Manual de Usuário

Organic:

Síntese de anomalias de código

Versão 1.0.0.1

Sumário

1. Introdução

Organic é um sistema de apoio para desenvolvedores de sistemas de software. O principal objetivo do *Organic* é ajudar desenvolvedores a identificar e raciocinar sobre anomalias de código que sejam arquiteturalmente relevantes, ou seja, anomalias de código que afetam a arquitetura do sistema.

Para cumprir seu papel, além de detectar anomalias de código, *Organic* também agrupa as anomalias de código usando informações extraídas de diferentes artefatos do projeto de software analisado. A esse processo damos o nome de *síntese de anomalias arquiteturalmente relevantes*. Os grupos de anomalias detectados pelo processo de síntese são chamados de *aglomerações*. O processo de síntese pode utilizar diferentes estratégias para detectar aglomerações. Por padrão *Organic* vem implementado com estratégias para três categorias de aglomerações: (i) *intra-method*, (ii) *intra-boundary* e (iii) *hierarchical*. A Tabela 1 apresenta uma breve descrição sobre cada estratégia. Além disso, conforme será explicado mais adiante, os usuários do sistema também podem especificar e implementar suas próprias estratégias, conforme julgue necessário. Para saber mais detalhes sobre o processo de síntese consulte as referências [1][2].

Tabela 1. Categorias de aglomerações.

Nome	Descrição
intra-method	Agrupa anomalias de código que ocorrem juntas em um mesmo método.
intra-boundary	Agrupa anomalias de código de um mesmo tipo que ocorrem em diferentes classes ou métodos de um mesmo componente (ou pacote).
hierarchical	Agrupa anomalias de código de um mesmo tipo que ocorrem em classes (ou métodos das classes) de uma mesma hierarquia (construída por meio de herança/generalização).

O restante desse documento está organizado da seguinte forma. A Seção 2 apresenta os passos para a instalação do sistema. A Seção 3 apresenta as possíveis configurações do sistema. Por fim, a Seção 4 apresenta uma visão geral sobre as funcionalidades do sistema.

2. Instalação

Antes de realizar a instalação do sistema, os seguintes requisitos mínimos deverão ser satisfeitos:

- Sistema operacional Windows Vista, Windows 7 ou Windows 8;
- Instalação do Eclipse IDE 4.3 (Kepler) ou superior;
- Instalação do *Java runtime environment* (JRE) 1.7;
- Memória de 2 GB.

Tendo atendido aos requisitos mínimos, a instalação pode ser realizada da seguinte forma:

1. Copie o arquivo *br.pucrio.inf.organic_1.0.0.1.jar* da pasta “Produto”, que faz parte do pacote de artefatos do *Organic*;
2. Abra a pasta de instalação da IDE Eclipse;

