Team <Scentaur>

Ga Jun Young	(ga.young@ucdconnect.ie)
Royal Thomas	(royal.thomas@ucdconnect.ie)
Zheng Ju	(zheng.ju@ucdconnect.ie)
William Ikenna-Nwosu	(William.ikenna-nwosu@ucdconnect.ie)

SCENTAUR

1. Introduction

1.1 Vision of the Project

Scentaur is a Java-specific “code smell” detector. The team behind Scentaur believes that many current code smell detectors are not particularly easy to use and understand (i.e. JDeodrant and Infusion). Thus, the team envisions Scentaur to be a user-friendly code smell detector tool. We want Scentaur to be accessible, performant, have clear data visualization and finally, to be reliable.

1.2 What Scentaur Set Out to Achieve

- **Web-based:** Team Scentaur has set out to achieve a web-based code smell detector tool. Providing a quick and easy way to detect smells with a simple quick drag and drop functionality. The web-based application brings forth a simple UI design and easy accessibility to reach a wide range of audience in hopes of detecting smells within their code.
- **Performant:** Scentaur had set out to reduce the amount of time required to detect smells unlike JDeodrant which takes a significant amount of time to detect smells; as it contains many types of detectors (22) along with the ability to refractor smelly code. As a result, Scentaur is currently providing up-to 4 sub smells for each smell category. This proves to increase performance and in addition, users are enabled to check smells they want to detect within their code.
- **Go-To Product:** When Scentaur first started, the team believed that in order to encourage an audience to code, we should provide a software to help beginners to think about the software design phase of development. To give them an understanding an appreciation for writing maintainable code. To allow them to think about code on a higher level so they can make an immediate impact when working in teams together. Scentaur has achieved this capability by providing users useful tool tips on the smells involved. After analysis Scentaur provides a general but impactful summary to users on the code at hand. This includes smell definitions, colored pieces of code that smells along with its line number, smell category and file location.

➢ **Reliability:** Testing can give one confidence that the code is **functioning** correctly, Source control gives one confidence that all team members are working on the same version of the code (**Communication** is up to date). Scentaur give users confidence that the system's (software) **design** is adaptable to change and if a change is made to the system that breaks functionality, it will be caught.

1.3 Main Goals Achieved

Team Scentaur has achieved several main goals. Here are the following:

1. To learn and understand the different types of code smells.
2. To understand and take responsibility in a larger team.
3. To enable Scentaur for future uses outside of the assignment background.
4. To enable easy implementations and execution of other code smells in the future.

1.4 Core Components and Bonuses

Throughout the project lifetime, Scentaur has become an amazing web-based application. Users are enabled to drag and drop repositories and Scentaur will respond by providing any detailed smells detected within a matter of seconds. This is all thanks to the following core components of Scentaur. The following is a list of core components:

1. **JavaParser:** Without JavaParser Scentaur would be incapable of sniffing Java files in a matter of seconds. The capabilities of JavaParser is immense, from parsing files to creating Abstract Syntax Tree to detecting smells. JavaParser is a huge component of Scentaur, where it is the one that helps provide the smells we wish to detect.
2. **HTML, CSS, JavaScript:** These front-end languages allowed Scentaur to represent itself visually to users by providing a web page. JavaScript was used in order to have functionality like drag and drop to upload a file. With CSS our page looked a lot more interesting and engaging to the audience. This helps to visually communicate to our users.
3. **Tomcat:** Tomcat plays the role of the server for Scentaur. The two components above are not able to complete any task individually. Thus,

Tomcat serves as a link between the two (links Java to HTML). Not only that Tomcat allows users to store their temporary files within a server. These files can only be accessed by the user during their session on the webpage. That is because each user is given a unique session ID and granted only permission to contents, they have stored to the server. As a result, Scentaur is a secure web-based application.

4. **Bonuses:** In addition to the above core components, Scentaur contains many plugin-and-play modules. New smells can be easily added into the Software without causing any problems or give dependency issues. We have also used interfaces to follow up the plugin-and-play module whilst using Generics. Statistics is another unique feature that Scentaur uses to give users a more comprehensive detail of the figures involved with the smells detected.

1.5 Unique Selling Point

Scentaur strives for uniqueness. The software overall uses JavaParser which greatly helps reduce the amount of coding required to parse and explore classes to find smells. However, that is not the main uniqueness about this program. Its Unique Selling Point is the fact that Scentaur can be run on a web browser available to users who do not wish to download and install a plugin for their favorite IDE. Scentaur overall has many capabilities one such is the fact that it can have a side by side comparison where one side contains the original java file and the other containing pieces of code that contains the code smell. Scentaur also approaches users by providing them with a simple User Interface with some interesting design choices. This allows for easy navigation and use of the web application.

