МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра програмних систем і технологій

Дисципліна

«Емпіричні методи програмної інженерії»

Практична робота № 6

“ПРАКТИЧНЕ ЗАСТОСУВАННЯ МЕТРИК”

Виконав:	Гоша Д. О	**Перевірила**:	Юрчук Ірина Аркадіївна
Група	ІПЗ-23	Дата перевірки	
Форма навчання	денна	Оцінка	
Спеціальність	121		

2022

Мета: Навчитися використовувати метрики на практиці при розробці програмного забезпечення.

Завдання

1. Дослідити та провести рефакторинг власної роботи (власного проекту), написаної раніше на C# в середовищі Visual Studio.

Виконання лабораторної роботи

Незважаючи на те, що розрахунок метрик може проводитися у повністю автоматизованому режимі, питання ефективного аналізу результату залишається невизначеним. Значні обсяги даних, отримані у результатах розрахунків, протилежні значення результатів певних метрик, складність і неоднозначність трактування та суб'єктивність обчислень деяких з них є серйозною перепоною на шляху до широкого практичного застосування метрик у якості надійного універсального інструменту оцінки якості програмного коду.

Рефакторинг є процес такого зміни програмної системи, у якому змінюється зовнішнє поведінка коду, але поліпшується його внутрішня структура. Це спосіб систематичного приведення коду до порядку, у якому шанси появи нових помилок мінімальні. По суті, при проведенні рефакторинг коду ви покращуєте його дизайн вже після того, як він написаний.

Проведемо аналіз метрик деякого проекту, а саме сайту з погодними даними, що отримуються безпосередньо з API openweather – сайт метерологічного центру.

Аналіз буде проводитися за такими метриками:

1. Maintainability Index, - Індекс зручності підтримки
2. Cyclomatic Complexity, - Складність організації циклів
3. Depth of Inheritance, - Глибина спадкування класів
4. Class Coupling, - Зв'язок класів
5. Lines of Source code, - Довжина вихідного коду
6. Lines of Executable code, - Довжина виконуючого коду

Таблиця клас – значення метрики Назва класу	Індекс зручності підтримки	Цикломатична складність	Глибина спадкування класів	Зв'язок класів	Довжина вихідного коду	Довжина виконуючого коду
AspNetCore	85	436	4	49	4231	1002
Weather	76	7	1	51	103	39
Weather.Areas.Admin.Controllers	77	13	3	22	94	32
Weather.Controllers	75	25	3	30	121	44
Weather.Domain	86	11	6	18	79	10
Weather.Domain.Entities	85	33	2	7	65	37
Weather.Domain.Repositories.Abstract	100	9	0	4	21	0

Weather.Domain.Repositories.EntityFramework	88	13	1	13	77	18
Weather.Migrations	32	4	2	37	1100	286
Weather.Models	89	44	2	30	149	52
Weather.Service	96	13	1	0	23	1
Weather.Servise	75	6	1	13	25	6

Індекс зручності підтримки

Обчислює значення індексу від 0 до 100, що відображає відносну простоту підтримки коду. Високе значення означає кращу ремонтопридатність. Кольорові оцінки можна використовувати для швидкого визначення проблемних місць у вашому коді. Зелений рейтинг становить від 20 до 100 і вказує на те, що код має хорошу ремонтопридатність. Жовтий рейтинг становить від 10 до 19 і вказує на те, що код помірно підтримується. Червоний рейтинг – це оцінка від 0 до 9 і вказує на низьку ремонтопридатність. У нашому віпадку усі класи мають більше ніж 75 пунктів індексу, що свідчить про хороший індекс пітдтримки, тому що значення значно більше 20. Тільки клас Migrations, що являє автогенерований код для створення бази даних має низьку підтримуваність, але як правило в подальшому першої міграції він не використовується. Отже можемо зробити висновки про відсутність потреби рефакторингу саме за цьою метрикою.

