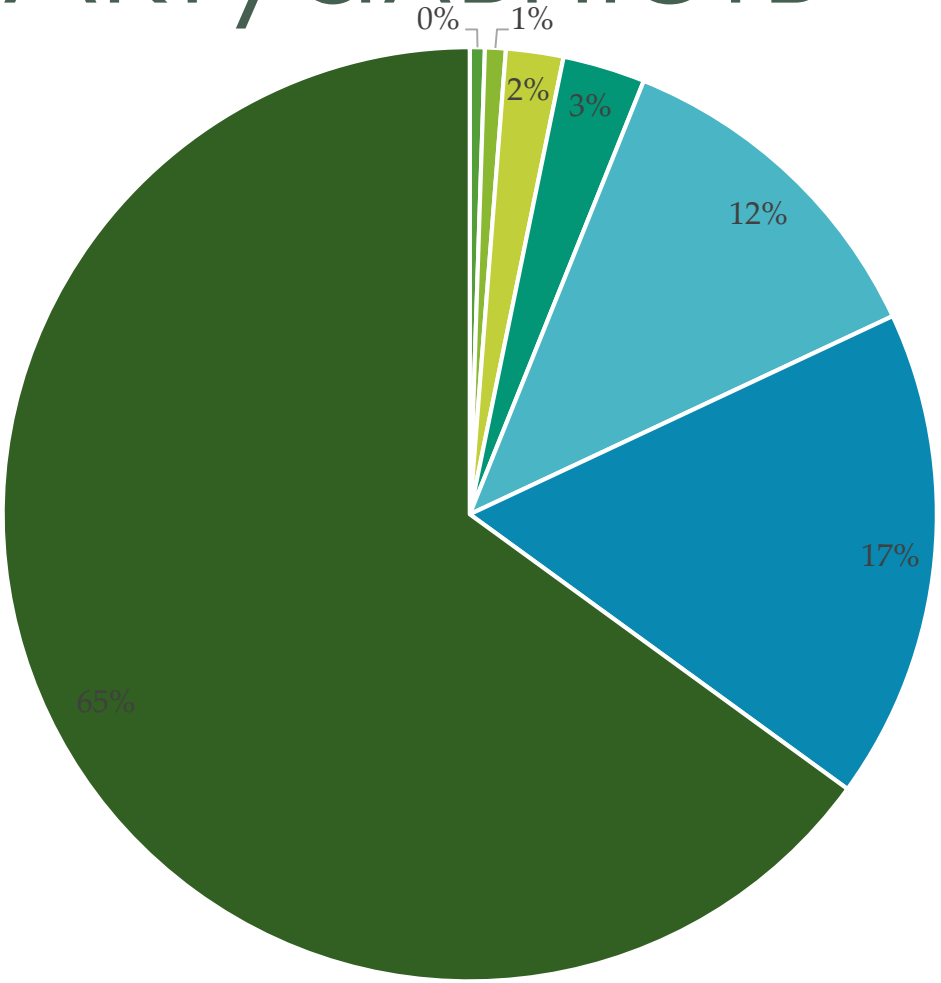
НТУУ «КПІ імені Ігоря Сікорського»
Теплоенергетичний факультет
Кафедра АПЕПС

Система комплексного рефакторинга для програм на мові C# .Net

Виконав: студент гр. ТВ-71мп *Лисяний* Є.С.
Керівник: к.т.н., доцент Гагарін О.О.

Актуальність



Популярність IT-КВЕДів

Станом на 12 квітня 2018

- Видання комп'ютерних ігор (653)
- Діяльність із керування комп'ютерним устаткованням (905)
- Видання іншого програмного забезпечення (2 505)
- Інша діяльність у сфері інформаційних технологій і комп'ютерних систем (3 601)
- Оброблення даних, розміщення інформації на веб-вузлах і пов'язана з ними діяльність (15 070)
- Консультування з питань інформатизації (21 418)
- Комп'ютерне програмування (81 970)

Мета та Завдання дослідження

- Проаналізувати існуючі засоби рефакторингу
- Проаналізувати існуючі генератори програмного коду
- Дослідити існуючі метрики коду та їх актуальність
- Удосконалити підхід до автоматизації рефакторингу
- Проаналізувати існуючі реалізації засобів автоматизації рефакторингу програмного коду
- Розробити програмний продукт для аналізу програмного коду для пошуку можливих проблем програмного коду

Об'єкт та предмет дослідження

- **Об’єктом дослідження** є програмне забезпечення автоматизованих систем рефакторингу.
- **Предметом дослідження** є засоби розробки програмного забезпечення рефакторингу програмного коду для мови C#.