2. System Features

2.1 Features Implemented

- **Smells:** The main feature of Scentaur. These small sniffers enabled Scentaur to collectively smell all sorts of warnings that a Java file may have. Users are enabled to detect from 12 smells through subcategories bloater, abuser, coupler and dispensable. These categories are then placed into HashMaps where we enabled focused class smells. i.e. smell only a specific class of the user's choice.

- **Report:** The report is essentially a compilation of all the smells detected. It incorporates statistics along with it enabling various calculations on data to check the most common smell etc. The report is essential in helping the servlet to connect with the Java back-end.
- **Web-Based:** Scentaur is a web-based application that uses Tomcat as the server and incorporates Java, CSS, HTML, and JavaScript to create a cohesive program. This feature was very important to Scentaur as it allowed users all over the world to explore Scentaur at an ease of access.

2.2 Not Implemented Features & Planned Features

- **Some Smells:** Our plug and play feature are a really handy aspect of Scentaur however, allowing for plug and play made it so that it was impossible to incorporate smells such as Refused Bequest which required knowledge of other classes.
- **Charts:** It is often very handy to display data in a visual aspect, however with the knowledge of servlet. We were unable to implement charts properly onto our webserver. However, we do have JavaScript files that are based on different charts.
- **Refracturing:** This feature would have been ideal if the team had a bit more time. We had planned to incorporate refactoring into our program and such methods like addComment is a steppingstone into this feature. However, this was scrapped due to the lack of available time.

2.3 Distinguish Your Project

The most distinguishable part of Scentaur is the fact that it is online and ready to use without having to download a plugin for your IDE. Users can access the web application and deposit their repository whilst having Scentaur detect any smells that might be present in the repository. The ease of access greatly entices users to use Scentaur along with how easy it is to detect smell within users' files. A user would simply have to type up Scentaur's URL, zip up their repository, upload the zip to Scentaur and in matter of seconds Scentaur would produce an overview of the smells detected. Afterwards Scentaur allows the user to check smells in their individual files, if they wish to do so. Thus, further allowing users to pinpoint what smells are current in a specific class.

3. Project Design

3.1 Overall Model

Scentaur is a web-based application that analyses user's code (Java) for code smells. It enables the user to upload their entire code base as a zip file which will then be analyzed by the system. Scentaur guarantees easy accessibility, as it is a web-based application that solely requires a standard web browser. Unlike standalone apps, the only requirement for a user is a web browser and a connection to the internet to access the application. The application runs java on the backend and uses HTML5, CSS3 and JavaScript on the front end. We use Java Servlets using .jsp files to execute Java code in the backend while at the same time providing user with their HTML, CSS and JavaScript code to the browser.

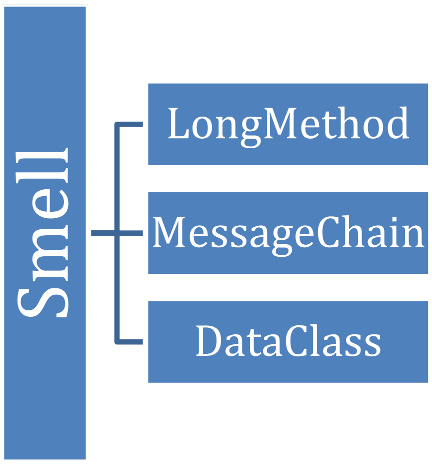
For the front end, we made use of various JavaScript libraries to perform various functions. We used ajax to lazy load data in order to make the web page seem smoother to the front end user. We utilized highlight.js (https://highlightjs.org/), a JavaScript based syntax highlighting tool which we used to display the java source codes for each file. We also utilized (highlightjs-line-numbers.js) in order to insert line numbers when displaying the code onto the user's browser. The design of the web page was kept minimal and dark in order to better suit developers. We wanted most of the page to be concentrated on what matters rather than random objects and shadows.

In terms of the backend, we were planning on doing spring boot as an MVC. While we did make a slight progress with Spring Boot, we decided that it would be easier and much simpler if we just stuck to JSP and Java Servlets.

3.2 Smell Model

At the beginning of Scentaur's build up, the team had envisioned to build Smells as an abstract class with subclasses (Abuser, Bloater, Coupler and Dispensable). This plan was in place because we felt that each concrete smell would have some methods that belong to all smells and then they would fall into their respective sub-categories (Abuser, Bloater, Coupler and Dispensable). i.e. Long Method would have a method from Smells and a unique method only for bloaters.

However, as time progressed, and smells were being built the design changed.



e.g. Smell Hierarchy Example

All the concrete smells did not require any unique methods in their subcategories meaning that the sub-classes abuser, bloater, coupler and dispensable was left empty and redundant. Thus, the team decided to cut off these dependencies and just have each concrete smell extend from Smell class. This seems rather messy as we would have many classes a sub class of smell (Abstract class) therefore, we packaged relevant smells to their respective smell package category e.g. Long Method Class would be within the Bloater package.

