OORPT Team 5 - ESE Evaluation

December 2006

Tests and Error Handling

Tests	Team 1	Team 2	Team 3	Team 4	Team 5
Number of test methods	52	17	80	26	17
Lines of code tests/project	763 (10.1%)	507 (7.0%)	3956 (8.3%)	968 (4.4%)	533 (6.9%)
Error free	yes	yes	yes	1 Failure	4 errors
What do they test?	Model	Model	Model and Database	Model	Model
Do they seem to be useful?	yes.	More or less. Some do something, but others don't (see LibrarySaverTest)	Yes, very detailed and in-depth!!	yes.	Hm. Not very much. See LibraryTest
Grade (0 – 2 points)	2	0.5	2	1.25	0.25

Bad examples:

Team 2, LibrarySaverTest:

```
public class LibrarySaverTest extends TestCase {

    public void testSaveLibrary(){

    }

}
```

This test is not useful!

Team 5, LibraryTest.testIsEmpty():

```
public void testIsEmpty() {
    // TODO this should be empty in my opinion, but is not
    //assertTrue(this.aLibrary.isEmpty());
    this.aLibrary.add(new UserPlayList(new CorePlayList()));
    assertFalse(this.aLibrary.isEmpty());
}
```

This is not the way we want tests to be.

Error Handling by example: What happens if a music file which is in a playlist does not exist anymore before launching the application?

	Team 1	Team 2	Team 3	Team 4	Team 5
Could the program start?	Yes	No. We had to remove the library.xml file.	Yes	Yes	yes

Error Handling by example 2: What happens if a broken music file is loaded?

	Team 1	Team 2	Team 3	Team 4	Team 5
Does it survive?	Yes	Yes	Yes	Yes	Yes
What else happens?	Displays an empty line, nothing happens	Does not import it	Does not import broken file	Displays an empty line, play button state is switched	Displays an empty line, nothing happens

Estimating a user story: How long will it take to add an import / export function for playlists in the m3u format?

Details	Team 1	Team 2	Team 3	Team 4	Team 5
Involved existing model Classes	SerializablePlaylist, LibraryFileHandler , Library.saveData() and .loadData()	LibrarySaver .parsePlaylist()..., ContentManager, Library	DataLoader, DataSaver, PlaylistLibrary	Loader, Saver.savePlaylist, javazoom BasePlaylist	XmlUtils .savePlaylist, AbstractPlaylist, UserPlaylist
Easy structure?	Simple structure	Simple structure	Big but well structured	A bit messy	Simple structure
Current persistence	xml	xml	Database	xml	xml
Importer: selects files or only directories?	files	files	files	directory	Directory, but files visible
Cyclomatic complexitiy average / max (whole)	2 / 43	2 / 19	2 / 28	2 / 27	1.5 / 7

To do	Team 1	Team 2	Team 3	Team 4	Team 5
M3u file reading / writing	5h	5h	5h	1h	5h
Add playlist to library and add files	1h	1h	1h	1h	1h
GUI: file and content menu / button	Menus => 3h	Buttons => 2h	Menus => 2h	Menus + Buttons => 3h	Buttons => 2h
GUI import and export windows	2h	2h	2h	5h	5h
Total:	11h	10h	10h	10h	13h

Documentation

Team1 uses Javadoc comments only in the modelclasses consistently. They have an UML-diagram in the repository which shows the basic structure of the model.
The Readme explaines only the most basic functions of the player. It does not tell how to edit Tracks or move them around in Playlists. There is no Pattern explicitly documented.
Score: 0.25

Team2 did not provide an UML diagram for their system and there are very few Javadoc comments. They should certanly use them more often.
The Readme matches the behaviour of the program and covers the functionality. There is no additional documentation on the teams website except for the autobuild reports.
Score: 0.25

Team3 provides an Eclipse-generated UML diagram of their Model but it is pretty hard to get an overview over the model out of it because it is very small and if you zoom in you loose the context of the classes. Maybe it would be better if there was not every constructor etc. listed so the classes would not take so much space and it would be easyer to see how they are relatet.
The Readmefile of this Team explaines only where the actual documentationfiles are stored. They are using a help system based on html pages. The intetion seems to be to make them accesible from within the running program by using the help-menu. Currently this does not

work. But the help-files can still be viewed by a browser. They explain in a detailed manner the usage of the program.
There are no design patterns documented. The website holds no additional userdocumentation. Javadoc is used in some classes but in most it is missing.
Score: 0.5

Team4 uses Javadoc quiet often. Many of their methodes have a javadoccomment and there are also many normal comments in the code.
We could not find any UML-diagram for the project though.
There is a user documentation on the website of the team but it is not 100% accurate. For example it states that during playing the progress is shown but this does not work at the moment. It also states that you can doubleclick on a song to play it but you have to click it once to select and then press play.
There are no desing patterns documented.
Score 0.5 (because of many javadoccomments)

Team5 has 3 UML diagrams in their repository and on their website. One shows the sequence diagram of file-importing, one the model and the last one documents how they implemented the MVC pattern.
There is also a readme that explaines how to use tha program.
Javadoc is used often. The core model classes in particular are very well documented. The Team also uses the TODO tag of eclipse alot. We think this is good because it is easyer to find sourcecode that should maybe be refactored for other programers.
Score: 1

Design and Code Conventions

All Teams use the MVC Pattern based JavaSwing framework. They all have separated packages for model view and controller. The only exception is Team 4 which has model and gui classes in the same package. So it is harder to find out where to look for the classes.
All Teams have an average of around 10 LOC per methode over the whole project. Also the CC averages under 2 per methode.

