Основы разработки ПО

Метрики проектирования

Кулаков Кирилл Александрович

Проблема

- Разработка ПО — решение сложных задач
- Одна задача — множество решений

- Проблема: выбор наилучшего решения

- Пример: Для вычисления значения X существуют 3 алгоритма:
  - Алгоритм A1 вычисляет за 10 секунд с потреблением памяти 10 Мб
  - Алгоритм A2 вычисляет за 20 секунд с потреблением памяти 1 Мб
  - Алгоритм A3 вычисляет за 5 секунд с потреблением памяти 50 Мб

Проблема

- Фредерик Брукс «No Silver Bullet – Essence and Accident in Software Engineering» определил 2 типа сложности:
  - **Essential complexity** – необходимая сложность, которая определяется самой проблемой и никак не может быть удалена.
  - **Accidental complexity** – непреднамеренная сложность, которая добавляется программистами во время проектирования и написания кода, и которая самими программистами и может быть устранена или хотя бы достаточно снижена.

Определение

- **Метрика программного обеспечения** (software metric) – численная мера, позволяющая оценить определенные свойства конкретного участка программного кода

- Использование метрик
  - Сопоставление результатов
  - Сравнение с контрольными показателями
  - История изменения/развития

Базовые метрики

- **Глубина дерева наследования (DIT, Depth of Inheritance Tree)** - максимальная длина от узла класса до корня дерева, измеряемая в предках класса
- Пример:

```
class A {…}
    class B: public A {…}
        class C: public B {….}
```

- Чем больше глубина тем большее количество методов наследуется
  - Непредсказуемое поведение класса

Базовые метрики

- **Количество потомков (NOC, Number of Chuldren)** — количество непосредственных потомков класса
- Пример:

```
class A {...}
    class B: public A {...}
    class C: public A {...}
```

- Чем больше потомков тем больше проверок требуется для родительского класса

Базовые метрики

- **Количество методов (NOM, Number of Methods)** — количество методов определенных локально в классе, без учета видимости класса
- Пример:

```
class A {
    public:
        void method_A();
        void method_B();
}
```

- Чем больше методов тем сложнее класс
- Мало методов — ограниченность функционала

Базовые метрики

- Уотсон, А. Х., & Маккейб, Т. Дж. (1996). Структурированное тестирование: методология тестирования с использованием метрики цикломатической сложности
- **Цикломатическая сложность (Cyclomatic Complexity)** определяется как измерение "объема логики принятия решений в функции исходного кода" (метрика Маккаба)
- **Цикломатическая сложность —** Число независимых линейных путей в программе.
- Цикломатическая сложность может быть рассчитана как: L - N + 2P, где L - число ребер/связей графа; N - число вершин графа; P - число несвязанных частей графа (например, граф вызова или подпрограмма).

Цикломатическая сложность

```
int sumOfPrimes(int max) {                    // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) {       // +1
    for (int j = 2; j < i; ++j) {             // +1
      if (i % j == 0) {                       // +1
        continue OUT;
      }
    }
    total += i;
  }
  return total;
}                    // Cyclomatic Complexity 4
```

```
String getWords(int number) { // +1
    switch (number) {
      case 1:                 // +1
        return "one";
      case 2:                 // +1
        return "a couple";
      case 3:                 // +1
        return “a few”;
      default:
        return "lots";
    }
}      // Cyclomatic Complexity 4
```

Базовые метрики

- 2017 год, компания SonarSource развила метрику цикломатической сложности — **Когнитивная сложность (Cognitive Complexity):**
  - Игнорировать структуры языков программирования, позволяющих сокращать написание кода и делать его более читаемым (это касается обновлённого синтаксиса любого из языков).
  - Увеличивать метрику на единицу для каждого оператора, прерывающего поток исполнения кода
    - Циклические конструкции for, while, do while, ...
    - Условные конструкции: if, #if, #ifdef, тернарные операторы
  - Увеличивать метрику на единицу в случае вложенных (nested) конструкций

Когнитивная сложность

```
int sumOfPrimes(int max) {
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) { // +1
    for (int j = 2; j < i; ++j) {       // +2
      if (i % j == 0) {                 // +3
        continue OUT;                   // +1
      }
    }
    total += i;
  }
  return total;
}                  // Cognitive Complexity 7
```

```
String getWords(int number) {
    switch (number) {          // +1
      case 1:
        return "one";
      case 2:
        return "a couple";
      case 3:
        return “a few”;
      default:
        return "lots";
    }
  }      // Cognitive Complexity 1
```

Базовые метрики

- **Количество взвешенных методов на класс (Weighted Methods Per Class, WMC)** - сумма цикломатической сложности методов определенных в этом классе
- Пример:

```
class A {
    public:
        void method1(); // CC=5
        void method2(); // CC=3
}
```

- Индикатор психологической сложности понимания класса

Базовые метрики

- Метрики кода
  - **Число строк (Source Lines of Code, SLOC)**
    - количество пустых строк,
    - количество комментариев,
    - процент комментариев (отношение числа строк, содержащих комментарии к общему количеству строк, выраженное в процентах),
    - среднее число строк для функций (классов, файлов),
    - среднее число строк, содержащих исходный код для функций (классов, файлов),
    - среднее число строк для модулей.

