



Базовые правила и принципы проектирования ПО

Евгений Кривошеев

EKrivosheev@luxoft.com

О вашем инструкторе



- **Имя:**
Евгений Кривошеев
- **Статусы:**
SCJP, SCWCD, SCBCD, BEA WLS
CA, IBM WAS CA
- **Контакты:**
EKrivosheev@luxoft.com

- Семинар призван систематизировать базовые правила и принципы проектирования ПО
- Представленные базовые принципы необходимы для понимания более сфокусированных техник и приемов - рефакторинга и шаблонов проектирования

- Данный семинар – вводный,
~ 3 мин на принцип или правило
- В дальнейшем будет разработан
полноценный курс

Слушатели должны:

- Иметь опыт разработки на одном из ООП-языков программирования
- Понимать ключевые концепции ООП

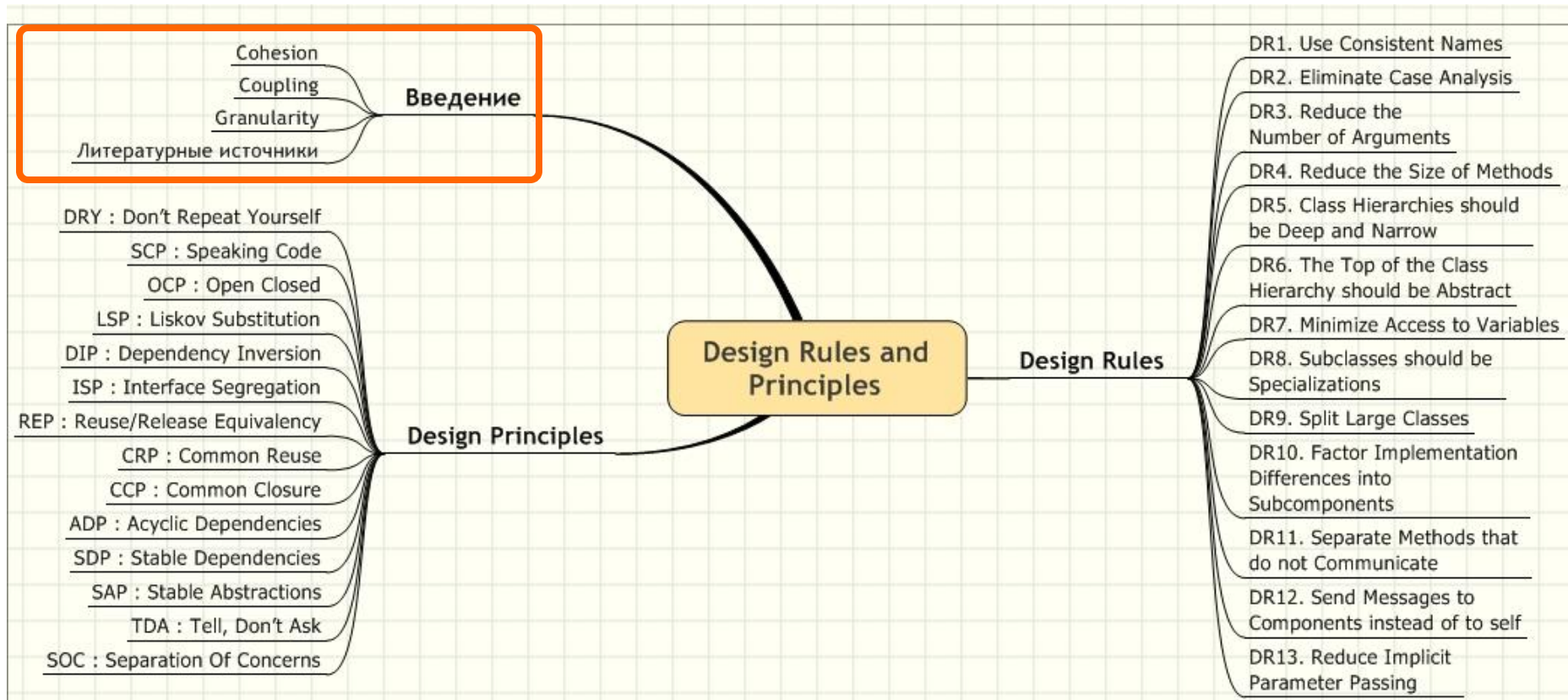
- Время начала и конца занятий
- Перерывы

- Конференции (www.soft-labs.ru):
 - 26 сентября, Киев: TEST Labs 2009, программа сформирована, регистрация участников
 - 17 ноября, Москва: Req Labs 2009, открыта регистрация докладчиков
 - 15 декабря, Москва: Arch Labs 2009, открыта регистрация докладчиков

Расписание:

- Класс руководителя группы разработки. Основные курсы (24.08.2009-15.09.2009)
- Класс менеджера проектов. Основы управления проектами (24.08.2009-17.09.2009)
- Класс тест-дизайнера. Дополнительные курсы (27.08.2009-11.09.2009)
- Класс java-разработчика. Разработка на базе платформы JavaSE. Экспертный уровень. (31.08.2009-30.09.2009)

<http://www.luxoft-training.ru/timetable>



Введение



Ключевые понятия

ООП-проектирования (общеизвестные)

- Наследование
- Полиморфизм

Ключевые понятия

ООП-проектирования (менее известные)

- Responsibility (ответственность)
- Intention (намерение)
- Coupling (связность)
- Cohesion
(сцепленность, сфокусированность)
- Granularity
(гранулярность, детальность)

Responsibility

- **Ответственность** – решаемая классом задача из предметной области
 - Функциональная задача
 - Инкапсуляция данных
- Чаще этот термин применяется для **классов**

Intention

- *Намерение* – решаемая разработчиком задача
- Чаще этот термин применяется для *МЕТОДОВ*

Coupling

- *СВЯЗНОСТЬ* – метрика, характеризующая степень зависимости классов друг от друга
- Loosely coupled vs. Tightly coupled
- Классы могут быть связаны (coupled) различными образами:
 - Зависимые
 - По управлению
 - По данным

Cohesion

- *Сцепленность (Сфокусированность)* – метрика, характеризующая узость *ОТВЕТСТВЕННОСТИ* класса
- Low cohesion vs. High cohesion
- Классы могут иметь различную сфокусированность (cohesion):
 - По функциональности
 - По данным

