**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)

Use a splash page image here [optional]

Use *LaTeX* if you wish, but use the general spacing and font/style you find here (1.5 spacing, 12 point font for text, etc.).

Be sure to submit a PDF (not a .DOC file) as your report. Overall it should be 12 to 15 pages, including diagrams.

**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.

**Ideally:** Have Scentaur integrated into an IDE to allow developers to have all tools at their disposal to tackle problems and work together to do more than write code; update their software systems (with confidence) and add features more efficiently. Scentaur could automatically run after a git pull command then refactor the code autonomously and report changes it made, otherwise it can display the report as a list of suggestions to guide refactoring.

**1.2 What Scentaur Hope to Achieve**

* **Web-based:** Team Scentaur’s aim is to make a web-based code smell detector tool. Providing a quick and easy way to detect smells and even enabling users to refactor fragments of code detected by Scentaur. Having Scentaur be a web-based product also allows us to reach a wider audience of users.
* **Performant:** Scentaur aims 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 refactor smelly code. As a result, Scentaur aims to use a smaller amount of Code Smell Detectors to increase performance. Detect smells that are common code smells and refractor code that will take lower effort.
* **Go-to product:** Since coding is being encouraged at large scale to the general public, Scentaur can be the go-to software product for beginners to learn how to think about the software design phase of development and give them an understanding and appreciation for writing maintainable code. Thinking about code on a higher level so they can make an immediate impact when working in teams together and having new people join or joining a team/company/open source project.
* **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 should 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 of Project**

Team Scentaur has set several main goals in mind. 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.

Scentaur strives to provide easy access, usage, lookup for all types of users. Being able to switch between a detailed and broad descriptor on the detected code smells. Followed by a color-coded scheme to display each individual smell. Team Scentaur encourages young Java users to understand the purpose of code smells by presenting easy to understand visualization of code smells, along with a brief description of the smells involved. Scentaur also provides for those that are more experienced with Java and data. By presenting a more in-depth explanation of code smells detected and help users to track down their smelly code.

**1.4 Typical User Experience**

A typical user would be able to open the website and upload their code directly without login, they would be able to see the system’s detection of code smells and suggestions. In order to save their suggestions/code smells they would be able to sign up to the website, this would save their past history of code uploads as well as code files don’t take up much space - we should be able to save their files and results on the server itself.

**2. Specification**

**2.1 Analyze Project**

Scentaur is planned to become a web application through Spring. Therefore, users are enabled to either submit a zip folder or java files directly to a web server. If a Zip folder is submitted, its contents will be extracted to a directory. Otherwise, files will be placed into a directory.

Designate a folder directory to contain the location of where Scentaur should analyze the project.

* For testing purposes, Scentaur will sniff out code from the *“testProject”* directory.
  + Note: Purposely made code smells will be available in *“testProject”*
* This will be a temporary directory to enable multi-users to run Scentaur.
* Once a user is done with Scentaur, the contents of the directory is wiped.

In order to analyze the entire directory given, Scentaur will be using **JavaParser.**

* A Parser class will take in the root directory path as a string.
* configureSymbolSolver method will set the symbols required to sniff out java files.
* The constructor will call configureSymbolSolver and parse all source file based on JavaParser-JUG-Milano slides.
* Parser will have a method that returns the compilation units of all java files within its root and sub directories.
  + Note: Information on compilation units is given in **Software Overview**

**2.2 Detect Code Smells**

* An abstract Smell Superclass will generalize all code smells.
* Sub directories will be made to accommodate smells that are categorized
  + These categories include:
    - Bloater, Abuser, Coupler, Dispensable
* The following interfaces will be made: *Smellable, Abusable, Bloatable, Coupleable and Dispensable.* To ensure that we enable plug-in-play system for the smells within the categories.

A sample hierarchy is shown to display the hierarchy specification.