Базовые метрики

- Метрики кода
  - **Метрики Холстеда**
    - n1 — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов),
    - n2 — число уникальных операндов программы (словарь операндов),
    - N1 — общее число операторов в программе,
    - N2 — общее число операндов в программе,
    - n=n1+n2 — словарь программы,
    - N=N1+N2 — длина программы,
    - V=N*log2n — объем программы,
    - ….

Метрики проектирования классов

- **Сцепление между классами объектов (Coupling Between Object classes, CBO)** — число связанных классов
- Связь:
  - Вызов метода другого класса
  - Использование атрибутов
  - Наследование
  - Полиморфный вызов
- Классы с чрезмерным сцеплением сложнее понимать, изменять и корректировать

Метрики проектирования классов

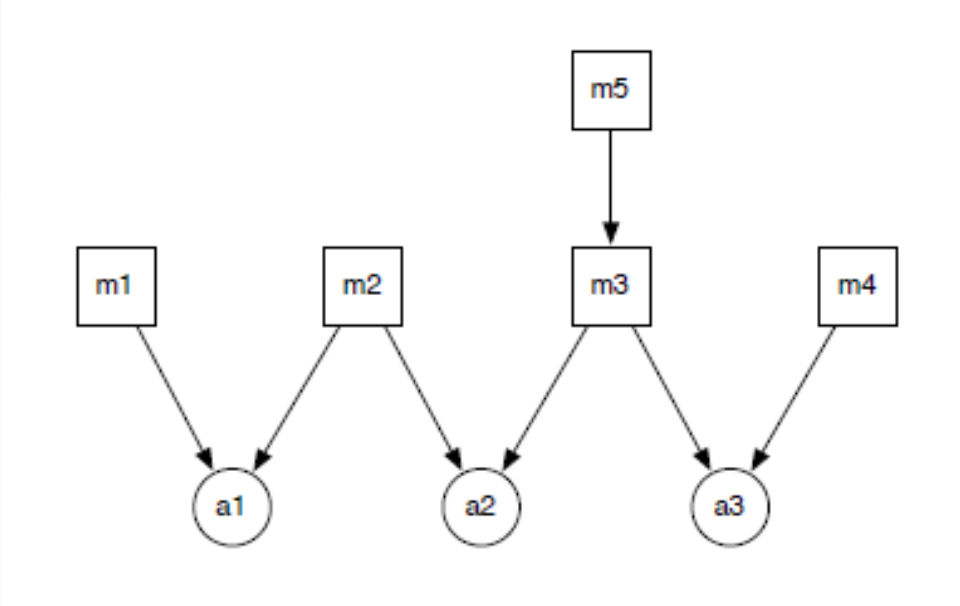
- **Коэффициент сцепления (Coupling Factor, CF)** — нормализованное соотношение между количеством клиентских отношений и общим числом возможных клиентских отношений
- Метрика вычисляется на всю программу
- Клиентское отношение:
  - Класс ссылается на метод другого класса
  - Класс ссылается на атрибут другого класса
  - Другой класс не является предком(!)

Метрики проектирования классов

- **Сцепление посылкой сообщений (Message Passing Coupling, MPC)** — количество посылаемых сообщений определенных в классе
- Вызов собственных методов класса не учитывается
- Позволяет измерить насколько реализация методов данного класса зависит от методов в других классах

Связанность классов

- **Отсутствие связанности в методах (Lack of Cohesion in Methods, LCOM)** — набор метрик
  - LCOM1 - количество пар методов в классе, которые не ссылаются на общий атрибут
    - Учет только собственных атрибутов и методов класса