3. Cole o arquivo *br.pucrio.inf.organic_1.0.0.1.jar* dentro da pasta “dropins” (Figura 1);
4. Reinicie a IDE Eclipse.

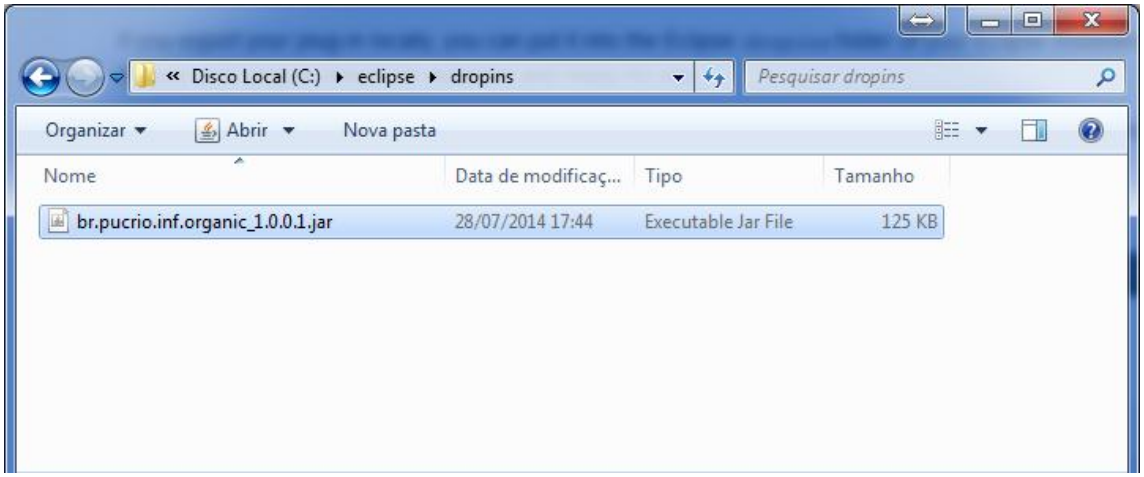

Figura 1. Instalação do sistema.

3. Configuração

A configuração do sistema *Organic* é simples. Além disso, ao instalar o sistema, os valores configurados por padrão permitem que o sistema seja executado imediatamente.

O sistema foi projetado para que as configurações fiquem associadas ao espaço de trabalho (*workspace*) que o usuário está utilizando. Assim, as configurações se aplicam a todos os projetos que fazem parte do espaço de trabalho.

Para acessar a tela de configuração, são necessários os seguintes passos:

1. Acessar o menu Janela (*Window*);
2. Selecionar a opção Preferências (*Preferences*);
3. Selecionar a opção Organic.

Após a execução dos passos supracitados, será aberto uma tela de configuração conforme exibido na Figura 2. Após abrir a tela de configuração, o usuário pode alterar os campos de acordo com sua necessidade. Para salvar as alterações, basta clicar no botão **Ok**. Caso o usuário altere os campos com valores inválidos, uma mensagem de validação é exibida (Figura 3).

