Spis treści

[1. Wstęp 5](#_Toc119969756)

[2. Istniejące rozwiązania 7](#_Toc119969757)

[2.1. ICloneable 7](#_Toc119969758)

[3. Wzorce projektowe 8](#_Toc119969759)

[3.1. Singleton 8](#_Toc119969760)

[3.2. Fabryka 8](#_Toc119969761)

[3.3. Fabryka abstrakcyjna 9](#_Toc119969762)

[3.4. Prototyp 9](#_Toc119969763)

[3.5. Fasada 9](#_Toc119969764)

[3.6. Pusty Obiekt 10](#_Toc119969765)

[3.7. Kontener IoC 10](#_Toc119969766)

[4. Kompilator Roslyn 11](#_Toc119969767)

[4.1. Model zestawu SDK .NET Compiler Platform 11](#_Toc119969768)

[4.2. Drzewo składni 11](#_Toc119969769)

[4.2.1. Węzły składni 11](#_Toc119969770)

[4.2.2. Tokeny składniowe 11](#_Toc119969771)

[4.2.3. Trivia składniowa 11](#_Toc119969772)

[4.3. Semantyka 12](#_Toc119969773)

[4.3.1. Kompilacja 12](#_Toc119969774)

[4.3.2. Symbole 12](#_Toc119969775)

[4.3.3. Model semantyczny 12](#_Toc119969776)

[4.4. Generator źródeł 12](#_Toc119969777)

[5. Projekt? 13](#_Toc119969778)

[6. Implementacja 14](#_Toc119969779)

[6.1. Architektura 14](#_Toc119969780)

[6.2. Użyte biblioteki 14](#_Toc119969781)

[6.3. Testy 14](#_Toc119969782)

[7. Porównanie z innymi rozwiązaniami 15](#_Toc119969783)

[8. Wyzwania 16](#_Toc119969784)

[9. Wnioski 17](#_Toc119969785)

[10. Bibliografia 18](#_Toc119969786)

[11. Spis rysunków 19](#_Toc119969787)

1. Wstęp

Celem tej pracy jest użycie kompilatora Roslyn do stworzenia rozszerzenia generującego kod typowych wzorców projektowych występujących w oprogramowaniu biznesowym. Wzorce projektowe to typowe rozwiązania problemów często napotykanych podczas projektowania oprogramowania. Każdy z nich stanowi plan, który po odpowiednim dostosowaniu pomaga poradzić sobie z konkretnym problemem w projekcie kodu. Wzorce projektowe różnią się poziomem skomplikowania, szczegółowością oraz skalą w jakiej można je zastosować.[[1]](#footnote-1) Wzorce, które zostały uwzględnione w tej pracy, zostały omówione w rozdziale 2.

Użyty w tej pracy kompilator Roslyn, zapewnia dostęp do szczegółowego modelu kodu aplikacji, który został utworzony przez kompilatory. Taki model, kompilatory tworzą, poprzez weryfikację składni i semantyki kodu aplikacji, oraz używają tego modelu do tworzenia wykonywalnych danych wyjściowych z kodu źródłowego.

Przez ostatnie lata, aby zwiększyć produktywność, coraz częściej programiści używają zintegrowanych funkcji środowiska projektowego (IDE), takich jak: IntelliSense, refaktoryzacja, inteligentna zmiana nazwy, czy wyszukiwania referencji. Korzystają oni z narzędzi do analizy kodu, aby poprawić jakość oraz z generatorów kodu, aby ułatwić tworzenie aplikacji. Z czasem, narzędzia te, stają się inteligentniejsze. Potrzebują dostępu do coraz większej liczby modeli tworzonych przez kompilatory podczas przetwarzania kodu aplikacji. Jest to podstawowa funkcja kompilatora Roslyn, czyli udostępnianie użytkownikom końcowym informacji o kodzie aplikacji.[[2]](#footnote-2) Pojęcia związane z Roslyn zostały omówione w rozdziale 3.

Generatory źródeł, w ramach Roslyn APIs, umożliwiają programistom C# sprawdzanie kodu podczas kompilowania[[3]](#footnote-3). Mogą tworzyć nowe pliki źródłowe języka C#, które są na bieżąco dodawane do kompilacji użytkownika. Generatory umożliwiają wykonanie dwóch głównych czynności:

1. pobieranie obiektu kompilacji, który reprezentuje cały kod użytkownika,
2. generowanie plików źródłowych języka C#, które można dodać do obiektu kompilacji podczas kompilacji.