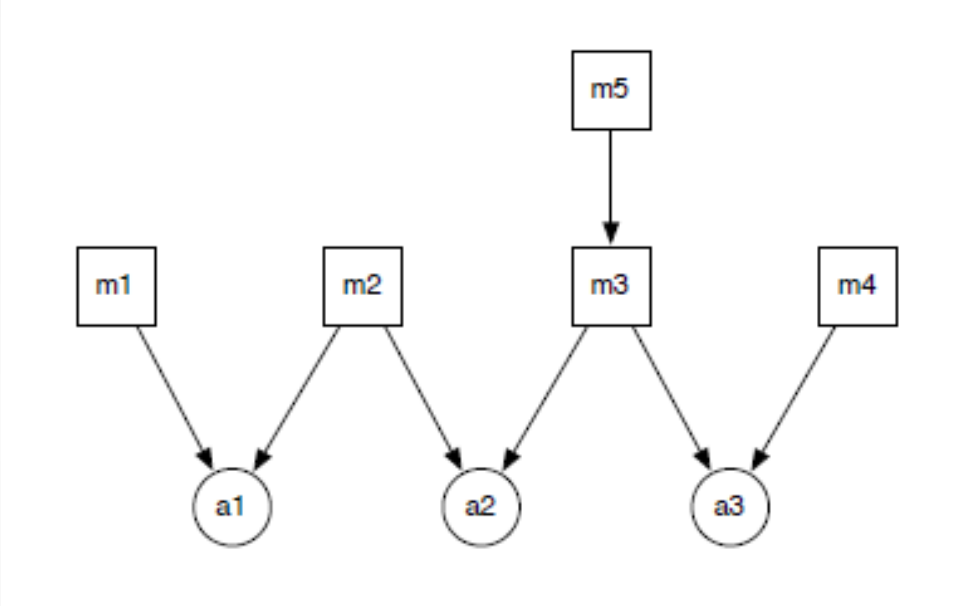


- LCOM1 = 7

  (m1, m3), (m1, m4),
  (m2, m4), (m5, m1),
  (m5, m2), (m5, m3), (m5, m4)

Связанность классов

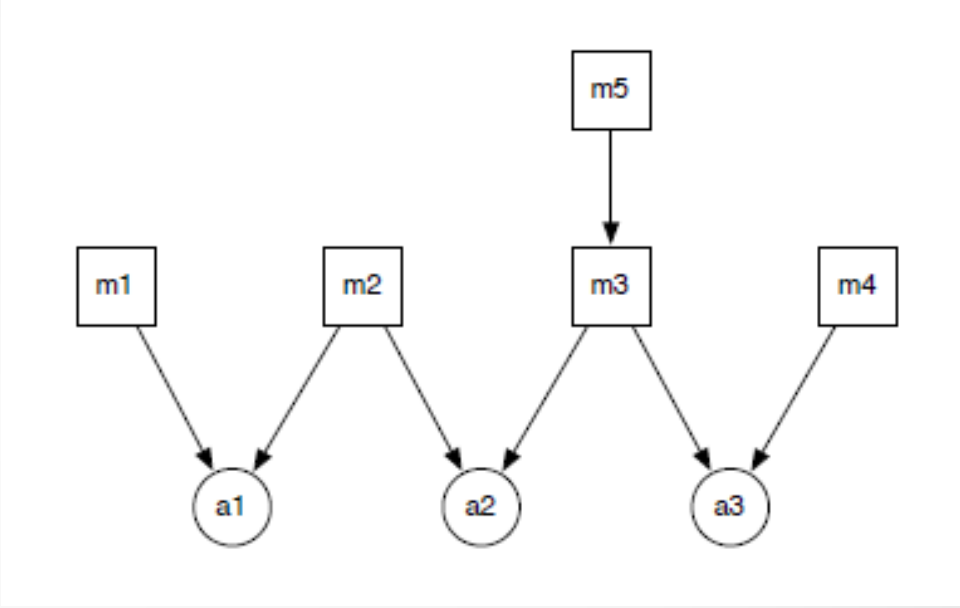
- **Отсутствие связанности в методах (Lack of Cohesion in Methods, LCOM)** — набор метрик
  - LCOM2=max(P-Q, 0), где
    - P — число пар методов, имеющих доступ к различному множеству переменных
    - Q — иначе
  - Если LCOM2 > 0, то класс можно поделить
    - Учет только собственных атрибутов и методов класса



- LCOM2 = 4
  - P = 7 ((m1, m3), (m1, m4), (m2, m4), (m5, m1), (m5, m2), (m5, m3), (m5, m4))
  - Q=3 ((m1, m2), (m2, m3), (m3, m4))