Furthermore, we were enabled to use polymorphism to encapsulate the instantiated class i.e. Smell longMethod = new LongMethod();

Which is a really handy feature as it allowed the team to make a list of smells and use this information in the report. Another great thing about this Smell class is that it is an extension of JavaParser's VoidVisitor Class which enables subclasses of smell to visit all of the class' specific features e.g. visit all methods within a class. Furthermore, Smell holds a list of node data. This is because JavaParser creates an abstract syntax tree for each class and the nodes are essentially the components of that class. Thus, if I wanted to store only Methods then I would find the method nodes of that class.

Since we removed the dependency of Abuser, Bloater, Coupler and Dispensable, the team decided to utilize these classes in another aspect. Each of these class is itself a plug-and-play system different to those of the concrete smells. They enable us to instantiate smells based on their category and place them into a HashMap for use on the server. The data that the HashMap holds is very important as it enables us to do a lot of unique things. Each HashMap holds a key to the class file name followed by the value – another HashMap with the key subcategory Smell (e.g. Abusers.DATA_HIDING) and the value which are the smells (In this case only Data Hiding smells).

3.3 Third Party

Scentaur was not able to achieve the state it is in now without the help of third-party libraries and components. Scentaur uses the Gradle plugin which is available on the Eclipse Marketplace. This is a free plugin that allows the team to import external libraries easily. Simply adding additional dependencies to the build.gradle file will import external libraries for Scentaur to use.

Scentaur uses many different libraries to become a Code Smell Detector Tool. The following is a list of libraries and their usage for this project.

1. **JavaParser:** This external library enabled Scentaur to easily parse Java files into an Abstract Syntax Tree which is then used to detect smells through a visitor class.
2. **JSON:** Support the communication between front-end and back-end. Response with multiple parameters will be stored into an array and then parsed to a json-String and then be returned to the front-end. Ajax at the front-end can use the json-String as an array directly.
3. **Servlet:** The core of the server-end. It receives the request sent from the front-end, analyzes the request then works with different functionality to respond to the user. It controls uploading and decompressing files, and also instructs when the core part of Scentaur should run and when the report should be generated. These servlets interact with user during the whole session.

3.4 Proud Aspect

The most rewarding aspect of Scentaur is its ability to project the software onto the web page. It helped us to learn a lot about web development using Java. While members on our team have had opportunities to work on the backend with languages such as Ruby and PHP, using Java to create the backend was a new challenge and we are happy to have taken that challenge.

Another proud aspect is the usage of different Java Features such as polymorphism and generics. At the start in building Scentaur we never really investigated these features and expected to use them. But as a result of developing the smells and appropriately calling them in Java, it was almost a must to use these two Java features. The generics part helped in creating a plug-and-play system for "Bloater, Abuser, Coupler and Dispensable" class and it allows us to add more categories in the future if we wished. Having the ability of reusability within these classes gives us a great advantage if we ever wish to look at these classes again. Changing some of its components will highly unlikely break the system which means that these classes have low coupling and high cohesion.

3.5 Benefits of Further Development

1. With further development of Scentaur we wish to implement some of the features that was not implemented – listed above. This would require more time and effort to research in order to come up with a solid solution e.g. We might have to re-imagine how Smells should be designed in order to accompany smells like "Refused Bequest" which requires information on another class.

2. Another benefit involves user experience. Currently our user interface displays the original code from the user (Similar to online compilers) versus pieces of code from the original code that contains smells (Similar to how HTML validators display their errors). However, an ideal development would to provide a perfect side to side comparison of the code before and after. The after would then indicate lines of code with

smells using colors. This would greatly benefit users visual experience and the ability to find and refactor their own code themselves.

3. We would have loved for the website to have more functionalities such as storage of reports for future references, uploading of multiple projects at once. On the backend side, with more work a lot of the smells that have been listed as "Possible Code Smell" could have been moved up to "Likely Code Smell" using more constraints and checks.

4. Successes and Failures

Everybody pulled their weight and wanted to contribute significantly to the project. We were all committed to our common goal and we accomplished it along with our objectives. We supported one another and co-operated as a cohesive unit; communicating our ideas effectively to each other; be it face-to-face or over the multiple communication channels we had at our disposal. Our effective communication and easy-going interpersonal relations especially came to the fore when we integrated the different components of the application together to form the final software product. Scentaur. The project was a success overall and a very enlightening learning experience. I would use Scentaur to smell my code any day.