Design evaluation Team 1

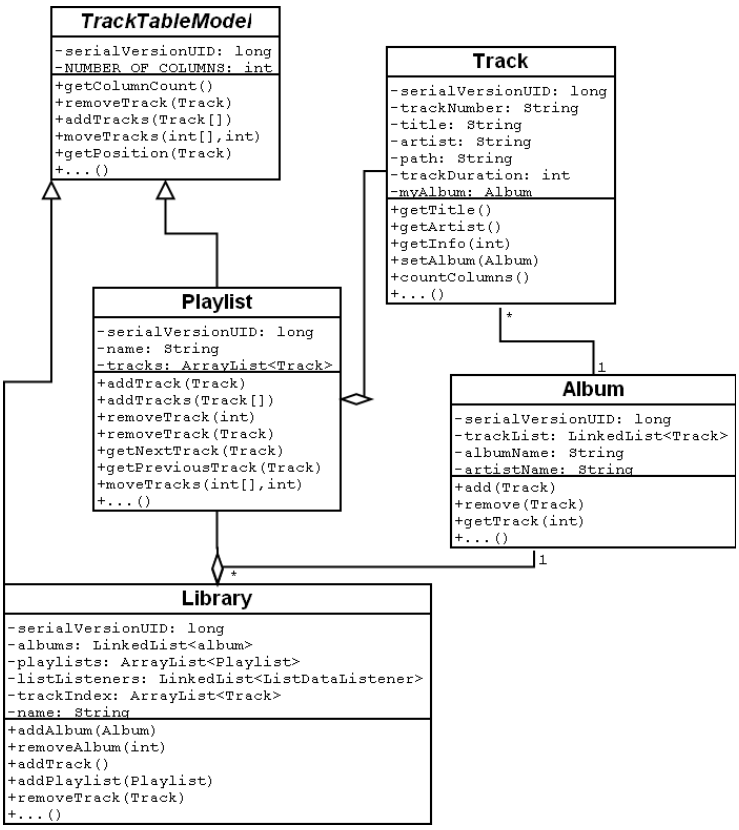
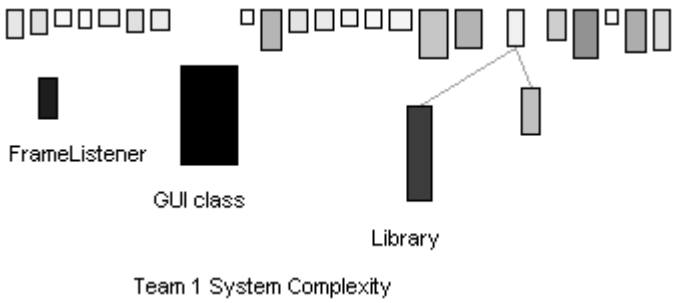


Abbildung 1: Team 1 Overview

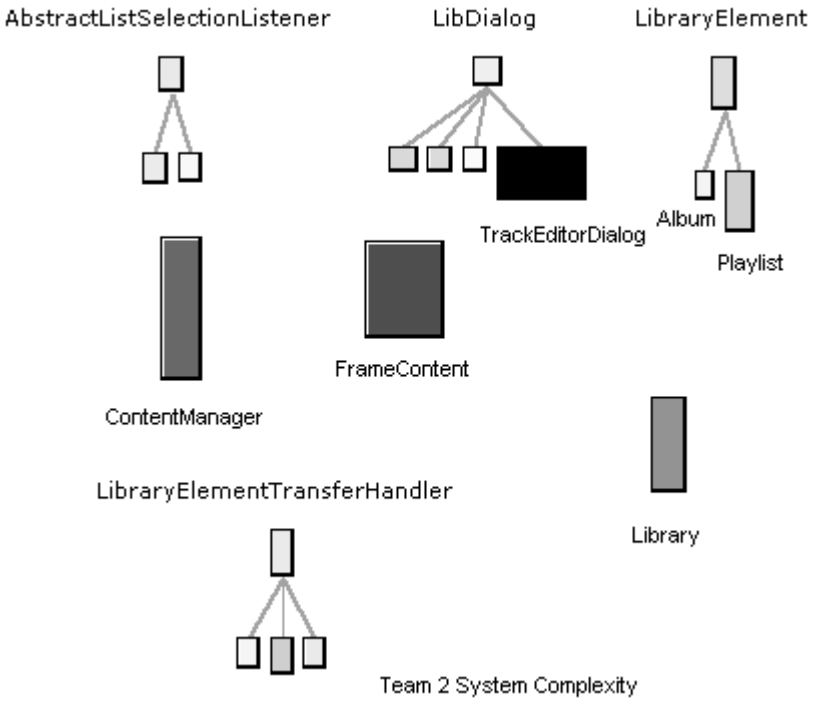
Their project is rather small. They have about 4000 LOC.
They have the whole gui in one Class. Everything is built there. We think it should be possible to split the gui in several subframes so we do not have such a Giant Class.
The FrameListener seems to have too much responsibility because its actionPerformed methode is very long and has a CC of 43. It manages the whole behaviour if a button is pressed.
The Library seems to be managing the Playlists. We think this is not a responsibility the library should have.



Team 1 System Complexity

Design Score: 1 + 0.5 CodeConventions

Design evaluation Team 2



Their project has about 5000 LOC.
By Looking at the SystemComplexity we found 3 Classes that seemd to be out of the norm.
The TrackEditorDialog class manages the editing of the Track Informations. This class breaks the MVC Pattern because it implements the gui the and the controller for track editing at the same time. We think this could be separeted.
They have implemented drag and drop to move songs into playlists which seems to be a good idea.
The ContentManager has no clear responsibility.
There is a user-defined exception class called library Exception but it does not have any special functionality. Nevertheless this is a good thing because the name of the exception already might give you a hint where the exception happened if it gets thrown.