Granularity

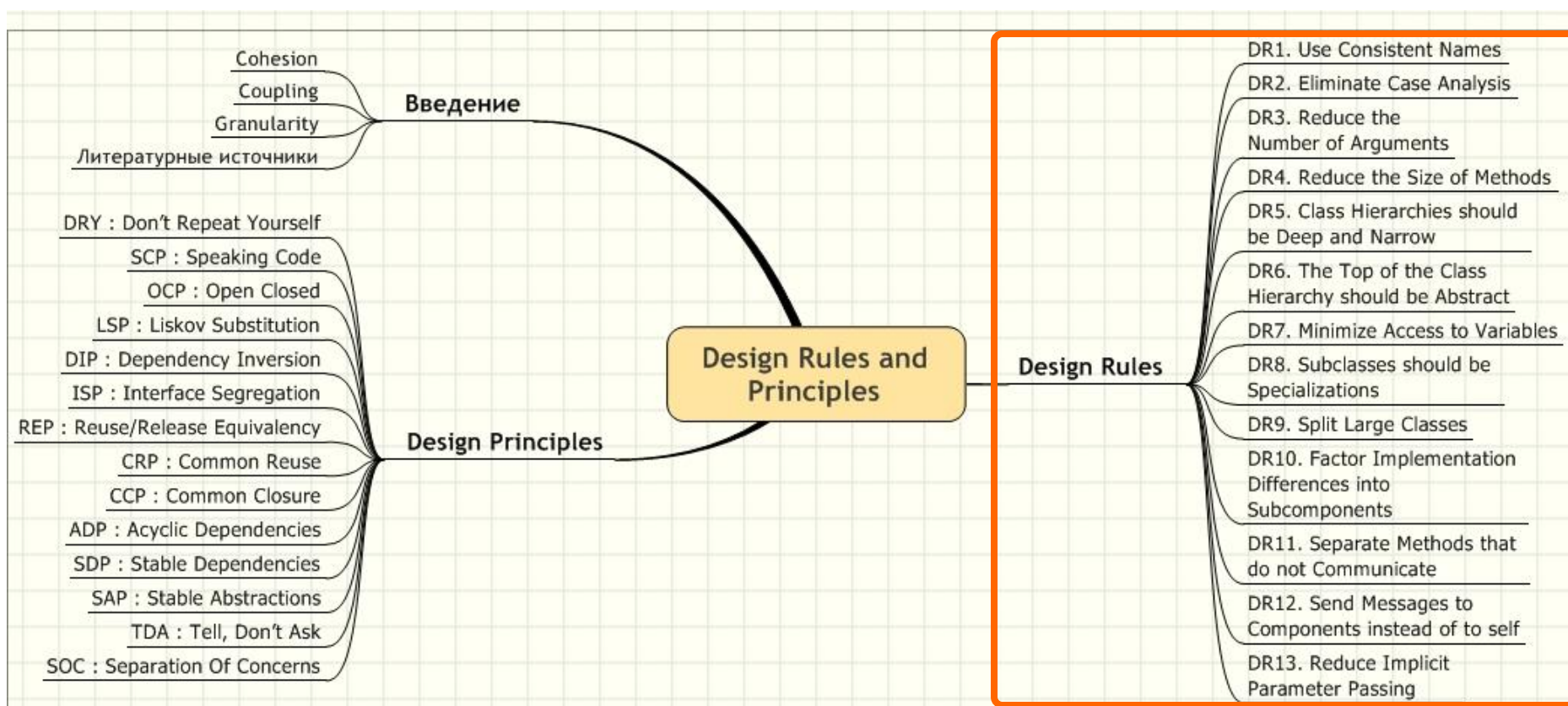
- *Гранулированность (Детальность)* – метрика, характеризующая способ реализации намерений
- Fine-grained vs. Coarse-grained
- Чаще этот термин применяется для *интерфейсов*, где намерение выражается методом

Инструменты code reuse в ООП

- Наследование
- Делегирование

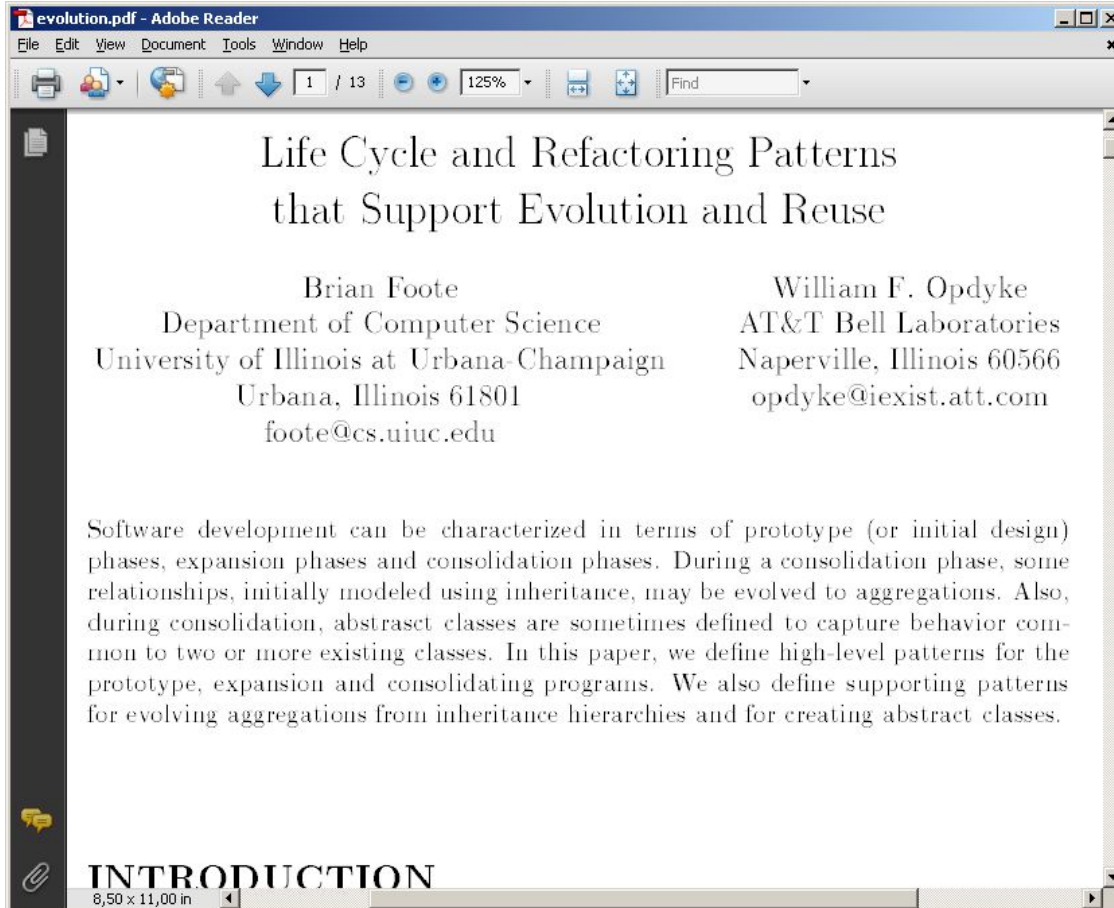
Литературные источники семинара

- Brian Foote and William F. Opdyke. Lifecycle and refactoring patterns that support evolution and reuse, 1995 (отдельно и в рамках книги Pattern Languages of Program Design vol. 1)
- Stefan Roock. Refactoring in Large Software, 2006
- Martin Fowler. Refactoring: Improving the Design of Existing Code, 1999



