Detecting software defects using relational rules mining

Симиютин Борис

научный руководитель: Тимофей Брыксин

СПб АУ НОЦНТ РАН

1 июня 2017 г.

Detecting software design defects using relational association rule mining
http://bit.ly/2rlv5DN

Как нам формально определить, хороший код, или нет?

Как нам формально определить, хороший код, или нет?

- Берем некоторый набор метрик

Как нам формально определить, хороший код, или нет?

- Берем некоторый набор метрик
- Определяем сущности нашего кода, например, классы

Как нам формально определить, хороший код, или нет?

- Берем некоторый набор метрик
- Определяем сущности нашего кода, например, классы
- Переводим их в вектора из пространства метрик

Как нам формально определить, хороший код, или нет?

- Берем некоторый набор метрик
- Определяем сущности нашего кода, например, классы
- Переводим их в вектора из пространства метрик
- Что-то делаем

Association rules

Association rules

- Пример: ”Если покупатель купит хлопья для завтрака, то скорее всего он купит и молоко.”
- В нашем случае: ”Если между метриками установилось какое-то соотношение, то скорее всего оно характеризует код”
- Например, $a_1 \leq a_2 \leq a_3$, где a_1, a_2, a_3 - какие-то метрики

Формализм

Формализм

- Association rule:
  $(a_1, a_2, a_m) \Rightarrow (a_1\mu_1 a_2\mu_2 ... a_{m-1}\mu_{m-1}a_m$, где a_i - атрибуты записи, $\{\mu_i\}$ - отношения. Например, $\{<, >, =\}$
- Support: $Support(X) =$ процент сущностей, в которых есть все атрибуты из правила
- Confidence: $Confidence(X \Rightarrow Y) =$ процент сущностей из $Support(X)$, на которых выполняется правило
- Правило $X \Rightarrow Y$ называется *интересным*, если $Support(X) \geq s_{min}, Confidence(X \Rightarrow Y) \geq c_{min})$

