Работа с чужд код

Съдържание

- Запознаване с проект
- Картографиране на проект
- Рефакториране

Запознаване с проект

- Идентифициране на словете
- Изследване структурата на всеки слой
- Изследване на комуникацията между словете
- Добавяне на нова функционалност на база проучването

Запознаване с проект – примерен процес

- Отпред назад
  - Изследвайте потребителския интерфейс – какво изпитва потребителя при работата си със софтуера
  - Какъв е първия изглед, който се показва?
  - Изисква ли се регистрация?
  - За какво устройство е направен софтуера?
  - На къде води всеки линк?
  - Какво се показва в браузърния инспектор?

Запознаване с проект – примерен процес

- Инструменти и технологии
  - Имате ли всички инструменти, нужни, за да се подкара проекта на вашата машина?
  - Ако разполагате с по-нови/по-стари инструменти, ще можете ли да подкарате проекта?

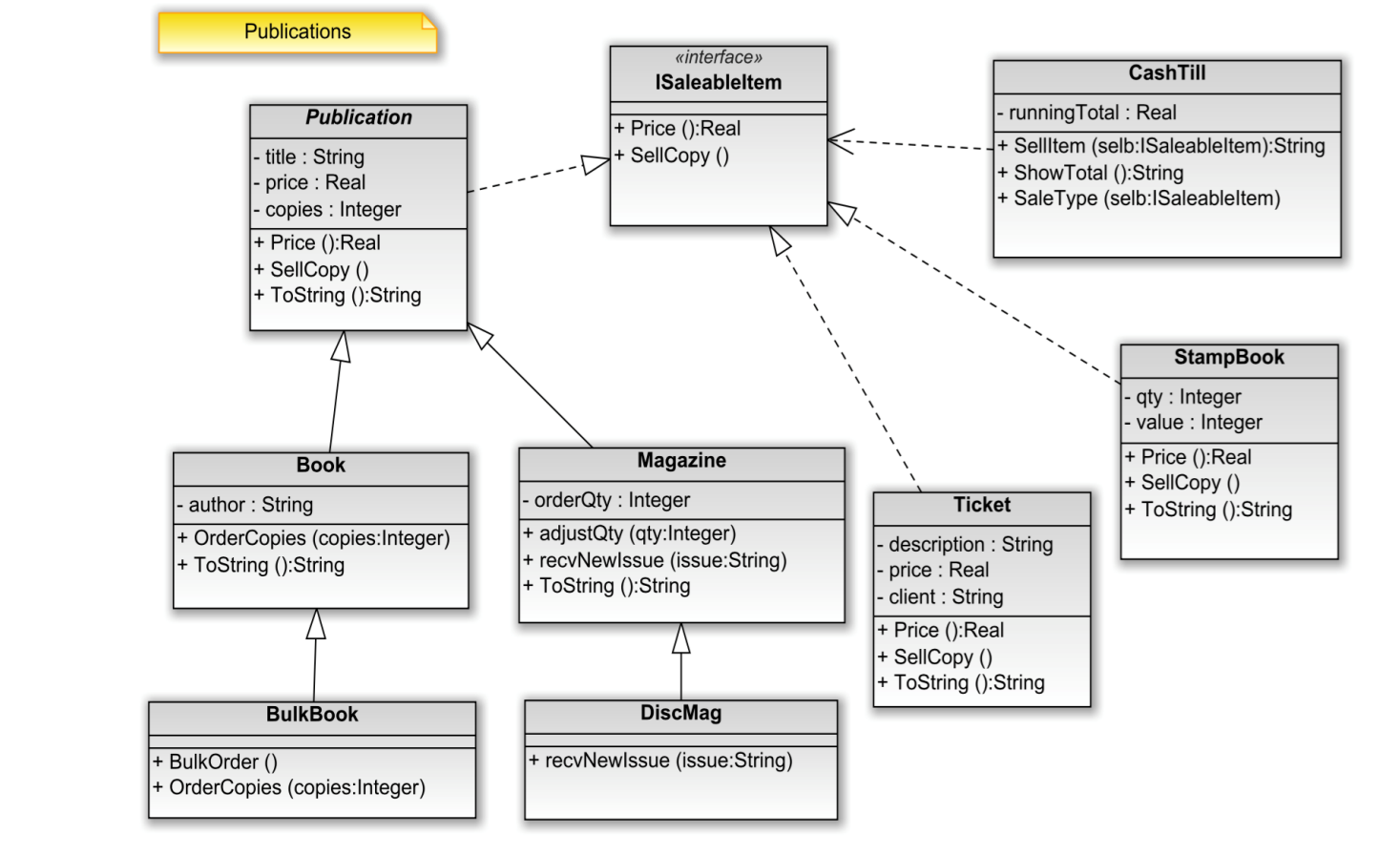
Запознаване с проект – примерен процес

- Обмен на данни
  - Как се показват данните на потребителския интерфейс?
  - Как се рендират данните?
  - Как се управлява състоянието? От едно място ли се управлява или от няколко?
  - Как се предават данните?
  - Под каква форма трябва да са данните?
  - Как се запазват данните и къде?

Картографиране на проект

- Клас-диаграма – тип статична диаграма, описваща структурата на дадена система, като изобразява класовете на системата, атрибутите им, методите им и връзките помежду им
  - Лесно запознаване със структурата на системата
  - Бизнес аналитиците могат да използват клас-диаграмите, за да моделират системите от бизнес перспектива