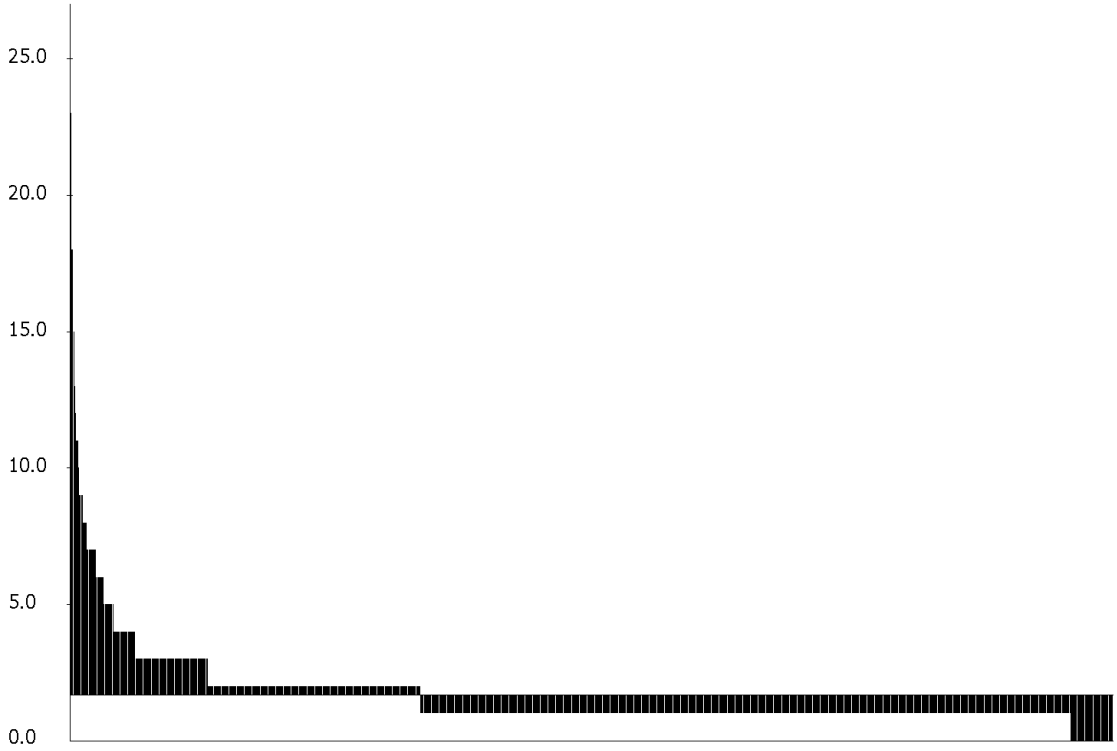
Score: 1.25 + 0.5 CodeConventions

Design evaluation Team 3

Their system is much bigger than the other ones. It has more than 18'000 LOC. Team 3 and 4 are both using the MP3 library and player jlgui from http://www.javazoom.net/. This might be the reason why their projects are bigger than the other ones. They partialy rely on functionality that is provided by the jlgui player (eg. Equalizer).
Like Team2 they created some user defined Exceptions. They are thrown if there occurs a problem while loading or saving from the database they use.
The gui creation is distributed over several Classes. But some of them are still pretty big. In the MainMenuBar class we could see that they are checking for the state in which they are to react

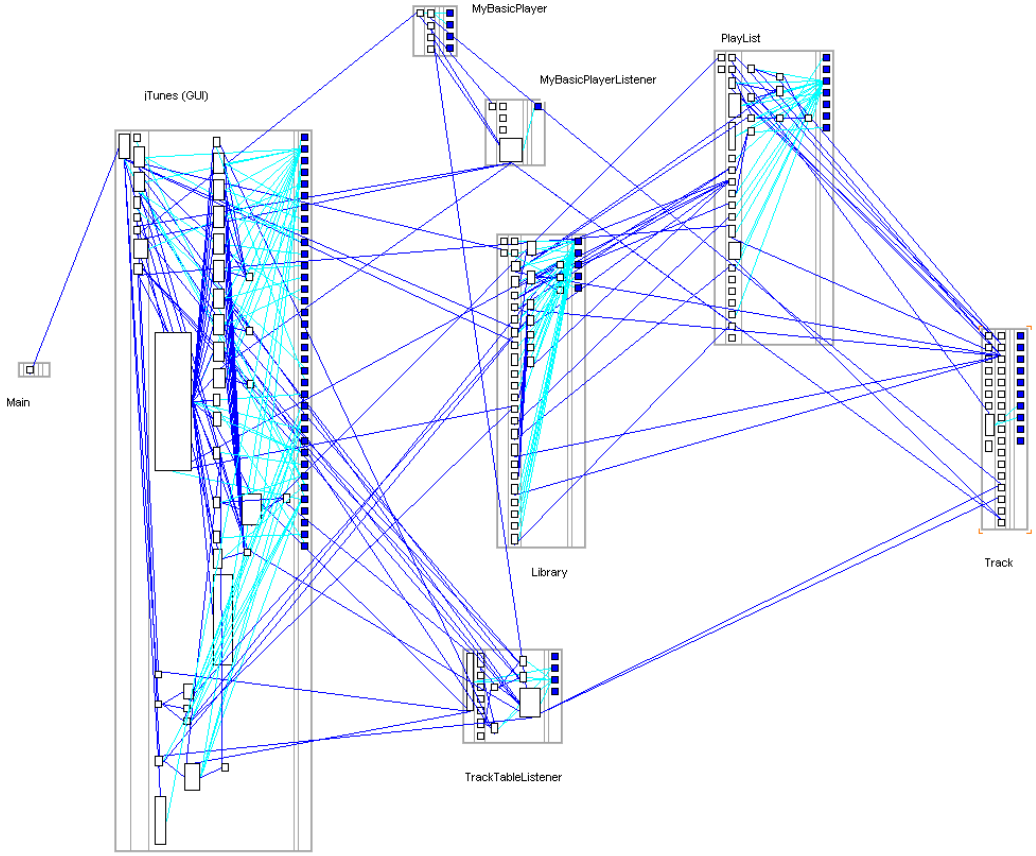
accordingly to events. This is not a problem at the moment but maybe they could use a State pattern later.