Design Rules

Литературные источники



evolution.pdf - Adobe Reader

File Edit View Document Tools Window Help

1 / 13 125% Find

Life Cycle and Refactoring Patterns that Support Evolution and Reuse

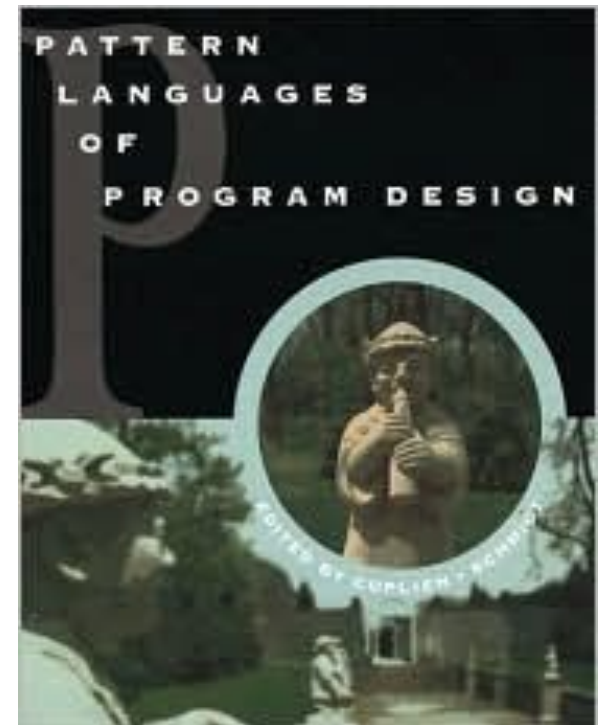
Brian Foote
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
foote@cs.uiuc.edu

William F. Opdyke
AT&T Bell Laboratories
Naperville, Illinois 60566
opdyke@iexist.att.com

Software development can be characterized in terms of prototype (or initial design) phases, expansion phases and consolidation phases. During a consolidation phase, some relationships, initially modeled using inheritance, may be evolved to aggregations. Also, during consolidation, abstract classes are sometimes defined to capture behavior common to two or more existing classes. In this paper, we define high-level patterns for the prototype, expansion and consolidating programs. We also define supporting patterns for evolving aggregations from inheritance hierarchies and for creating abstract classes.

INTRODUCTION

8,50 x 11,00 in



Design Rules

Design Rules

- DR1. use consistent names
- DR2. eliminate case analysis
- DR3. reduce the number of arguments
- DR4. reduce the size of methods
- DR5. class hierarchies should be deep and narrow
- DR6. the top of the class hierarchy should be abstract
- DR7. minimize access to variables
- DR8. subclasses should be specializations
- DR9. split large classes
- DR10. factor implementation differences into subcomponents
- DR11. separate methods that do not communicate
- DR12. send messages to components instead of to self
- DR13. reduce implicit parameter passing.

<http://www.laputan.org/drc/drc.html>

Design Rules

- В дальнейшем обсуждении мы рассмотрим не дословный перевод правил проектирования, а расширенные современные трактовки
- Важно помнить, что эти правила хоть и распространенные, но контекстные, т.е. их применимость следует обосновывать

Design Rules

DR1. Use Consistent Names

DR1. Recursion introduction

- *Используйте непротиворечивые имена методов [и свойств]*
- Непротиворечивость здесь – это одинаковые имена для одинаковых намерений
- В книге [MF] есть аналогичный smell/refactoring

Design Rules

DR2. Eliminate Case Analysis

- *Избегайте смешивания бизнес-логики и логики выбора/ветвления*
- В книге [MF] есть аналогичный smell/refactoring

Design Rules

DR3. Reduce The Number Of Arguments

- *Уменьшайте число аргументов/параметров*
- В книге [MF] есть аналогичный smell/refactoring

