Monday, February 14, 2022 9:28 AM

Объектно-ориентированный анализ и проектирование 2

Рабочий поток проектирования

Стадии: уточнение, конструирование

Задача: построить проект системы

Состоит из:
- пакетов
- проектных подсистем (компонентов)
- проектных классов
- интерфейсов
- проектных реализаций прецедентов
- диаграмм развертывания

Две модели или одна?

	1	2
Функциональная	Ан. модель дополняется до проектной	В точке "Проработка" создается аналитическая модель, которая дорабатывается до проектной
Поддерживаемая	Ан. модель дополняется до проектной, с помощью CASE-средства восстанавливается "Analysis View"	Поддерживаются обе модели, выполняется их синхронизация

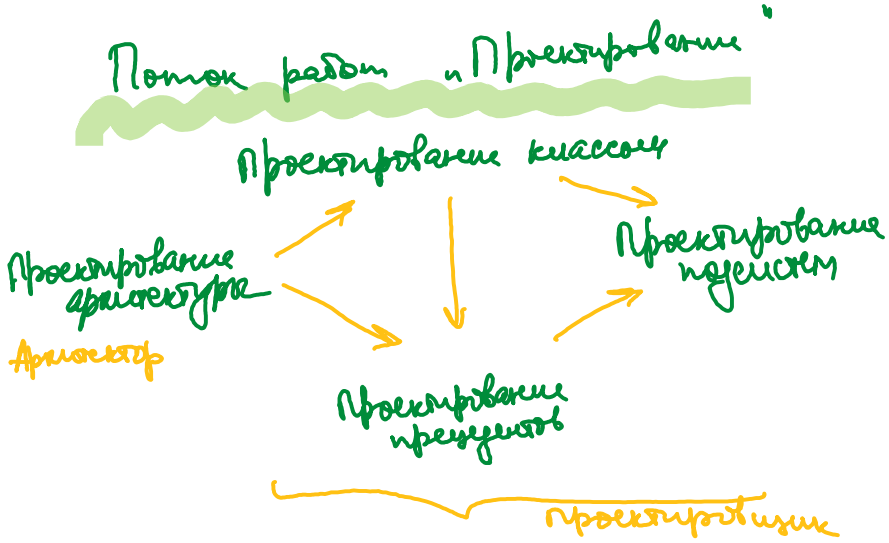
↓ Трудоемко

Аналитическая модель

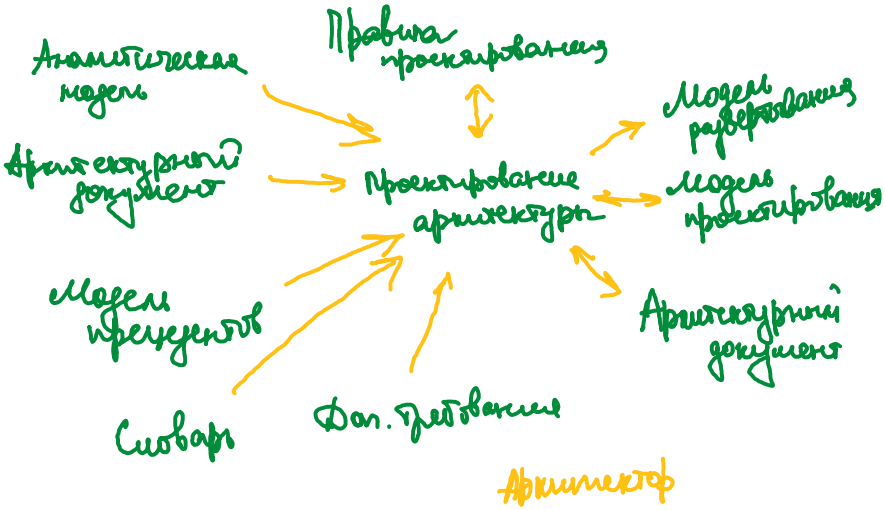
Полезна для:
- инструктажа новых участников
- понимания legacy-кода
- проверки трассируемости требований
- написания требований
- планирования усовершенствований

— документирование

Поток работ «Проектирование»



Работа «Проектирование архитектура»