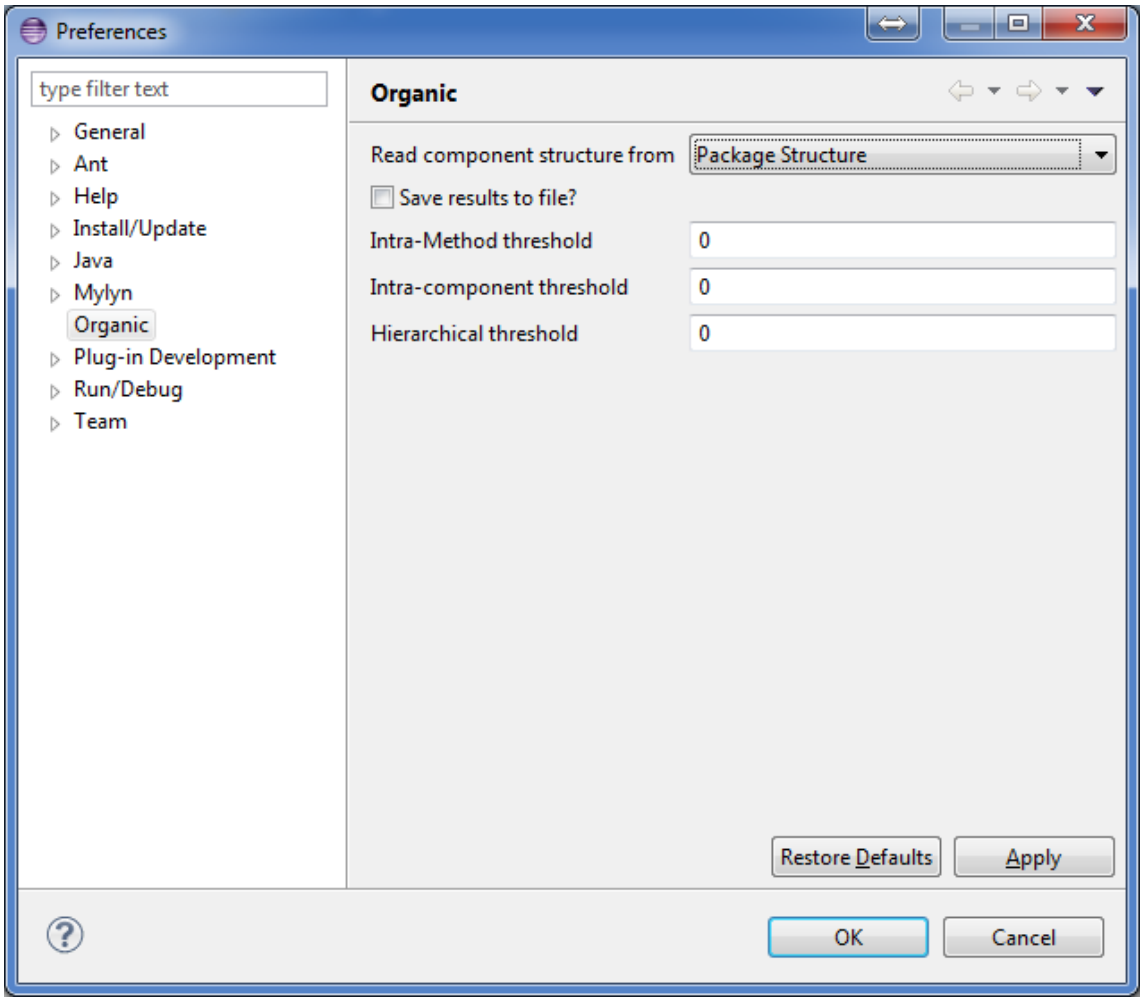

Figura 2. Tela de Configuração.

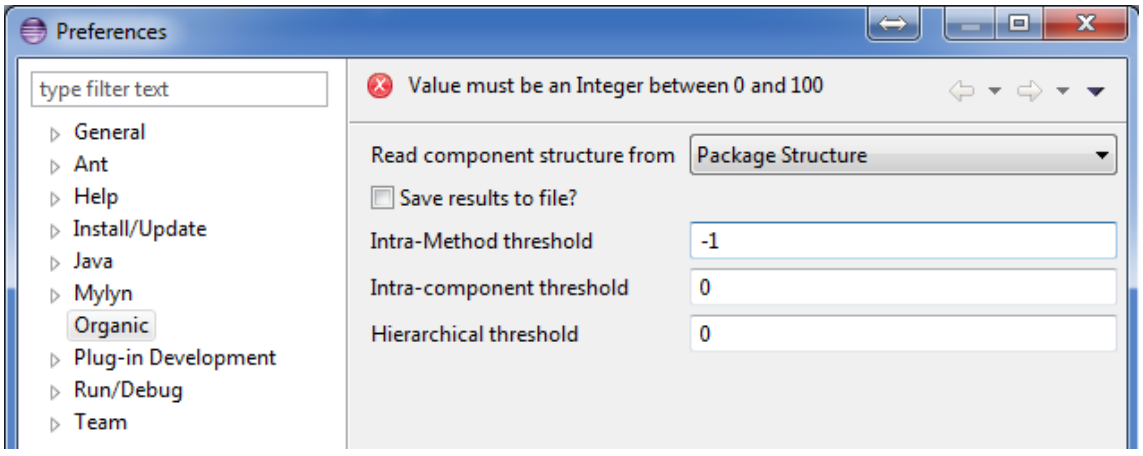

Figura 3. Configuração com valores inválidos.

4. Funcionalidades

A principal funcionalidade do *Organic* é a síntese de anomalias de código arquiteturalmente relevantes. Essa funcionalidade é composta por uma sequência de processos que buscam e processam as informações necessárias. A síntese pode ser realizada de duas formas distintas, de acordo com as configurações do usuário.

Configuração Padrão

Caso o sistema esteja configura para ler a estrutura de componentes a partir da estrutura de pacotes (configuração padrão), não será necessário fornecer nenhum artefato, além do código fonte, para avaliar um projeto. O processo de síntese é iniciado da seguinte forma:

1. Clique com o botão direito no projeto que será avaliado;
2. No menu *pop-up* que abrir, selecione a opção "*Organic-> Find Agglomerations*" (Figura 4);
3. Aguarde até que a *view* (aba) de aglomerações seja exibida ou atualizada (Figura 5).