Алгоритм

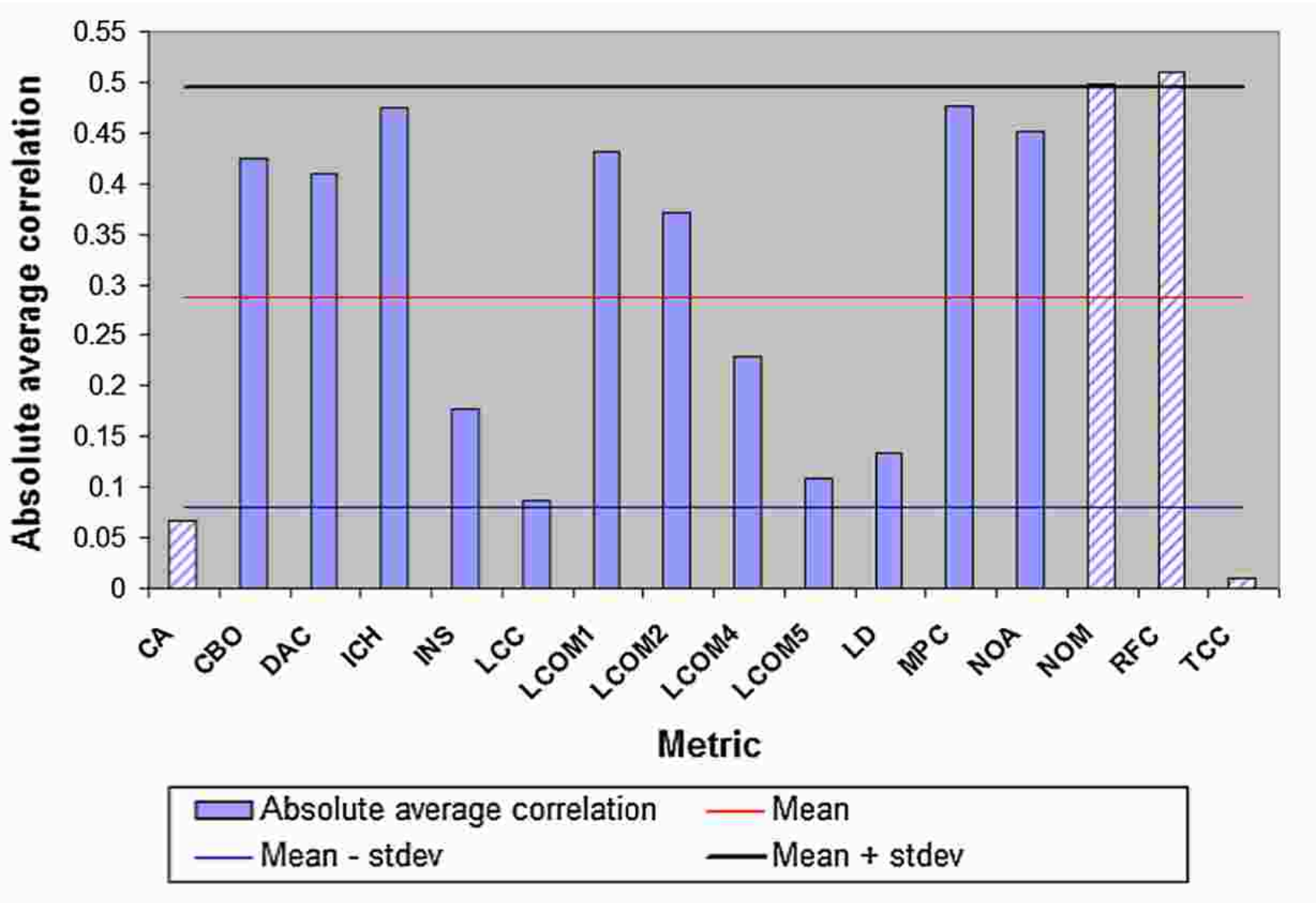
Алгоритм обучения

1. Среди набора метрик отбираем перспективные
2. Ищем интересные правила
3. Определяем критерии, по которым будем отделять хороший код от плохого

Алгоритм обучения: отбор метрик

1. Pearson correlation coefficient
2. $avg(sm_i)$ - среднее значение корреляции между sm_i и всеми остальными метриками
3. $mean = mean(avg), stddev = stddev(avg)$
4. Оставляем только те, для кого $|avg(sm_i) - m| > stddev$

Алгоритм обучения: отбор метрик. пример



Алгоритм обучения: поиск правил

1. Выбираем множество отношений, например, $\{<, >, =\}$
   1. Ищем все интересные правила длины 2
   2. Ищем все интересные правила длины 3
   3. Оставляем наиболее длинные правила
   4. Ищем все интересные правила длины 4
   5. ...
   6. Пока на новой итерации не нашли новых правил

Алгоритм построения: поиск правил. пример

```
public class Class_A {
  public static int attributeA1;
  public static int attributeA2;

  public static void mA1(){
      attributeA1 = 0;
      mA2();
  }

  public static void mA2(){
          attributeA2 = 0;
          attributeA1 = 0;
    }

  public static void mA3(){
          attributeA2 = 0;
          attributeA1 = 0;
          mA1();
          mA2();
  }
}
```

```
public class Class_B {
  private static int attributeB1;
  private static int attributeB2;

  public static void mB1(){
          Class_A.attributeA1=0;
          Class_A.attributeA2=0;
          Class_A.mA1();
    }

  public static void mB2(){
          attributeB1=0;
          attributeB2=0;
    }

  public static void mB3(){
          attributeB1=0;
          mB1();
          m2();
  }
}
```

Найденные правила:

Length	Rule	Confidence
2	DIT > NOC	1
2	NOC < FI	0.625
2	FI > FO	0.5
3	DIT > NOC < FI	0.625
3	DIT > NOC < FO	0.75
3	NOC < FI > FO	0.5
4	DIT > NOC < FI > FO	0.5

После сокращения:

Length	Rule	Confidence
3	DIT > NOC < FO	0.75
4	DIT > NOC < FI > FO	0.5

Метрики:

- Depth of inheritance tree(DIT)
- Number of children(NOC)
- Fan-In(FI)
- Fan-Out(FO)

Алгоритм классификации

1. Пусть на вход дали множество сущностей S_{new}
2. Проверяем на них все правила
3. Находим множество P потенциально ошибочных сущностей - тех, для кого процент ошибок p_e выше порога
4. Отбираем из них тех, у кого p_e выше среднего по множеству
5. Если $P = \varnothing$, то отбираем тех, у кого $n_e(s_i) > meanErr(S_{new}) + stddevErr(S_{new})$

