ESE Report

Juan, Victor, Markus, Micheal

2006-12-19

Contents

Introduction

This report evaluates and compares the internal quality of the ESE projects at the *University of Bern* in 2006.

1 CVS Analysis

The CVS analysis is based on the java files only. So pictures, jars and so are not visualized on the pictures in this section. Why we do this? A CVS analysis can show us if people worked together on the same files. If they collaborate with eachother it also means that the sourcecode is understandable for more than one man and that is a sign for good code quality. And by the way, we also have to understand the code, so it would be easier for us.

From a cvs point of view, team 1, team 4 and team 5 seems to be good ESE groups in 2006.

Team 1

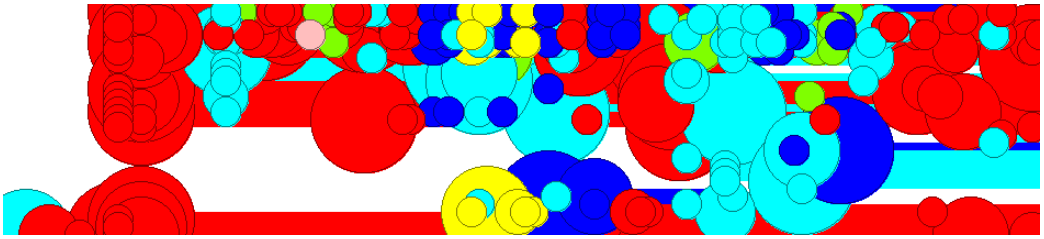

Figure 1: Visualisation from ESE group 1

In Figure 1 (p.2), we see that allmost all team members worked on the sourcecode. It seams they worked together, except the green author who hasn't a lot of commits.

Team 2

Figure 2 (p.3) shows us that this group has not a lot of collaboration in their project, because the red authos stick out a lot.

Team 3

In group 3 the one author did a lot as well, but we also see in Figure 3 (p.4) at the bottom, that they worked together. We also see that they have much more commits that other groups but for this you can look at Section 2 (p.3).

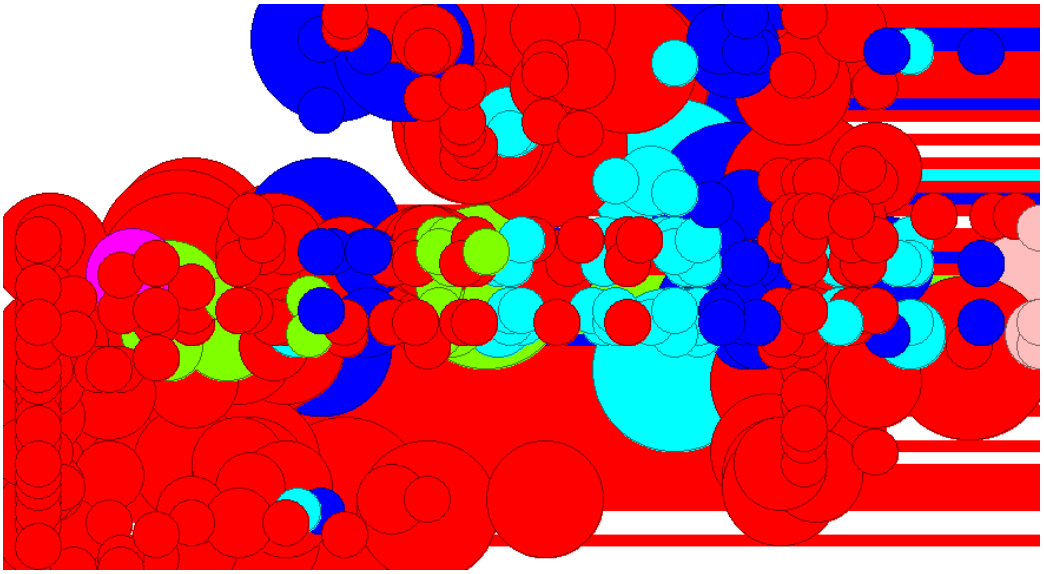

Figure 2: Visualisation from ESE group 2

Team 4

Team 4 has some collaboration on some files on others they don't.

Team 5

Team 5 has the best collaboration of all teams, because almost eveyone worked on all files.

2 Metrics

Team 1

This project is the smallest one. It bears the most reduced number of code lines (2252) and classes (26). Further exploring through the metrics obtained showed that there was something weird at the main frame class. Its complexity at the actionListener was high (47 McAbe) and the nested block depth was huge (7). It was caused by almost all the behaviour of the application being concentrated there. It would be likely to be difficult to maintain, perhaps it should be dealt with.

