Course: SE 6356.MS1 - Software Maintenance, Evolution, and Re-Engineering - F15  
Instructor: Andrian Marcus

Assignment #: 2

Group Members: Prabin Gautam, Raza Mohammed, Haitham Muhsen

Deliverables

[1. Part 1:​ the outcome of task# 3.](#h.pn7immatf4cw)

[1.1 aTunes](#h.btcx8crmuv5b)

[1.2 jEdit](#h.p10fb6b2537c)

[2. Part2:​ the outcome of task# 5.](#h.a6yas4nz1a8)

[2.1 aTunes](#h.wln1q9lggd9v)

[2.2 jEdit smells](#h.wmu5pl1fbndy)

[3. Automatic Refactor](#h.lylpgxlw0zep)

[3.1 aTunes smell # 3](#h.bqhcncu9n6yh)

[3.2 jEdit smell #](#h.tknsjfrncwuq)

[4. Manual Refactor](#h.a5q1jz5end0v)

[4.1 aTunes smell #2](#h.p4skqlyahxj)

[4.2 jEdit smell #1](#h.njqgp53z87y0)

[5. Difference between automated and manual refactoring.](#h.dgz58ic2mkcc)

# 1. Part 1:​ the outcome of task# 3.

## 1.1 aTunes

**1.1.a Lowest Cohesion**  
1. net.sourceforge.atunes.gui.views.panels.PlayerControlsPanel  
PlayerControlsPanel is big and has four responsibilities packed into one class, and therefore lacks cohesion. In other words, it packs unrelated responsibilities together. This class could be split into four classes for time/progress-bar, shuffle/repeat, volume/balance, play/pause/stop/next.

2. net.sourceforge.atunes.kernel.handlers.RipperHandler  
This class has common tasks, and two unrelated tasks packed together. It can be broken into 3 classes: cdRipper, SongsFromAmazon, and generalized common tasks shared by them.

3. net.sourceforge.atunes.kernel.handlers.PlayListHandler  
The class groups filter share functionality with other playlist functionalities, and there is a room to improve cohesion. It can be split into two classes one with filter functionalities and the PlaayListHandler with just the play list functionalities.

**1.1.b Highest Cohesion**  
1. net.sourceforge.atunes.gui.ToolBar  
This class is cohesive as it groups related Toolbar items together and nothing else.  
  
2. net.sourceforge.atunes.gui.views.panels.FilePropertiesPanel  
This class is highly cohesive as it groups only file properties items together and nothing else.

**1.1.c Difference**The key difference between cohesive and noncohesive classes is that cohesive classes only packed highly related attributes and functionalities together, whereas low cohesion classes grouped unrelated responsibilities together.

## 1.2 jEdit

**1.2.a Lowest Cohesion**

1. org.gjt.sp.jedit.pluginmgr.InstallPanel

This class has low cohesion. It places together core functionality of InstallPanel with closely related and slightly related helper functionalities. This class can be broken into a class with core functionality, and few other classes with helper functionalities.

2. org.gjt.sp.jedit.View  
This module is too big and has low cohesion. It has multiple classes packed into one module, and can be broken down to have module for each of them. It can be broken down to three modules for ViewConfig, WindowHandler, and the rest.

**1.2.b Highest Cohesion**

1. org.gjt.sp.jedit.Abbrevs  
This class has tightly related functionalities all handling Abbreviation related tasks packed together and therefore is highly cohesive.

2. org.gjt.sp.jedit.JARClassLoader  
This class does one only thing loading class from jar files and is highly cohesive.

**1.2.c Difference**  
The main difference is that low cohesion modules try to do too much and pack unrelated logic into one place, while high cohesion has a clear objective of doing one thing or at least packing tightly related stuff together.

# 

# 2. Part2:​ the outcome of task# 5.

## 2.1 aTunes

**2.1.1 aTunes Smell1**

**2.1.1.a Smell Description**

Smell type: God Class  
smell detector: inCode  
Classes, methods, attributes:  
class: /aTunes/src/net/sourceforge/atunes/kernel/controllers/playList/PlayListController.java

methods:

2.1.2.b Smell reason:  
God Class

2.1.2.c Are they actual smell? Justify.  
We agree that PlayListController tries to do too much. Although the functionalities are related, they can be categorized and the class can be broken down to classes for each category. For example there could be a class for Favorites, Navigation control, PlayList control.