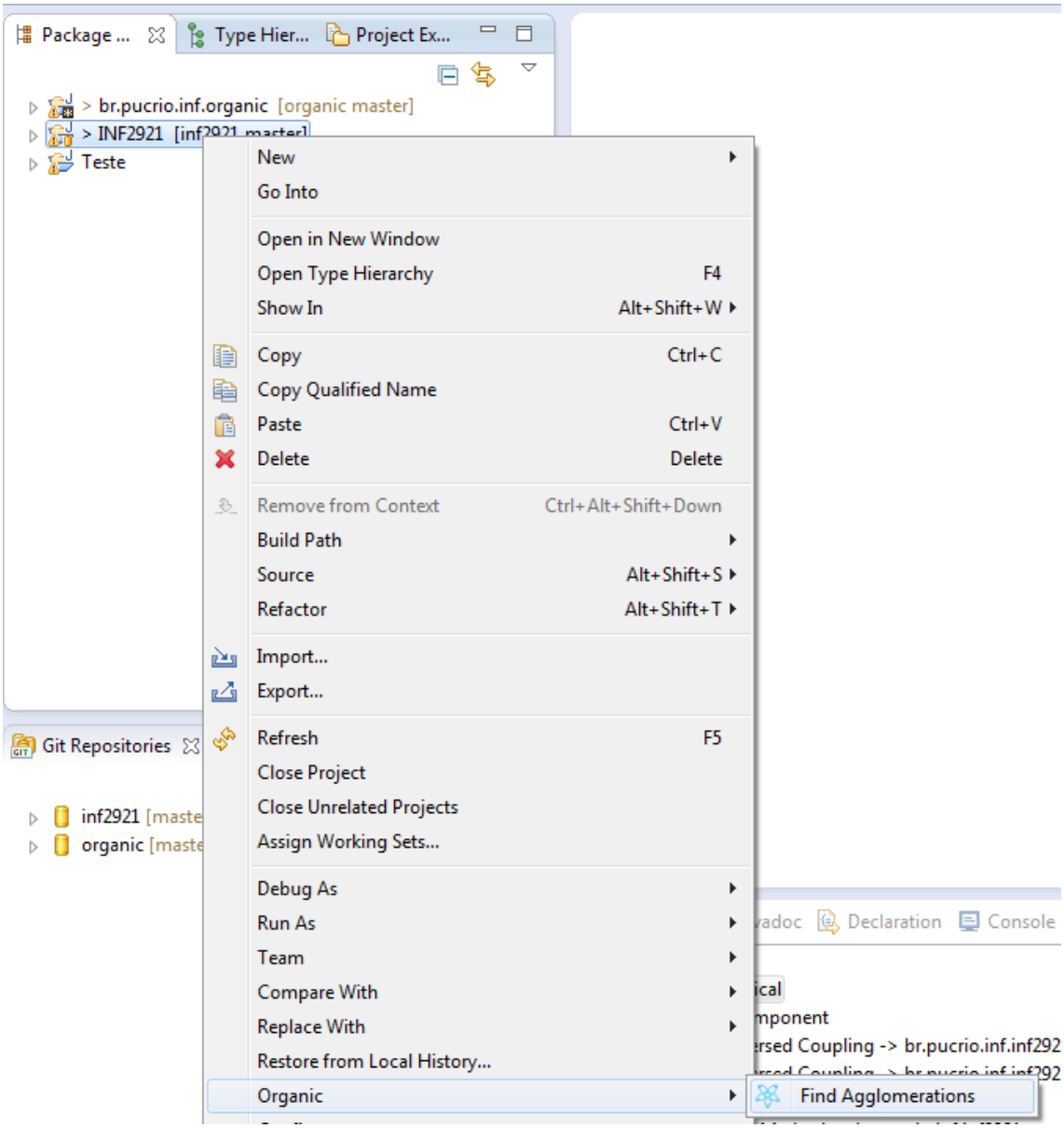

Figura 4. Menu para executar o processo de síntese.