Связанность классов

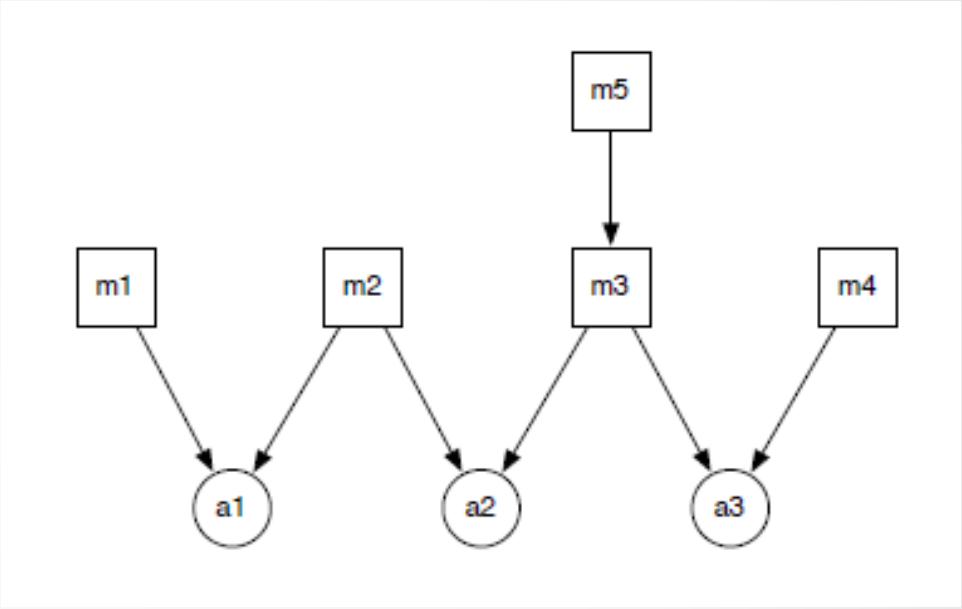
- **Отсутствие связанности в методах (Lack of Cohesion in Methods, LCOM)** — набор метрик
  - LCOM3 - количество соединенных компонент в графе методов
  - Методы в графе соединены, если методы имеют доступ к тому же атрибуту
  - Учет только собственных атрибутов и методов класса



- LCOM3 = 2
  - (m1, m2, m3, m4)
  - (m5)

Связанность классов

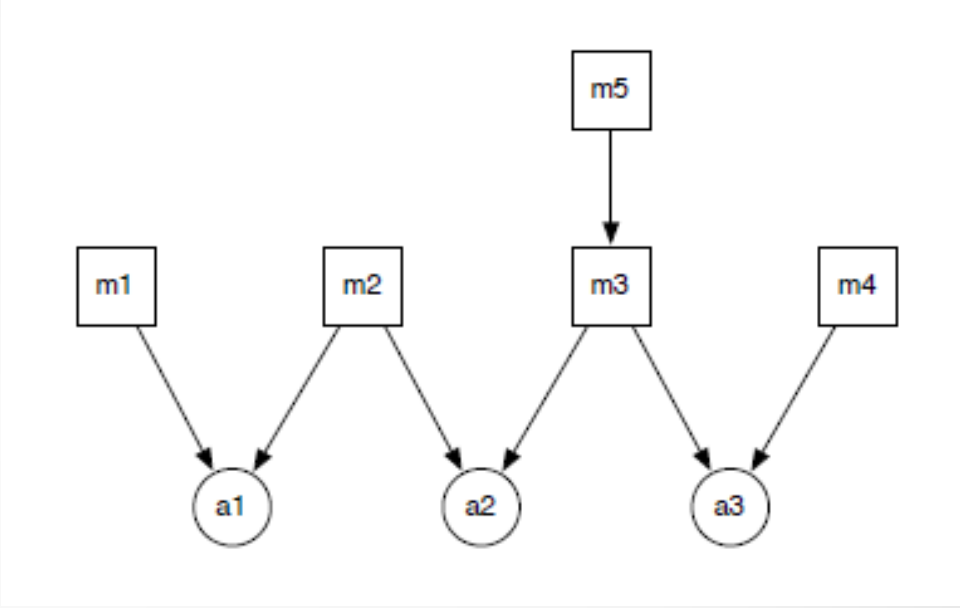
- **Отсутствие связанности в методах (Lack of Cohesion in Methods, LCOM)** — набор метрик
  - LCOM4 - количество соединенных компонент в графе методов
  - Учет транзитивных связей(!)
  - Учет только собственных атрибутов и методов класса



- LCOM4 = 1

Связанность классов

- **Отсутствие связанности в методах (Lack of Cohesion in Methods, LCOM)** — набор метрик
  - LCOM5 — вычисление по формуле
    - M — множество методов
    - NOM – количество методов
    - NOA – количество атрибутов
    - NOAcc(m) – количество атрибутов класса доступных методу m
  - Учет только собственных атрибутов и методов класса

