Гибкая модульность инструментами АОП

Вынос сквозной функциональности.

Цели

- Дать определение АОП
- Определить какие проблемы решает АОП
- Построить модульную систему применяя АОП
- Сравнить динамическое и статическое связывание
- Дать рекомендации по использованию АОП

Для чего нужно АОП

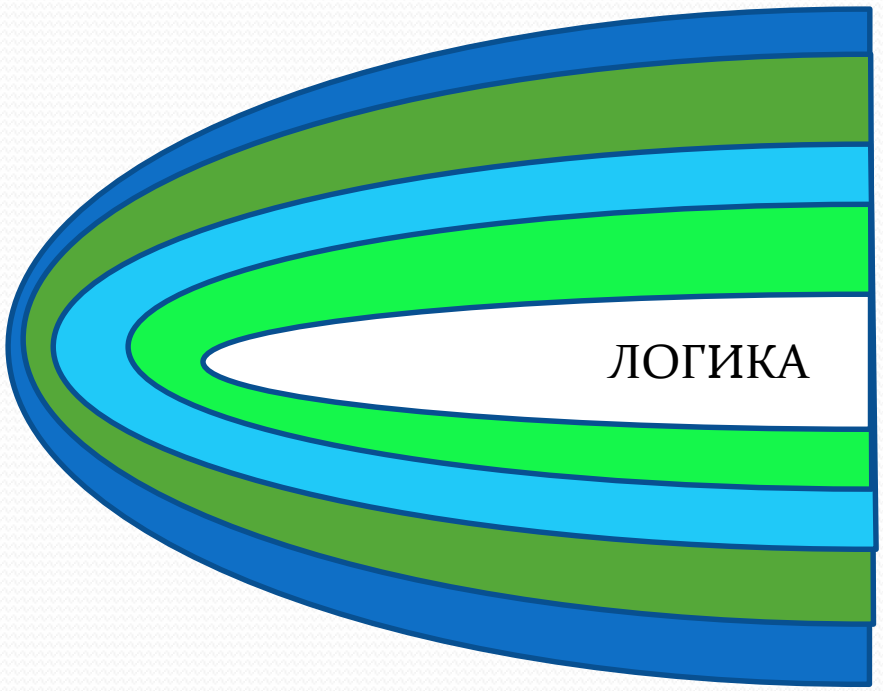
- Пересекающаяся логика она же сквозная функциональность – части программы присутствующие «везде», но не относящиеся к конкретному классу.
- АОП позволяет вынести сквозную функциональность в отдельные модули - аспекты.
- Аспекты можно включить или отключить, без «переписывания» кода.

Сквозная функциональность

- Журнал – он же логгер
- Профилирование
- Проверка прав доступа
- Кэш
- INotifyPropertyChanged
- И все что не относится к логике приложения.

Проблема сквозной функциональности

- Ctrl+C -> Ctrl+V – в народе копипаст
- Сомневающийся заказчик – изменяющиеся требования.
- Тяжело разбираться



.

Проблема сквозной функциональности

- Чистая функция

```
public double Up(double value)
{
  if (value - (int)value > 0.764131 && value - (int)value < 0.764133)
        Thread.Sleep(15000);
  return value /= 3;
}
```

- Спрятанная функция

```
public double Up(double value)
{
   var sw = new Stopwatch();
   sw.Start();

   if (value - (int)value > 0.764131 && value - (int)value < 0.764133)
        Thread.Sleep(15000);
   var result = value /= 3;

   sw.Stop();
   Logger.Info($"Recalc value:{value} elapsed:{sw.Elapsed.Seconds}", result);

   return result;
 }
```

Истоки АОП

- 2001 год компания Xerox – расширение для Java AspectJ.
- ***Аспект*** — модуль или класс, реализующий сквозную функциональность.
- ***Совет*** — реализация сквозной функциональности.
- ***Точка соединения*** — точка в выполняемой программе, где следует применить совет.
- ***Срез*** — набор точек соединения.
- ***Связывание (внедрение)***— смешивание аспекта с выполняемым кодом.

Как это работает
Два типа связывания в АОП

- Статическое – путем изменения кода приложения.