*Smell* extends *VoidVisitorAdapter<Void>* inherits *Smellable*

*Bloater* extends *Smell* inherits *Bloatable*

*LongParameterList* extends *Bloater*

*VoidVisitorAdapter<Void>* enables code smell detectors to visit nodes for a compilation unit related to the code smell. *E.g. LongParameterList visits methods of a class and checks if the method has a long parameter*

The following is possible due to the hierarchy above:

*Smell* longParameterList = new LongParameterList();

**2.3 Generate Report**

The report will be available in multiple different forms.

* A Report class will take in all the smells that were detected
* It will have an object inside the report to enable calculations in generating data for the smells. This is to **show distribution of different smells** that exists within the code.
  + - Sample text: Bloater Smells – 23  
       LongParameterList – 15  
       Long Method - 8
    - Generate percentages in terms of smells.

Bloater/Total Problems \* 100%

Abusers/Total Problems \* 100%

Coupler/Total Problems \* 100%

Dispensable/Total Problems \* 100%

Furthermore, the report can be obtained in a text file. Displayed on a table like manner.

* The report can also generate classes in either text or java format with comments added to wherever the code smells existed.

**2.4 Visualize the Code-Base & Identify Trouble-Spots**

The code base is planned to be visualized on the web browser using spring and CSS.

* Each smell will have their own identifying color in hexadecimal for CSS to interpret.
* Scentaur plans to visualize problems by either commenting problems above the smell or color coding the specific problems.
* Using the compilation unit to locate the line at which the problems exists we can add color to it.
* Problems will be highlighted.
* A cross comparison will be shown where the left-hand side displays the original code and the right-hand side displays the updated version containing comments or highlighted text describing the code smell.
* Users will be enabled to choose different smells to detect from through a selection bar. The default option is the one where all smells are sniffed for. [Drop Down Menu]
* Percentages like the calculations in the report will be shown in terms of pie charts, histograms and other visual representations.

**3. Software Overview**

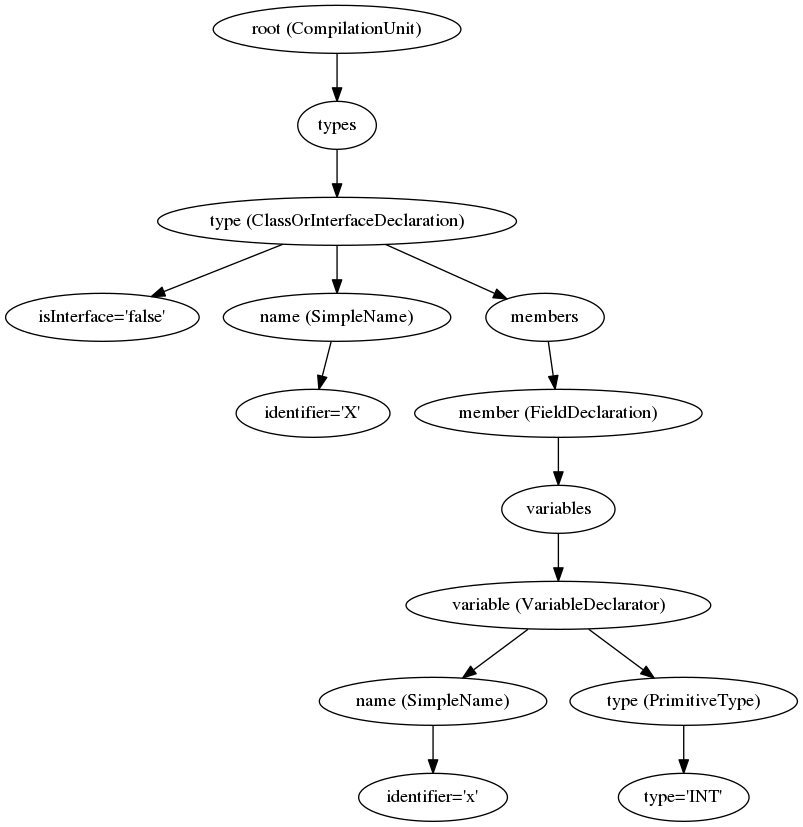
**3.1 Smell Overview**

Below is a schematic UML view of the planned Scentaur Design.