**2.1.2 aTunes Smell2**  
**2.1.2.a Smell Description**  
Smell type: Internal Duplication  
smell detector: incode  
Classes, methods, attributes:  
class: /aTunes/src/net/sourceforge/atunes/kernel/modules/mplayer/MPlayerHandler.java

methods: previous() and next()

**2.1.2.b Smell reason**

Internal Duplication

**2.1.2.c Are they actual smell? Justify.**  
We agree as Previous and next share same logic, that can be abstracted, into a general method used by both. Solution: Extract clone refactoring.  
  
**2.1.3 aTunes Smell 3**  
**2.1.3.a Smell Description**  
Smell type: feature envy  
smell detector: jDeodrant  
Classes, methods, attributes:  
class: net.sourceforge.atunes.kernel.modules.audioscrobbler.AudioScrobblerService

method: getImage

class: net.sourceforge.atunes.kernel.modules.audioscrobbler.AudioScrobblerSimilarArtists

method: getPicture  
  
**2.1.3.b Smell reason**

feature envy

**2.1.3.c Are they actual smell? Justify.**

We agree as the AudioScrobblerService.getImage method uses AudioScrobblerSimilarArtists’s picture attribute through AudioScrobblerSimilarArtists.getPicture method, and thus the client method can be moved to where the data is.

## 2.2 jEdit smells

**2.2.1 jEdit Smell 1**  
**2.2.1.a Smell Description**  
Smell type: internal duplication  
smell detector: inCode  
Classes, methods, attributes:  
 path: /jEdit/org/gjt/sp/jedit/EditPane.java

class: EditPane

methods: prevBuffer(), nextBuffer()  
  
**2.2.2.b Smell reason**

internal duplication

**2.2.2.c Are they actual smell? Justify.**

We afree as prevBuffer and nextBuffer share some logic, which can be exrtracted into a method, being called by both.

**2.2.2 jEdit Smell 2**  
**2.2.2.a Smell Description**  
Smell type: God class  
smell detector: inCode  
Classes, methods, attributes:  
class: /jEdit/org/gjt/sp/jedit/gui/DockableWindowManagerImpl.java

methods:  
  
**2.2.2.b Smell reason**  
God Class

**2.2.2.c Are they actual smell? Justify.**  
We agree. This class packs too much logic into one class. The class also uses many attributes from external classes. This class can be decomposed into many classes.

**2.2.3 jEdit Smell 3**  
**2.2.3.a Smell Description**  
Smell type: Data Clumps  
smell detector: inCode  
Classes, methods, attributes:  
class: /jEdit/org/gjt/sp/jedit/gui/BufferSwitcher.java

methods: getListCellRendererComponent()  
  
**2.2.3.b Smell reason**  
Data Clumps

**2.2.3.c Are they actual smell? Justify.**

We agree. getListCellRendererComponent() method has a long parameter list, and its signature or a significant fragment thereof is duplicated by other methods. Group of parameters, being passed around collectively to multiple methods in the system, could form a new abstraction, that could be extracted to a new class

# 3. Automatic Refactor

## 3.1 aTunes smell # 3

System: aTunes  
smell type: feature envy

smell detector: jDeodrant  
refactor: auto by jDeodrant

3.1.a Refactoring need justification

Smell reason: feature envy

AudioScrobblerService.getImage method uses AudioScrobblerSimilarArtists’s picture attribute through AudioScrobblerSimilarArtists.getPicture method, and thus the client method can be moved to where the data is.

3.1.b Refactoring description

jDeodrant moves logic of AudioScrobblerService.getImage() method to new method AudioScrobblerSimilarArtists.getimage(). AudioScrobblerService.getImage() is still there, but uses AudioScrobblerSimilarArtists.getimage(). This way anything dependent on AudioScrobblerService.getImage() function as before.

3.1.c Manual code changes  
None

3.1.d Test case

Reference search and trace

1. AudioScrobblerService.getImage is used by   
   /aTunes/src/net/sourceforge/atunes/kernel/modules/audioscrobbler/AudioScrobblerSimilarArtistsRunnable.java AudioScrobblerSimilarArtistsRunnable.run()
2. /aTunes/src/net/sourceforge/atunes/kernel/modules/audioscrobbler/AudioScrobblerRunnable.java
3. /aTunes/src/net/sourceforge/atunes/kernel/handlers/AudioScrobblerServiceHandler.java retrieveInfo()
4. /aTunes/src/net/sourceforge/atunes/kernel/controllers/audioScrobbler/AudioScrobblerController.java updatePanel(AudioFile file)
5. /aTunes/src/net/sourceforge/atunes/kernel/handlers/**PlayListHandler**.java, and  
   /aTunes/src/net/sourceforge/atunes/kernel/handlers/VisualHandler.java showAudioScrobblerPanel()