Więcej informacji na temat generatorów źródeł zostało zamieszczone w rozdziale 3.3.

1. Istniejące rozwiązania
   1. ICloneable

Wzorzec prototyp jest dostępny w C# od razu dzięki interfejsowi IClonable. Obsługuje on klonowanie, które tworzy nowy obiekt klasy o tej samej wartości co istniejący obiekt. Interfejs zawiera tylko metodę Clone, która ma zapewnić klonowanie poza elementem dostarczonym przez Object.MemerwiseClone.[[4]](#footnote-4)

1. Wzorce projektowe
   1. Singleton

Singleton to kreacyjny wzorzec projektowy, którego celem jest ograniczenie możliwości tworzenia obiektów danej klasy do jednej instancji. Ponadto singleton zapewni globalny dostęp do stworzonego obiektu.

Wzorca Singleton należy używać w następujących warunkach:

1. jeśli musi istnieć dokładnie jeden egzemplarz klasy dostępny klientom w znanym miejscu,
2. kiedy potrzebna jest możliwość rozszerzenia jedynego egzemplarza przez tworzenie podklas, a klienci powinni móc korzystać ze wzbogaconego egzemplarza bez konieczności wprowadzania zmian w ich kodzie.[[5]](#footnote-5)

Wzorzec ten zapewnia kilka korzyści:

1. zapewnia kontrolę dostępu do jedynego egzemplarza,
2. pozwala zmniejszyć przestrzeń nazw,
3. umożliwia dopracowywanie operacji i reprezentacji,
4. umożliwia określenie dowolnego limitu liczby egzemplarzy,
5. jest bardziej elastyczny od operacji statycznych.[[6]](#footnote-6)

Singleton to najbardziej znienawidzony wzorzec projektowy w historii stosowania wzorców.[[7]](#footnote-7)

* 1. Fabryka

Fabryka to kreacyjny wzorzec projektowy, który udostępnia interfejs do tworzenia obiektów w ramach klasy bazowej. Jest to niejednoznaczny termin mogący oznaczać metodę lub klasę która ma za zadanie coś wytworzyć. W tej pracy została użyta Fabryka prosta[[8]](#footnote-8), która przedstawia klasę posiadającą jedną metodę kreacyjną z obszerną instrukcją warunkową, która na podstawie parametrów metody decyduje jakiej klasy produktu instancje stworzyć i zwrócić.

* 1. Fabryka abstrakcyjna

Fabryka abstrakcyjna to kreacyjny wzorzec projektowy, którego celem jest dostarczenie interfejsu do tworzenia różnych obiektów jednego typu dla tej samej rodziny klas bez specyfikowania ich konkretnych klas.

Ten wzorzec należy stosować w następujących warunkach:

1. kiedy system powinien być niezależny od sposobu tworzenia, składania i reprezentowania jego produktów,
2. jeśli system należy skonfigurować za pomocą jednej z wielu rodzin produktów,
3. jeżeli powiązane obiekty-produkty z jednej rodziny są zaprojektowane do wspólnego użytku i trzeba wymusić jednoczesne korzystanie z tych obiektów,
4. kiedy programista chce udostępnić klasę biblioteczną produktów i ujawnić jedynie ich interfejsy, a nie implementacje.[[9]](#footnote-9)

Wzorzec Fabryka abstrakcyjna ma następujące zalety i wady:

1. izoluje klasy konkretne,
2. ułatwia zastępowanie rodzin produktów,
3. ułatwia zachowanie spójności między produktami,
4. utrudnia dodawanie obsługi produktów nowego rodzaju.
   1. Prototyp
   2. Fasada
   3. Pusty Obiekt
   4. Kontener IoC
5. **Kompilator** Roslyn

Roslyn to implementacja typu open-source kompilatorów C#, jak i Visual Basic, który umożliwia tworzenie analizatorów i poprawek kodu. Analizatory interpretują składnię i semantykę w celu wykrywania błędów, które należy poprawić. Poprawki kodu natomiast zawierają co najmniej jedną sugerowaną poprawkę do rozwiązywania błędów kodowania znalezionych przez analizatory lub diagnostykę kompilatora.[[10]](#footnote-10)

* 1. Model zestawu SDK .NET Compiler Platform

Kompilatory przetwarzają kod, zgodnie z regułami ustrukturyzowanymi, które często różnią się od sposobu odczytywania i zrozumienia kodu przez ludzi. Podstawowa wiedza na temat modelu używanego przez kompilatory jest niezbędna do zrozumienia interfejsów API używanych podczas kompilowania narzędzi opartych na platformie Roslyn.