Design Rules

DR4. Reduce The Size Of Methods

- *Уменьшайте объем методов*
- В книге [MF] есть аналогичный smell/refactoring

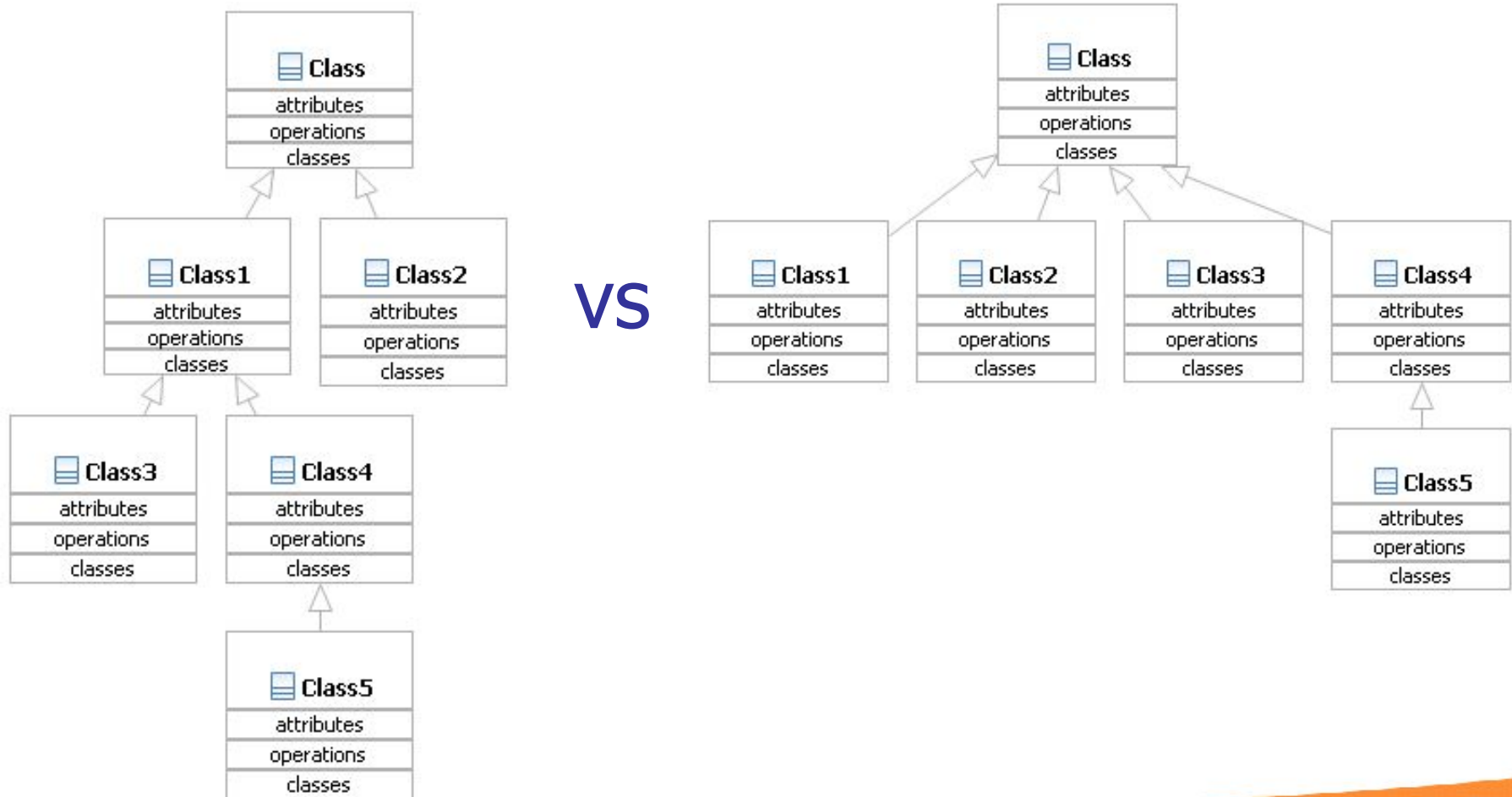
Design Rules

DR5. Class Hierarchies Should Be Deep And Narrow

- *Иерархию наследования стоит проектировать глубокой и узкой*

Design Rules

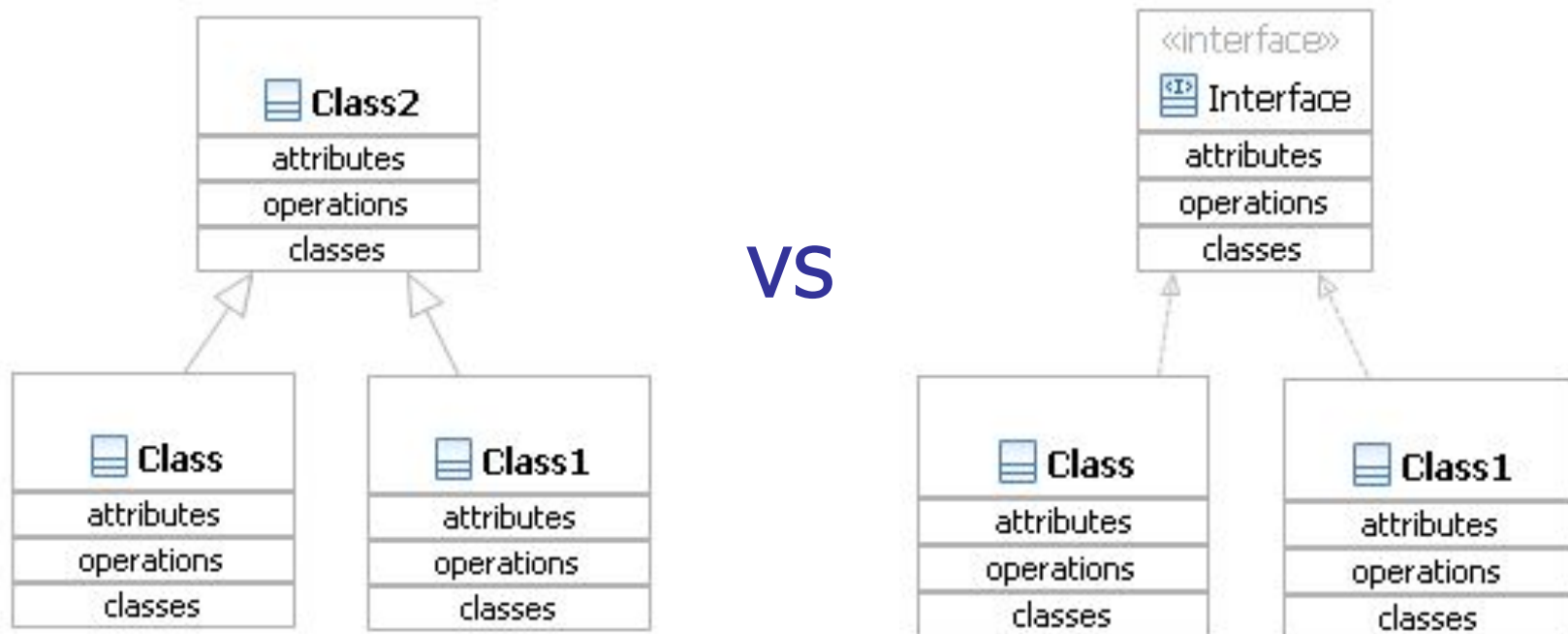
DR5. Class Hierarchies Should Be Deep And Narrow



Design Rules

DR6. The Top Of The Class Hierarchy Should Be Abstract

- *На вершине иерархии наследования стоит размещать абстракцию*



DR7. Minimize Access to Variables

- *Стоит минимизировать прямой доступ к переменным классов и экземпляров*
- Encapsulation aka Hiding
- В книге [MF] есть аналогичный smell/refactoring

DR8. Subclasses Should Be Specializations

- *Наследники должны быть специализациями базовых классов*
- Специализация – отношение «IS-A», «является»
- В книге [MF] есть аналогичный smell/refactoring (Refused Bequest)

DR9. Split Large Classes

- *Разбивайте большие классы*
- В книге [MF] есть аналогичный smell/refactoring

DR10. Factor Implementation Differences Into Subcomponents

- *В случае серьезной разницы в реализации метода двумя «братьями» стоит задуматься о целесообразности описания этого метода в их «отце»*
- Потенциально можно вынести этот метод в иной класс (делегировать)

Design Rules

DR11. Separate Methods That Do Not Communicate

- *Разделяйте несвязанные методы*
- **Связь:**
 - по управлению
 - по данным
- В книге [MF] есть аналогичный smell/refactoring