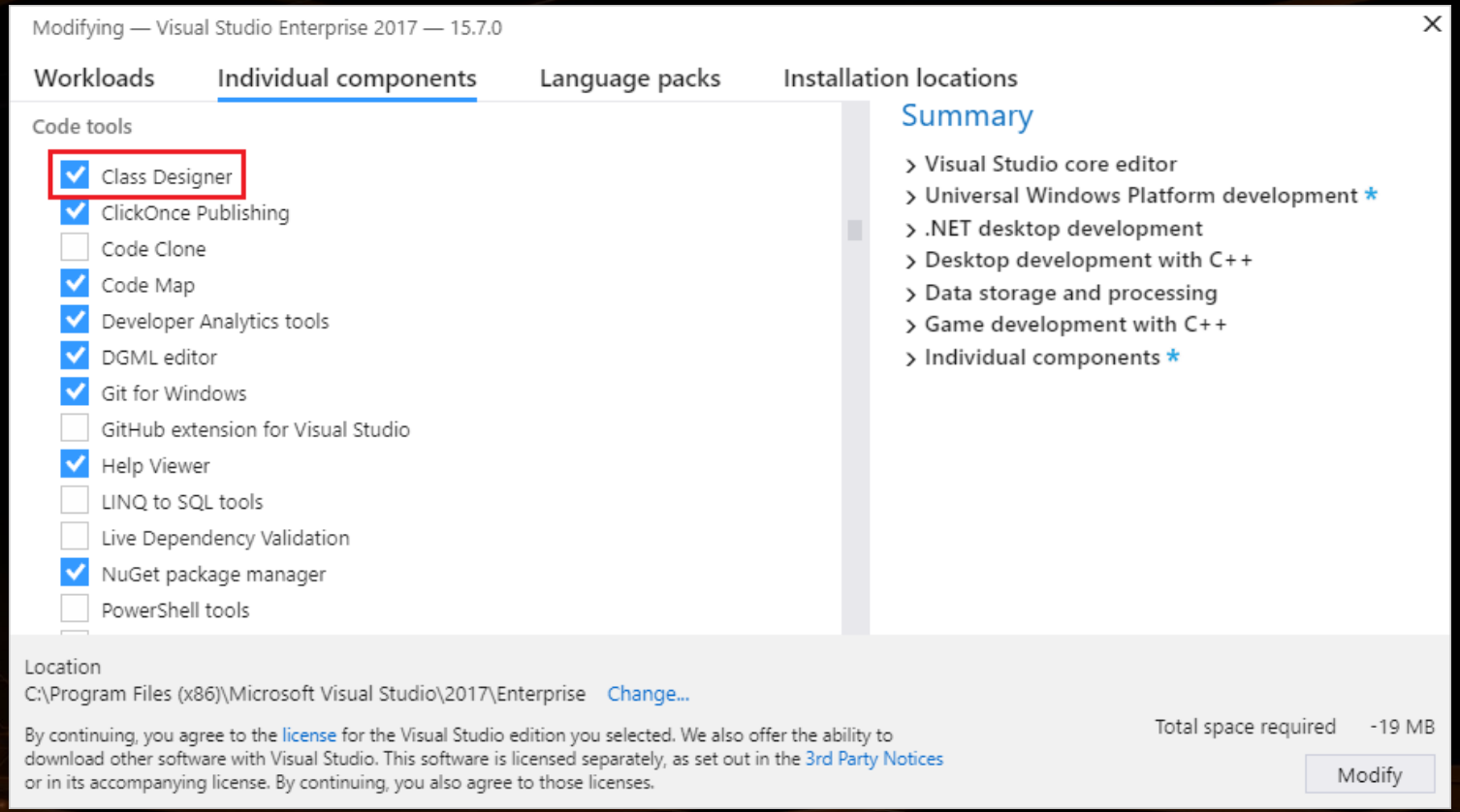
Картографиране на проект



Картографиране на проект – Visual Studio

- Добавяне на клас-диаграма към проект с Visual Studio
- Инсталация на Class Designer компонента
  - От Visual Studio Installer изберете Tools > Get Tools and Features
  - Изберете Individual Components и навигирайте до Code Tools
  - Изберете Class Designer, след което Modify

Картографиране на проект – Visual Studio



Картографиране на проект – Visual Studio

- След успешна инсталация чрез Visual Studio изберете с десен бутон Solution Explorer-а, след което изберете View > View Class Diagram
  - Вашата диаграма трябва да се отвори

Забележка: .NET Core проектите не поддържат Class Designer

Рефакториране

- Рефакториране (нарицателно) – промяна във вътрешната структура на софтуера, въведена с цел да се улесни разбирането на кода и да се направи по-лесно въвеждането на нови промени, но без да се променя наблюдаваното поведение

Рефакториране – защо да го правим?

- Улеснява въвеждането на бъдещи промени
- Подобрява производителността
- Рефакторирането е редактиране

Рефакториране – кога да го правим?

- След написването на unit тест (Test-Driven Development)
- Ако текущото състояние на кода причинява неудобство (Pain-Driven Development)
- Като част от поправянето на грешки (bug fixing)

Рефакториране – кога да не го правим?

- Кода не работи в текущото си състояние
- При изоставане от крайния срок за завършване на проекта

Рефакториране – процесът

- Остави кода по-добър, отколкото го намери

1. Запазва се работеща версия на проекта
2. Проверява се текущото поведение (по възможност с автоматизирани тестове)
3. Рефакториране