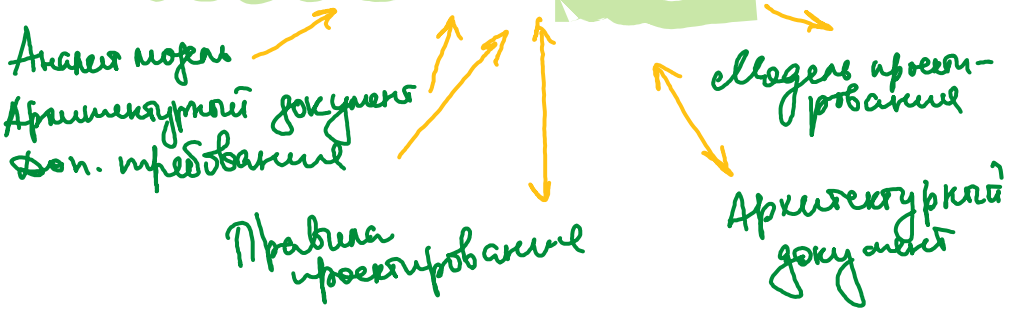
Работы ООАиП

Архитектор:
- архитектурный анализ
- идентификация проектных механизмов и эл-тов
- описание исполнимой архитектуры
- описание распределения
- внедрение сущ-ших проект. эл-тов

Проектировщик:
- анализ прецедентов
- проект. прецедентов

проект. подсистем, ...

Задача архитектора
«Идентификация проектных эл-тов»



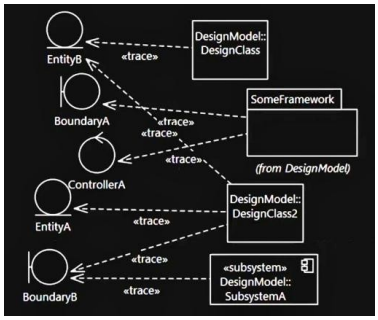
Архитектор

Цели работы

1. Идентифицировать проектные классы и подсистемы
2. Идентифицировать интерфейсы подсистем
3. Обновить проект
4. Выявить и описать возможности использования готовых решений (проектных и технологических)

Эл-ты проекта, трассируемые к эл-там модели анализа

* в общем случае НЕТ взаимно-однозначного соответствия



Проектные классы

- Очень простые классы
- Классы, представляющие логическую абстракцию с ограниченной областью ответственности

часто превращаются в проектные классы

Наши классы начинают
- разделяются на несколько классов
- становятся пакетами
- становятся подсистемами

Распределение проектных классов по пакетам

В одном пакете:
1. относящиеся к одной единице сборки/конфигурации
2. относящиеся к одной библиотеке/каркасу, которые поставляются одной командой разработчиков
3. относящиеся к одной группе типов
4. унаследованного/используемого продукта/модуля

Граничные классы

две стратегии:
- внутренний класс — интерфейсов с окружающей системой средой в отдельном специальном пакете
- поместить граничные классы вместе с функц. связанными с ними классами (предпочтительнее вместе)

Класс функционально связан, если

- изменение одного влечет изменение другого
- удаление класса повлияет на другой класс
- большой обмен сообщениями между классами
- граничный класс является представителем класса-сущности
- неск. классов реализуют функц-ть, необходимую для реализации прецедентов другого этапа
- имеют связи (отношения) друг на друга
- класс создает экземпляр другого класса

Подсистема

- модуль системы, реализующий один или несколько интерфейсов. Поведение подсистемы полностью опред. интерфейсом

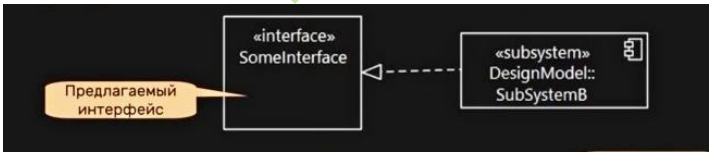
+ Полностью инкапсулирует поведение

+ Представляет модуль, готовый к повторному использованию

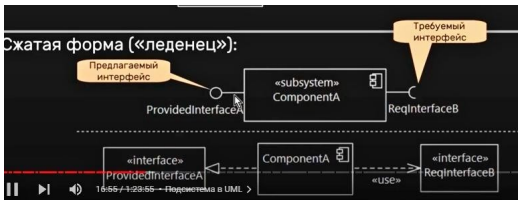
+ Используется для моделирования разных вариантов реализации

UML имеет 2 формы обозначений

Каноническая форма:



Сжатая форма. „леденец"



Подсистема – Пакет

Пакет:
- используется для структурной группировки элементов
- не имеет поведения
- не полностью инкапсулирует своё содержимое
- не всегда легко заменяемый

Подсистема:
- используется для высокоуровневой функционально-структурной декомпозиции программы
- имеет поведение
- полностью инкапсулирует своё содержимое
- всегда легко заменяемый

Как найти подсистемы?