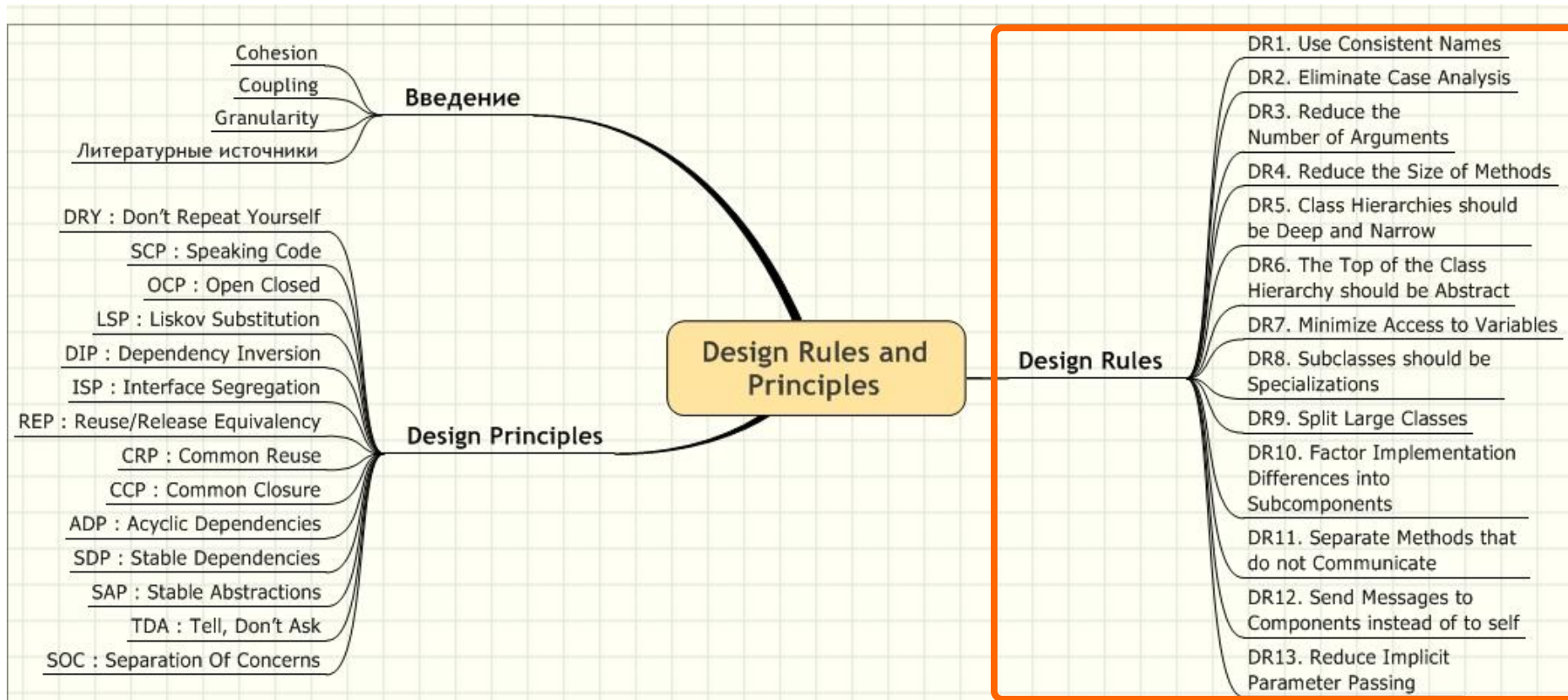
Design Rules

DR12. Send Messages To Components Instead Of To Self

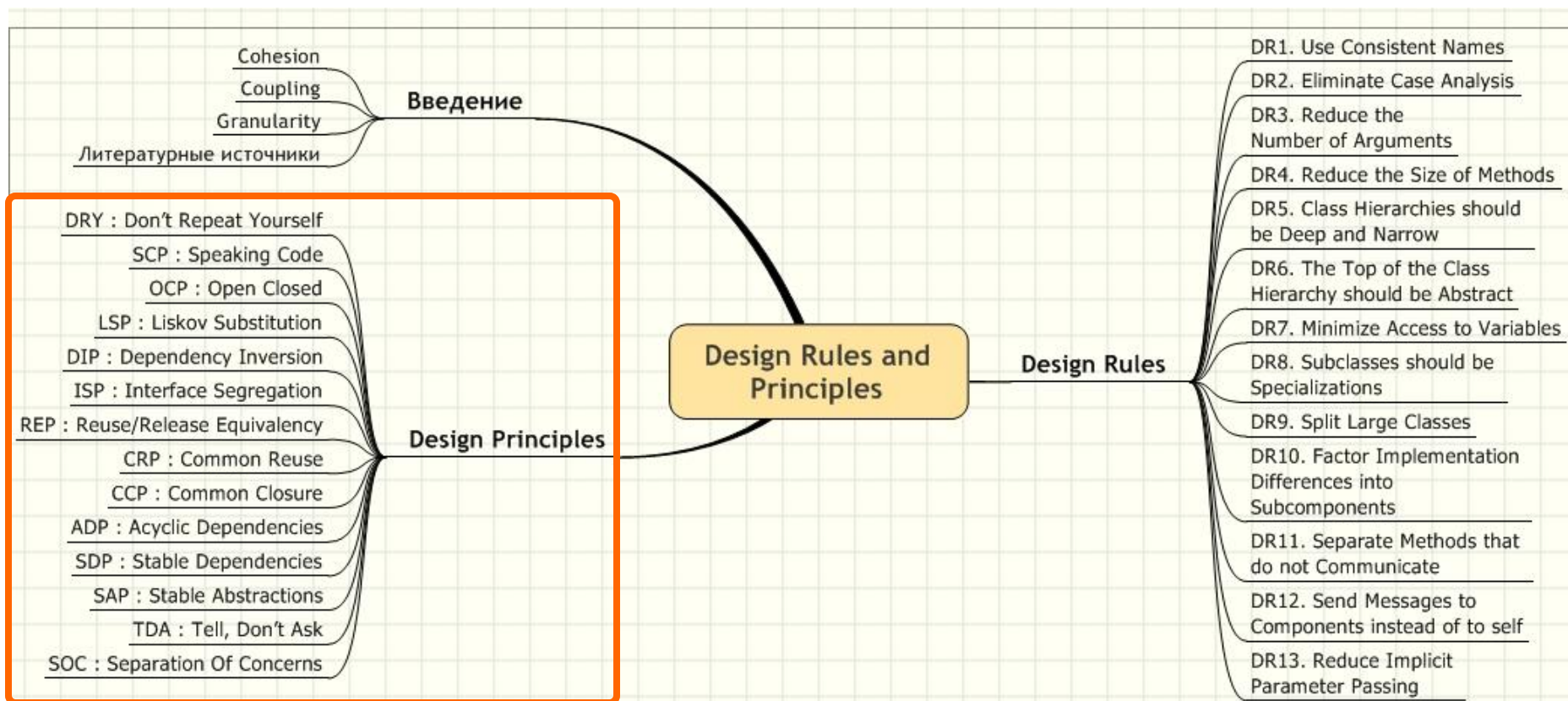
- *Делегируйте*
- Inheritance-based framework vs component-based framework – где ниже связность?

DR13. Reduce Implicit Parameter Passing

- *Передавайте параметры явно*
- **Варианты неявной передачи:**
 - глобальные переменные
 - состояние
 - внешние источники данных
- **Вызов метода, результат которого зависит только от входных параметров - *идемпотентный***