4. Проверява се запазено ли е първоначалното поведение

Рефакториране – характеризиращи тестове

1. Пише се тест, за който е сигурно, че ще се провали
2. Използва се изхода от провалення тест, за да се хване текущото поведение
3. Променя се теста, за да съвпада с текущото поведение
4. Теста се провежда отново, като този път трябва да премине

Рефакториране – по-чист код

- Премахване на дупликацията
- Подобряване на именуването
- Разбиване на големи сегменти код
- Редуциране на зависимостите
- Редуциране на комплексността
- Разделяне на отговорностите

Рефакториране – проблеми (code smells)

- Проблем, изискващ рефакторирането на код (code smell) – повърхностна индикация, която обикновено сигнализира за по-дълбок проблем в системата

Рефакториране – принцип на най-малката изненада

- Правете това, което потребителя очаква
- Изграждайте API-та от перспективата на разработчиците, които ще ги използват
- Пишете прост код
- Пишете ясен код

Рефакториране – класификация на проблемите

- Набухватели (bloaters)
- Насилници на обектно-ориентираната парадигма (object-orientation abusers)
- Възпрепятстващи промени (change preventers)
- Ненужни (dispensables)
- Обвързващи (couplers)
- Разсейващи (obfuscators)

Рефакториране – bloaters

- Правят кода излишно голям
- Обикновено влиянието им се усеща на по-късен етап
- Предотвратяват се с писането на чист, фокусиран код

Рефакториране – object-orientation abusers

- Нарушават полиморфизма
- Създават излишни и неподходящи зависимости
- Изискват повторение

Рефакториране – change preventers

- Влияят на голяма част от системата
- Създават тесни зависимости
- Липсва разпределение на отговорностите