The Scentaur team is advised to follow the layout above. This will ensure that the team moves forward with Scentaur consistently and cooperatively towards a common goal. This hierarchy and structure provide a means to maintain a highly cohesive and low coupling software design.

Multiple interfaces have been built to date to ensure that team-members follow a set structure when designing code smells. The following code smell interfaces have been designed to date:

1. Smellable
2. Bloatable
3. Coupleable
4. Abusable
5. Dispensable

Smellable contains abstract methods which all smells should be capable of implementing. This is the general solution to all smells. Bloatable, Coupleable, Abusable and Dispensable are interfaces which are unique to their code smell category. E.g. LongParameterList class will be Bloatable and therefore isBloatable().   
Code smells detected are stored in a list of nodes which are specific to the JavaParser Abstract Syntax Tree. We use JavaParser.ast to enable the team to pinpoint exactly the sections of code which contains a specific

ref: <https://javaparser.org/inspecting-an-ast/>

From this diagram, we can observe that the JavaParser creates CompilationUnits for each Java file. With this Compilation Unit we can obtain different nodes specific to the Java file such as variables, methods and comments. Scentaur uses this implementation by storing nodes with specific code smells. E.g. **Primitive Obsession** using nodes, can check how many times a variable is used within that class, where the variable is declared and therefore obtain its scope. If the variable is a primitive obssession then the variable will be stored as a node within a list in Smell.

This node is a very powerful object as it can obtain various types of information such as the line it was declared on the java file, the class it was declared in and the directory the node came from.

Furthermore, Scentaur allows team members to add more code smells without adding complicated dependencies. Code smells created will only have to appear in the code smell category that it belongs to and implement and inherit some interfaces/classes which are all smell related. The created smell object is then accepted by all the compilation units in the following way.

all.forEach(c -> {

c.accept(longParameterList, null);

});  
Note: all – List of compilation units.  
To accept more code smells a simple addition of c.accept(“code smell object”, null); is called. Thus, allowing Scentaur to smell “code smell object” and longParameterList.

The Scentaur team has also generated a testProject directory which contains sample code that we can test for code smells. Specifically made code smells are present in this directory. To detect code smell within Scentaur a simple change of root directory path to System.Property(“user.dir”); is called.

**3.2 Report Overview**

A report class is made in order to do some functionalities. The report will be taking an array of smell objects that are not null (smells that are present within the java files). The report will be capable of generating a text file containing a table of relevant code smell data . For the report to generate these data it will need a calculation object that will calculate statistical anaylsis of the code smells present.   
This statistcal object will perform some of the following calculations: average testing, range, occurences etc. Which is then supplied back to the report to use when generating a text file.  
The report is also capable of generating java files. This is possible because the nodes stored in the smell objects can return their compilation unit.  
  
**3.3 Spring Visualization**

As the team is mainly focused on the standard concept of the program much knowledge of spring will come after everything else is implemented. If time is the issue the team may plan to create a UI instead (As Spring is a new concept to the team); this ultimately puts us back. However, we have one team member who has some knowledge of Spring.

**4. Major Responsibilities and Work Breakdown**

Provide a modular view of the work here, with an assignment of responsibilities to each team member. A Gantt diagram is useful here.

**5. Team Communication**

Provide an overview of the tools and methodology you will be using to ensure productive teamwork and a transparent flow of information. Will you be using any groupware applications to coordinate your efforts? How often will you be meeting as a team? How will you use resources like GitHub to maximum effect? How will you deal with disagreements or scheduling conflicts or misalignment of goals or plans?

**6. Concluding Remarks**

Provide a brief statement of your team’s philosophy here. Mention any special of noteworthy aspects of your approach to the problem. Highlight any risks you feel may impact the work, and offer mitigation strategies if necessary.

**Acknowledgements**

Every team member should contribute an equal effort to this report. Use this optional section to provide report credits, or to highlight a special contribution by a given team member.

**References**

List any bibliographical citations here [optional]