  Пост обработка, влияет на время компиляции.
- Динамическое – на лету. Шаблон Proxy. Dynamic Proxy генерируются на этапе выполнения. Проигрывает по производительности и возможностям, но выигрывает по гибкости изменений.

Статическое Post связывание

Я использую переводчик, чтобы показать принцип статического связывания в АОП

Пост обработка исходного кода

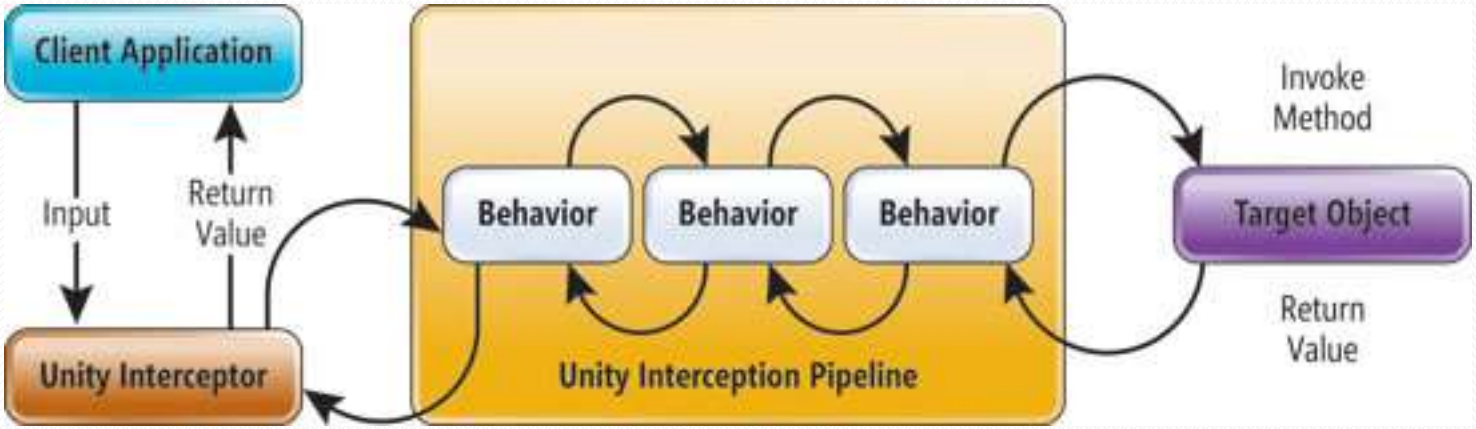
私はAOPでインタプリタを使用して静的バインディングの原則を示します

Попытка восстановления исходного кода

Я использую интерпретатор в АОП, чтобы показать принцип статической привязки

Interception

- Перехват



АОП и .Net

- Можно написать, CIL в помощь
- PostSharp
- Fody
- Spring .Net
- Unity IoC
- Ninject
- Autofac
- Castle.Project (Windsor, DynamicProxy)
- И многие другие.

Практика АОП

Пример последовательного приведения монолитной части программы к модульной архитектуре основанной на АОП. (см. AopModules.sln)

- Stage1 – этап на котором возникает проблема подвисания приложения. Код построен таким образом, что не возможно протестировать. Определить в чем проблема не удается.

Практика АОП

- Stage2 – было выделено несколько программных модулей. Модули покрыты тестами, но проблема не выявлена.
- Stage3 – в код модулей добавлен профайлер. Проблема исправлена, но код модулей загрязнен и нет возможности отключить профилирование.
- Stage4 – пересекающаяся логика вынесена в отдельные Proxy. Внутреннее состояние профилируемых объектов стало прозрачным. Профилирование легко включать и выключать. Осталось дублирование кода и при необходимости профилирования новых модулей нужно писать новые Proxy.

Практика АОП

- Stage5Factory – отказ от отдельных Proxy в пользу АОП. Код профилирования содержится в аспекте. Легко повторно пере использовать.
- Stage5IoC – большинство IoC поддерживают динамическое АОП, не стоит изобретать велосипед.
- Bonus – демонстрация дополнительных возможностей АОП. Вынос ввода вывода в аспект.