Цикломатична складність програми

Вимірює структурну складність коду. Він створюється шляхом обчислення кількості різних шляхів коду в потоці програми. Програма, яка має складний потік керування, вимагає більше тестів для досягнення хорошого покриття коду і менш підтримується.

Цикломатична складність = кількість країв – кількість вузлів + 1.

Вузол представляє логічну точку розгалуження, край - лінію між вузлами. Правило повідомляє про **порушення**, якщо цикломатична складність перевищує 25.

Перевіримо, чи не перевищує рівень складності коду у методах, простори імен яких мають велику цикломатичну складову. А саме:

- Weather.Models
- Weather.Domain.Entities
- AspNetCore

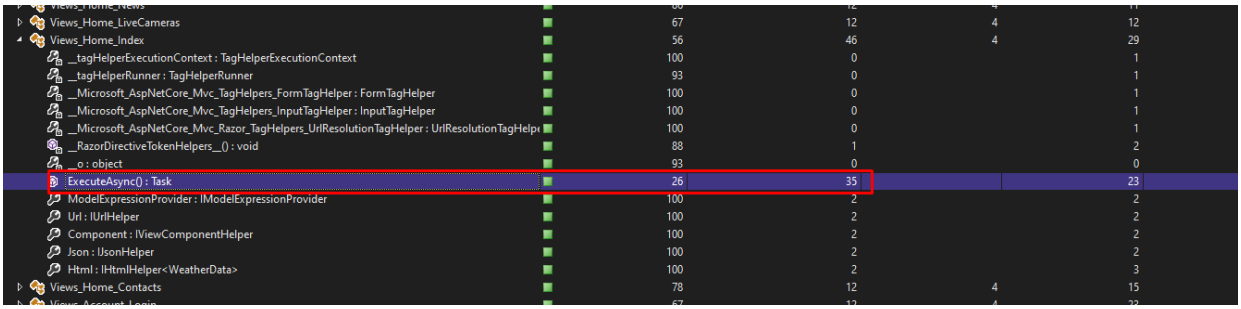
Models

CloudsInfo	6
ErrorViewModel	4
LoginViewModel	6
TemperatureInfo	14
WeatherData	6
WeatherService	8