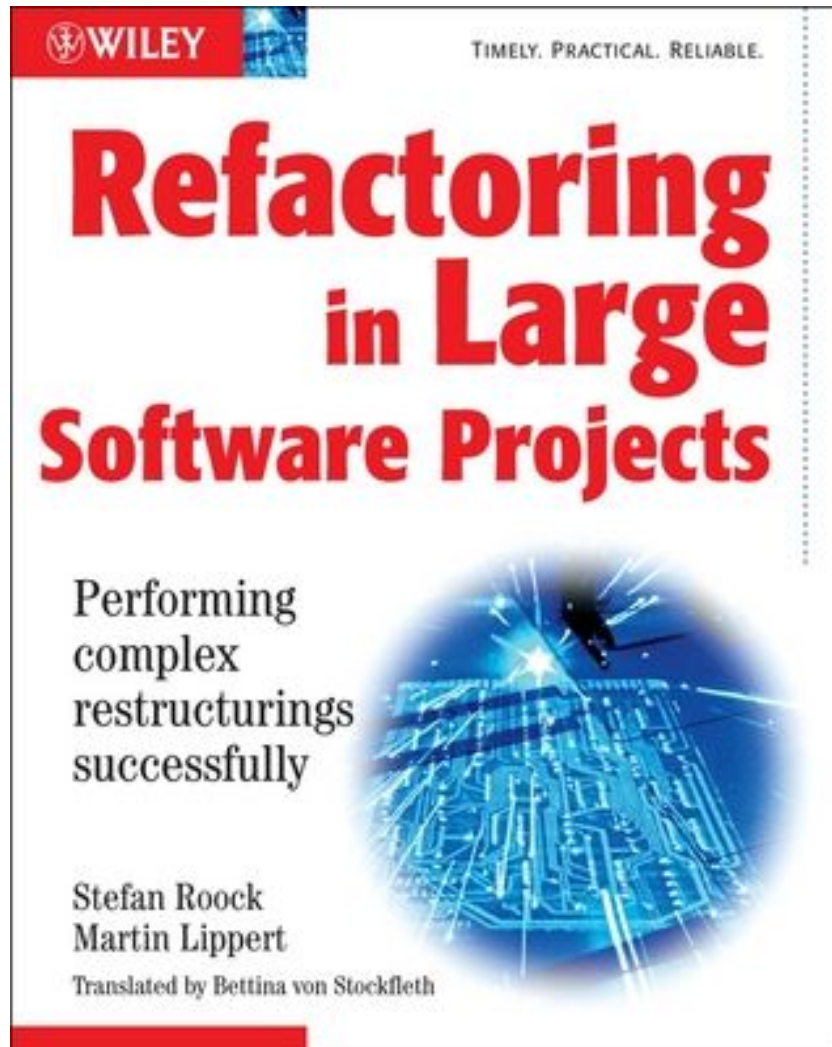


Буду рад ответить на Ваши вопросы



Design Principles

Литературные источники



Design Principles



Design Principles

DRY : Don't Repeat Yourself
SCP : Speaking Code
OCP : Open/Closed
LSP : Liskov Substitution
DIP : Dependency Inversion
ISP : Interface Segregation
REP : Reuse/Release Equivalence
CRP : Common Reuse
CCP : Common Closure
ADP : Acyclic Dependencies
SDP : Stable Dependencies
SAP : Stable Abstractions
TDA : Tell, Don't Ask
SOC : Separation Of Concerns

Stefan Roock. Refactoring in Large Software. 2006

Design Principles

DRY: Don't Repeat Yourself

- *Минимизируйте повторение кода для снижения затрат на поддержку*
- *ака Single Point of Truth or Single Point of Maintenance*
- В книге [MF] есть аналогичный smell/refactoring

Design Principles

SCP: Speaking Code

- *Код должен явно и однозначно отражать намерение*
- В книге [MF] есть аналогичный smell/refactoring

Design Principles

OCP: Open/Closed

- *Программные сущности (классы, модули, функции) должны быть открыты для расширения и закрыты для изменения*
 - «Открыты для расширения» - возможно расширять и изменять поведение приложения при изменении требований
 - «Закрыты для изменения» - расширение поведения не приводит к изменению исходного или бинарного кода

OCP: Open/Closed

- Принцип OCP используется в двух контекстах его реализации:
 - *Dr. Bertrand Meyer's Open/Closed Principle*
«Написанная реализация класса модифицируется только для исправления ошибок, новые ответственности или изменение существующих потребует создание нового класса, возможно, наследника. Этот новый класс не обязан реализовать тот же интерфейс.»
 - **Polymorphic Open/Closed Principle**
Более современная версия. «Множество реализаций классов можно использовать полиморфно, через один и тот же интерфейс.» Здесь зафиксирована интерфейсная часть, а реализация вариативна.
- См. так же «Protected Variation»

далее

Design Principles

OCP: Open/Closed



OPEN CLOSED PRINCIPLE

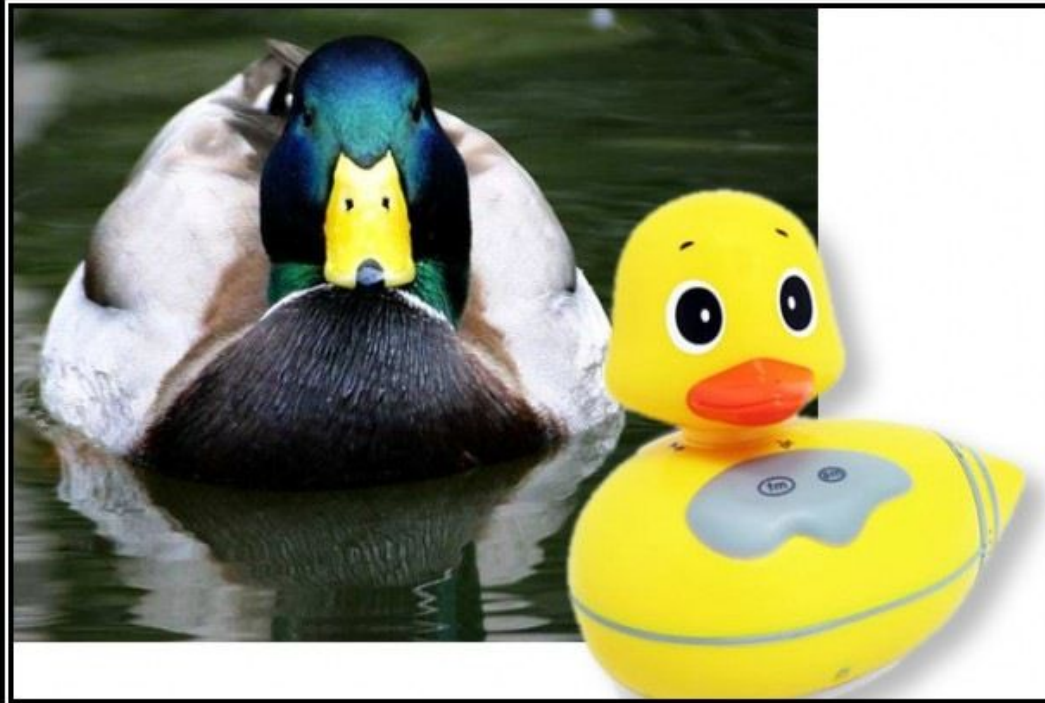
Open Chest Surgery Is Not Needed When Putting On A Coat

LSP: Liskov Substitution

- Существует два базовых определения:
 - Barbara Liskov
«В коде приложения класс всегда можно заменить его наследником»
 - Bertrand Meyer ("Design-by-Contract" formulation)
«Наследники должны соблюдать контракт предка»

Design Principles

LSP: Liskov Substitution



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

DIP: Dependency Inversion

- *Высокоуровневые модули не должны зависеть от низкоуровневых. И те, и другие должны зависеть от абстракций.*
- *Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракции.*

