

# 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
- ...
- Динамические библиотеки (связывающий 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)

Программные компоненты – единицы развертывания

Программные компоненты – выстраивают граф зависимостей

# Как спроектировать компонент?

К какому компоненту отнести тот или иной класс?

# Принципы связности 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 (ССР)

*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 (ССР)

*«ОТКРОВЕНИЕ»:*

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

# Принцип согласованного изменения (ССР)

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

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

# Принцип согласованного изменения (ССР)

*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)

*Развитие принципа разделения интерфейсов из SOLID*

- Не вынуждайте пользователей компонента зависеть от того, что им не требуется
- Классы, не имеющие тесной связи, не должны включаться в компонент

Не создавайте зависимостей от чего-то неиспользуемого

# Баланс: главная задача архитектора



# Как связывать компоненты между собой?

Как сочетать несколько компонентов между собой?

# Принципы сочетаемости 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

Граф зависимостей – ациклический ориентированный граф

# Принцип ацикличности зависимостей (ADP)

Разрыв цикла:

1. Применить принцип инверсии зависимостей (DIP)
2. Создать новый 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 и слабо отражает функции приложения

*Поэтому она не проектируется полностью в начале разработки*

# Принцип устойчивых зависимостей (SDP)

Зависимости должны быть направлены в сторону устойчивости

Устойчивость – способность сохранять свое состояние при внешних воздействиях

Метрика неустойчивости ( $I$ ) = выходы / (входы + выходы)

$I = 0$  – максимальная устойчивость,  $I = 1$  – максимальная неустойчивость

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

# Принцип устойчивости абстракций

Устойчивость компонента пропорциональна его абстрактности

Абстрактность (A) =  $\frac{\text{число абстрактных классов и интерфейсов}}{\text{число классов}}$

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