Zestaw SDK kompilatora platformy .NET składa się z kilku warstw interfejsów API:

1. interfejsów API kompilatora,
2. interfejsów API diagnostycznych,
3. interfejsów API skryptorów,
4. interfejsów API obszarów roboczych.[[11]](#footnote-11)
   1. Drzewo składni

Drzewo składni jest podstawową niezmienną strukturą danych uwidacznianą przez interfejsy API kompilatora. Reprezentuje ono leksykalną i składniową strukturę kodu źródłowego. Każde drzewo składni składa się z węzłów, tokenów i triwii.

* + 1. Węzły składni

Węzły składni są jednym z podstawowych elementów drzew składniowych. Reprezentują one konstrukcje składniowe, takie jak deklaracje, instrukcje, klauzule i wyrażenia.

* + 1. Tokeny składniowe

Tokeny składniowe reprezentują najmniejsze fragmenty składni kodu. Składają się one ze słów kluczowych, identyfikatorów, literałów i interpunkcji. Nigdy nie są one rodzicami innych węzłów lub tokenów.

* + 1. Trivia składniowa

Trivia składniowa reprezentuje część tekstu źródłowego, które w dużej mierze są nieistotne dla działania kodu, takie jak białe znaki, komentarze czy dyrektywy preprocesora. Trivia, podobnie jak tokeny, to typy wartości.

* 1. Semantyka
     1. Kompilacja
     2. Symbole
     3. Model semantyczny
  2. Generator źródeł

1. Projekt?
2. Implementacja
   1. Architektura
   2. Użyte biblioteki
   3. Testy

1. Porównanie z innymi rozwiązaniami
2. Wyzwania
3. Wnioski
4. Bibliografia

Freeman, E., Robson, E., Bates, B. i Sierra, K. (2004). *Head First Design Patterns. A Brain-Friendly Guide.* (P. Koronkiewicz i G. Kowalczyk, Tłumacze)

Gamma, E., Helm, R., Johnson, R. i Vlissdes, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* (T. Walczak, Tłum.) Addison-Wesley Professional.

Micorosoft Corporation. (2021, Wrzesień 15). *Microsoft - The .NET Compiler Platform SDK*. Pobrano z lokalizacji https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/

Micorosoft Corporation. (2022, Październik 7). *ICloneable*. Pobrano z lokalizacji https://learn.microsoft.com/en-us/dotnet/api/system.icloneable.clone?view=net-6.0

Micorosoft Corporation. (2022, Czerwiec 06). *Microsoft - Source Generators*. Pobrano z lokalizacji https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/source-generators-overview

Microsoft Corporation. (2021, Wrzesień 15). *Microsoft Corporation - Omówienie modelu zestawu SDK .NET Compiler Platform*. Pobrano z lokalizacji https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/compiler-api-model

Nesteruk, D. (2019). *Design Patterns in .NET: Reusable Approaches in C# and F# for Object-Oriented Software Design.* (R. Meryk, Tłum.) APress Media.

1. Spis rysunków

1. (Gamma, Helm, Johnson i Vlissdes, 1995, str. 16) [↑](#footnote-ref-1)
2. (Micorosoft Corporation, 2021) [↑](#footnote-ref-2)
3. (Micorosoft Corporation, 2022) [↑](#footnote-ref-3)
4. (Micorosoft Corporation, 2022) [↑](#footnote-ref-4)
5. (Gamma, Helm, Johnson i Vlissdes, 1995, str. 130) [↑](#footnote-ref-5)
6. (Gamma, Helm, Johnson i Vlissdes, 1995, str. 131) [↑](#footnote-ref-6)
7. (Nesteruk, 2019, str. 69) [↑](#footnote-ref-7)
8. (Freeman, Robson, Bates i Sierra, 2004) [↑](#footnote-ref-8)
9. (Gamma, Helm, Johnson i Vlissdes, 1995, str. 102) [↑](#footnote-ref-9)
10. (Micorosoft Corporation, 2021) [↑](#footnote-ref-10)
11. (Microsoft Corporation, 2021) [↑](#footnote-ref-11)