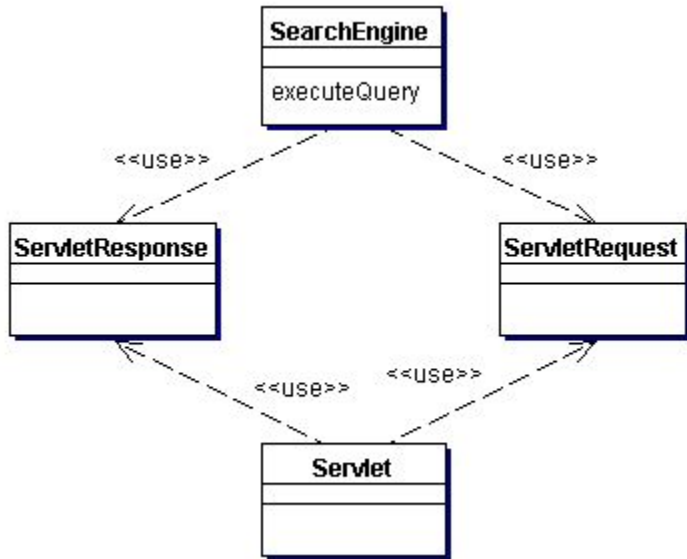
Design Principles

DIP: Dependency Inversion

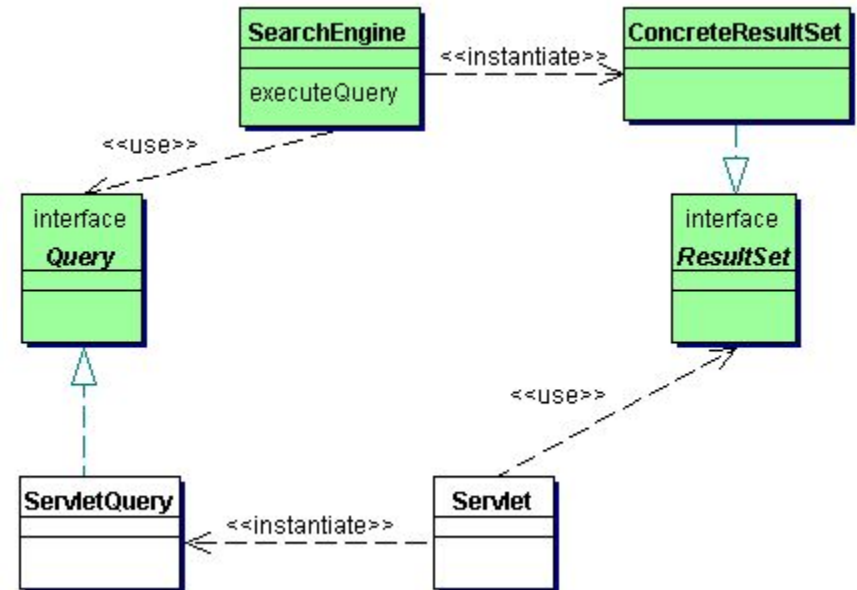
- С помощью абстракций детали системы изолируются друг от друга
- Легко менять детали реализации без модификации высокоуровневой логики
- Шаблоны, с помощью которых реализуется принцип DIP:
 - Plug-in, [A] Factory [M], Service Locator, Inversion of Control, Dependency Injection

Design Principles

DIP: Dependency Inversion



VS



Design Principles

DIP: Dependency Inversion



DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

далее

Design Principles

DIP → IoC → Dependency Injection

- *Inversion Of Control*

- Принцип *инверсии управления потоком выполнения* по сравнению с процедурным программированием
- Основа всех *каркасов (frameworks)*
- aka *Hollywood Principle*

- *Dependency Injection*

- Шаблон проектирования
- *Не мы сами получаем необходимые объекты, а внешняя среда нам их передает*

Design Principles

ISP: Interface Segregation

- *Не стоит заставлять клиентов зависеть от ненужных им интерфейсов*
- Вместо одного многофункционального интерфейса лучше выделить несколько узкоспециальных

Design Principles

ISP: Interface Segregation



INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

REP: Reuse/Release Equivalence

- *The granule of reuse is the granule of release. Only components that are released through a tracking system can be effectively reused. This granule is the package*
- *Многократно используемая единица кода должна пройти завершённый цикл разработки – система контроля версий, багтрекер, тесты. Эта единица – пакет.*
- REP и ряд следующих принципов – макропринципы организации разработки и пакетирования кода

Design Principles

CRP: Common Reuse

- *Классы в пакете используются совместно. Если используется один класс из пакета, считается что используются все.*
- *Здесь использование – многократное использование при дальнейшей разработке*

Design Principles

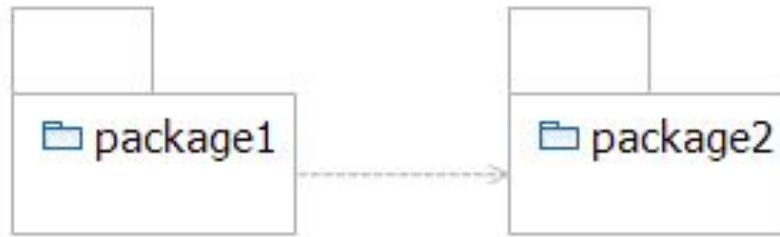
CCP: Common Closure

- *Классы в пакете должны быть связаны одинаковой причиной их изменения. Изменение пакета (одного из классов) касается всех классов в нем.*

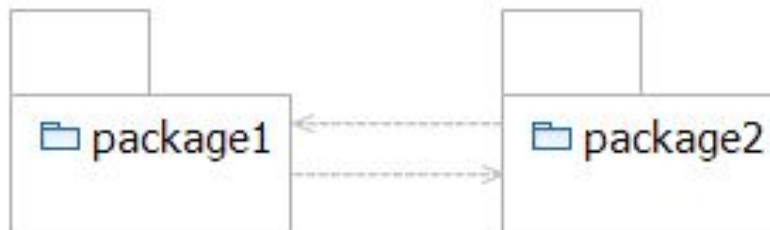
Design Principles

ADP: Acyclic Dependencies

- *Не должно быть взаимной зависимости между пакетами, только односторонняя.*



VS



SDP: Stable Dependencies

- *Зависимости между пакетами должны быть в сторону более стабильного. Пакет должен зависеть только от более стабильного пакета, чем он сам.*
- *Стабильность модуля, класса или пакета – степень сложности его изменений*
- *Стабильные классы – независимые классы (незачем менять) или сильнозависимые (множество причин не менять)*