Domain.Entities

EntityBase	19
ServiceItem	8
TextField	6

AspNetCore



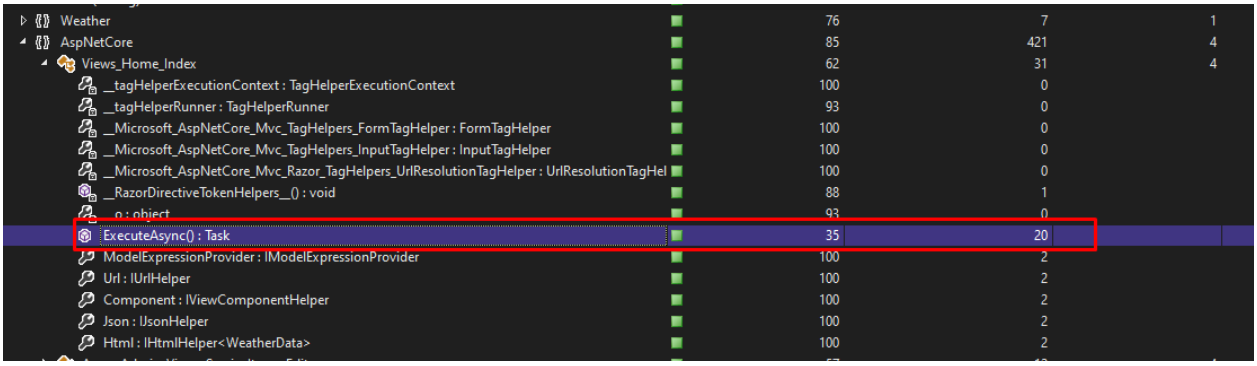
Аналіз метрик до рефакторингу

Побачимо, що в представлені View – Home – Index має завелику цикломатичність, а саме – 35. Виправимо це.

```
ViewData["Title"] = "Home Page";
string img = "http://openweathermap.org/img/wn/"+@Model.Current.weather[0].icon+".png";

wait Html.PartialAsync("CssPartial")
iv class="hero" data-bg-image="images/banner.png">
  <div class="container">
      <form asp-controller="Home" asp-action="Index" method="post" class="find-location">
          <input asp-for="City" type="text" placeholder="Find your location...">
          <input type="submit" value="Find">
      </form>
  </div>
div>
iv class="forecast-table">
  <div class="container">
      <div class="forecast-container">
          <div class="today forecast">
              <div class="forecast-header">
                  <div class="day">@dt.DayOfWeek</div>
                  <div class="date">@dt.ToString("dd MMMM")</div>
              </div> <!-- .forecast-header -->
              <div class="forecast-content">
                  <div class="location">@Model.timezone</div>
                  <div class="degree">
                      <div class="num">@(Convert.ToInt32(Model.Current.temp))<sup>o</sup>C</div>
                      <div class="forecast-icon">
                        <img src= @img alt="" width=90>
                      </div>
                  </div>
                  <span>
                      <img src="~/images/icon-umberella.png" alt="">@if (Model.Current.pop == 0)
                      {
```

Виправлений код



Аналіз метрик після рефакторингу

Після виправлення шматку коду можемо побачити, що індекс зручності підтримки класу зріс на 10 пунктів, а цикломатична складність зменшилась до 20, тим самим увійшла в проміжок норми.

Глибина спадкування класів

Вказує кількість різних класів, які успадковують один від одного, аж до базового класу. Глибина успадкування подібна до з'єднання класів, оскільки зміна базового класу може вплинути на будь-який із його успадкованих класів. Чим вище це число, тим глибше успадкування і тим вище ймовірність того, що модифікації базового класу призведуть до руйнівних змін. Для глибини успадкування низьке значення є хорошим, а високе - поганим.

У нашому випадку середнє значення метрики глибини спадкування складає 2,16. Рекомендований діапазон значення 2-5, тому що , якщо значення менше двох то принцип ООП спадкування використовується не правильно, а якщо більше 5, то це може сказатися на важкості підтримки і сильної залежності спадкувальників від батьківського класу. Саме моє значення у зоні рекомендованої норми, тому проводити рефакторинг недоцільно.

Зв'язок класів

Вимірює зв'язок з унікальними класами за допомогою параметрів, локальних змінних, типів повернення, викликів методів, загальних або шаблонних екземплярів, базових класів, реалізацій інтерфейсу, полів, визначених у зовнішніх типах, і декорування атрибутів. Хороший дизайн програмного забезпечення вимагає, щоб типи та методи мали високу згуртованість і низьку зв'язність. Високе зчеплення вказує на конструкцію, яку важко повторно використовувати та підтримувати через багато взаємозалежностей від інших типів.

Низько класне зчеплення загалом краще, але іноді це неможливо реалізувати. Наскільки це можливо, я безумовно мінімізував залежність між просторами імен, оскільки це дає набагато менше причин для залежностей. Але ASP.NET CORE, платформа ,на якій розроблений застосунок пропонує модульну архітектуру, що обов'язує таку кількість зв'язків модулів. Саме тому немає рекомендованого значення. Все індивідуально до кожного з проектів.

Довжина вихідного коду

Вказує точну кількість рядків вихідного коду, які присутні у вашому вихідному файлі, включаючи порожні рядки.

Довжина виконуючого коду

Вказує приблизну кількість виконуваних рядків або операцій коду. Це підрахунок кількості операцій у виконуваному коді. Значення зазвичай близьке до попередньої метрики, яка є метрикою на основі інструкцій MSIL, яка використовується в застарілому режимі.

Довжини коду не являються самостнійними метриками, тому слугують для знаходження інших. Наприклад якщо вирахувати значення по наступній формулі наведеній нижче, отимаємо розмір методу.

$$N = \frac{N_{\text{р}}}{N_{\text{м}}}$$

Саме так ми зможемо знайти завеликі методи і розділити їх на дві або більше частин.