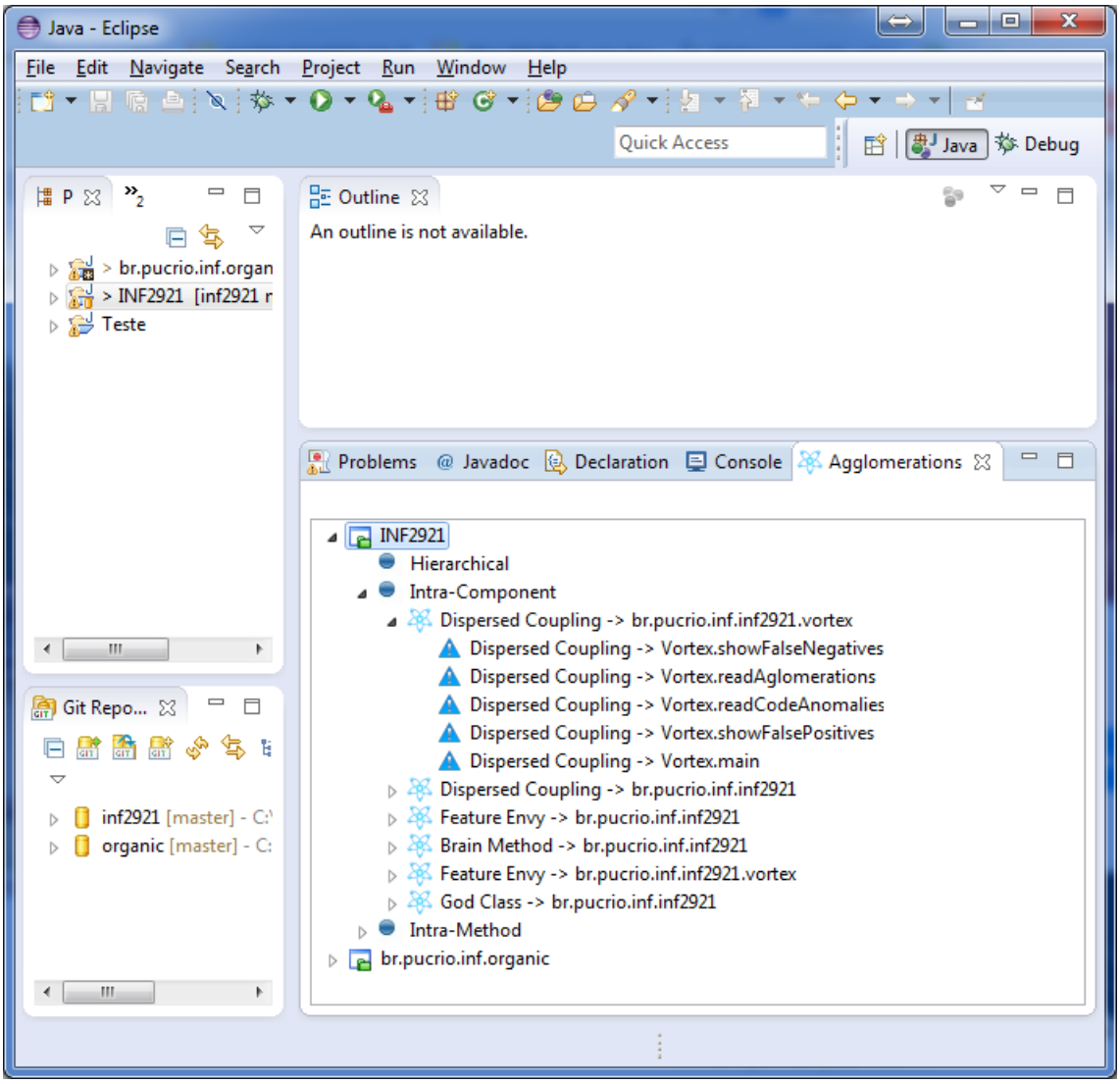

Figura 5. View que exibe as aglomerações.