4.1 What Problems Arose and How Did You Deal With Them?

1. Problem: Creating the hierarchy of smells
   How we dealt with problem
   - We referenced the taxonomy we mentioned in the interim report to guide our smell packaging hierarchy layout. We created interfaces for each type of smell (Bloater, Coupler, OOPAbuser, Dispensable) initially but after adding some features and implementing smells; we realised that they were not necessary for our design.
   - We made an abstract class called Smell that is inherited from by all concrete smells. This allowed us to reference the concrete smells in a more general way, giving us access to more powerful features. This is because once the smells have been detected by the concrete

classes, no clients of the concrete classes will need to know the implementation details of them. They will only seek what the data processed by the concrete classes refer to. They will not need the concrete knowledge of the class to solve their problems. Clients of the concrete classes will only need higher level abstract knowledge of the concrete classes e.g. clients don't want to know how to calculate longMethod, they just want to know which files the longMethod smell was detected in, or, may want a string of that information. This is making sure that clients only know enough about concrete classes to solve their problems. Low coupling. https://www.geeksforgeeks.org/referencing-subclass-objects-subclass-vs-superclass-reference/
[Really exploited this feature of the Java Programming language in our design]

Ex.
Smell x = new LongMethod()
LongMethod x = new LongMethod() /* We would only be able to define smells like this without the abstract class.*/

- This Smell class also contained enums grouping categories of smells together. e.g. Abusers{DATAHIDING, SWITCHSTATEMENT}. This allows the server side to quickly and easily retrieve data it seeks. It also allows us to group smells together to make calculations which aggregate smells less cumbersome to compute. i.e. it reduced the amount of code we had to write to get something done.
- We used packages to group types of smells(Bloater, Coupler, OOPAbuser, Dispensable) together and in each package we had a Superclass denoting the type of smell (Bloater, Coupler, OOPAbuser, Dispensable). Each instance of the type of smell inherited from the superclass. e.g. DataHiding is an OOPAbuser, LongMethods is a Bloater. Each superclass in each package

contained methods that would be a common necessity across its children e.g. retrieving all file names where the concrete smell was detected, retrieving list of concrete smells of package e.g. list of bloater type smells.

2. Problem: Deciding how users should upload code to Scentaur

   How we dealt with problem: We decided that users should upload their source code in a zip file. This is how we wanted users to interface with our product. We felt that it would require the least amount of effort on the user's side.
3. Problem: Deciding how to compute smell detection on server

   How we dealt with problem: We had two options; (1) Compute smell detection for all smells regardless of whether user selected smell or not then for those smells selected by user, give them the computed results. (2) Compute smell detection for smells user selected. Then on update (user selects different smells) recompute. We decided to opt for (1) because it allowed us to do one expensive computation then reap the benefits by giving the user their requested smells (amortized). (2) would have required us to compute at run time and added loading delay since the server would have to send the results to display to the user. On an amortized analysis, (1) was the better choice [compute once expensive operation. Every time user updates selections, we don't have to re-compute. Would have to do so otherwise]

4.2 How Would You Do Things Differently In The Future, or If You Had To Do It All Over Again?

1. Spend more time researching code smells.
2. It took us a while to get into the groove of the project because the initial design phase was new to us. If we had more information and suggestions about the overall design of the project, we would have been able to implement features earlier.
3. Assigning one or two people to the server and website, while the remaining others work on the core application software from the very start.

4. Have everyone understand the necessary libraries for the project. Have a session where we discuss and educate each other on a library so we don't have to revisit this topic.

4.3 Who Did What in The End?

William Ikenna-Nwosu: Implemented smells and worked on Statistics class

Ga Jun Young: Implemented smells, abstract smell class, generic sub categories, report, parser and detector classes.

Royal Thomas: Implemented smells, worked on Front-end, Parser.java, unit tests

Jackie Ju: Implemented smells, worked on Server side back-end integration

4.4 Who Deserves Special Mention For Going Above and Beyond The Call of Duty?

Jackie Ju: Server-side functionality [Honourable mention: introducing interfaces with inheritance]

Ga Jun Young: Lead by example, compilation units, design decision [referencing subclass objects through superclass]

Royal Thomas: JavaParser, compilation units, drag-and-drop

William Ikenna-Nwosu: Identifying when note-worthy design decisions were being made for the report.

5. Team Communication

Team communication in general went much better than expected. Meetings were regularly held, and plans were made. We had group meetings and calls on platforms such as discord and messenger where we discussed our plans and ideas, and critiques were well welcomed. These were crucial in ensuring that the team works as a unit instead of each person doing their own thing. Each person was assigned specific tasks for them to do before the next meeting. For our team, this method worked very well.

However, like all other teams, we have had our off weeks where due to the accumulation of work and other engagements, we were not able to do as well as we should have had. In terms of ‘tools of software development’ we were able to make proper use of a few softwares including Slack, Github, Discord and Messenger.