Алгоритм аналізу

Попередня компіляція

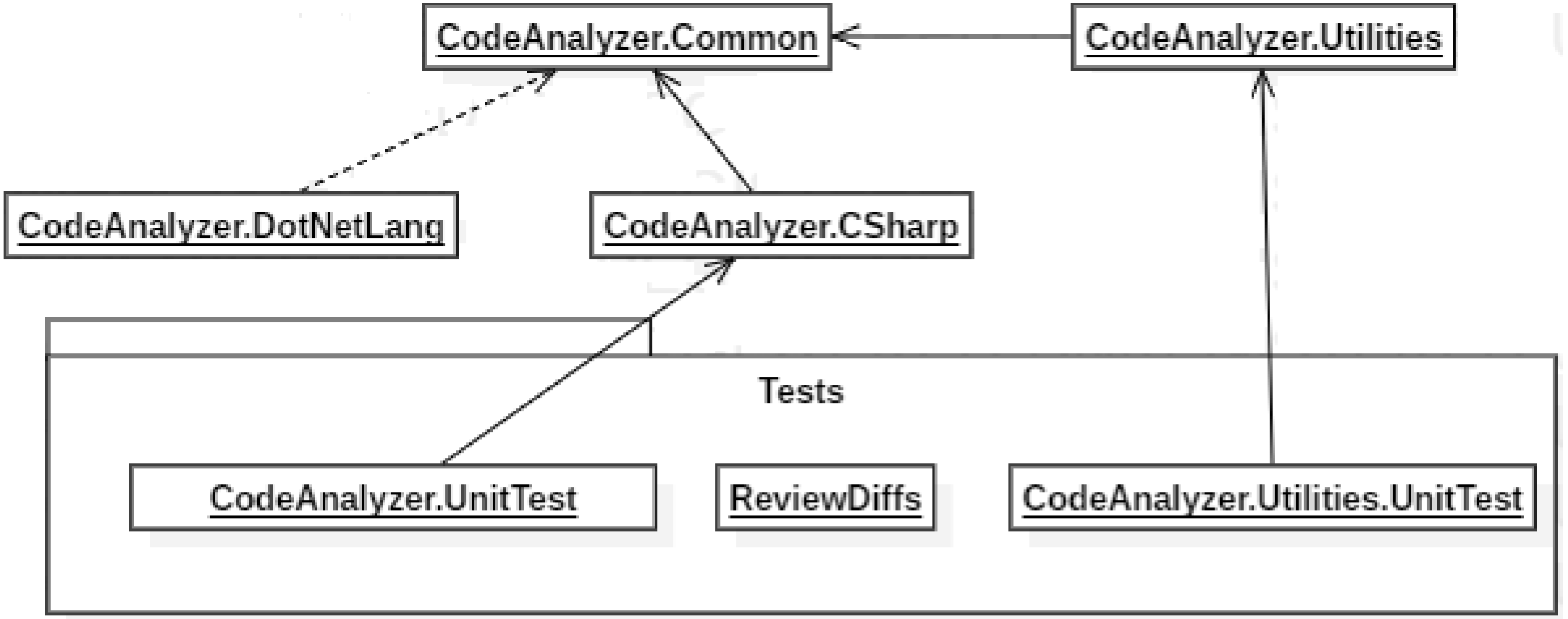
Визначення структури

Перевірка на відповідність правилам

Класифікація правил

- Дефекти
- Вразливості
- Некоректності коду (Code Smells)

Структура створеного рішення



Технологія використання

1. Додати зовнішнє посилання на бібліотеку до власного рішення
2. Використовуючи класи-утиліти, підключити розбиття коду на структурні елементи
3. Зареєструвати правила, що мають бути перевірені
4. Передати для перевірки файли:
   a) Рішення або проекту, для повної перевірки
   b) Файл з кодом, для часткової перевірки
5. Відобразити результати перевірки