Рефакториране – dispensables

- Допринасят с много малка или никаква стойност
- Могат безопасно да се премахнат

Рефакториране – couplers

- Въвеждат прекомерни зависимости
- Свързват части от системата, които нямат нищо общо

Рефакториране – obfuscators

- Пречат на ясната комуникация
- Объкрват читателя
- Скриват намеренията

Рефакториране – организация на проблемите според C# йерархията

- Операторни проблеми (statement smells)
- Методни проблеми (method smells)
- Класови проблеми (class smells)

Рефакториране – statement smells

- Примитивна мания [primitive obsession] (bloater) – прекомерна употреба на примитивни типове, вместо изпозлването на по-добри абстракции или типове данни, което изисква повече код, за да се приложи дадено ограничение

Рефакториране – statement smells

- Вертикално разделяне [vertical separation] (obfuscator) – дефинирайте и използвайте променливите и функциите близо до местата, където се ползват
- Инициализирайте локални променливи там, където за пръв път се използват
- Дефинирайте private функции под мястото, където се използват
- Избягвайте да карате читателя да скролва

Рефакториране – statement smells

- Неконсистентност [inconsistency] (obfuscator) – бъдете постоянни в именуването, форматирането и използването на шаблони в рамките на дадено приложение

Рефакториране – statement smells

- Лошо именуване [poor naming] (obfuscator) – Не доброто именуване е често посочвано като един от най-трудните проблеми в компютърните науки
- Използвайте описателни имена и избягвайте абревиатури и криптиране, когато е възможно

Рефакториране – statement smells

- Характеристики за идеално именуване:
  - Описателно
  - Подходящо ниво на абстракция
  - Следване на стандарти
  - Недвусмислено
  - По-дълги имена при по-големи полета на живот (scopes)
  - Без съкращения

Рефакториране – statement smells

- [switch statements] (object-orientation abuser) – switch оператори и комплексни вериги от if-else могат да са знак за неподходящо използване на обектно-ориентиран дизайн

Рефакториране – statement smells

- Повтарящ се код [duplicate code] (disposable) – повтарящият се код е корена на всяко зло за софтуера
- Следвайте DRY принципа (Don’t Repeat Yourself), т.е. избягвайте повторения в кода си

Рефакториране – statement smells

- Мъртъв код [dead code] (disposable) – отървете се от безполезения код, който никога не се извиква – той не добавя никаква стойност

Рефакториране – statement smells

- Скрита зависимост на последователността [hidden temporal coupling] (coupler) – определени операции трябва да се свършат в дадена последователност или няма да работят
- Нищо в дизайна не налага тази последователност като задължителна – разработчиците трябва сами да я открият от контекста

Рефакториране – method smells

- Голям метод [long method] (bloater) – предпочитайте по-кратки методи
- Малките методи имат по-добри имена, защото правят по-малко
- След като могат да бъдат добре именувани, по-кратки са и не правят много – малките методи са по-лесни за разбиране

Рефакториране – method smells

- Колко малко е „малко“?
  - В идеалния случай методите трябва да се събират на екрана (без скролване)
  - Около 10 реда код

Рефакториране – method smells

- Сложна условност [conditional complexity] (change preventer) – методите трябва лимитират количеството условност, което съдържат
- Броя на уникални логически разклонения в метод може да се измери като цикломатическа сложност, която трябва да е не повече от 10
  - Може да използвате Visual Studio, за да измерите цикломатическата сложност, като от Solution Explorer изберете проект -> десен бутон -> Analyze and Code Cleanup -> Calculate Code Metrics

Рефакториране – method smells

- Неконсистентно ниво на абстракция [inconsistent abstraction level] (change preventer) – методите трябва да работят на консистентно ниво абстракция – не смесвайте високото с ниското ниво на поведение в един и същи метод

Рефакториране – специфично за методи

- Изкарване на метод [extract method]
  1. Идентифицирайте кода за изкарване в нов метод
  2. Създайте нов метод с подходящо име
  3. Копирайте кода от източника в новия метод
     1. Временните променливи, използвани само в този код могат също да бъдат копирани
  4. Идентифицирайте промененните локални променливи
     1. Трябва ли новия метод да връща стойност?
     2. Ако е повече от една, помислете за изкарване на по-малък метод

