# Design & module code metrics

## Завдання 1 - code metrics continued

### Prerequisites

1. Для першої частини завдання вам знадобиться 2 приклади коду (snippets). Ви маєте обрати 2 приклади коду з відкритого доступу - використовуючи <https://github.com/> . Для прикладу ви можете взяти:

| Мова | Посилання |
| --- | --- |
| загальні | <https://github.com/alan2207/awesome-codebases>  |
| .Net | <https://github.com/quozd/awesome-dotnet?tab=readme-ov-file#application-frameworks>  |
| JS | <https://github.com/sorrycc/awesome-javascript?tab=readme-ov-file#mvc-frameworks-and-libraries>  |
| PHP | <https://github.com/ziadoz/awesome-php?tab=readme-ov-file#frameworks>  |
| для інших | для будь-якої мови є список “awesome-\*”  |

Буде корисно обрати фреймворк з яким ви вже знайомі, і працювали. Фреймворк обов'язково має бути у відкритому доступі.

### Опис задачі

1. Проаналізувати design рішення даного фреймворку. Відповісти на наступні питання (письмово, розгорнуто, подумавши):
	1. Яку задачу вирішує фреймворк?
	2. Яким чином? Підхід?
	3. Яку він має структуру (файли, слої/layers)?
	4. Які використовуються патерни?
		1. Знайти 1+ design patterns в codebase. Навести лістинг або скріншоти коду. Пояснити для чого цей патерн використовується.
	5. Знайти використання 3-х design principles в коді. Ваші спостереження пояснити. (Будь-ласка, для SOLID - кожну букву розглядаємо окремо).
2. Для обраних вами сніпетів коду порахувати (вручну, з деталізацією розрахунків) для прикладів - дивіться [Appendix](#_1osxlkm5g13d):
	1. LCOM
	2. Abstractness
	3. Instability
	4. Distance from the main sequence

## Як здати лабораторну роботу

Результатом виконання вашої лабораторної роботи має бути звіт (з титульною сторінкою, титульна - з рамкою, далі не вимагається). Звіт має бути структурований відповідно до завдань, містити код для перевірки.

Звіт надсилається на пошту викладача kkn\_vra@ztu.edu.ua у форматі PDF!

Subject має БУТИ ОБОВЯЗКОВО ‘{ваша група}, {СУЯІС - абревіатура предмета}, {ПІБ} - Лаб {N}’. - наприклад

ІСТ-20-1, СУЯІС, Вольський Ростислав - Лаб 3

## Appendix

### Корисні посилання:

* <https://en.wikipedia.org/wiki/Programming_complexity>
* <https://en.wikipedia.org/wiki/Software_metric>
* <https://www.researchgate.net/publication/221598130_A_framework_analysis_of_the_open_source_software_development_paradigm>

#### Приклади розрахунку метрик

#### 1. Lack of Cohesion in Methods (LCOM)

* Definition: LCOM measures how well the methods of a class are related to each other. High LCOM values indicate a class where methods have little in common, suggesting the class might be doing too much or should be split into multiple classes.
* Calculation Example:
	+ Consider a class with methods M = {m1, m2, m3} and fields F = {f1, f2, f3}.
	+ Suppose m1 uses f1, m2 uses f2, and m3 uses f3 only.
	+ There are no shared fields between methods, so each method is a separate part.
	+ LCOM can be calculated in various ways, but a simple approach is LCOM = P - Q, where P is the number of unshared (unique) field accesses across methods, and Q is the number of shared field accesses across methods. In this case, P = 3 (each method accesses a unique field) and Q = 0 (no shared field accesses), so LCOM = 3 - 0 = 3.
	+ A high LCOM suggests considering splitting the class into smaller, more cohesive classes.
* Origin: LCOM was originally introduced by Chidamber and Kemerer in their paper "A Metrics Suite for Object Oriented Design" published in IEEE Transactions on Software Engineering in 1994. The original LCOM metric has been revised and extended over time (e.g., LCOM2, LCOM3, LCOM4), but the foundational concept originates from Chidamber and Kemerer's work.

#### 2. Abstractness (A)

* Definition: Abstractness measures the proportion of abstract types (interfaces and abstract classes) in a module. It ranges from 0 (no abstract types) to 1 (all types are abstract), aiming to balance between abstraction and concreteness.
* Calculation Example:
	+ Consider a module with 10 classes, including 2 abstract classes and 1 interface.
	+ The number of abstract types Na = 2 (abstract classes) + 1 (interface) = 3.
	+ The total number of types Nt = 10.
	+ Abstractness A = Na / Nt = 3 / 10 = 0.3.
	+ This value indicates the level of abstractness of the module, with 0.3 suggesting a moderate level of abstraction.
* Origin: These metrics are part of the package metrics introduced by Robert C. Martin in his work on the principles of Object-Oriented Design and the metrics for package cohesion and coupling. Martin's metrics, including Abstractness (A) and Instability (I), were detailed in his book "Agile Software Development, Principles, Patterns, and Practices" and various articles on the topic of software design principles, often referred to as the SOLID principles.

#### 3. Instability (I)

* Definition: Instability measures the ratio of efferent coupling (Ce, outgoing dependencies) to total coupling (Ce + Afferent coupling (Ca), incoming dependencies). It ranges from 0 (stable, no outgoing dependencies) to 1 (unstable, many outgoing dependencies).
* Calculation Example:
	+ Consider a module with 5 outgoing dependencies (Ce) and 3 incoming dependencies (Ca).
	+ Instability I = Ce / (Ce + Ca) = 5 / (5 + 3) = 5 / 8 = 0.625.
	+ This value indicates the module's tendency to be affected by changes in other parts of the system.
* Origin: same as for Abstractness

#### 4. Distance from the Main Sequence (D)

* Definition: The main sequence is a hypothetical line that represents the ideal balance between abstractness and stability. Distance from the main sequence (D) measures a module's deviation from this ideal, aiming for a D value close to 0.
* Calculation Example:
	+ Given a module with Abstractness A = 0.3 and Instability I = 0.625.
	+ Distance D = |A + I - 1| = |0.3 + 0.625 - 1| = |0.925 - 1| = 0.075.
	+ A low D value indicates the module is well-balanced in terms of its design for changeability and stability.
* Origin: The concept of the "Main Sequence" and the Distance from it (D) is also from Robert C. Martin's work on package design principles, particularly in the context of the Dependency Inversion Principle and his discussions on the balance between flexibility and understandability in package design.