The Graph below shows the cyclomatic complexity of the project. Each bar represents a class. The Bars are drawn starting at the average over all bars and extending to the top if the class has a cc above average to the bottom if it has a cc below average.



Score 1.5 Design + 0.7.5 CodeConventions

Design evaluation Team 4



Overview Team 4

This project has about 10'000 LOC. They are using a Factory Pattern and a decorater pattern in the TableSorter class but it seems like this classes were taken of some extern library and they have not written it themselfs. They also use a very big gui class that should be split in some subframes. At the moment this guiclass is also building/instantiating the rest of the system.
It seems they also already have a M3U importer.
In the repository there is a file in the Files/mp3 folder named Classes.rtf. We found it a bit strange that there are some classes listet that are not existing in the project. For example there is an Album class that does not exist in the system. It might be a good idea to introduce this class. Maybe it would be possible to use a Composite pattern for the Playlist and Album.

Score 1.25 Design + 0.5 CodeConventions

Design evaluation Team 5

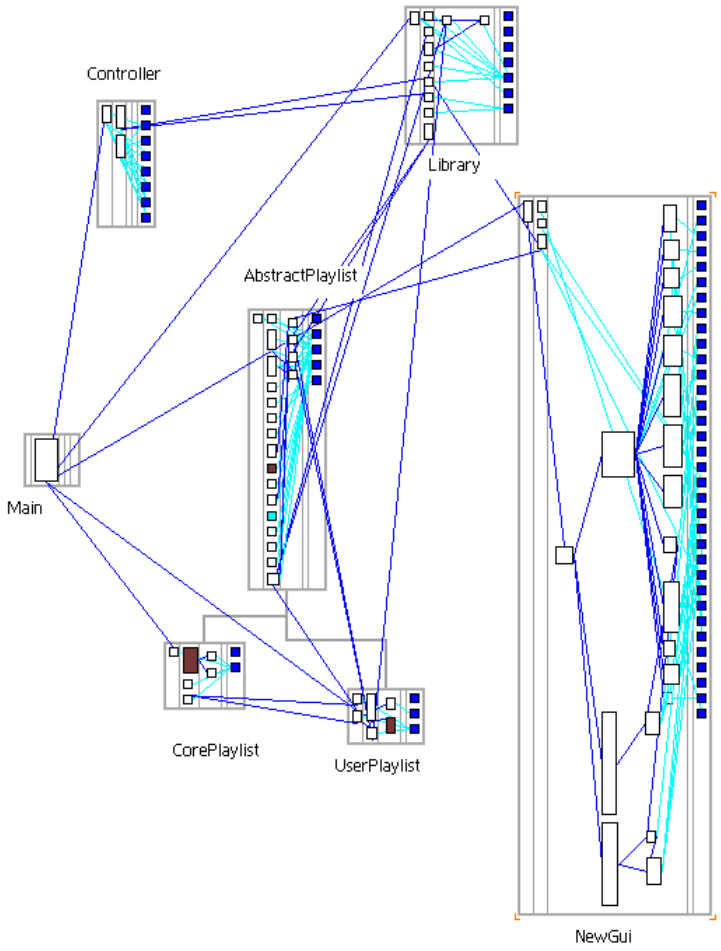
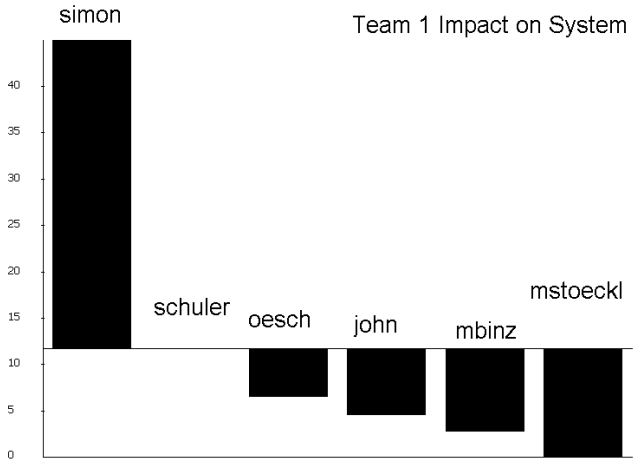