Подсистемами могут стать:
- классы со сложным поведением и многими обязанностями
- классы-сервисы
- классы, предоставляющие набор инструментов
- синглтон-классы („контроллер всего")

- граничные классы

Внешние компоненты могут рассматриваться как подсистемы

- СУБД
- общие картотеки методов
- собрания структур данных
- системные утилиты

и так далее

Что учитывать при выявлении подсистем?

- Взаимодействие сущностей
- Сплоченность классов и связи между ними
- Необязательность элементов
- Подстановочность элементов
- Распределение вычислений
- Изменчивость элементов
- Взаимодействие с миром вне системы

* Контроллеры могут стать подсистемами.

Для чего нужны подсистемы?

Выделенные подсистемы НЕЗАВИСИМО:

- организованы
- скоординированы
- развернуты
- установлены
- разработаны
- обновлены
- заменены

Подсистемы могут использоваться для:

- контроллеры безопасности
- создание фасадов для унаследованных продуктов
- создание фасадов для аппаратных средств

Определение компонентов

Компонент – модуль, для которого явно определен интерфейс, и который может быть развернут в среде

Шаг 2

Определение интерфейсов

1. Для каждой подсистемы опред. НЕ менее одного интерфейса-кандидата. Что должна будет делать подсистема?
2. Выявить сходные интерфейсы, объединить схожие операции
3. Установить другие зависимости между интерфейсами
4. Установить связи интерфейсов и подсистем
5. Определить поведение, задаваемое каждым интерфейсом
6. Распределить интерфейсы по пакетам

Хороший интерфейс

- Имя отражает некоторую роль в системе
- Описывает законченный набор обязанностей
- Каждая операция имеет понятное и короткое имя, проработанный набор параметров и возвращаемое значение (включая тип)
- (В рамках RUP) имеет проработанное вспомогательное артефакты

Обновление проекта

- перемещает организацию проекта с помощью пакетов аналит. модели в структуру в имплементац

- не больше 4-х
- по слоям распределяются подсистемы

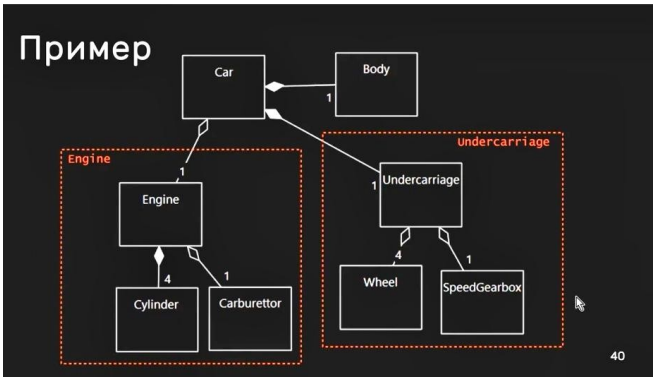
- избегать ситуации циклической зависимости

В процессе моделирования

- следить за уровнем доступа пакетов
- следить за осмысленностью имён
- классы библиотек / сред / каркасов помещать в пакеты компактно.

- классы из одной иерархии - в одном пакете
- классы, связанные агрегацией и композицией — в одном пакете
- классы, которые некого взаимодействуют - в один пакет

Пример:



Внешнее видимость повторного использования

- найти схожие интерфейсы в системе или подкорпусные компоненты вне системы
- модифицировать интерфейсы-кандидаты так, чтобы они соответствовали друг другу

- Заменить интерфейс [illegible] найденное готовое.
- Связать интерфейс с разрабатываемым компонентом в системе или готовым компонентом, включенным в неё.