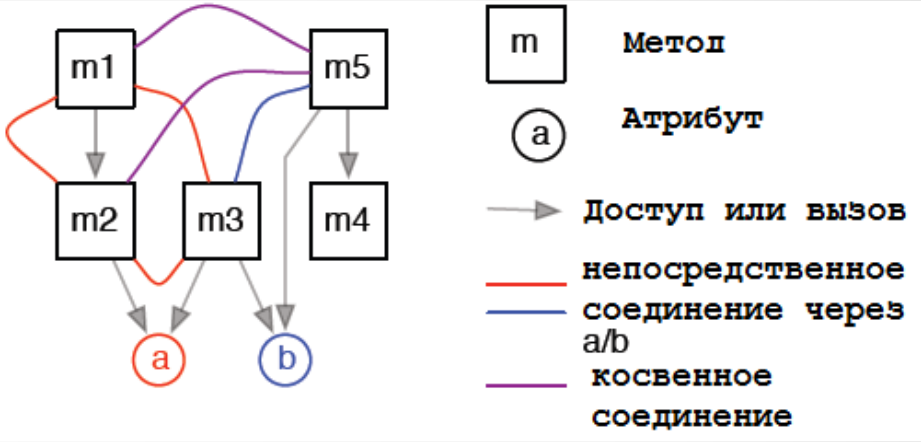


- LCOM5 = (5 - (1 + 2 + 2 + 1 + 0)/3) / (5-1) = 0,75

$$LCOM5 = \frac{NOM - \frac{\sum_{m \in M} NOAcc(m)}{NOA}}{NOM - 1}$$

Связанность классов

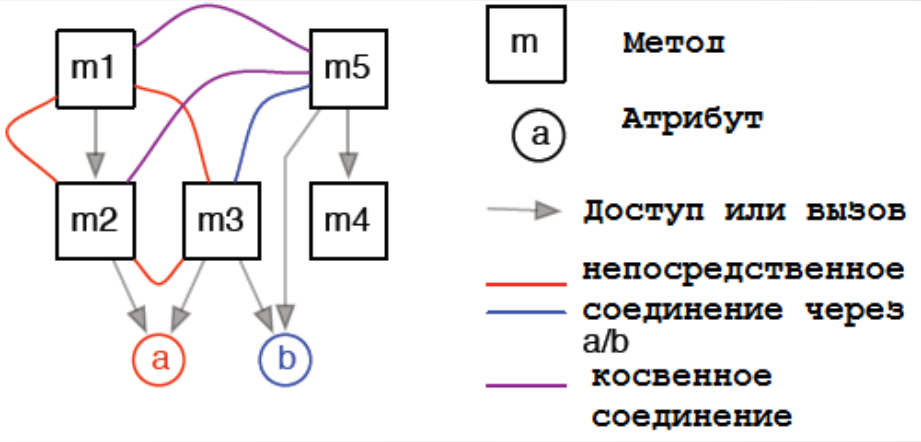
- **Плотность связанности класса (Tight Class Cohesion, TTC)** - нормализованное соотношение между количеством непосредственно связанных через атрибуты методов к общему количеству соединений между методами
- Игнорируются конструкторы и деструкторы, а также методы реализующие интерфейсы или обрабатывающие события.
- TCC = NDC / NP, где
  - NDC: количество непосредственных соединений
  - NP = N * (N-1) / 2
- Класс со значением метрики меньшим 0.5 не связан



- NDC=4
  - ((m1,m2), (m1,m3), (m2, m3)) через a
  - (m3,m5)) через b
- N = 5
- NP = 5 * (5-1) / 2 = 10
- TCC = 4/10

Связанность классов

- **Потеря сцепления классом (Loose Class Cohesion, LCC)** - нормализованное соотношение между количеством непосредственно или **косвенно** связанных через атрибуты методов к общему количеству соединений между методами
- Игнорируются конструкторы и деструкторы, а также методы реализующие интерфейсы или обрабатывающие события.
- LCC = (NDC + NIC) / NP, где
  - NDC: количество непосредственных соединений
  - NIC: количество косвенных соединений
  - NP = N * (N-1) / 2
- Класс со значением метрики меньшим 0.5 не связан