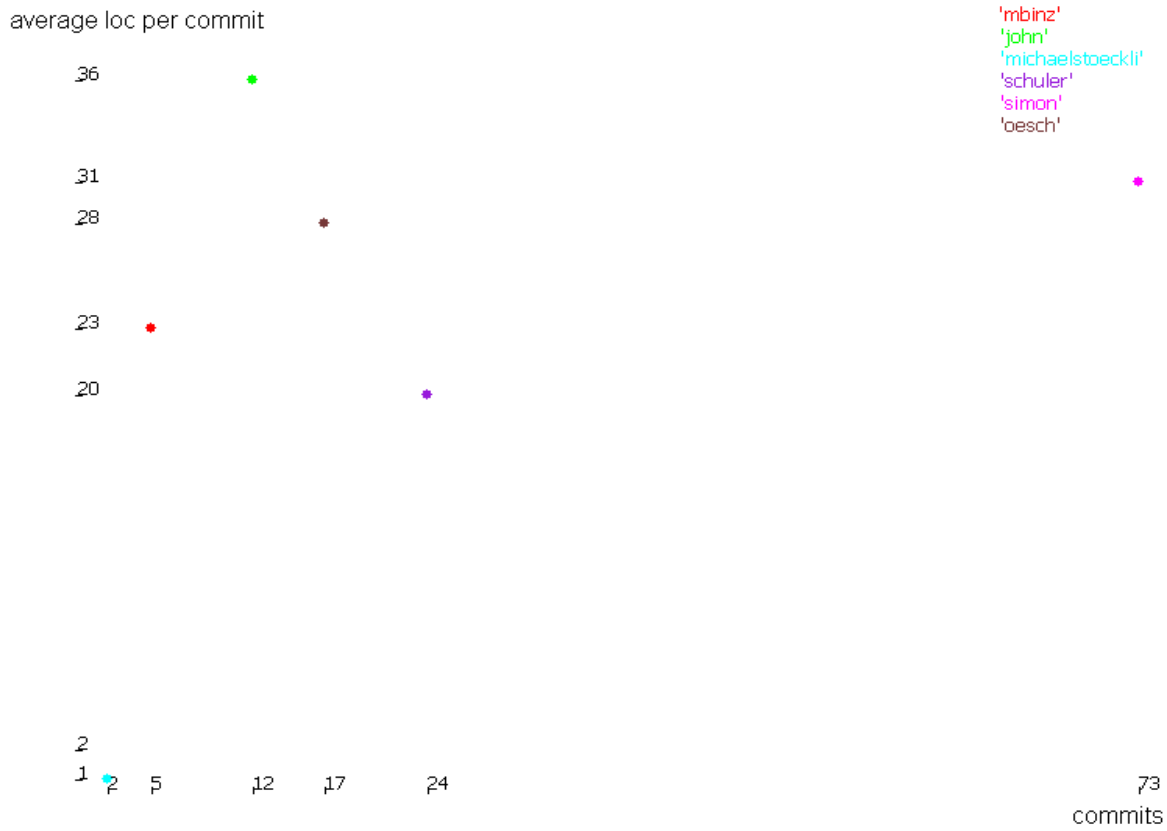
Abbildung 2: Overview Team 5

The system has about 4000 LOC.
Score 1 Design + 0.5 CodeConventions.