5.1 Slack

Slack was used as an official mode for communication and a place to store ideas, suggestions and report various issues. We created various channels for various purposes so that everything wasn’t bundled up into one single channel making it difficult for us in the future to find information we could be looking for. We integrated GitHub onto each of our Slack apps so that we received constant updates and notifications for all changes that occurred on GitHub. This was not limited to commits but also included various other things such as changes on the project board and issues raised. We were able to use @channel and @here to send notifications to all members to raise issues and questions and ensure swift communication.

5.2 Discord

Discord on the other hand was primarily just used for making calls as it was something everyone in the group was familiar with. Like Slack, we were able to make channels on discord to have smaller calls and discussions when we divided into teams to tackle specific tasks. Discord being available on all devices was a godsent! The calls on discord were generally 60 to 90 minutes long and were always fruitful and professional.

5.3 GitHub

GitHub was our primary system for version control. Being well versed with git, using GitHub is and has always been a pleasure. Along with the version control system, the project board enabled us to organize tasks in a much simpler way. For each sprint, we were able to assign users their tasks and then let each task be placed from ‘to-do’ to ‘in progress’ to ‘done’. The issues section on GitHub allowed us to report bugs and suggest future changes that would help better the project.

5.4 Messenger

Messenger on the other hand was used for general discussions along with other off topic discussions. Things were not kept entirely official in messenger and was used as an area where discussions took place for things not covered during team meetings. Messenger also allowed us to set up and organize the next dates for meetings and team calls.

5.5 Communication Conclusion

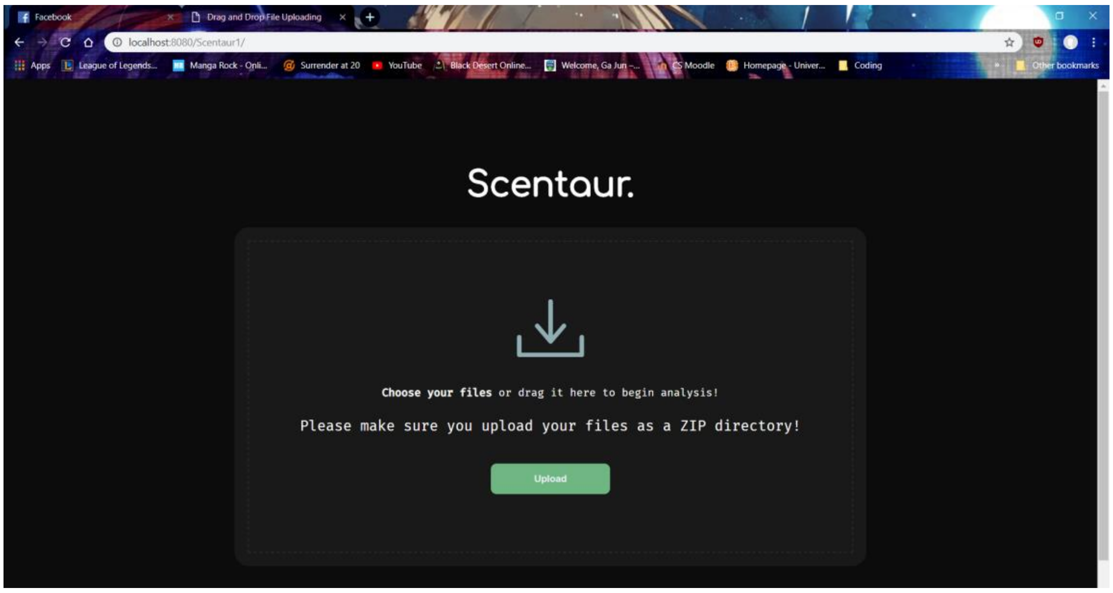
Finally, the most important of all were the face to face meetings done during the semester. In 1920, social psychologist Floyd Allport found that “people worked better in teams even if they weren’t collaborating, competing, or actively communicating with each other”. This proved true for us too, working together as a team produced better productivity rates than when people decided to tackle tasks in a solo manner. These, being more personal, facilitated better communication among the team. Being able to be in a room and discuss what we were planning on doing while doing it helped prevent a lot of issues that might have risen if we would have decided only to discuss these later.

While yes, discussions and meetings on topic were very important, a good practice that a well-functioning team must have is scheduling time for virtual camaraderie building, including chatting in an informal context. Researchers at MIT’s Human Dynamics Laboratory have found conversations outside of formal meetings are the most important factor that contributes to team success. On various occasions, the team went together for lunch and coffee breaks where the discussions were kept off topic to encourage team building camaraderie.