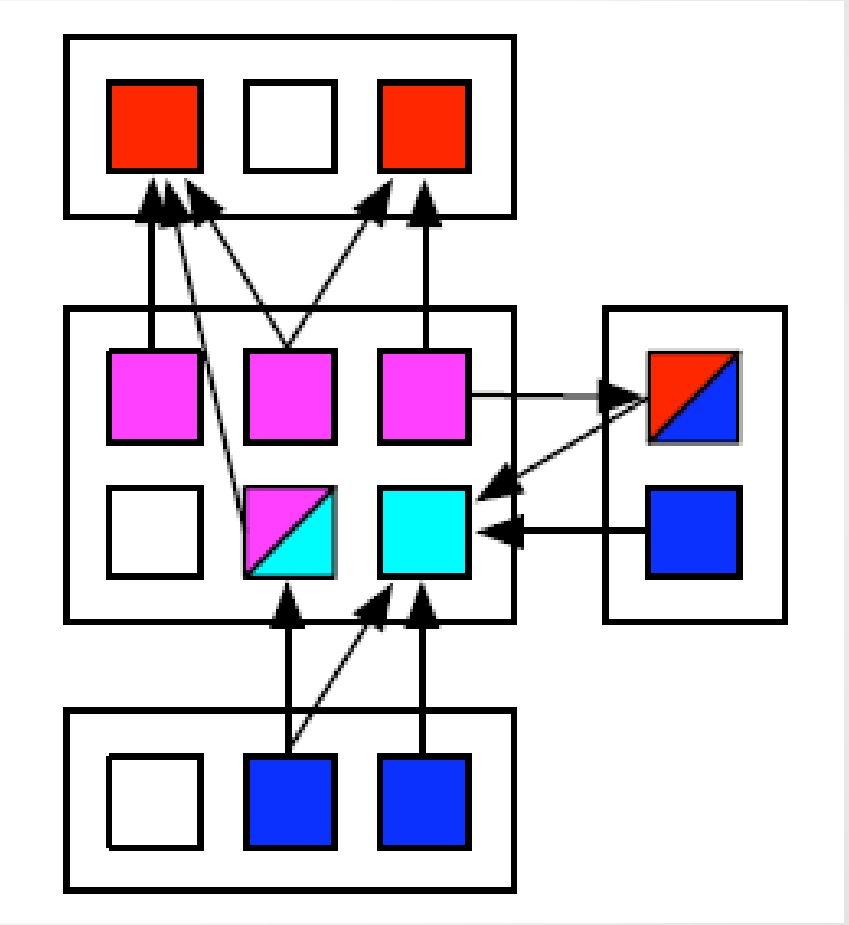
- NDC=4
  - ((m1,m2), (m1,m3), (m2, m3)) через a
  - (m3,m5)) через b
- NIC=2 ((m1,m5) и (m2,m5)) через m3
- N = 5
- NP = 5 * (5-1) / 2 = 10
- LCC = 6/10

Архитектура пакетов

- Пакет (модуль, библиотека) — набор классов
- Связанность классов <> Связанность пакетов
- Связанность / сцепление пакетов можно улучшить перемещением классов между пакетами, рафакторинга классов, расщепление классов
- Стоимость сопровождения пакета зависит от связности сцепления и сложности

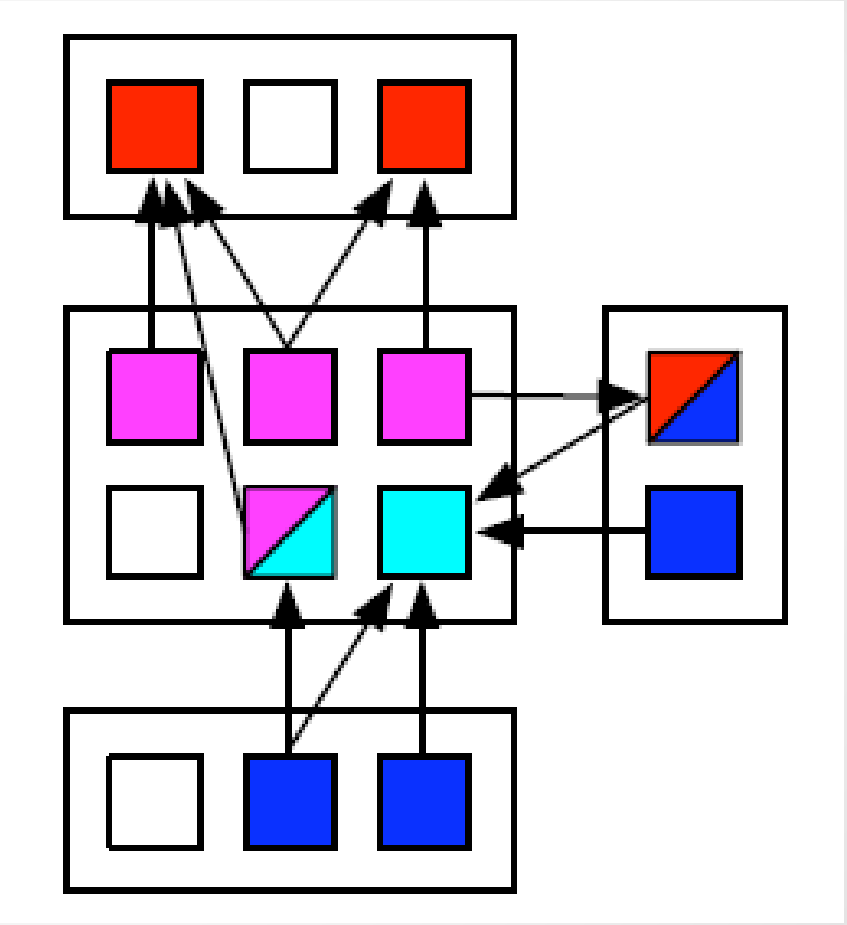
Архитектура пакетов

- Простейшие метрики
  - **Количество классов в пакете (Number of Classes in Package, NCP)**
    - NCP = 2, 3, 3, 6
  - **Количество экспортируемых классов (Out-port Classes, OutC)** — количество классов в пакете, которые зависят от классов из других пакетов
    - OutC = 2, 0, 2, 4