Приклад дефекту

```
1   public class TestClass : IDisposable
2   {
3       private const string TempFileName = "TestTemp.tmp";
4
5       private readonly string _tempFilePath;
6
7       private StreamWriter _writeStream;
8
9       public TestClass()
10      {
11          _tempFilePath = Path.Combine(Path.GetTempPath(), TempFileName);
12          _writeStream = File.CreateText(_tempFilePath);
13      }
14
15      // Class logic
16
17      private void ReleaseResources()
18      {
19          if (File.Exists(_tempFilePath))
20          {
21              File.Delete(_tempFilePath);
22          }
23          else
24          {
25              throw new FileNotFoundException();
26          }
27      }
28
```

```
29      private void Dispose(bool disposing)
30      {
31          ReleaseResources();
32          if (disposing)
33          {
34              _writeStream?.Dispose();
35          }
36      }
37
38      public void Dispose()
39      {
40          Dispose(true);
41          GC.SuppressFinalize(this);
42      }
43
44      ~TestClass()
45      {
46          Dispose(false);
47      }
48  }
```

Приклад вразливості

```
string text = "";
try
{
    text = File.ReadAllText(fileName);
```

```
public class SqlInjection : Controller
{
  private readonly UsersContext _context;

  public SqlInjection(UsersContext context)
  {
    _context = context;
  }

  public IActionResult Authenticate(string user)
  {
    var query = "SELECT * FROM Users WHERE Username = '" + user + "'";
    var userExists = _context.Users.FromSql(query).Any();
    return Content(userExists ? "success" : "fail");
  }
}
```

‘ OR 1=1 OR ‘’=‘

```
string text = "";
try
{
    text = File.ReadAllText(fileName);
```

```
public class SqlInjection : Controller
{
  private readonly UsersContext _context;

  public SqlInjection(UsersContext context)
  {
    _context = context;
  }

  // GET /SqlInjection/Authenticate
  public IActionResult Authenticate(string user)
  {
    var query = "SELECT * FROM Users WHERE Username = {0}"; // Safe
    var userExists = _context.Users.FromSql(query, user).Any();
    return Content(userExists ? "success" : "fail");
  }
}
```

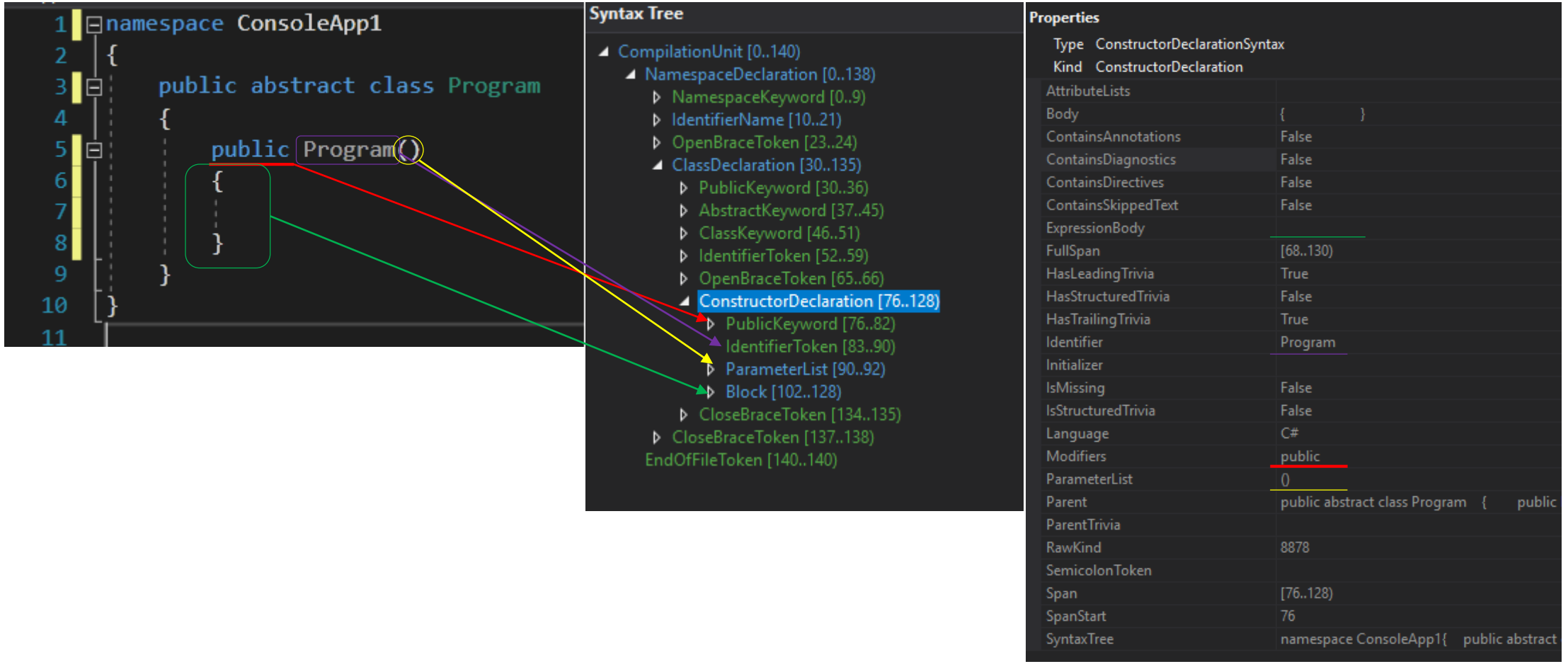
SELECT * FROM Users WHERE Username = ‘’ OR 1=1 OR ‘’=‘’

