**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 refractor smelly code. As a result, Scentaur aims to use fewer Code Smell Detectors to increase performance and will detect code smells that are more frequently created by users. Scentaur plans to only refractor pieces of code that are not costly to performance.
* **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. For users to obtain their history of code smells/suggestions of refraction, users must stay on the webpage without closing the web browser. Closing the web page will automatically close all data related to the user.

**2. Specification**

**2.1 Analyze Project**

The development of Scentaur is accelerated by Gradle. Allowing the team to freely import libraries outside of the standard. Scentaur is also planned to become a web-based application using Spring. Therefore, users are enabled to either submit a zip folder or java file 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 are wiped.
* To analyze Scentaur itself use System.Properties(“user.dir”); to obtain the String path of Scentaur’s root directory.

*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 files 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>* implements *Smellable*

*Bloater* extends *Smell* implements *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**

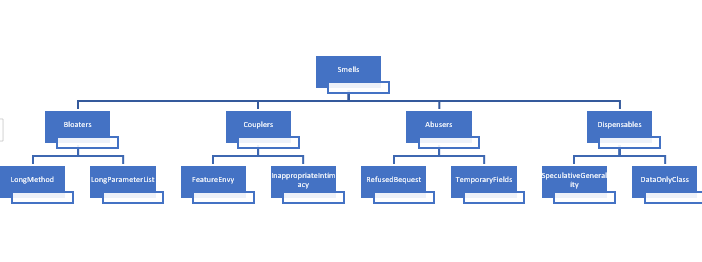
The code base is planned to be visualized on the web browser using spring and CSS.

* 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**

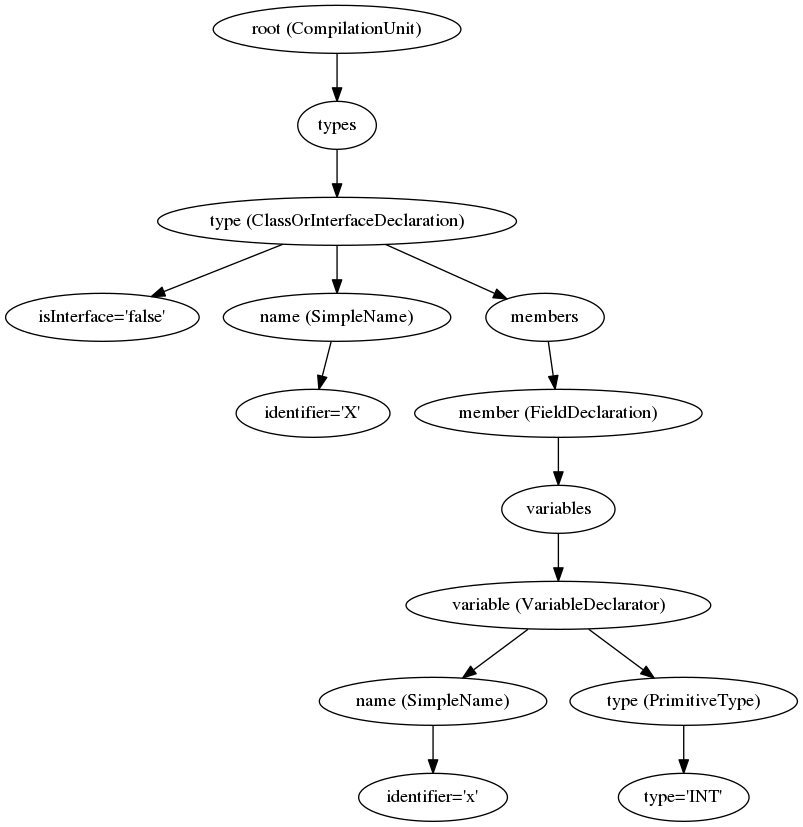
Below is a schematic UML view of the planned Scentaur Design in detecting code smells.



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 smell.

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

From this diagram, we can observe that the JavaParser creates Compilation Units 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 in a list.

E.g. **Primitive Obsession** checks how many times a variable is used within that class. If the variable is a primitive obsession, then the variable will be stored as a node within a list in Smell.

This node is a very powerful object as it can also retrieve information such as the line it was declared on in 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**

The report class 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. A general idea on how to solve this visualization problem is included in specialization.

**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.

**4.1 Major Responsibilites**

Having a diverse team from various backgrounds, we were able to divide players based on their expertise and utilize the best abilities of each member.

* + - 1. Lead Designers  
         - Gajun Young and William Ikenna-Nwosu are responsible of overall design and identifying major classes. They were able to use well thought out and concrete class hierarchy structure to make the project as a whole much easier to develop.
      2. Reflection API  
         - Zheng Ju and Royal Thomas implemented the reflection API. The team used actions provided within JavaParser to reflect on the code and extract various information from it.
      3. File Analysis   
         - Gajun Young and Royal Thomas are in charge of making an interface to allow users to import and read files from the user’s preferred folder. While at the moment it reads from the user’s folders on the computer it runs, the ideal plan for the team is to allow users to upload their entire projects onto the server for analysis and report generation.
      4. Smell Detection  
         - Smell detection is broken up into four parts and two members are in charge of each,   
         - 1. **Bloaters -** Zheng and Will.   
         - 2. **Abusers -** Gajun and Royal.  
         - 3. **Couplers –** Will and Gajun.  
         - 4. **Dispensables –** Royal and Zheng.
      5. Interface Design  
         - Zheng and William were able to implement interfaces ensuring that each smell acted as a plug and play module. Knowing what each smell detector should be able to do, we were able to properly exploit polymorphism without worrying how each module is implemented on its own.
      6. Code Analysis   
         - Royal and Gajun are responsible for calling the detectors onto the java files one at a time to produce reports.
      7. Smell Visualization   
         - The whole team is responsible for the implementation of smell visualization. The ideal plan is to be able to print out the code for each file after the process and point out to the users the issues through color codes implemented using css libraries and html.
      8. GUI Interactions  
         - Like smell visualization, the entire team is responsible for the GUI interactions. As a team, we are hoping to form an entire web service capable of receiving projects from a user on an html page which will be then processed by a Java backend (Spring). We are hoping to implement various CSS and JavaScript libraries to make it very user friendly while being simple and well structured.