Задача архитектора. «Идентификация проектных механизмов»

Наша работа

1. Запланировать превращение аналитических механизмов в проектные
2. Документировать архитектурные механизмы

Документ с механизмами анализа

1. Построить таблицу «Аналит класс → механизм»
2. Внести характеристики всех механизмов в применении к конкретным элементам.
3. Структурировать эл-ты, к которым должны быть применены механизмы в соответствии с типами и характеристиками механизмов
4. Для каждой группы эл-тов проекта предложить вариант реализации механизма с помощью шаблона проектирования, стиля архитектуры или готового компонента.

Шаблоны проектирования

- традиционное решение традиционной задачи.

Проектные механизмы могут быть разработаны с использованием шаблонов проектирования

В УП шаблон представляется параметризованным Collaboration.

Документирование

– в архитектуре документе

Проектирование классов

Цели работы:

1. Идентифицировать доп. проектные классы и подсистемы, отношения для реализации
2. Включить автоматные классы, спроектировать их работу
3. Проработать все отношения, операции, атрибуты классов

Стратегический подход

Архитектор и проектировщик делают стратегический проект системы.

Тактическое решение – программисты

Шаги работы

1. Создать проектные классы
2. Проработать операции
3. Определить методы
4. Определить состояния
5. Определить атрибуты
6. Определить зависимости
7. Определить ассоциации
8. Определить внутренние структуры
9. Определить наследование
10. Разрешить противоречия
11. Проработать нефункциональные требования

Два источника информации

1. Аналитическая модель
2. Технологические механизмы и ресурсы

Как реализовать требуемое поведение системы?

Проектный класс

Проработанный:

1. Атрибуты → в программном коде
2. операции → методы в исх. коде
3. типы возвр. знач. для операций, параметры.
4. конструкторы (или + деструкторы)

– отражающая программную сущность
– не предмет стереотипа

– декомпозирован основа исх. кода
– атр закрытые, свойства открытые

Хороший проектный класс

– достаточен для реализации
– простой
– высокая внутренняя сплочённость
– низкая связность – минимум ассоциаций

– не содержит реализаций алгоритмов
– не появляется "из воздуха"

Граничные классы — ПОДСИСТЕМЫ

Классы-сущности — КЛАССЫ КОЛЛЕКЦИЙ (персистентны)

Классы-контроллеры — • обведомляются
• предв. в персистенты

• остаются отдельно
нет класса

Метод
– реализация операции

Класс-автомат
– класс с конечным набором состояний

Проработка отношений
– надо стремиться к более свободным отношениям

(нет отношения, зав-ть, ассоциация,
слабее →
агрегация, композиция, обобщение (наследование)
→ сильные

реализация интерфейса

Зависимости
– не является структурным отношением
– ассоциация ан. может почти никогда не могут стать зависимостями

Ассоциация
> атрибут у класса

не может быть
1. двунаправленных ассоциаций
2. ассоциаций без кратностей
3. ассоциаций с кратностями много-ко-многим
4. классов-ассоциаций

Агрегация и композиция

⇒ Наличие связи ЦЕЛОЕ-ЧАСТЬ
- должно быть направленным

Композиция или атрибут?

Композиция, если:
- свойство имеет собств. идентичность
- есть у многих классов
- сложно устроено и имеет свои св-ва
- имеет собств. поведение
- имеет неявн. отношение

В остальных случаях – АТРИБУТ

Ассоциация 0..1 к 0..*

0..1 – с помощью указателя или ссылки

0..* – дроби не может быть реализован напрямую ⇒ с помощью коллекции

ассоциация * к *

Должна быть разбита с использованием шаблона item.

Не может быть реализована в большинстве языков программирования

* к 1 – скорее всего агрегация

Симметричность

Ассоциация симметрична
Агрегация ассиметрична
Агрегация транзитивна

Агрегация и наследование

Наследование — "является {is a}"
Агрегация — "часть целого {part of}"
Наслед. сильнее ассоциации и агрегации

Фасадирование

- часть операций из класса в другой класс
- разрываем некоторое отношение

* Замена наследования делегированием

Суть ООП

! Абстрагирование
- отделить способ использования объекта от способа его внутренней реализации.

- Инкапсуляция
- Полиморфизм