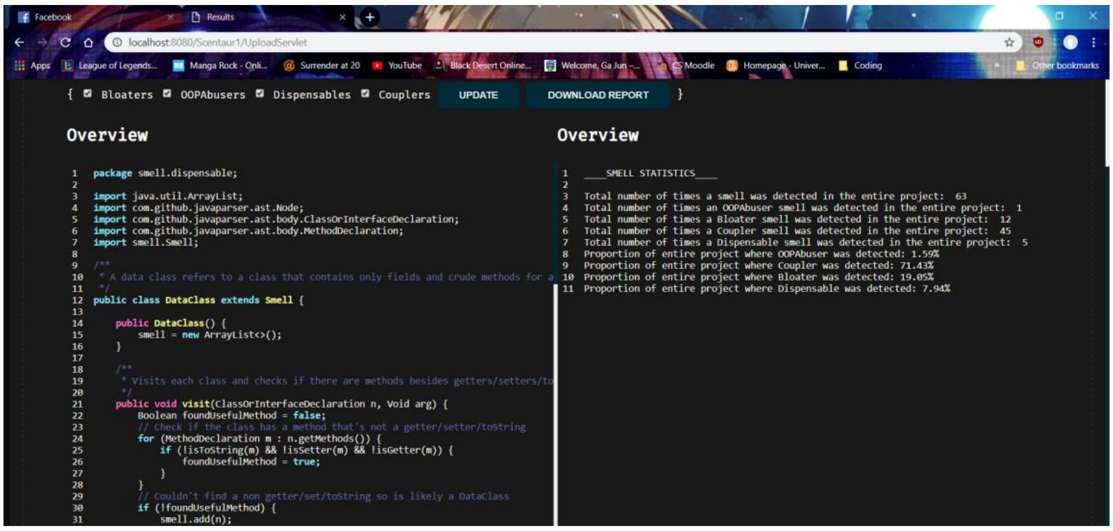
6. Your Project In Detail

Applying Scentaur’s detection tool onto itself provides a great sample for the team to understand the problems Scentaur may experience as a software. Along with the ability to help team members to strengthen their coding style especially in Java. The following is a worked example of Scentaur:

After the Tomcat server is switched on, it is possible to host the web server on our local machine. To obtain a more impressive view of the web page, it is ideal to run localhost on your selective browser. In our case, we will be running on Google Chrome. When you first open the page, the user is met with a simplistic, modern, and minimal style landing page. Where the user is requested to send a zip folder to the server either through "Drag-and-Drop" or click on the link and search the zip file from one of your own directories.


localhost:8080/Scentaur1/
Scentaur.
Choose your files or drag it here to begin analysis!
Please make sure you upload your files as a ZIP directory!
Upload

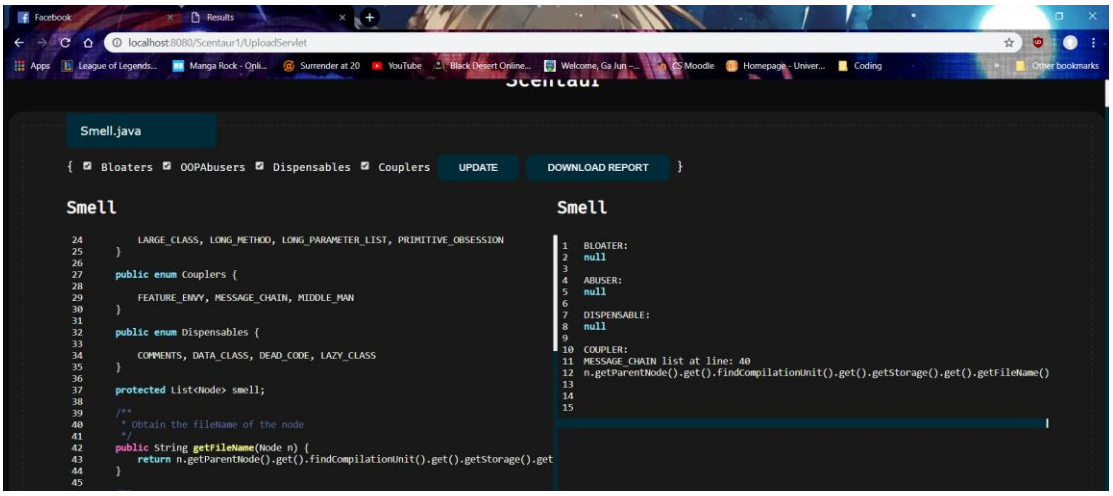
Once the user has drag and dropped, the user then must click on the update button to request the next action. This action will send the user to the result page. The result page contains many information about the user's zip file. The backend handles the zip file by decompressing the information inside and takes out all the relevant Java files. This is clearly visible on the left-hand side with displays the original files that the user zipped up for Scentaur.