Base on the search and trace, it is understood that getImage() is ultimately used to display image for selected song in the playlist, therefore the test case is:   
Select a song in playlist panel -> top menu -> Playlist -> Info -> pops window with song info including image (if any)

Shortcut: Select a song in playlist panel -> F1 -> pops window with song info including image (if any)

**Test results:**  
Before refactor: It displayed mage

After refactor: It displayed image

**3.1.e smell status**  
Smell removed: yes

## 

## 3.2 jEdit smell #

**Note:** jDeodrant did not run on jEdit, it just hung up without giving meaningful information.

I have 2 options for automatic refactor:

1. either find smell manually, and refactor manually
2. or do more smell detection and refactoring for aTunes (where both inCode and jDeodrant run)

We choose to refactor 1 more smell for “**aTunes”** instead.

System: aTunes

smell type: feature Envy

smell detector: jDeodrant  
refactor: auto by jDeodrant

**3.2.a Refactoring need justification**  
src/net/sourceforge/atunes/kernel/controllers/fileProperties/FilePropertiesController.java: fillSongProperties() method uses   
properties from   
src/net/sourceforge/atunes/gui/views/panels/FilePropertiesPanel.java   
This is a feature envy smell, and can be solved by moving that method to the class with the attributes it is using

**3.2.b Refactoring description**

moved method fillSongProperties() from FilePropertiesController to FilePropertiesPanel

included necessary imports

FilePropertiesController now refers fillSongProperties() from FilePropertiesPanel  
  
**3.2.c Manual code changes**  
None

**3.2.d Test case**  
Explored areas in aTunes, to see if the file properties are being displayed.  
**Test results:**Before refactor: file properties are displayed

After refactor: file properties are displayed

**3.2.e smell status**  
Smell removed: yes

# 4. Manual Refactor

## 4.1 aTunes smell #2

System: aTunes

Smell type: Internal Duplication  
smell detector: incode  
refactor: manual

**4.1.a Refactoring need justification**

Since both methods share common logic, the code is duplicated. Any change to that logic then has to be done at both places, which is inefficient and can create errors. If the common logic is extracted and encapsulated in a single method, any future change can be done only there.  
  
**4.1.b Rationale for the chosen refactoring operation**

Since both methods, use resources from the current module/class, the extracted common method, and both refactored methods will stay in the same module/class.  
  
**4.1.c Refactoring description**  
Internal duplication is resolved by extracting common logic refactoring.  
Visualized diff of two methods, using tkdiff, to see what is shared and what is unique to the methods. Turns out all logic is shared. Based on this analysis extracted common logic to a third method, which takes next or previous as selection parameter.  
Also added additional feature: when current song was first, on clicking previous it did not go to the last. It does that now. It already had go to first on clicking next on the last song.

**4.1.d Test case**  
Based on the method and class names MPlayerHandler.previous() and MPlayerHandler.next(), it was speculated to be tied to next and previous buttons in the playlist panel.   
Test 1: logging.debug() was used in methods to print on button click.   
Test 2: Visual confirmation that the song highlight/selection moved back 1 step on previous button press, and moved forward 1 step on next button press  
Test results:Before refactor: test 1 and test 2 passed

After refactor: test 1 and test 2 passed

**4.1.e smell status**  
Smell removed: yes

## 4.2 jEdit smell #1

System: jEdit  
smell type: internal duplication

smell detector: inCode  
refactor: manual

**4.2.a Refactoring need justification**

Since both methods share common logic, the code is duplicated. Any change to that logic then has to be done at both places, which is inefficient and can create errors. If the common logic is extracted and encapsulated in a single method, any future change can be done only there.

**4.2.b Rationale for the chosen refactoring operation**

Since both methods, use resources from the current module/class, the extracted common method, and both refactored methods will stay in the same module/class.

**4.2.c Refactoring description**  
Internal duplication is resolved by extracting common logic refactoring.  
Steps:   
 1. Tested buffer is working in reading pane

2. Created method navBuffer(String navType)  
 3. called the common method from prevBuffer() and nextBuffer()  
 4. Tested buffer is working in reading pane

**4.2.d Test case**

Opened jEdit, open a file with large content that needs scrolling the viewing pane area, scrolled to see the view is changing.  
**Test results:**Before refactor: passed

After refactor: passed

**4.1.e smell status**  
Smell removed: yes

# 5. Difference between automated and manual refactoring.

Based on our experience the difference was in comfort level.

We we more comfortable with automated refactoring when the refactoring was simple, straight-forward and easy to understand. Automated was good when refactoring similar simple changes that repeated multiple times.

For more complex refactoring we were more comfortable with manual refactoring. However code correction, auto completion, dependency validation features from eclipse were helpful during manual refactoring.