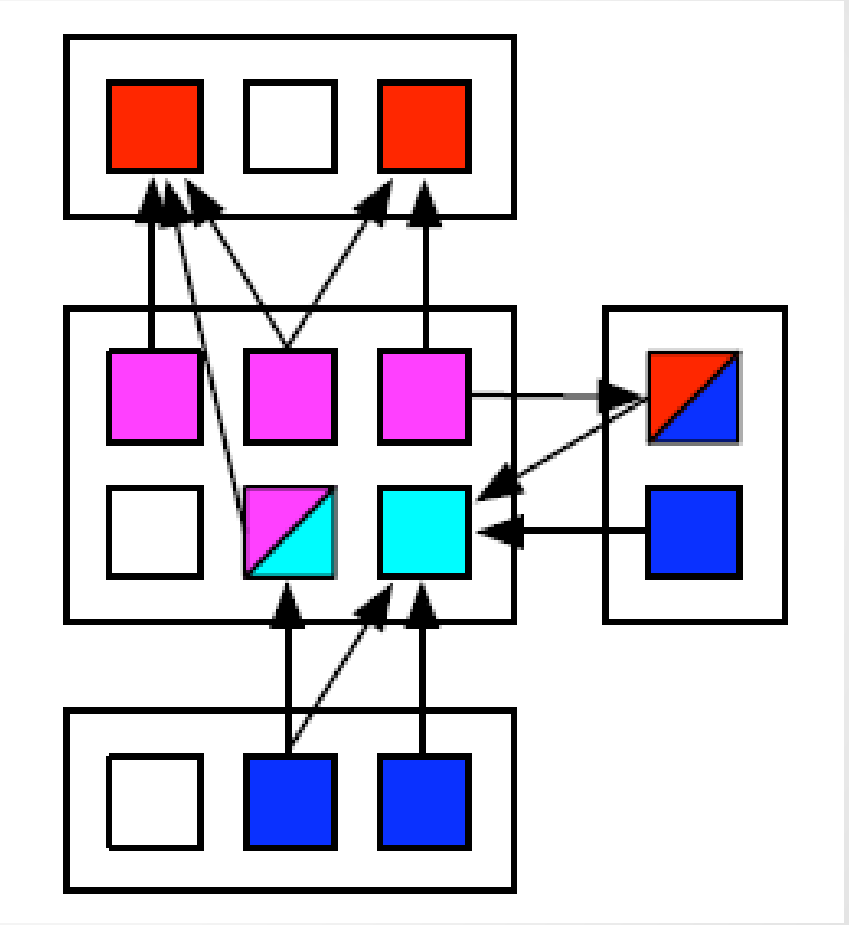
Архитектура пакетов

- Простейшие метрики
  - **Количество импортируемых классов (In-port Classes, InC)** — количество классов в пакете, от которых зависят классы в других пакетах
    - InC = 1, 4, 0, 2
  - **Скрытые классы (Hidden Classes, HC)** — количество классов, которые не имеют связей с классами из других пакетов
    - HC = 0, 1, 1, 1



Архитектура пакетов

- Простейшие метрики
  - **Количество классов-провайдеров системы классов (System Providers Classes, SPC)** - количество классов в системе, от которых зависят классы пакета
    - SPC = 1, 0, 2, 3
  - **Количество классов-клиентов пакета (System Client Classes, SCC)** — количество классов в пакете, от которых зависят классы в других пакетах
    - SCC = 1, 2, 0, 4