localhost:8080/Scentaur1/UploadServlet
{ Bloaters OOPAbusers Dispensables Couplers UPDATE DOWNLOAD REPORT }
Overview
Overview

```java
package smell.dispensable;

import java.util.ArrayList;
import com.github.javaparser.ast.Node;
import com.github.javaparser.ast.body.ClassOrInterfaceDeclaration;
import com.github.javaparser.ast.body.MethodDeclaration;
import smell.Smell;

/**
 * A data class refers to a class that contains only fields and crude methods for a
 */
public class DataClass extends Smell {

    public DataClass() {
        smell = new ArrayList<>();
    }

    /**
     * Visits each class and checks if there are methods besides getters/setters/to
     */
    public void visit(ClassOrInterfaceDeclaration n, Void arg) {
        Boolean foundUsefulMethod = false;
        // Check if the class has a method that's not a getter/setter/toString
        for (MethodDeclaration m : n.getMethods()) {
            if (!isToString(m) && !isSetter(m) && !isGetter(m)) {
                foundUsefulMethod = true;
            }
        }
        // Couldn't find a non getter/set/toString so is likely a DataClass
        if (!foundUsefulMethod) {
            smell.add(n);
```

```
____SMELL STATISTICS____

Total number of times a smell was detected in the entire project:  63
Total number of times an OOPAbuser smell was detected in the entire project:  1
Total number of times a Bloater smell was detected in the entire project:  12
Total number of times a Coupler smell was detected in the entire project:  45
Total number of times a Dispensable smell was detected in the entire project:  5
Proportion of entire project where OOPAbuser was detected: 1.59%
Proportion of entire project where Coupler was detected: 71.43%
Proportion of entire project where Bloater was detected: 19.05%
Proportion of entire project where Dispensable was detected: 7.94%
```

On the right-hand side, the statistics of the overall zip project is present. It is not visually appealing however; it nails down numbers straight to the users. Along with this text of statistics, we have back-end JavaScript charts. Unfortunately, we are unable to display them properly without introducing bugs into the program. But these charts would represent the different smells within the project.



```
24         LARGE_CLASS, LONG_METHOD, LONG_PARAMETER_LIST, PRIMITIVE_OBSESSION
25     }
26
27     public enum Couplers {
28
29         FEATURE_ENVY, MESSAGE_CHAIN, MIDDLE_MAN
30     }
31
32     public enum Dispensables {
33
34         COMMENTS, DATA_CLASS, DEAD_CODE, LAZY_CLASS
35     }
36
37     protected List<Node> smell;
38
39     /**
40      * Obtain the fileName of the node
41      */
42     public String getFileName(Node n) {
43         return n.getParentNode().get().findCompilationUnit().get().getStorage().get
44     }
45
```

```
1   BLOATER:
2   null
3
4   ABUSER:
5   null
6
7   DISPENSABLE:
8   null
9
10  COUPLER:
11  MESSAGE_CHAIN list at line: 40
12  n.getParentNode().get().findCompilationUnit().get().getStorage().get().getFileName()
13
14
15
```

Finally, the user is then enabled to check the type of smell for each class using a selection of check boxes and a tab list of classes available. In this sample below, I choose to check the Smell.java class and checked the boxes for bloaters, abusers, couplers and dependables. Immediately on the right provides snippets of code where the smell occurred. Each snippet is accompanied by the file name, smell type and the line at which it was detected. We use this to map the smell to the line number of the original file. Scentaur enables us to do this for infinite numbers of time searching through multiple combinations of smells. *Note: the left-hand side does not accurately portray the original file as it was beautified. Meaning the line numbers on the left may be slightly inaccurate.*

<u>6.1 What Does your System Say About Your Code?</u>

Having looked at the statistic overview of Scentaur. It has detected a total of 63 smells and 71% of them being couplers and the least amount being OOP abusers. We believe this is accurate as a lot of the concepts within Scentaur requires links between classes and as a result, couplers are fairly high. We see that there is only 1 such OOP abuser which shows that the team members are coding in a fashion that puts abusers to a minimal. Which in fairness is a good sign.

Code smells are smells but they are not always a bad sign, these coupling might be a necessity for this project for it to work otherwise it wouldn't have been possible to carry out some functionalities.

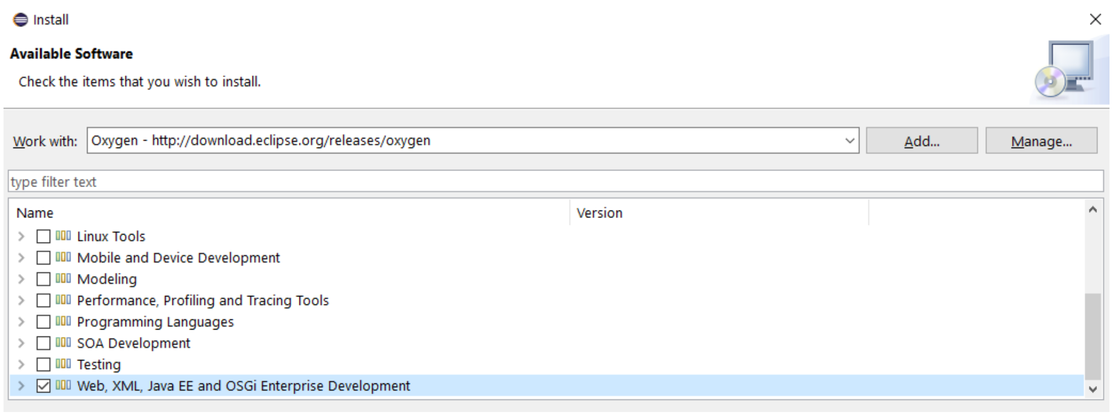
We believe that the analysis is fair, as we expected the results as such to occur. We expected couplers to be fairly high around 50% above compared to the rest.