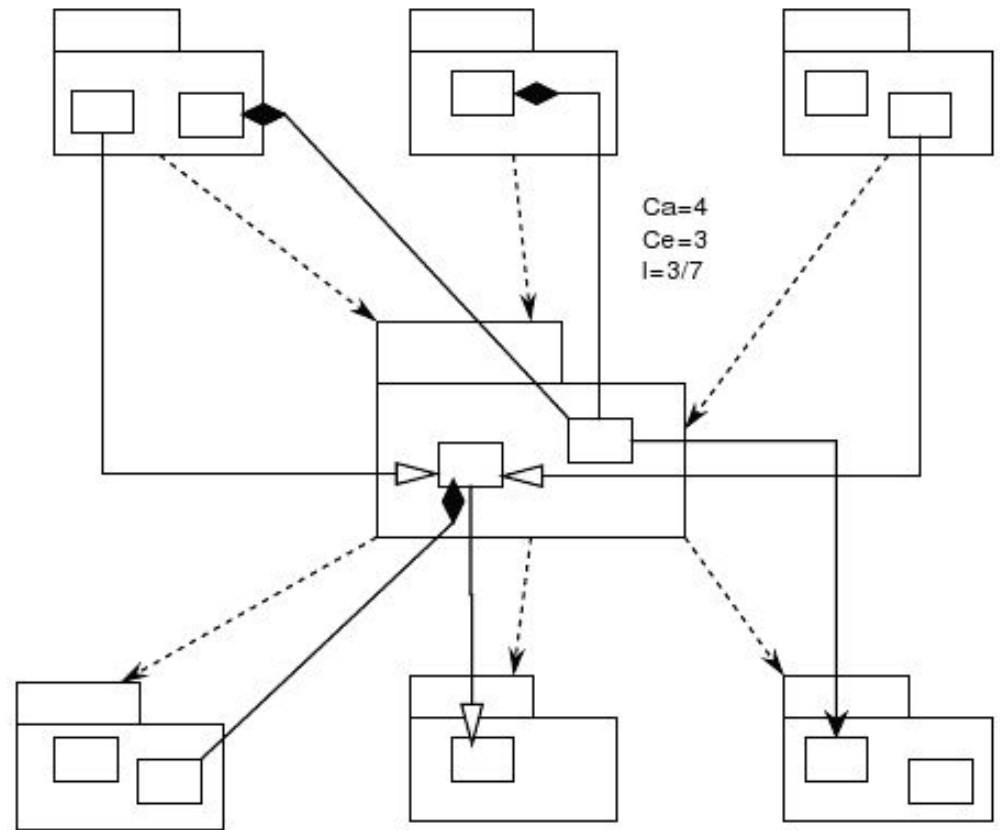
Design Principles

SAP: Stable Abstractions

- *Самые стабильные пакеты должны быть самыми абстрактными. Нестабильные пакеты должны быть конкретными. Степень абстракции пакета должна зависеть от его стабильности.*

Design Principles

REP : Reuse/Release
Equivalence
CRP : Common Reuse
CCP : Common Closure
ADP : Acyclic Dependencies
SDP : Stable Dependencies
SAP : Stable Abstractions



Design Principles

TDA: Tell, Don't Ask

- *TDA – стиль ООП, при котором объекта сигнализирует объекту Б выполнить его ответственность, вместо того, чтобы что-либо спрашивать у него и выполнять ответственность самому*

Design Principles



TDA: Tell, Don't Ask

- Объекты берут на себя сфокусированные ответственности и делегируют остальные ответственности другим объектам
- ООП vs Процедурный стиль
- См. так же «Low Of Demeter»

Design Principles

SOC: Separation Of Concerns

- *Разделяйте ответственности по сфокусированным классам*
- aka Single Responsibility Principle
«Класс должен иметь только одну причину изменения»

Design Principles

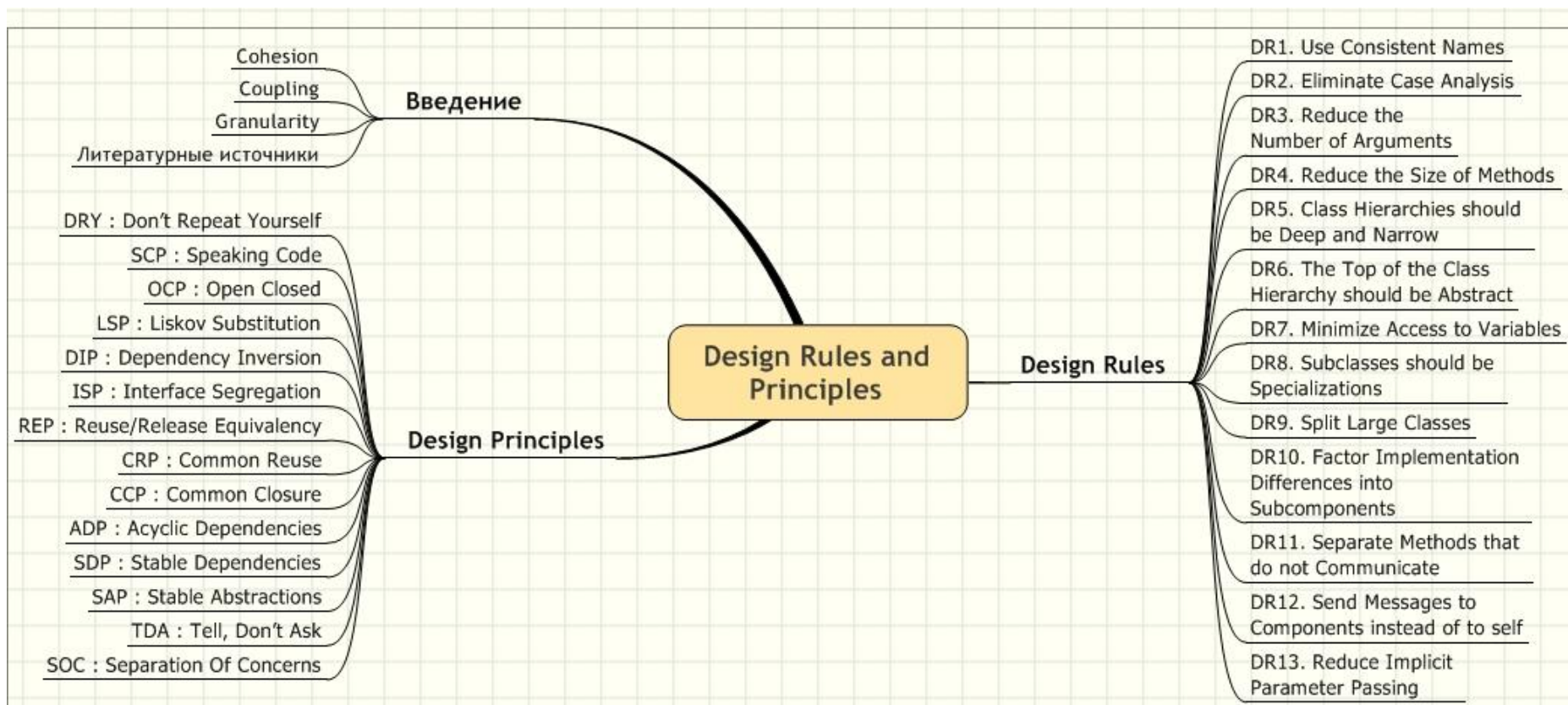
SOC : Separation Of Concerns



SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

Выводы



Буду рад ответить на Ваши вопросы

Ссылка на оценочную форму семинара:
<http://www.luxoft-training.ru/events/vote>

- УЦ Luxoft предлагает *более 200* курсов и тренингов по различным направлениям промышленной разработки ПО
- Наши инструкторы – практики, готовые передать свою экспертизу

<http://luxoft-training.ru>

- Конференции (www.soft-labs.ru):
 - 26 сентября, Киев: TEST Labs 2009, программа сформирована, регистрация участников
 - 17 ноября, Москва: Req Labs 2009, открыта регистрация докладчиков
 - 15 декабря, Москва: Arch Labs 2009, открыта регистрация докладчиков

Расписание:

- Класс руководителя группы разработки. Основные курсы (24.08.2009-15.09.2009)
- Класс менеджера проектов. Основы управления проектами (24.08.2009-17.09.2009)
- Класс тест-дизайнера. Дополнительные курсы (27.08.2009-11.09.2009)
- Класс java-разработчика. Разработка на базе платформы JavaSE. Экспертный уровень. (31.08.2009-30.09.2009)

<http://www.luxoft-training.ru/timetable>



Базовые правила и принципы проектирования ПО

Евгений Кривошеев

EKrivosheev@luxoft.com