Software Reengineering

Assignment 2

January, 2015

Spyros Foniadakis - 4318773
P.S.N. Chandrasekaran Ayyanathan - 4314506

Single Responsibility Principle Violation

Violating Classes and Packages

The Developer class of the eu.sqooss.service.db package served the purpose of retrieving and storing objects related to the Developer in the database. It is an extension of the DAObject abstraction which serves the functionalities of storing and retrieving from the database. The problem was that it contained two distinct methods:

- static methods for only retrieving objects using conditions (filtering by name, email, etc.) from the database through explicitly queries
- fields and methods for modeling and managing a Developer object

In the same sense, we suggest the same modifications to be applied to all relevant classes, like Bug and BugPriority.

Refactorings

1. A new class called “DeveloperDB” is created in the same package as Developer class
2. The methods getDeveloperByUsername and getDeveloperByEmail were moved from Developer class to DeveloperDB class.
3. All calls to these two kind of methods (getDeveloperByUsername, getDeveloperByEmail) and their overloads were modified (refactored) so as to be called via the DeveloperDB class instead of the Developer class. The following files were adjusted
    a. GitUpdater (eu.sqooss.plugins.updater.git)
    b. MailMessageJob (eu.sqooss.plugins.maildir)
    c. BugzillaXMLJob (eu.sqooss.plugins.bugzilla)
    d. TestGitUpdater (eu.sqooss.plugins.git.test) - a unit test!
    e. SVNUpdaterImpl (eu.sqooss.plugins.updater.svn)

Tests

A test will be performed for each method edited; In this sense, a Developer object will be created and stored in the database. Then it will be retrieved by the following methods, and it will be tested on its existence.

- getDeveloperByEmail
- getDeveloperByUsername
- getDeveloperByName

Acyclic Dependecy Principle Violation

Violating Classes and Packages

The three classes of eu.sqooss.service.db package - MailMessage, MailingList, MailingListThread were cyclically dependent on one another each referring to the objects of the other.

Refactorings

We refactored the code in such a manner that there was a flow in dependency as shown in Figure 1.

Figure 1: Refactoring of Mail classes

Therefore if MailingList has to retrieve corresponding MailMessages then it first must retrieve the relevant MailThreads and then only retrieve the MailMessages. This preserves the hierarchy as was intended and does not make overriding dependencies which would ruin the structure and may possibly cause problems when extending the code later on.

Any associations of MailMessage towards MailingList were removed and vice versa. The following were namely refactored.

MailMessage	**MailingList**
Attribute MailingList list removed	Attribute Set<MailMessage> messages removed
Method getList() now gets relevant thread and then returns the relevant list	Method getMessages() now retrieves relevant threads and through a TreeSet retrieves relevant messages
Method setList() now gets relevant thread and then sets the relevant list	Method getMessages() now retrieves relevant threads and through a TreeSet sets relevant messages

In addition, the explicit queries in the methods of MailingList class - getMessagesNewerThan(), getLatestEmail(), getLatestThread() have also been modified to reflect the refactoring done.
Figure 2 shows the class diagrams created through ObjectAid before and after refactoring.