Наведемо приклад **поганого** методу.

```
  public WeatherData NewCity(string City)
        {
            try
            {
                Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("ru-RU");
                Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
                if (City == null)
                {
                    City = "Kyiv";
                }

                MapPoint geo = getCoordinats(City);
                if (geo == null)
                {
                    try
                    {
                        MapPoint point;
                        var locationService = new GoogleLocationService(Config.GoogleApi);
                         point = locationService.GetLatLongFromAddress("Kyiv");
                    }
                    catch (System.Net.WebException ex)
                    {
                        Console.WriteLine("Google Maps API Error {0}", ex.Message);
                        return null;
                    }
                }
                string url =
$"https://api.openweathermap.org/data/2.5/onecall?lat={geo.Latitude}&lon={geo.Longitude}&exclude=hourly,daily&appid={Config.Weatherapi}&mode=json
&units=metric";
                HttpWebRequest = (HttpWebRequest)WebRequest.Create(url);
                HttpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();
                string responce;
                using (StreamReader = new StreamReader(httpWebResponse.GetResponseStream()))
                {
                    responce = streamReader.ReadToEnd();
                }
                WeatherData = JsonConvert.DeserializeObject<WeatherData>(responce);
                _memoryCache.Set($"Somekey", weatherData, TimeSpan.FromMinutes(1440));
                return weatherData;
            }
```

Наведемо приклад **виправленого** коду

```
public MapPoint getCoordinats(string address)
        {
            try
            {
                MapPoint point;
                var locationService = new GoogleLocationService(Config.GoogleApi);
                return point = locationService.GetLatLongFromAddress(address);
            }
            catch (System.Net.WebException ex)
            {
                Console.WriteLine("Google Maps API Error {0}", ex.Message);
                return null;
            }

        }
        public WeatherData NewCity(string City)
        {
            try
            {
                Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("ru-RU");
                Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);
                if (City == null)
                {
                    City = "Kyiv";
                }
                MapPoint geo = getCoordinats(City);
                if (geo == null)
                {
                    geo = getCoordinats("Kyiv");
                }
                string url =
$"https://api.openweathermap.org/data/2.5/onecall?lat={geo.Latitude}&lon={geo.Longitude}&exclude=hourly,daily&appid={Config.Weatherapi}&mode=json
&units=metric";
                HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(url);
                HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();
                string responce;
                using (StreamReader streamReader = new StreamReader(httpWebResponse.GetResponseStream()))
                {
                    responce = streamReader.ReadToEnd();
                }
                WeatherData weatherData = JsonConvert.DeserializeObject<WeatherData>(responce);
                _memoryCache.Set($"Somekey", weatherData, TimeSpan.FromMinutes(1440));
                return weatherData;
            }
            catch (OperationCanceledException)
```

```
            {
                return null;
            }
        }
```

Висновки

Таким чином, задача оцінки якості програмних проектів поки що не отримала прийнятного розв’язку у сучасній науці. Існуючі поширені підходи на основі метрик нерідко призводять до результатів, які погано узгоджуються між собою та є достатньо складними для інтерпретації і прийняття рішень на їх основі. Академічні проекти дозволяють по-новому підійти до вирішення проблеми, водночас поки що не придатні для масового практичного застосування в основному з причин як своєї новизни, так і незавершеності концепцій, використаних для представлення показників якості по проектам. Подальші дослідження у цьому напрямку, на мій погляд, мають бути зосереджені на тому, щоб привести існуючу базу метрик програмного коду до такого вигляду, у якому вони могли б слугувати реальною опорою для прийняття рішень керівниками програмних проектів, використовуючи комплексні і узгоджені між собою показники. Використання природно зрозумілих людині аналогій (структура міста, вигляд дерева та ін.) допомогло б подати інформацію більш наглядно, розширити коло людей, які могли нею оперувати при реалізації проекту.

У останній лабораторній роботі 6 були здобуті навички з рефакторингу власної роботи (власного проекту), написаної раніше на C# в середовищі Visual Studio.