# 3. Refactoring Assistants
(Last Update: 2024-02-27)

In this session, we will learn about two tools that offer refactoring assistance. Using tools, we can plan the refactoring tasks that could improve our design. We will search for bad smells (symptoms of design problems) that give us **hints** on where and how to refactor. When planning refactoring tasks, remember the pattern **"Keep it Simple"** (OORP, p.37), as it is a common mistake for people to over-complicate the design of a refactored artefact. Another important pattern to remember when refactoring is **"Most Valuable First"** (OORP, p.29), meaning you should prioritise the refactoring tasks that bring more benefits.

During this session, we will perform simple refactoring tasks. However, it is important to remember that for the Reengineering Course, we focus on Strategic Refactoring, i.e., we should refactor with a clear reason or goal in mind.

**Sample Project**
django-cms/django-cms

**Materials & Tools**
- PyCharm IDE
- JPacman repository.
- CodeScene - *no* installation necessary, but it requires a GitHub account. This tool's integration with GitHub allows it to visualise your repositories. The Technical Debt part shows refactoring targets. The Code Biomarkers show a more detailed analysis of smells, but it is only available to paid subscribers.
- SonarQube is a tool/platform that performs static analysis on source code. Download the free community edition or run its Docker container.

**Setup / Preparation**
1. You can read the slides here.
2. Fork the repository above.
3. If you have not already, download the book for this course *"Object-Oriented Reengineering Patterns"* *(Note: OORP, p.xx refers to a page in the pdf version of this book)*

**Task 1: Django-CMS on CodeScene**

For our first task, we will use CodeScene to suggest which artefacts need refactoring.
To do this, select the "Code" menu on the left side, and then the "Hotspots" submenu. In this visualisation, the hotspots are artefacts with a lot of commit activity (i.e., they change a lot during the software evolution and maintenance).
On that visualisation, you can check the tab on Refactoring Targets. Look at the recommended refactoring targets.

If you select a specific file in this visualisation (or the hotspots visualisation), on the right, it will display more details. When you scroll down to the details section, you can see a few actions. They include Review, Source code, and X-ray. Check out these options and see for yourself what information CodeScene can provide.

Also, notice that, for some files, CodeScene highlights other coupled files. Explore these code couplings as well.

Questions:
1. Did the CodeScene visualisation help you identify possible targets for refactoring?
2. Did CodeScene give you hints or clues about how to refactor the proposed targets?
3. Did CodeScene help you visualise the extent of the refactoring activity?

**Task 2: Django-CMS on SonarQube**

For the second task, we will use a more complex and dedicated tool to find refactoring targets. Follow the instructions in the documentation to install and run SonarQube. You may either install it locally or run it in a Docker container.

If you are successful, you should be able to run an analysis of your local clone of Django-CMS.

Click on the "Code Smells" and analyse the detected smells. You should see that SonarQube also explains the smells ("Why is this an issue?").

Questions:

1. Which one is more useful when locating refactoring targets: CodeScene or SonarQube?
2. Which tool provides better reasoning/explanation for the possible refactoring targets: CodeScene or SonarQube?
3. Have you found many common artefacts in the Code Smells in SonarQube and Refactoring targets in CodeScene?

**Task 3: Django-CMS Strategic Refactoring**

For the Reengineering Course, we value the concept of Strategic Refactoring, which is refactoring with a goal. Tools can help identify artefacts with smells that could lead to potential issues. However, only a developer can identify the necessary artefacts for a specific goal. Let's do that for Django-CMS.

Browse through the issue tracker of Django-CMS and search for issues with the following keywords “**is:open label:"kind: enhancement"**”. You will notice there are some issues which require a patch.

Choose a few issues and plan the necessary refactoring task(s) to support them. You can start with the simplest refactoring tasks to avoid "breaking" the code or go big according to the pattern **"Most Valuable First"**. There is no wrong path. Do whichever you find easier or most logical for you. Please note, you need only to **"plan"** the refactoring activities for this lab session, there is no need to implement the refactoring tasks (of course, if you want to do it, go ahead --- just remember that planning is the important part here).

Questions:

1. What were your strategies and reengineering patterns for planning this refactoring?
2. Why did you consider these refactoring tasks important for your goal?
3. Did the previous tools (CodeScene or SonarQube) identify the refactoring targets you deemed necessary for this goal?
4. Do you prefer to refactor to improve code quality or to refactor with a goal?

**Optional Task 1: Django-CMS Strategic Refactoring (Part 2)**

To complete this optional task, you should implement the planned refactoring tasks. Remember to ensure that your refactoring change is not breaking the application.

**Optional Task 2: Duplicate Code Detection on SonarQube**

For this optional task, let's remind ourselves of the previous lab session on Duplicate Code. If you looked over the SonarQube analysis on Django-CMS, you may have noticed this tool also detects duplicated code. Check out how SonarQube presents the duplicated code it found.

Did you like SonarQube visualisation of duplicated code snippets? Or do you prefer another tool? Why do you like your chosen tool?

## Additional Reading Material

To dive deeper into refactoring, you may read the materials below:

- M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. Refactoring: Improving the Design of Existing Code. Object Technology Series. Addison-Wesley, 1 edition, June 1999.
- M. Lanza and R. Marinescu. Object-Oriented Metrics in Practice - Using Software Metrics to Characterise, Evaluate, and Improve the Design of Object-Oriented Systems. Springer, 2006.
- M. Fowler and J. Kerievsky. Smells to refactorings quick reference guide. Reference sheet, 2005.: http://www.industriallogic.com/blog/smells-to-refactorings-cheatsheet/
- F. Khomh, M. D. Penta, Y.-G. Guéhéneuc, and G. Antoniol. An exploratory study of the impact of antipatterns on class change- and fault-proneness. Empirical Softw. Engg., 17(3):243–275, June 2012. http://link.springer.com/article/10.1007%2Fs10664-011-9171-y