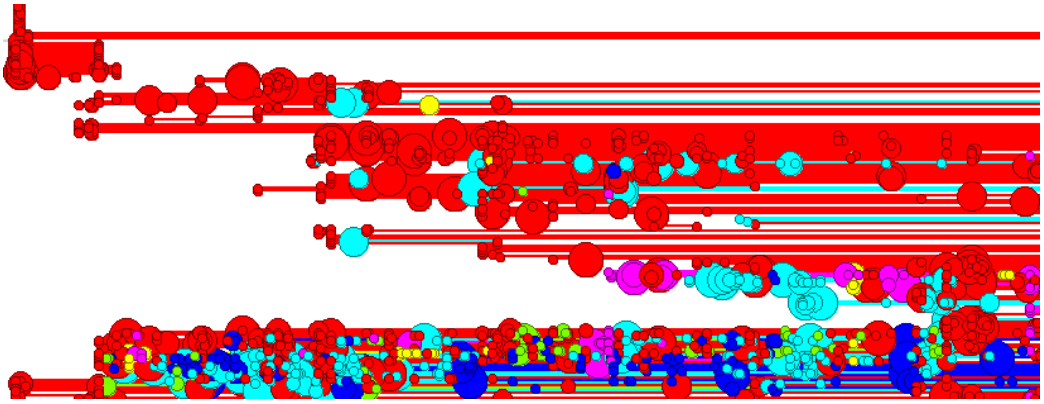

Figure 3: Visualisation from ESE group 3

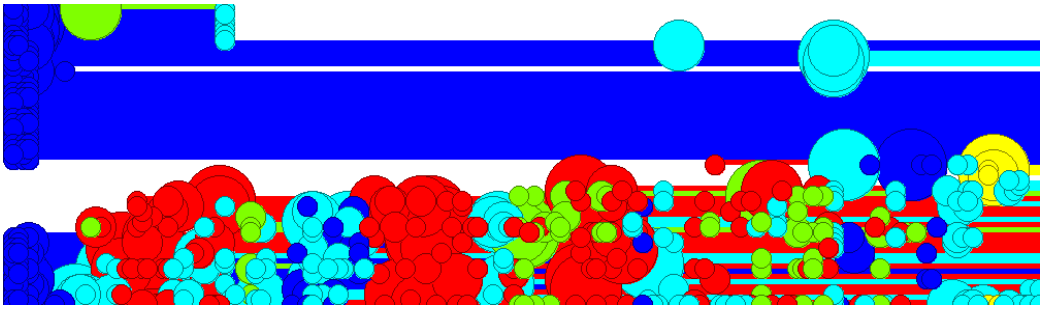

Figure 4: Visualisation from ESE group 4

Team 2

The number of code lines is 3148 and we have 37 classes. The classes are more or less the same size as in the first teams application. We found a problem of similar characteristics here also. There was another listener at the ListPane class that was too complex (18 McAbe) but its block depth was under normal levels, so we didnt consider this a problem that should be focused on or taken into account.

Team 3

This was by far the largest project. For example, compared with the first it came to be 5 times larger. It has 11296 code lines and 105 classes. The classes were, in average, quite bigger than in the first and second projects. We found third party code that had quite a complexity and nested block depth. As it was third party, we didnt

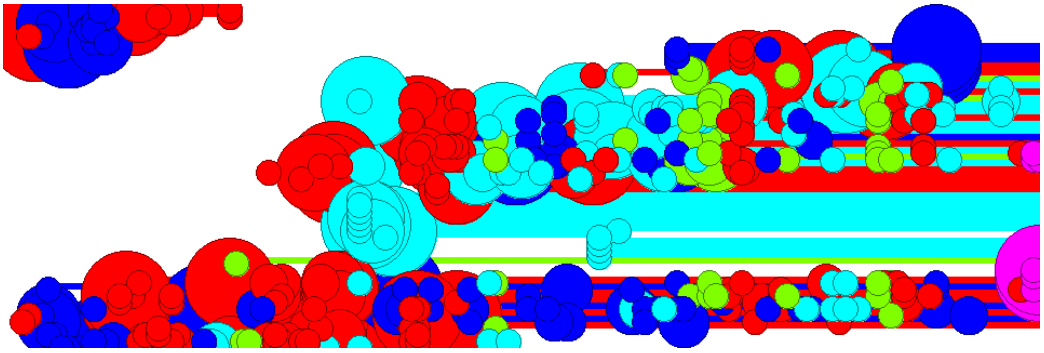

Figure 5: Visualisation from ESE group 5

try to find out more about it, as far as we are concerned its just used as a black box. We only got to know it was related with the tag extraction system, among other tasks. We also found there was a class called EspectrumAnalyzerDisplayPanel which would need 6 parameters but it didnt look like a big setback .

Team 4

This project would make for an intermediate point among the whole set. It has 6253 code lines and 51 classes. The classes are, in average, quite bigger than in the first and second projects. We found a BasePlayList class with a nested block depth of 9. Also, here was also present the same third party code that we found in the third project. Just focusing on their own code, we found a Track class that had a constructor that would need 7 parameters. That, nevertheless, wasnt something bad in itself. There were lots of constructors in that class that would have less requirements. Its existence was of optative nature (we werent forced to use it as there were so many additional constructors) and perhaps it could come to be even handy.