**4.2 Work Breakdown**Gajun Young

As the team leader, he is responsible for group meeting organization and task assignment through the whole development. At the beginning of the project, he and Royal took the responsibility of figuring out how to pass source files from user into the software and conducted the preliminary file analysis. As the project progresses, he plays the main role of designing the software, coordinating the team and making necessary adjustment.

Royal Thomas

One of the main developers of File Analysis System. He worked with Gajun on how to pass the file to the software and conduct the preliminary analysis. Also, he is in charge of designing the I/O system and visualizing the code smells. As an important member of team Scentaur, he should assist the leader to make decisions and evaluate any adjustments to the team.

William & Zheng

Main developers of project Interfaces. Worked together to decide the interfaces the software would use. Decided carefully what hierarchy the software would follow. Test the File Analysis System. Take the main responsibility in Code Smell Detection Development. Assist the leader to make decisions and evaluate any adjustments to the team.

**5. Team Communication**

For better communication between team members, several Apps such as Slack, Messenger and Discord, were used by team Scentaur. A GitHub Project Board was also used. Apart from online communication, face-to-face group meetings were scheduled for each week. Usually, decisive agreements such as project interface design and user interface selection, were reached during the face-to-face meeting.

**5.1 Messenger**

In order to make the software development more specific, smooth and efficient, for general discussion, Messenger was our primary platform to share ideas. Questions were carefully discussed and addressed by group. However, given the rudimentary nature of the platform, it was not used as a primary method to record crucial communication information.

**5.2 Discord**

Discord was where we conduct daily Scrums, Spring Planning, Sprint reviews and Spring retrospectives. It ended up being a very user-friendly group voice chat platform to conduct these meetings. Team Scentaur had 3 channels in total. General channel was used for general talk. Team members discussed advantages and disadvantages, agreements and disagreements of the project and real-time ideas, information or links found were shared immediately here. Screen sharing was highly encouraged because it is a good way for every team member to give and receive advice and help. This platform allows team Scentaur to ensure group work and communication are pushed in progress simultaneously. Another two channels were for sub-groups. Our 4-memeber team was divided into 2 2-member sub-groups during different period of development. These channels were used for sub-groups to work on specific modules of the project. Therefore, different modules can be developed at the same time.

**5.3 Slack**

Basically, real-time information was shared during the group discussion, but any of the useful or important materials were also posted on Slack. Team Scentaur used Slack primarily to record the progress of the project. Only critical decisions or task assignment were pushed on Slack. Usually, team members were not allowed to talk on Slack. This makes Slack a clean and useful reference for team members to investigate.

**5.4 GitHub Project Board**

GitHub Project Board was used as a Scrum board to assign responsibilities and to track progress. Using the board, we began by producing an MVP (Minimum Viable Product) which was able to detect a few smells and read files in and parse them. Incrementally, we added more and more features onto this to produce better software. This helped us review the software and the path we are taking each time we ran a sprint.

**5.5 GitHub**

During the development, GitHub was used for version control, scrum board and issue/bug tracking. The version control system helped us pull back commits that had issues and made it simpler for us to share our code with each other. The GitHub issues section was used to report bugs, issues and possible enhancements which was not communicated during the daily meets.

Team Scentaur holds team meetings frequently. Weekly meeting is fixed on Wednesday and other meetings are held if necessary, on campus. During break, daily talk on Messenger is compulsory. Team Scentaur needs to have the knowledge of everyone’s work progress. Voice meeting are planned. Usually once every 3 or 4 days but team Scentaur will hold a voice meeting every day when a topic requires a discussion.

**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**

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

|  |  |
| --- | --- |
| Name [Contribution %] | Responsibilities |
| Ga Jun Young [25%] | Introduction, Specification, Software Overview, Referencing |
| Royal Thomas [25%] | Introduction, Team Communication, Major Responsibilities & Breakdown |
| Zheng Ju [25%] | Introduction, Team Communication, Major Responsibilities & Breakdown |
| William Ikenna-Nwosu [25%] | Introduction [Compile], Software Overview, Proof Read and Editing |

**References**

Sandi Metz (2017), *Code Refactoring: Learn Code Smells And Level Up Your Game!.* Available at: <https://www.youtube.com/watch?v=D4auWwMsEnY> (Accessed: 18th March 2019)

Federico Tomassetti (2017), *JavaParser-JUG-Milano.* Available at: <https://tomassetti.me/wp-content/uploads/2017/12/JavaParser-JUG-Milano.pdf>

*Alexander S., Gerhard F., Marina P. (2006 – 2019), Code Smells*, Available at: <https://sourcemaking.com/refactoring/smells>

Danny V.B (2018), *Inspecting an AST,* Available at: <https://javaparser.org/inspecting-an-ast/>