Arquitetura de Arquivo XML

Caso a arquitetura do projeto avaliado não seja alinhada com a estrutura de pacotes da implementação, existe a alternativa de especificar os componentes por meio de um arquivo xml de mapeamento. Esse arquivo deve ser criado usando a ferramenta *concern mapper* [3][4]. A ideia, basicamente, é mapear quais classes do projeto implementam quais componentes arquiteturais.

O arquivo de mapeamento gerado pelo *concern mapper* deverá ser salva no diretório do projeto. Além disso, o arquivo deverá ter o nome *"components.cm"*, pois *Organic* vai buscar no diretório do projeto um arquivo com esse nome.

Explorando Aglomerações

Como pode ser observado na Figura 5, após a execução da síntese, as aglomerações detectadas são exibidas em uma estrutura hierárquica. O primeiro nível da hierarquia mostra os projetos avaliados. No nível seguinte, são exibidos os tipos de estratégias utilizados (*Hierarchical, Intra-Component*, etc). Para cada tipo de estratégia, são exibidas as aglomerações encontradas. Cada aglomeração é apresentada com uma descrição significativa. Por exemplo, *"Dispersed Coupling*

- > *br.pucrio.inf.inf2921.vortex”* indica que aquela aglomeração é formada por anomalias do tipo *Dispersed Coupling*, dentro do componente *br.pucrio.inf.inf2921.vortex*.

Por fim, no último nível da hierarquia são exibidas as anomalias de código que compõe cada aglomeração. Similarmente ao nível anterior, a descrição de cada anomalia diz o tipo da anomalia e o elemento de código afetado (classe ou método).

Como pode ser observado na Figura 6, ao fazer um clique duplo em uma anomalia de código, o eclipse abre no editor de texto o elemento de código afetado. Além disso, nos elementos de código afetados, são exibidos marcadores (*markers*), indicando a participação em uma ou mais aglomerações.

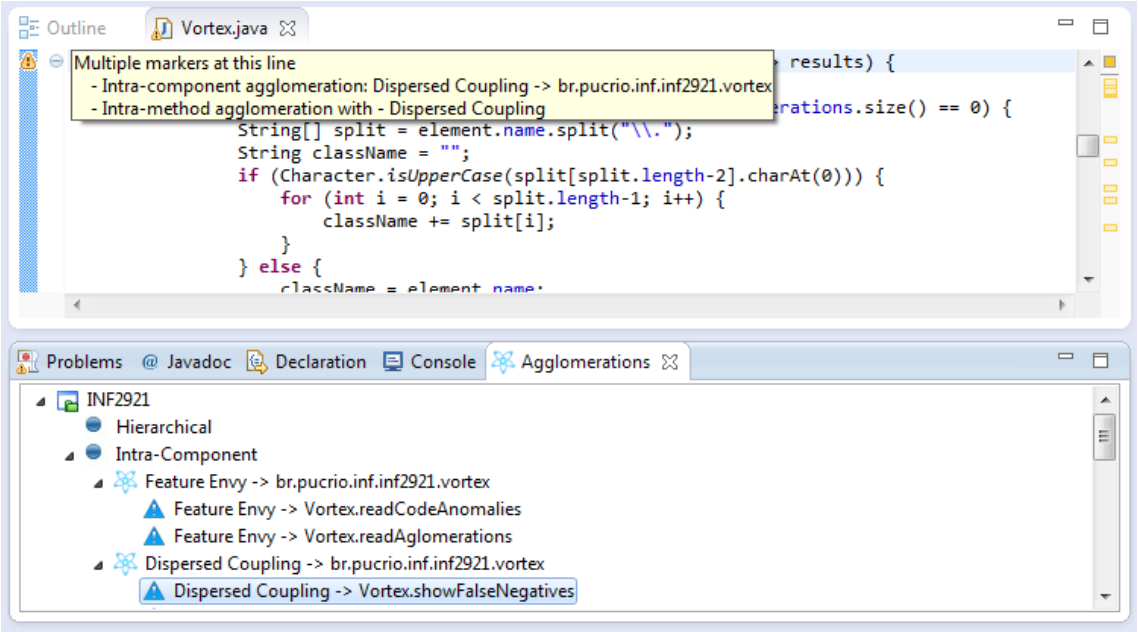

Figura 6. Marcadores no código fonte.