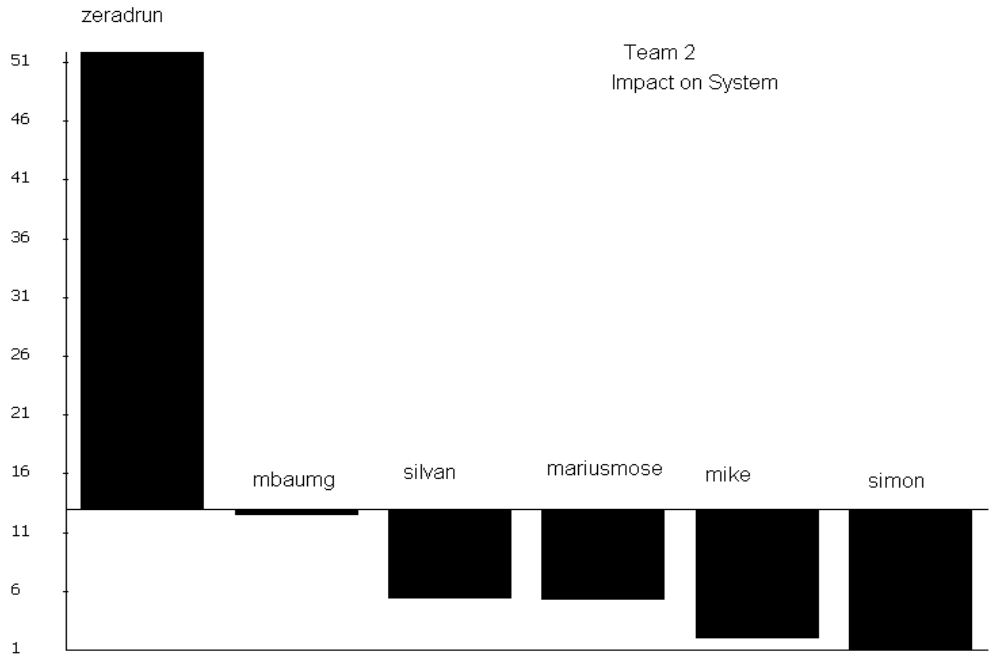
Participation

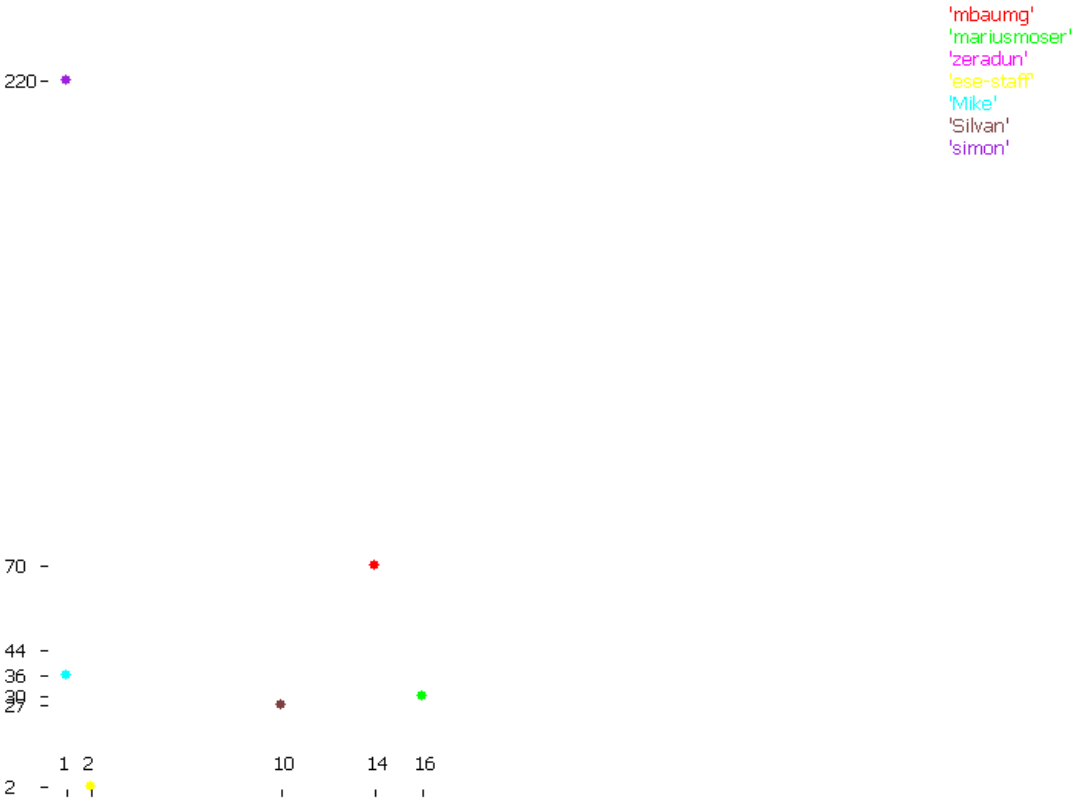
Team 1



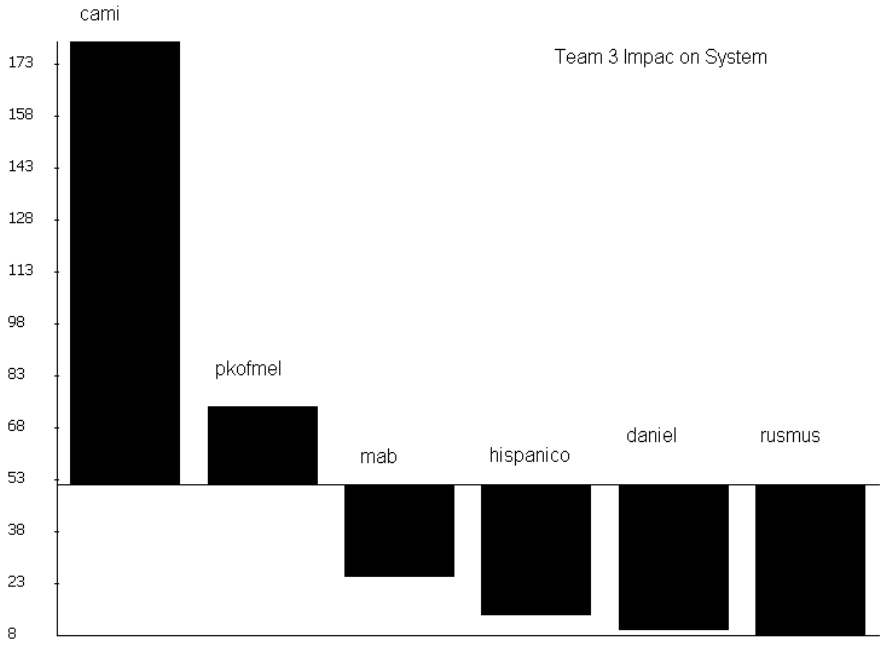


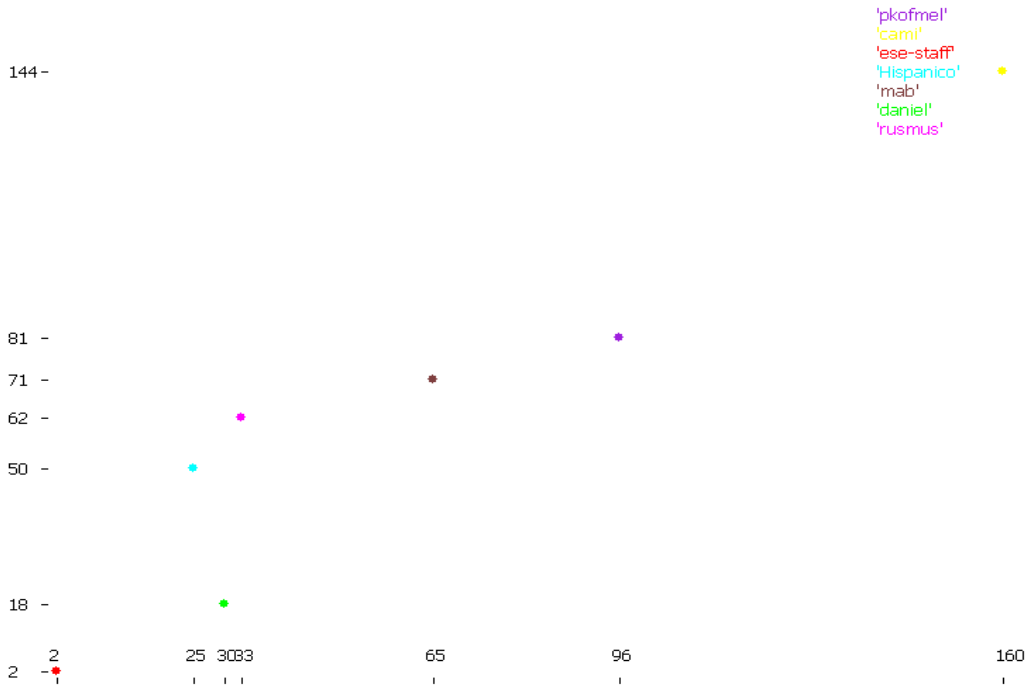
Team 2


zeradrun
Team 2
Impact on System
51
46
41
36
31
26
21
16
11
6
1
mbaumg
silvan
mariusmose
mike
simon

