Traditional Detection of Code Anomalies – Eclipse Plugin

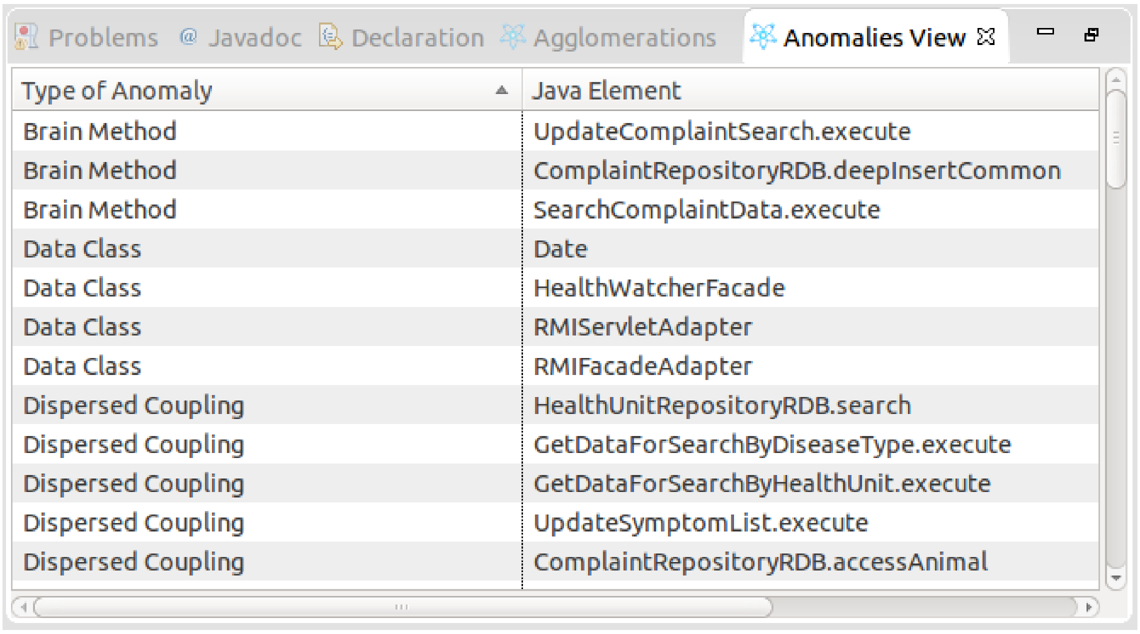
Code anomalies are symptoms in the source code that may indicate maintainability problems, such as **design problems**. Code anomalies are not bugs, instead they only indicate weakness in the source code design that may cause maintainability problems or increase the risk of bugs and failures in the future. Traditional techniques for code anomaly detection commonly present the detected code anomalies in a table or in a list. Moreover, traditional techniques presents the following information about the detected code anomalies: (1) type of anomaly, (2) name of the affected code element and (3) file and position in which the affected element is located.

In this document we present the traditional implementation that will be used in this experiment. This implementation was made as a plug-in for the eclipse platform. Next we present the features that will be used in the experiment.

Anomalies View

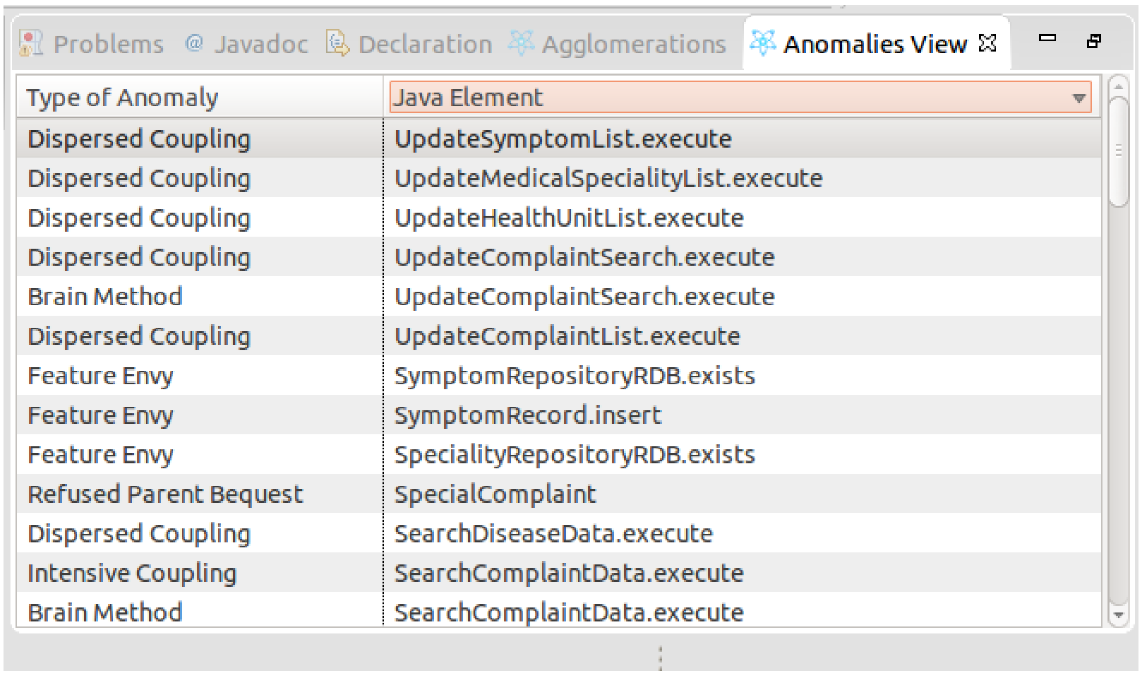
The Anomalies view shows all the detected code anomalies. This view contains a table with two columns: (1) type of anomaly and (2) Java Element. The Java Element column shows the name of the code element (method or class) that is affected by the anomaly. Figure 1 shows a snapshot of the Anomalies View filled with code anomalies.