Реализация

Реализация

- Алгоритм реализован на Java

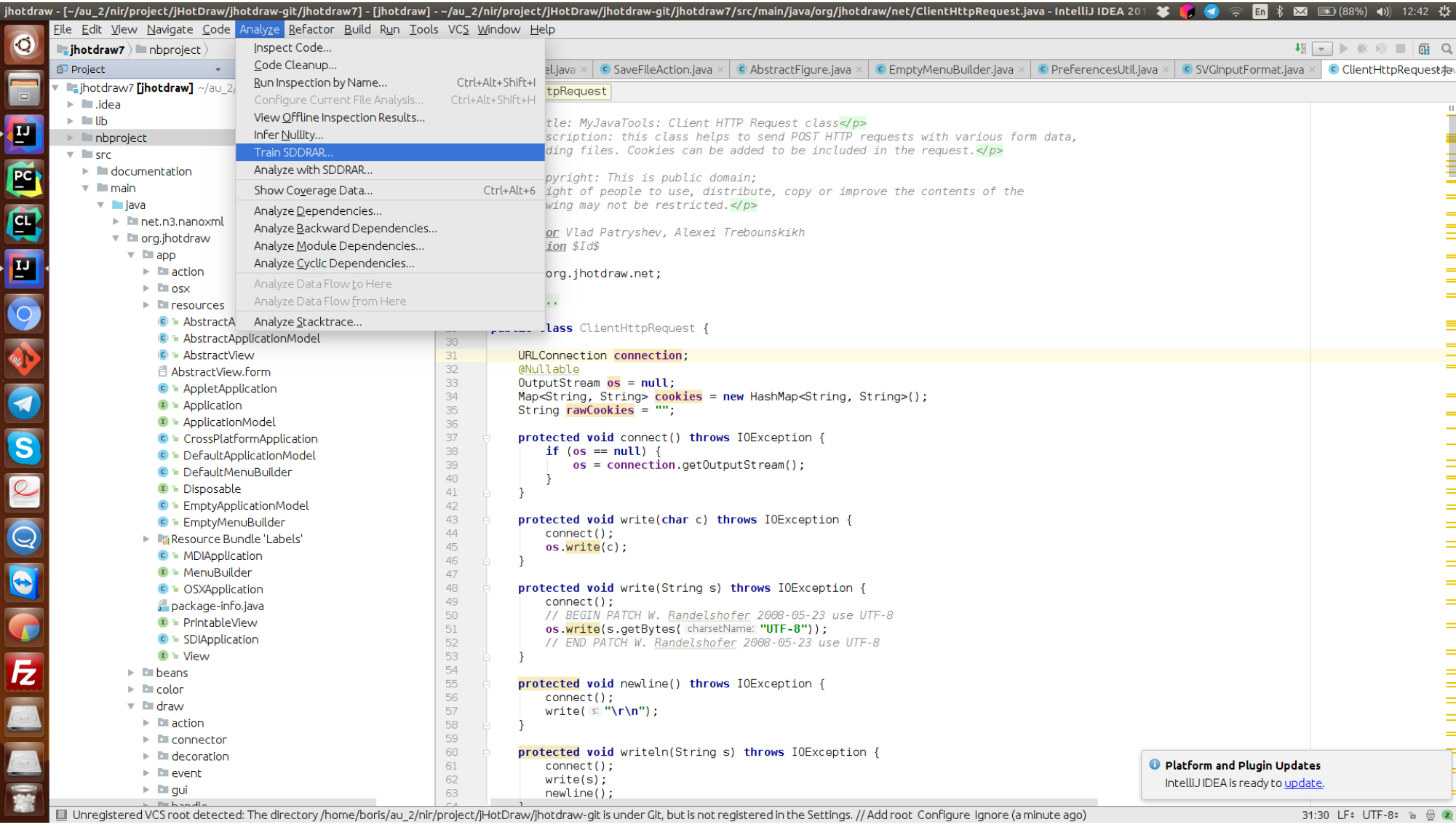
Реализация

- Алгоритм реализован на Java
- В качестве кодового окружения использован плагин MetricsReloaded для IDEA