Рефакториране – специфично за методи

- Изкарване на метод [extract method]
  5. Компилирайте
  6. Сменете изкараният код с извикване на новия метод
  7. Компилирайте и тествайте

Рефакториране – специфично за методи

- Преименуване на метод [rename method]
  1. Имплементиран ли е метода от под-/надкласове
     1. Ако е, повторете стъпките за всяка имплементация
  2. Създайте нов метод със същите параметри и новото име
  3. Копирайте (не изрязвайте) тялото на стария метод в тялото на новия
  4. Компилирайте
  5. Променете стария метод така, че да извиква новия (опционално)
  6. Компилирайте и тествайте
  7. Намерете всички извиквания на стария метод и ги променете към новия
  8. Премахнете стария метод. Компилирайте и тествайте

Рефакториране – специфично за методи

- Въвеждане на поясняваща променлива
  1. Декларирайте променлива и задайте за стойността ѝ част от сложен израз
     1. Бъдете сигурни, че е добре именувана
  2. Сменете частта от израза с променливата
  3. Компилирайте и тествайте

При нужда повторете и за други части от израза

Рефакториране – специфично за методи

- Смяна на поясняващата променлива със заявка
  1. Потвърдете, че на променливата се задава стойност само веднъж
  2. Вземете дясната част от израза и я превърнете в метод
  3. Компилирайте и тествайте
  4. Сменете всички референци на променливата с извикявания към метода
  5. Компилирайте и тествайте
  6. Премахнете променливата

Рефакториране – class smells

- Голям клас [large class] (bloater) – големия клас най-вероятно има прекано много отговорности и може да бъде разбит на два или три по-малки, по-фокусирани класове
  - Извлечете метод [extract method]
  - Извлечете клас
  - Извлечете подклас или интерфейс

Рефакториране – class smells

- Клас, който не прави много [class doesn’t do much] (bloater) – понякога след рефакториране даден клас не прави много
- Ако това, за което класа служи в сегашното си състояние има по-голяма логика да се извършва другаде – извлечете го и изтрийте класа
- Мързелив клас [lazy class] (dispensable) – клас, който не върши достатъчно, за да оправдае съществуването си

Рефакториране – class smells

- Временно поле [temporary field] (object-orientation abuser) – клас, който има поле, използвано само в определени случаи, обикновено, за да се предаде състояние между методи, вместо да се използват параметри

Рефакториране – class smells

- Алтернативни класове с различни интерфрейси [common classes with different interfaces] (object-orientation abuser) – общите операци трябва да споделят общи семантики като имена, параметри и ред на параметрите

Рефакториране – class smells

- Паралелна йерархия на наследяване [parallel inheritance hierarchy] (change preventer) – две концепции са моделирани с наследствена йерархия – всяка промяна в един наследик изисква съответната промяна и в друг наследник

Рефакториране – class smells

- Клас за данни [data class] (dispensable) – клас, който няма никакво поведение, а съдържа само свойства и полета

- Забележка: Не става въпрос за класовете, които Entity Framework използва или за DTO класове

Рефакториране – class smells

- Скрита зависимост [hidden dependency] (coupler) – класове, имащи външни зависимости, които не са оказани в конструктора им

Рефакториране – специфично за класове

- Енкапсулиране на поле
  1. Създайте get и set методи за полето
     - Опционално използвайте автоматично свойство
  2. Намерете всички референции към полето
  3. Променете всички референции така, че да използват полето
  4. Компилирайте след всяка промяна
  5. Променете полето на private
  6. Компилирайте и тествайте
- Бъдете внимателни при премахването на get/set метод – Entity Framework изисква set (дори и да е private)

Рефакториране – специфично за класове

- Енкапсулиране на колекция
  1. Добавете изрични Add/Remove методи
  2. Инициализирайте полето с празна колекция
  3. Компилирайте
  4. Намерете референциите, които задават стойност на колекцията
     1. Модифицирайте така, че да използват Add/Remove методите
  5. Компилирайте и тествайте
  6. Намерете референциите, които извличат и после променят колекцията. Компилирайте и тествайте
  7. Модифицирайте get метода така, че да връща read-only колекция. Компилирайте и тествайте

Обобщение

- Запознаване с проект
- Картографиране на проект
- Рефакториране