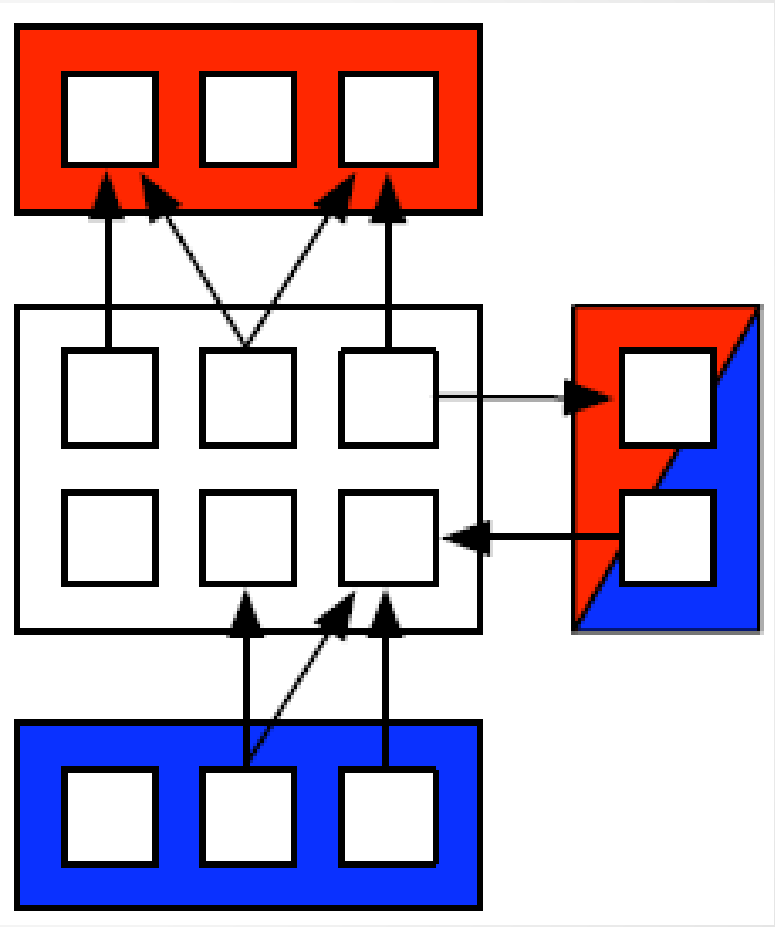


Метрики проектирования пакетов

- **Принцип стабильных зависимостей (Stable Dependencies Principle, SDP)** - зависимость должна быть направлена в направлении стабильных классов и пакетов
  - Пакет с большим числом входящих отношений зависимости из других пакетов, должен быть стабилен, поскольку ответственен за эти пакеты
  - Пакеты без каких либо входящих зависимостей рассматриваются как независимые и нестабильные
- **Принцип стабильных абстракций (Stable Abstractions Principle, SAP)** — стабильные пакеты должны быть абстрактными пакетами
  - Для увеличения гибкости приложений нестабильные пакеты должны быть легко изменяемыми
  - Стабильные пакеты должны быть легко расширяемыми, и, следовательно, должны быть насколько возможно абстрактными

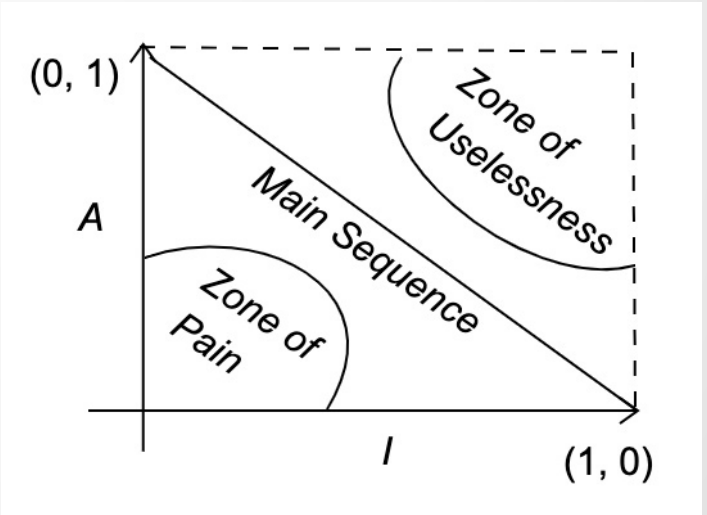
Метрики проектирования пакетов

- **Центробежное сцепление (Efferent Coupling, Ce)** — количество модулей от которых он зависит
  - Выходящие зависимости
  - Ce = 1, 0, 1, 2
- **Центростремительное сцепление (Afferent Coupling, Ca)** — количество модулей, которые зависят от этого модуля
  - Входящие зависимости
  - Ca = 1, 1, 0, 2



Метрики проектирования пакетов

- **Абстрактность (Abstractness, A)** — соотношение между количеством абстрактных классов и общим количеством классов пакета
  - Значение метрики изменяется в пределах [0..1]
  - Значение 0 означает полностью конкретный пакет
  - Значение 1 – полностью абстрактный пакет.
- **Нестабильность (Instability, I)** — отношение «ответственных и безответственных»
  - I = Ce(P) / (Ce(P)+Ca(P))
    - Ce — Центробежное сцепление (выходящие)
    - Ca — Центростремительное сцепление (входящие)
  - Значения меняются в пределах [0; 1]
- **Расстояние (Distance, D)** — расстояние до баланса
  - D = |A + I - 1|
  - Баланс: A + I - 1 = 0
  - Стабильные пакеты должны быть абстрактными пакетами (A = 1 и I = 0)
  - Нестабильные пакеты должны быть конкретными пакетами (A = 0 и I = 1)



Заключение

- ЭТО НЕ ВСЕ МЕТРИКИ!!!
- Использование метрик позволяет оценить, сравнить и выбрать лучшее решение
- Числовые значения метрик можно отслеживать
- Использование плагинов среды или специальных программных инструментов для расчета метрик

- Метрики неспособны показать объективную картину
  - Оптимизация решения под метрики
  - Формальное использование метрики без учета контекста