Приклад Code Smell

- Дублювання коду
- Довгі методи
- Великі класи
- Багато параметрів
- Залежності в методах
- Паралельні ієрархії наслідування
- Мертвий код
- • •

```
public AcxZonesViewModel(
        IAcxDataModel dataModel,
        IStudioZoneSet studioZoneSet,
        IEventAggregator eventAggregator,
        IWellServiceProvider wellServiceProvider,
        IZonesUndoRedoManager<ITimeZoneEntity, IParameterEntity> zonesU
        ISampleParameters sampleParameters,
        IConstantsService constantsService,
        IDispatcherService dispatcherService,
        IToastMessageService messageService,
        IZoneValidatorFactory zoneValidatorFactory)
        : base(string.Empty, studioZoneSet, eventAggregator, wellServic
    {
        _dataModel = dataModel;
        _studioZoneSet = studioZoneSet;
        _eventAggregator = eventAggregator;
        _sampleParameters = sampleParameters;
        _messageService = messageService;
        _zoneValidatorFactory = zoneValidatorFactory;
```

Приклад визначеної структури коду



Приклад ініціалізації правила

```
24      protected override void Initialize(CodeAnalysisContext context)
25      {
26          context.RegisterSyntaxNodeActionInNonGenerated(
27              c =>
28              {
29                  var ctorDeclaration = (ConstructorDeclarationSyntax)c.Node;
30
31                  var isAbstractClass = c.Node.Parent is ClassDeclarationSyntax classDeclaration &&
32                      classDeclaration.Modifiers.Any(SyntaxKind.AbstractKeyword);
33
34                  var invalidAccessModifier = ctorDeclaration.Modifiers.FirstOrDefault(
35                          m => m.IsKind(SyntaxKind.PublicKeyword) ||
36                               m.IsKind(SyntaxKind.InternalKeyword));
37
38                  if (isAbstractClass && !invalidAccessModifier.IsKind(SyntaxKind.None))
39                  {
40                      c.ReportDiagnosticWhenActive(Diagnostic.Create(rule, invalidAccessModifier.GetLocation()));
41                  }
42              }, SyntaxKind.ConstructorDeclaration);
43      }
```

Результат перевірки

Код для аналізу

```
1  namespace ConsoleApp1
2  {
3      public abstract class Program
4      {
5          public Program()
6          {
7
8          }
9      }
10 }
11
```

Абстрактний клас не повинен мати публічних конструкторів, оскільки екземпляр такого класу не може бути створений.

"abstract" classes should not have "public" constructors

Since `abstract` classes can't be instantiated, there's no point in their having `public` or `internal` constructors. If there is basic initialization logic that should run when an extending class instance is created, you can by all means put it in a constructor, but make that constructor `private` or `protected`.