Salvar Resultados em Arquivo

Caso o sistema seja configurado para salvar os resultados em arquivo, a cada execução do *Organic* é gerado um arquivo *csv* no diretório do projeto avaliado. O arquivo tem o seguinte padrão de nomenclatura:

“`OrganicResults_[Nome do Projeto]_[Data e Hora].csv`”.

Cada registro do arquivo gerado contém as seguintes informações (separadas por ponto e vírgula):

1. Sequência;
2. Tipo de estratégia;
3. Descrição da aglomeração;
4. Anomalias de código que fazem parte da aglomeração.

Especificar Estratégias Próprias

Conforme explicado em [1][2], diferentes estratégias de síntese podem apresentar resultados satisfatórios, ou não, dependendo do projeto de software avaliado. Por exemplo, a estratégia de

detecção para aglomerações do tipo *hierarchical*, muito provavelmente, não será efetiva em sistemas que não possuem muitos relacionamentos do tipo herança (generalização). Sendo assim, para avaliar determinados projetos os desenvolvedores podem querer utilizar estratégias de detecção próprias. Para possibilitar isso, o sistema *Organic* utiliza o mecanismo de integração e extensão de plug-ins disponibilizado pela IDE Eclipse. Por ser um mecanismo amplamente documento, ele não será explicado de maneira detalhada neste documento. Sugere-se a leitura de [5] para maiores detalhes sobre a integração entre plug-ins no Eclipse.

Para estender *Organic* com estratégias de detecção customizadas, é necessário que seja feita a implementação de um novo plug-in. Esse plug-in deverá conter implementações concretas para as seguintes interfaces e classes abstratas:

- **AgglomerationDetector.** Essa interface define as classes que serão selecionadas (usando reflexão) e invocadas para a detecção de aglomerações. Ela faz parte da definição do ponto de extensão (*extension point*) provido pelo *Organic*. Caso o desenvolvedor queira especificar uma estratégia de detecção própria, ele obrigatoriamente deverá implementar essa interface.
- **AgglomerationModel.** Essa classe abstrata define as operações e dados que toda instancia de aglomeração deve ter. Ou seja, cada estratégia de detecção implementada deverá prover sua própria implementação de *AgglomerationModel*, de acordo com as características das aglomerações detectadas.
- **AgglomerationStrategy.** O processo de detecção de aglomerações deve ser realizado por subclasses de *AgglomerationStrategy*. Tal classe provê as operações e dados necessários para buscar e analisar anomalias de código nos projetos analisados.
- **AgglomerationTextMarker.** Por fim, recomenda-se que seja criada uma subclasse de *AgglomerationTextMarker* para cada estratégia implementada. Apesar de não ser necessário, já que a criação de marcadores é independente do *Organic*, isso ajuda a padronizar os plug-ins e manter a compatibilidade em versões futuras.

Referências

[1] W. Oizumi, A. Garcia, T. Colanzi, M. Ferreira & A. Staa, “When Code-Anomaly Agglomerations Represent Architectural Problems? An Exploratory Study”, in the 28th Brazilian Symposium on Software Engineering (SBES), to appear, 2014.

[2] W. Oizumi, A. Garcia, L. Sousa, D. Albuquerque e D. Cedrim, “Towards the synthesis of architecturally-relevant code anomalies”, in the 11th Workshop on Software Modularity, to appear, 2014.

[3] M. Robillard, Concern Mapper. Disponível em http://www.cs.mcgill.ca/~martin/cm/. Visitado em 30/07/2014.

[4] M. Robillard and F. Weigand-Warr “ConcernMapper: Simple View-Based Separation of Scattered Concerns”, in the 2005 OOPSLA workshop on Eclipse technology eXchange, 2005.

[5] Eclipse Foundation, “Plug-in Extension Points”, Disponível em http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fconcepts%2Fextension.htm. Visitado em 30/07/2014.