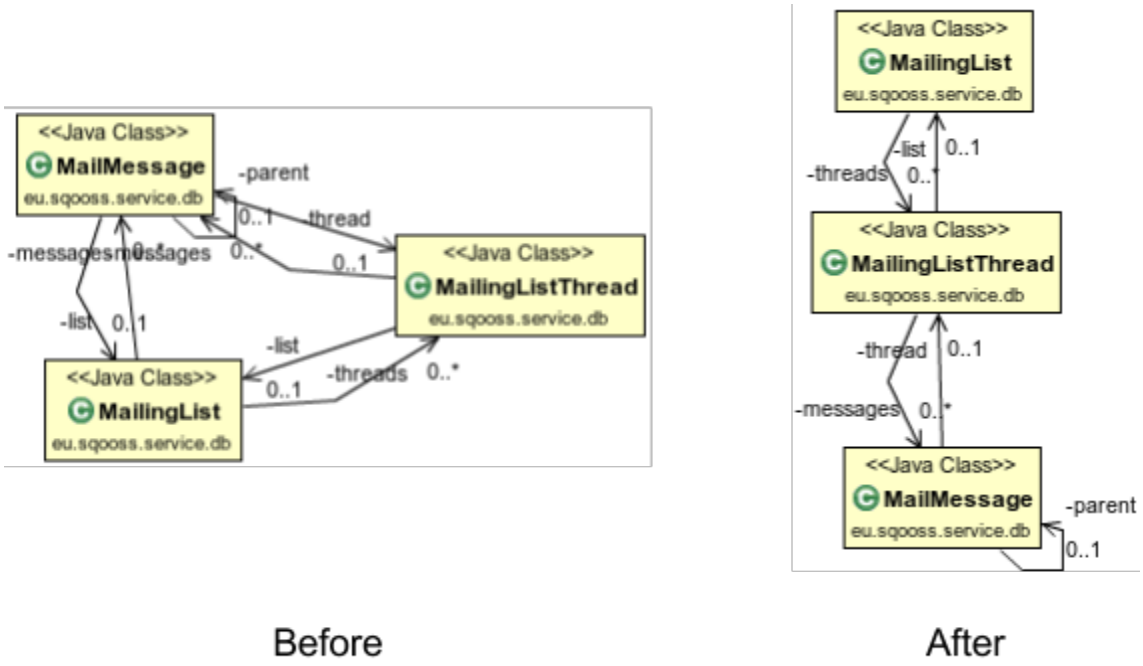

Figure 2 : Before and after refactoring of Mail Classes

Tests

The major changed introduced was the link-removal between MailingList and MailMessage class. Hence from now on, the mail messages of a mailing list will be retrieved via its MailingListThreads. So the tests developed regarded the creation of a mailing list along with a set of mailing list threads and mail messages related to it. Then the mailing list object was checked on whether the getMessages() and the getThreads() methods returned the appropriate number of objects.

Improvements

The following tabulation shows the test coverages calculated using Eclemma including the new tests that were created

Test Package	Test Coverage
eu.sqooss.admin.test	1.9%
eu.sqooss.developer.test	0.5%
eu.sqooss.mail.test	0.6%

Figure 3 shows a snapshot of Eclemma coverage test on the mail test package.

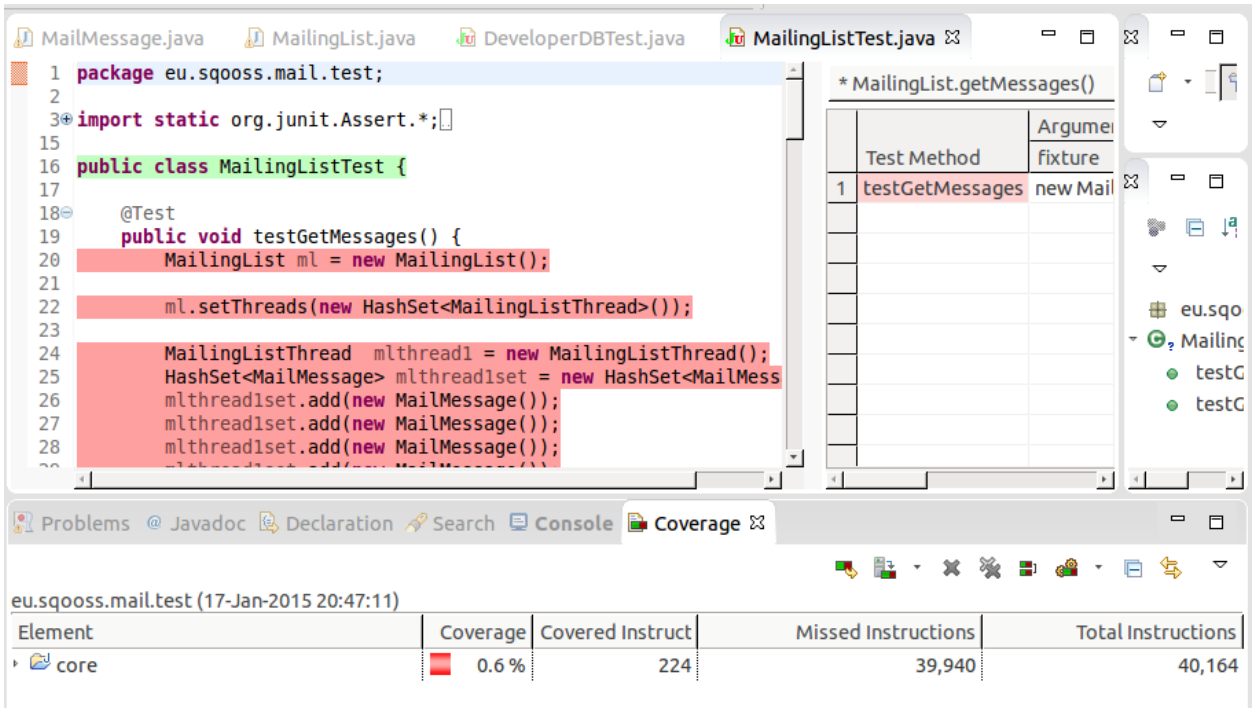

Figure 3 : Snapshot of coverage test for Mail test

The following tabulation shows the signficant metrics as computed through Google CodePro before and after the SRP and ADP violations were rectified.

Metric	**Before**	**After**
Abstractness	5.8%	6.0%
Average Cyclomatic Complexity	1.42	1.41
Efferent Couplings	43	42

The decrease in Cyclomatic Complexity and the Efferent Couplings is attributed to the ADP refactoring of the mail classes whereas the increase in abstractness is attributed to the SRP refactoring of the developer as the number of classes dependent on the developer class has decreased. If a similar refactoring was done to classes such as the developer there would be a further increase in the abstractness.

Suggestions

ISP - Interface Segregation Principle

We consider AlitheiaPlugin, in the package: eu.sqooss.service.abstractmetric, interface to serve four different purposes; (1), (2), (3) and (4). In this context, we suggest the following:

1. To divide this functionalities into four different interface
2. The AbstractMetric abstract class to implement the complete set of the four interfaces discussed above