Team 5

This was also a small project. It had 2413 code lines and 41 classes. The classes would have a normal size (very similar relation among code lines and classes if compared with the first project). However, we had our attention required over the class that would make for the main part of the GUI. It took 9 parameters for its

constructor. When we checked it, we found that it needed to receive every button as a parameter. That wouldnt be a very good decision if we were about to extend this concrete part of the application. If we just added some buttons, the constructor header would get huge. If we were to remove them, we would have to take into account more things than we should normally have.

3 Test

Team 1

Pros:

- Most of the methods are selfexplaining.
- Complete set of tests

Cons:

- No comments in any Test File!!
- There aren't tests that uses real MP3 files!!
- They comment a few tests out that don't work :P ¿¿ They removed testRemoveTrack and testRemoveTrackByTitleAndArtist from the test because generate errors

Team 2

Pros:

- Comments in the tests

Cons:

- 100% of the test pass but
  - testRemoveTrack pass but in the application remove a track doesn't work
  - testSaveLibrary is empty
- Not too many tests and a lot of useless tests

- There aren't tests that uses real MP3 files

Team 3

Pros:

- Use MP3 files in the tests
- Complete set of tests

Cons:

- No comments in the test source code
- Many tests but then the app throw up exceptions everywhere

Team 4

Pros:

- Not too many tests but quite useful to test the main behavior of the application

Cons:

- no comments

Team 5

Pros:

- Uses MP3 files

Cons:

- Not useful set of tests.
- Not many operations with the playlist

	Team 1	Team 2	Team 3	Team 4	Team 5
% of tests passed	100%	100%	100%	100%	76.5%
number of tests	51	17	88	26	17
use of MP3 files	no	no	yes	yes	yes
All tests runs at once	no	no	yes	yes	no

Table 1: Overview over ESE tests

3.1 Overview

4 Questions

4.1 How extensible is the design?

Team 1

The design seems to be well organized. Though there is no high use of inheritance among their own classes, it would only be necessary to adopt certain patterns already built into the system in order to extend it. The fact that the model is completely disengaged from everything else will certainly help.

Team 2

Again here the extension seems a likely possibility. In this case, the GUI is quite easier to extend than in the first project, as its a bit more divided and disengaged. The model shouldnt give any problem and additional functionality should fit in properly.

Team 3

This project is huge compared with the others. The first impression is it would take a little bit more to extend it, especially the GUI as it is bigger and needs more work than the others. The fact is the GUI, although not necessarily the best designed, has the widest functionality among all the GUI found at this projects. However, the third party code that is used for certain tasks could be a problem.

Team 4

Here the GUI is mixed a little bit with the model. It could make things more complicated. Moreover, we have the same third party code that could get the extending process even more complex. If this project would need to go through it, then perhaps we should work on this before doing anything else.

Team 5

Once more, the model is disengaged from the view and we should be able to extend almost every aspect on the application. The GUI seems to be somewhat more complex than the one from the first teamss project, for example. But then, thats probably because its more divided and structured. Adding new modules shouldnt be very difficult.

4.2 How easy is it to plug in a different view (for example a Web Interface) into the application?

The interface could be easily extended in all the projects because all of them uses a Model-View-Controller Pattern that splits the functionality to work separately with the data and the representation of the information (interface).

4.3 How are errors handled?

In all projects no error handling is implemented. All *try catch* looks like in Listing 1 (p.9).

Listing 1: Sample try-catch code

```
try {
  ...
} catch (IOException e1) {
  e1.printStackTrace();
} catch (Exception e2) {
  // e2.printStackTrace();
}
```

4.4 Are there any team members that do not participate?

In all teams are members who did't made a lot. In team 2 it's horrible, because the red author made moslty everything.

4.5 Nice to know!

Team 1

No unhandled exceptions found.

Team 2

java.lang.ArrayIndexOutOfBoundsException when removing a trak from a libery. This bug does not occurs in every situation. It seams it happens after a special sequence of actions.

Team 3

java.lang.ArrayIndexOutOfBoundsException when removing a trak from a libery. This bug does not occurs in every situation. It seams it happens after a special sequence of actions.

Team 4

No unhandled exceptions found.

Team 5

Pushing "Next" on a empty libery throws a **java.lang.NullPointerException**

5 Evaluation

- Software Design (2 grades)
- Unit Tests (2 grades)
- Documentation (1 grade)
- Coding Style (1 grade)

	Team 1	Team 2	Team 3	Team 4	Team 5
Software Design	1.5	1.9	1.2	1.0	0.8
Unit Tests	1.2	1.0	1.8	1.7	0.5
Documentation	0.8	0.4	0.3	0.8	0.9
Coding Style	0.8	0.2	0.3	0.8	0.9
Final Grade	**4.3**	**3.5**	**3.6**	**4.3**	**3.1**

Table 2: Grade Overview

6 Estimate a user story

You are to estimate the following user story for each project: "The user wants to share his favorite playlist with all his friends that have access to his shared hard-drive. He exports the playlist in the M3U playlist format so that others can import the playlist in their favorite music player. Some of his friends are using 3rd party software, others are using an ESE MusicPlayer."

For this we should choose the project from team 4, because

- good design
- the development version already includes support for importing and exporting m3u files.
- if we had to choose a project with looking only at the firstrelease we would have choosen the project from team 1