Проектирование программных компонентов

Часть 2. Принципы организации компонентов

Уровни проектирования

- Уровень функций и методов
- Уровень классов
- Уровень организации компонентов
- Архитектурный уровень

Компонент – единица развертывания

DLL GEM JAR NPM EXE

История

- Неперемещаемые библиотеки
- Перемещаемые библиотеки (связывающий загрузчик)
- Компоновщик (редактор связей) + загрузчик

Любая программа растет, пока не заполнит все доступное время на компиляцию и компоновку

Компоненты

Программные компоненты – динамически связываемые файлы, которые можно подключать во время выполнения (связывающий загрузчик)

Связность компонентов

- **REP:** Reuse/Release Equivalence Principle

(Принцип эквивалентности повторного использования и выпусков)

- **CCP:** Common Closure Principle

(Принцип согласованного изменения)

- **CRP**: Common Reuse Principle

(Принцип совместного повторного использования)

Принцип эквивалентности повторного использования и выпусков (REP)

«Выпуск»:

- Номер версии
- Описание новой версии
- Change Log (Лог изменений)

Единица повторного использования == Единица выпуска

Классы и модули, объединяемые в компонент, должны выпускаться вместе

Объединение в один выпуск должно иметь **смысл** для автора и пользователей

Принцип согласованного изменения (CCP)

Развитие принципов «единственной ответственности» (SRP) и «открытости/закрытости» (OCP) из SOLID

- В один компонент должны включаться классы, изменяющиеся по одним причинам и в одно время
- В разные компоненты должны включаться классы, изменяющиеся по разным причинам и в разное время

Принцип согласованного изменения (CCP)

ОТКРОВЕНИЕ:

Для **большинства** приложений простота сопровождения **важнее** возможности повторного использования

Принцип согласованного изменения (CCP)

Идея: Объединение в компонент классов, закрытых для одного и того же вида изменений.

Изменение требований => изменение **МИНИМАЛЬНОГО** количества компонентов

Принцип согласованного изменения (CCP)

Собирайте вместе все, что изменяется по одной причине и в одно время.

Разделяйте все, что изменяется в разное время и по разным причинам.

Принцип совместного и повторного использования (CRP)

- Не вынуждайте пользователей компонента зависеть от того, что им не требуется

- Классы, не имеющие тесной связи, не должны включаться в компонент

Баланс


МНОГО НЕНУЖНЫХ ВЫПУСКОВ
REP
Объединение для удобства пользователей
???
CCP
Объединение для удобства сопровождения
CRP
Разделение для устранения лишних выпусков
ИЗМЕНЕНИЯ ЗАТРАГИВАЮТ МНОГО КОМПОНЕНТОВ
ПРОБЛЕМЫ С ПОВТОРНЫМ ИСПОЛЬЗОВАНИЕМ

Сочетаемость компонентов

- Принцип ацикличности зависимостей
- Принцип устойчивых зависимостей
- Принцип устойчивости абстракций

Принцип ацикличности зависимостей (ADP)

Циклы в графе зависимостей недопустимы

Отдельные компоненты – отдельные разработчики/команды

Появление цикла – появление одного БОЛЬШОГО компонента

Принцип ацикличности зависимостей (ADP)

Разрыв цикла:

1. Применить принцип DIP
2. Создать новые компонент, от которого зависят проблемные

Проектирование сверху вниз?

- Граф зависимостей формируется для защиты стабильных и ценных компонентов от влияния изменчивых компонентов
- Структура компонентов отражает удобство сборки сопровождения

Поэтому она не проектируется полностью в начале разработки

Принцип устойчивых зависимостей

Зависимости должны быт направлены в сторону устойчивости

Метрика неустойчивости = выходы / (входы + выходы)

Метрика неустойчивости компонента должна быть выше метрик неустойчивости компонентов, от которых он зависит

Принцип устойчивости абстракций

Устойчивость компонента пропорциональна его абстрактности

Пример: Компоненты, содержащие только интерфейсы в C# и Java.