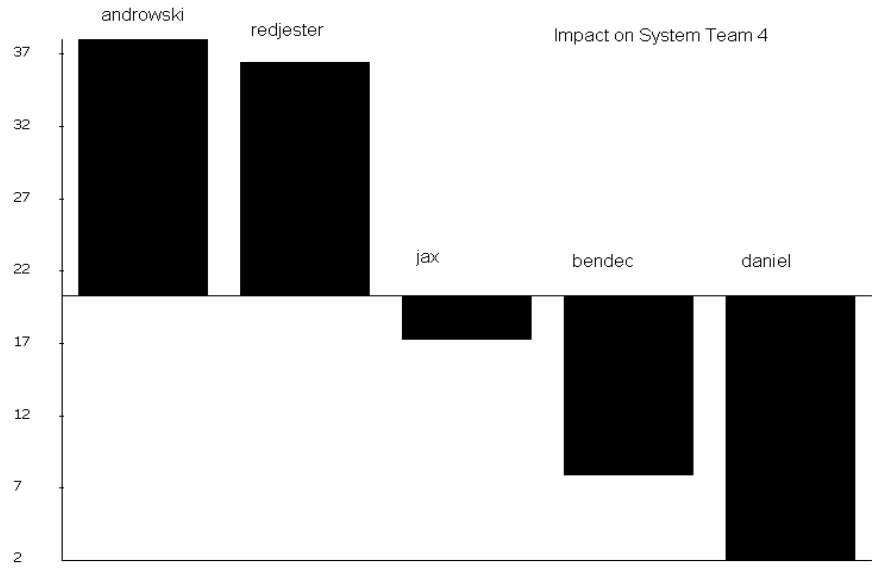

'mbaumg'
'mariusmoser'
'zeradun'
'ese-staff'
'Mike'
'Silvan'
'simon'
220
70
44
36
30
27
2
1
2
10
14
16
44

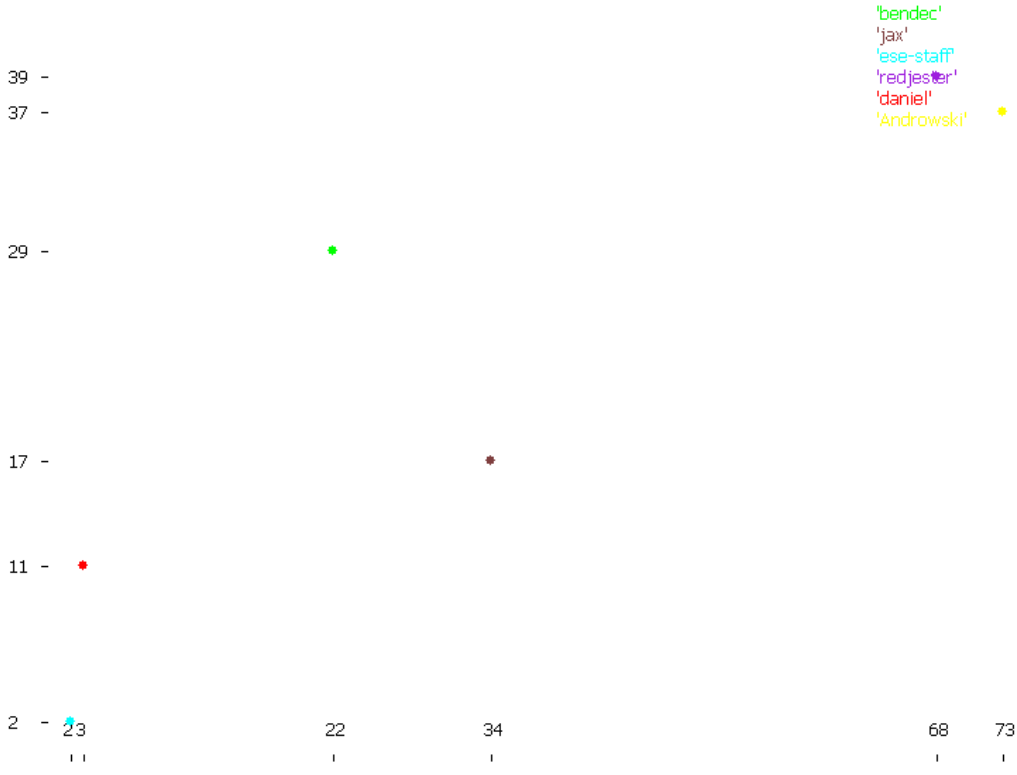
Team 3


cami
Team 3 Impac on System
173
158
143
128
113
98
83
68
53
38
23
8
pkofmel
mab
hispanico
daniel
rusmus