It is possible to sort the anomalies in this view by any column in any direction. That is, this view allows the programmer to sort anomalies in both ascending and descending order using any of the two columns. In Figure 1 anomalies were sorted by the first column in ascending order. On the other hand, in Figure 2 they were sorted by the second column in descending order.



Type of Anomaly	Java Element
Brain Method	UpdateComplaintSearch.execute
Brain Method	ComplaintRepositoryRDB.deepInsertCommon
Brain Method	SearchComplaintData.execute
Data Class	Date
Data Class	HealthWatcherFacade
Data Class	RMIServletAdapter
Data Class	RMIFacadeAdapter
Dispersed Coupling	HealthUnitRepositoryRDB.search
Dispersed Coupling	GetDataForSearchByDiseaseType.execute
Dispersed Coupling	GetDataForSearchByHealthUnit.execute
Dispersed Coupling	UpdateSymptomList.execute
Dispersed Coupling	ComplaintRepositoryRDB.accessAnimal

Figure 1. Anomalies view sorted by Type of Anomaly

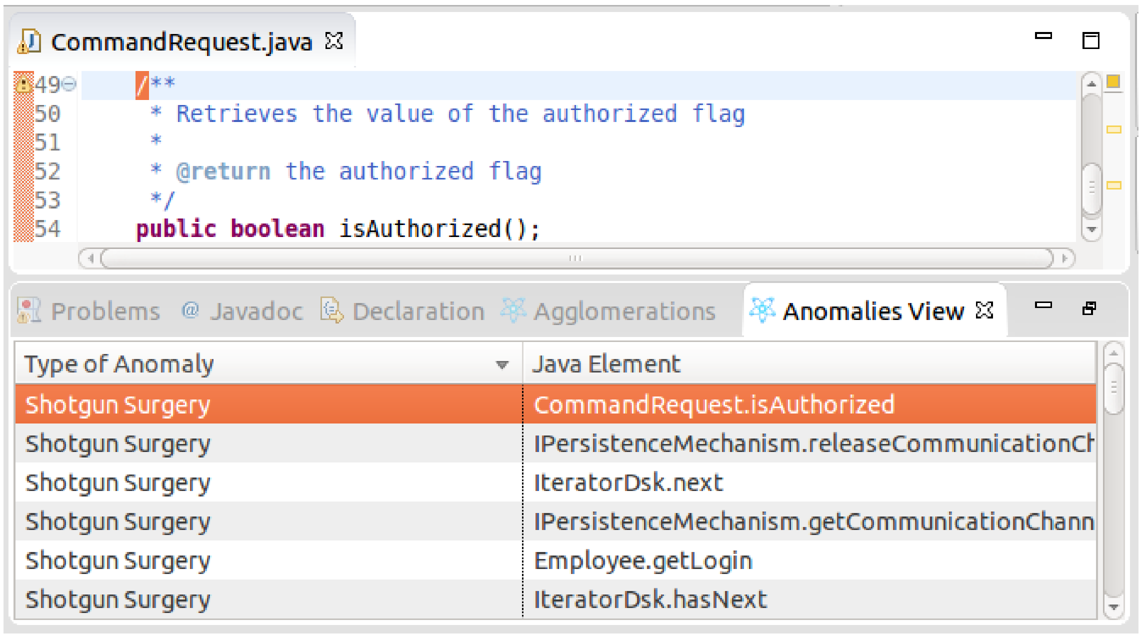


Type of Anomaly	Java Element
Dispersed Coupling	UpdateSymptomList.execute
Dispersed Coupling	UpdateMedicalSpecialityList.execute
Dispersed Coupling	UpdateHealthUnitList.execute
Dispersed Coupling	UpdateComplaintSearch.execute
Brain Method	UpdateComplaintSearch.execute
Dispersed Coupling	UpdateComplaintList.execute
Feature Envy	SymptomRepositoryRDB.exists
Feature Envy	SymptomRecord.insert
Feature Envy	SpecialityRepositoryRDB.exists
Refused Parent Bequest	SpecialComplaint
Dispersed Coupling	SearchDiseaseData.execute
Intensive Coupling	SearchComplaintData.execute
Brain Method	SearchComplaintData.execute

Figure 2. Anomalies view sorted by Java Element

Open Code Element in Editor

Besides providing a table of the detected code anomalies, the Anomalies View also help developers to “navigate” through the anomalous code elements. Using this view it is possible to open the source code of each anomalous code element by double clicking on the anomaly. For example, in Figure 3, a double click in the first line of the Anomalies View opened the anomalous code element, which is the `isAuthorized()` method of the `CommandRequest` class.



```
    /**
     * Retrieves the value of the authorized flag
     *
     * @return the authorized flag
     */
    public boolean isAuthorized();
```

Type of Anomaly	Java Element
Shotgun Surgery	CommandRequest.isAuthorized
Shotgun Surgery	IPersistenceMechanism.releaseCommunicationCh
Shotgun Surgery	IteratorDsk.next
Shotgun Surgery	IPersistenceMechanism.getCommunicationChann
Shotgun Surgery	Employee.getLogin
Shotgun Surgery	IteratorDsk.hasNext

Figure 3. Opening code element in editor