In running this overall software, we noticed that Dead Code was not producing the right answers as it made Dispensables obtain a rather high value. We intended to fix this problem however, due to the lack of time this was not possible and therefore we omitted Dead Code from the list of dispensables which then reduced our overall inaccurate results.

6.2 Setup

A. Using Eclipse [Hard Method]

a. Download the following plugins
   i. Buildship Gradle Integration 3.0
   ii. Eclipse Java EE Developer Tools 3.12
   iii. Eclipse JST Server Adapters (Apache Tomcat, Jon…
b. Make sure that Eclipse can run Java Web projects. If not go to:
   i. Help -> Install New Software -> Select a site and download



Install

Available Software

Check the items that you wish to install.

Work with: Oxygen - http://download.eclipse.org/releases/oxygen Add... Manage...

type filter text

Name	Version
☐ Linux Tools	
☐ Mobile and Device Development	
☐ Modeling	
☐ Performance, Profiling and Tracing Tools	
☐ Programming Languages	
☐ SOA Development	
☐ Testing	
☑ Web, XML, Java EE and OSGi Enterprise Development	

c. On the web browser – Download Tomcat 8.0
   i. Install it onto your Java JRE folder
   ii. If you can't see Tomcat on your server
      1. Window preferences on Eclipse -> Search Tomcat -> config the server -> select version 8.5 -> server location, other fields will be automatically filled
d. Importing Scentaur
   i. File -> Import -> Gradle -> Existing Gradle Project -> Scentaur

        ii. Accept Everything

e. Right Click on Scentaur -> Gradle -> Refresh Gradle
f. Right Click Scentaur -> Properties -> Java Build Path -> Configure Build Path -> Libraries -> Add JARs -> Scentaur -> WebContent -> WEB-INF -> lib -> Everything -> Apply
g. Right Click Scentaur -> Properties -> Deployment Assembly -> Add -> Java Build Path Entries -> Select Everything -> Finish -> Apply
h. Right Click Scentaur -> Properties -> Project Facets (Make sure dynamic Web module is ticked and Tomcat v8.5 is applied in Runtimes
i. Right Click Scentaur -> Run as -> Run on Server, Select Tomcat v8.5 Server at localhost -> Finish

Note: Refreshing Gradle will remove all external dependencies in the build path and on the server. Make sure you configure the build path again (Step f to g) each time you refresh Gradle

B. Easier Method

1) Install this : https://www-us.apache.org/dist/tomcat/tomcat-8/v8.5.40/bin/apache-tomcat-8.5.40.exe
2) - Open installation directory of it and then open webapps, for me it was
- C:\Program Files\Apache Software Foundation\Tomcat 8.5\webapps
3) Put Scentaur.war file in there
4) Open bin folder inside tomcat installation directory (Tomcat 8.5) (
C:\Program Files\Apache Software Foundation\Tomcat 8.5\ )
and open
'Tomcat8w'
5) Press on start, make sure port 8080 is open else it won't start!
6) Visit : http://localhost:8080/Scentaur/

Acknowledgements

The Scentaur report was an equally contributed effort. A breakdown is as follows:

Name [Contribution %]	Responsibilities
Ga Jun Young [25%]	Smells, Categories, Report, Final Report (Intro, Feature, Project In Detail, Project Design, Smell Model)
Royal Thomas [25%]	HTML, CSS, JavaScript, Smell, Final Report (Communication, Project Design, References)
Zheng Ju [25%]	Web Server Connection, Report, Statistics, Final Report (Set Up)
William Ikenna-Nwosu [25%]	Statistics, Smell, Final Report (Success and Failures, Proof Read)

References

1. Cisco Study Finds Telecommuting Significantly Increases Employee Productivity, Work-Life Flexibility and Job Satisfaction.
   Available at:
   https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=5000107
2. Floyd Henry Allport. "The Influence of the Group Upon Association and Thought." Journal of Experimental Psychology, 3, 1920: 159-182.
   Available at:
   https://brocku.ca/MeadProject/Allport/Allport_1920a.html
3. The hard science of teamwork (March 20, 2012).
   Available at:
   https://hbr.org/2012/03/the-new-science-of-building-gr