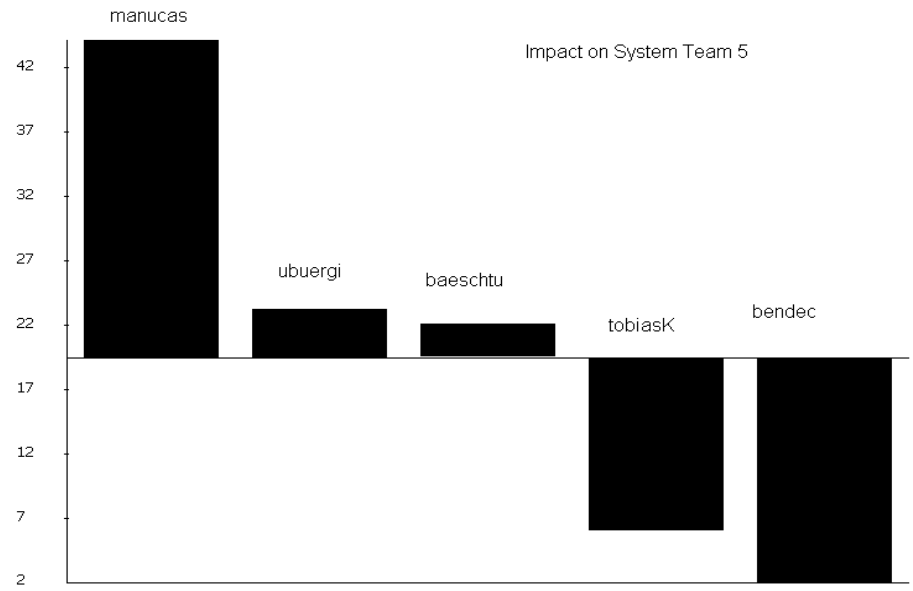

'pkofmel'
'cami'
'ese-staff'
'Hispanico'
'mab'
'daniel'
'rusmus'
144
81
71
62
50
18
2
2
25
30
33
65
96
160

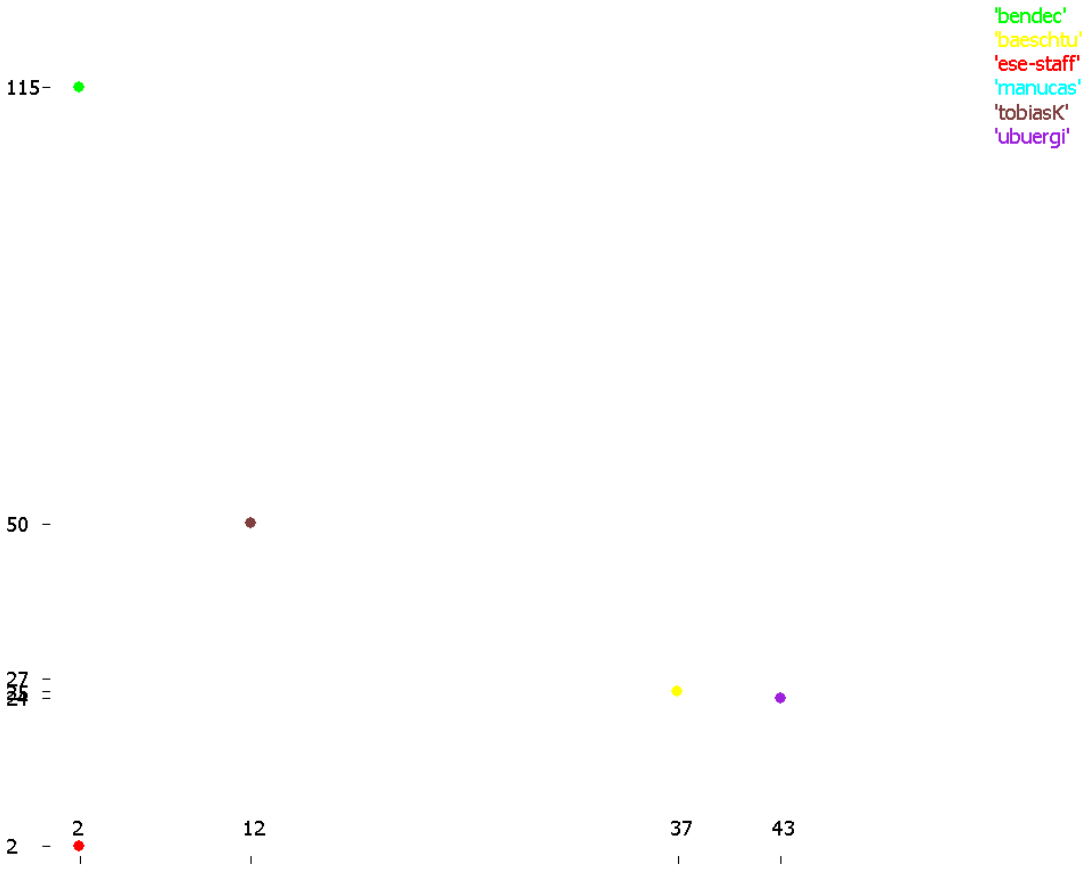
Team 4


androwski
redjester
Impact on System Team 4
37
32
27
22
17
12
7
2
jax
bendec
daniel


'bendec'
'jax'
'ese-staff'
'redjester'
'daniel'
'Androwski'
39
37
29
17
11
2
23
22
34
68
73

Team 5


manucas
Impact on System Team 5
42
37
32
27
22
17
12
7
2
ubuergi
baeschtu
tobiasK
bendec


'bendec'
'baeschtu'
'ese-staff'
'manucas'
'tobiasK'
'ubuergi'
115
50
27
24
2
2
12
37
43
64

Total Grades:

Team 1: 3.75

Team 2: 2.5

Team 3: 4.5

Team 4: 3.5

Team 5: 2.75