Noncompliant Code Example

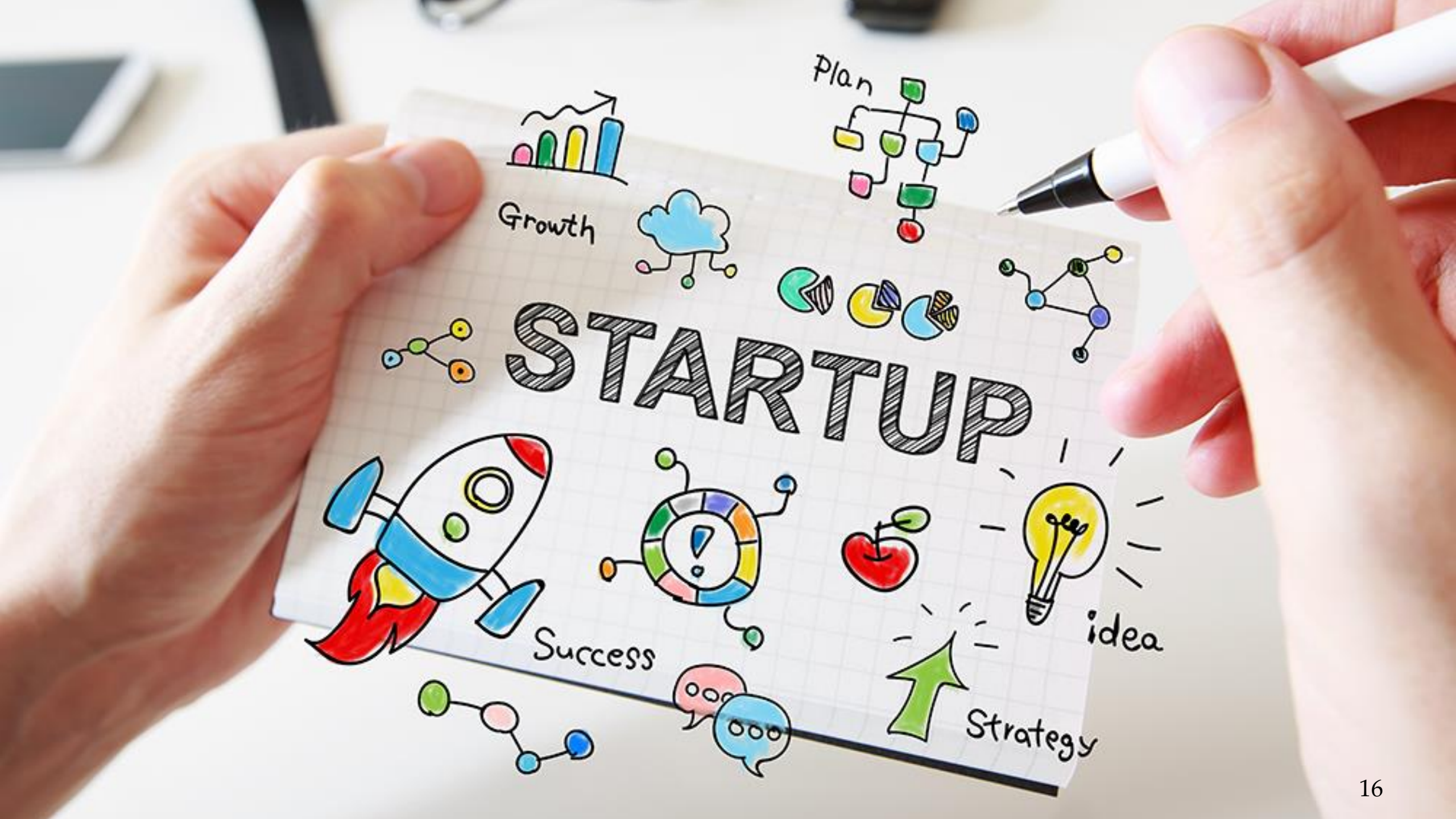
```
abstract class Base
{
    public Base() // Noncompliant, should be private or protected
    {
      //...
    }
}
```

Compliant Solution

```
abstract class Base
{
    protected Base()
    {
      //...
    }
}
```

Висновки

- Було проаналізувати існуючі засоби рефакторингу
- Було проаналізувати існуючі генератори програмного коду
- Досліджено існуючі метрики коду та їх актуальність
- Удосконалено підхід до автоматизації рефакторингу
- Проаналізовано існуючі реалізації засобів автоматизації рефакторингу програмного коду
- Розроблено програмний продукт для аналізу програмного коду та пошуку можливих проблем в ньому


Plan
Growth
STARTUP
Success
idea
Strategy

Підсумовування

- Використовує передову систему попередньої компіляції
- Готова до розширення правилами для інших мов
- Може бути використана в широкому спектрі програмних продуктів
- Перевіряє на порушення понад 350 правил, з них
  - 256 некоректностей коду (Code Smells)
  - 74 дефекти
  - 21 вразливість

Подальші дії

- Розширення набору правил
- Створення додатків для Visual Studio та VS Code
- Створення хмарного аналізатора
- Вихід на ринок

Дякую за увагу!