Практика АОП

- Post – пример использования АОП в статических классах с использованием PostSharp. Используются все те же Interseptor'ы. Заплатив «всего» 500$, вы получите в подарок наборы аспектов

Logging	Contracts	INotifyPropertyChanged	Weak Event
XAML	Parent/Child, Visitor and Disposable	Undo/Redo	Caching
Multithreading	Aspects Validating	Architecture	Testing and Debugging

- http://doc.postsharp.net/conceptual-documentation

Практика АОП

- Fody – пример использования INotifyPropertyChanged. Легко устанавливается через NuGet, открыт и бесплатен, очень просто использовать.
- Очень много расширений https://github.com/Fody/Fody
- АОП лишь небольшая часть проекта.

Что мешает использовать АОП

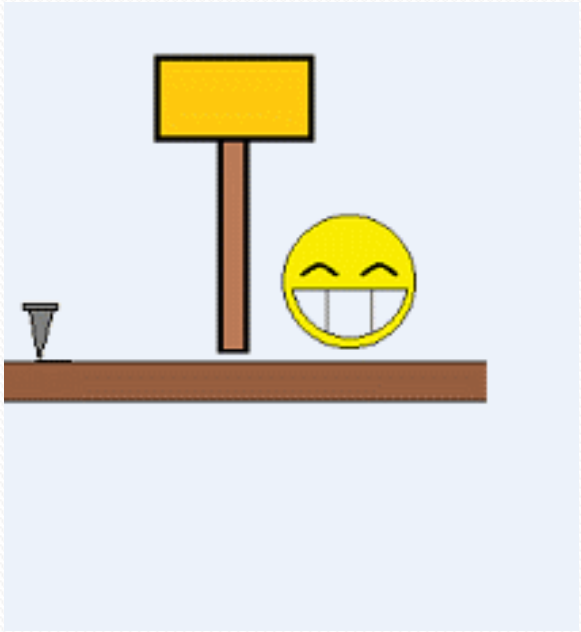
Утверждение	Реальность
АОП это сложно	Как и любая новая технология требует время на изучение
АОП не решает ни каких новых задач	АОП решает старые задачи новым способом
АОП затрудняет понимание кода	Любой дополнительный уровень абстракции усложняет понимание
Шаблоны проектирования и интерфейсы заменяют АОП	АОП хорошо дополняет модульную архитектуру
При изменении классов нужно менять аспекты	Зависит от качества проектирования. Плохое проектирование не устойчиво к изменениям
АОП не тестируемо	Аспекты тестируются точно так же как и обычные объекты

Что мешает использовать АОП

Утверждение	Реальность
АОП способствует плохому проектированию	Нужно понимать что АОП – не золотой молоток
АОП нельзя внедрять в уже разработанный продукт	АОП можно внедрять на любой стадии разработки

Вывод по внедрению АОП

- Стадия Bonus – хороший пример плохого проектирования и использования АОП в качестве Gold Hummer. Ввод вывод не является сквозной функциональностью.



Вывод по внедрению АОП

- Стадия Post - С помощью статического связывания можно смешать аспект даже со статическими классами. Такой подход требует минимальных усилий, но в общем случае является сырым проектированием.
- Стадия Fody – при внедрении АОП таким образом нужно соблюдать определенные правила, без которых система работать не будет.

Когда не стоит использовать АОП

- Функциональность не сквозная или используется в ограниченном числе модулей.
- Необходимо работать с внутренним состоянием объектов.
- Команда не готова. Нужно ориентироваться на тех кто это будет сопровождать.

Как использовать АОП

- Разделять крупные модули на более мелкие.
- Инвертировать зависимости модулей.
- Делать внутреннее состояние прозрачным.
- Использовать Proxy классы для реализации сквозной функциональности.
- Заменять Proxy на Аспекты.
- АОП очень хорошо работает вместе с S.O.L.I.D.
- Соблюдать правила диктуемые Аспектом.
- Помнить: АОП, не Золотой Молоток.

Заключение

- Используйте АОП в своих проектах.
- Будьте гибкими и не бойтесь новых технологий и подходов.

Изменения неизбежны.