDM
- **SCQL**: usando como exemplo a ferramenta .QL
- **RM**: usando como exemplo a ferramenta SAVE
- **ACL**: usando como exemplo a ferramenta DCL

- DSMs representam instrumento simples e poderoso para visualizar e raciocionar sobre arquiteturas de software, pois:
 - DSMs são estruturas inerentemente hierárquicas, o que provê escalabilidade
 - Possuem algoritmos de reordenação
- Regras de Projeto (*Design Rules*) não são expressivas
 - Não possuem expressões regulares ou subtipos
 - Limitação aos tipos de dependência

- .QL representa uma simples, mas poderosa linguagem de consulta em código fonte
 - Poder deve-se sua origem em *Datalog*
 - Simplicidade deve-se a sintaxe inspirada em *SQL*
- Dificuldades para visualizar, navegar e raciocinar sobre representações arquiteturais
 - Mas, será esse o foco?

- Possui um processo completo e bem definido para verificação de conformação arquitetural
- Controle sobre a granularidade e o nível de abstração dos componentes
- Mapeamento é uma tarefa árdua e deve ser mantida
- A computação do modelo de reflexão pode tomar tempo
 - Não é viável ser continuamente aplicado
 - Mas, viável antes de lançamento de versões, após uma interação do desenvolvimento etc

- Restringe o espectro de dependências que podem ser estabelecidas em sistemas orientados a objetos
- É simples e auto-explicativa
- Alto poder de expressão
 - Definição de módulos: pacotes, subtipos, expressões regulares etc
 - Restrições: Todos os tipos de restrições
- Não provê mecanismos para raciocinar ou visualizar sobre a arquitetura
 - Foco na conformação arquitetural



L. Passos; R. Terra; R. Diniz; M. T. Valente; N. Mendonça. **Static Architecture Conformance Checking: An Illustrative Overview**. IEEE Software, 2010



N. Sangal *et al.* **Using dependency models to manage complex software architecture**. OOPSLA, 2005



O. Moor *et al.* **Keynote address: .ql for source code analysis**. SCAM, 2007



J. Knodel *et al.* **Static evaluation of software architectures**. CSMR, 2006



G. Murphy; D. Notkin; K. Sullivan. **Software reflexion models: Bridging the gap between source and high-level models**. ACM SIGSOFT FSE, 1995



R. Terra; M. T. Valente. **A dependency constraint language to manage object-oriented software architectures**. SPE, 2009



Obrigado!!!