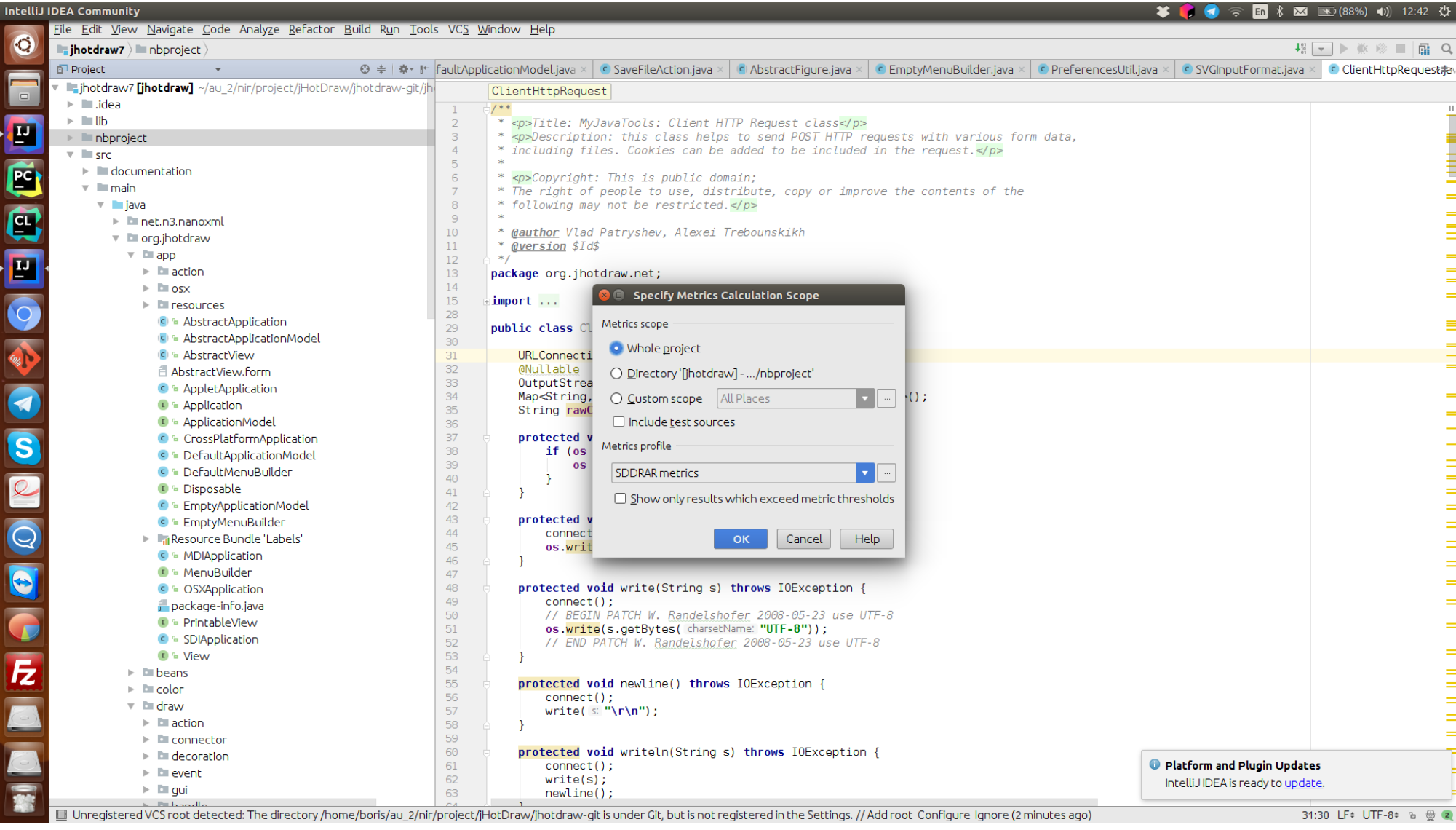
Реализация

- Алгоритм реализован на Java
- В качестве кодового окружения использован плагин MetricsReloaded для IDEA
- С новыми метриками, добавленными в другом НИРе

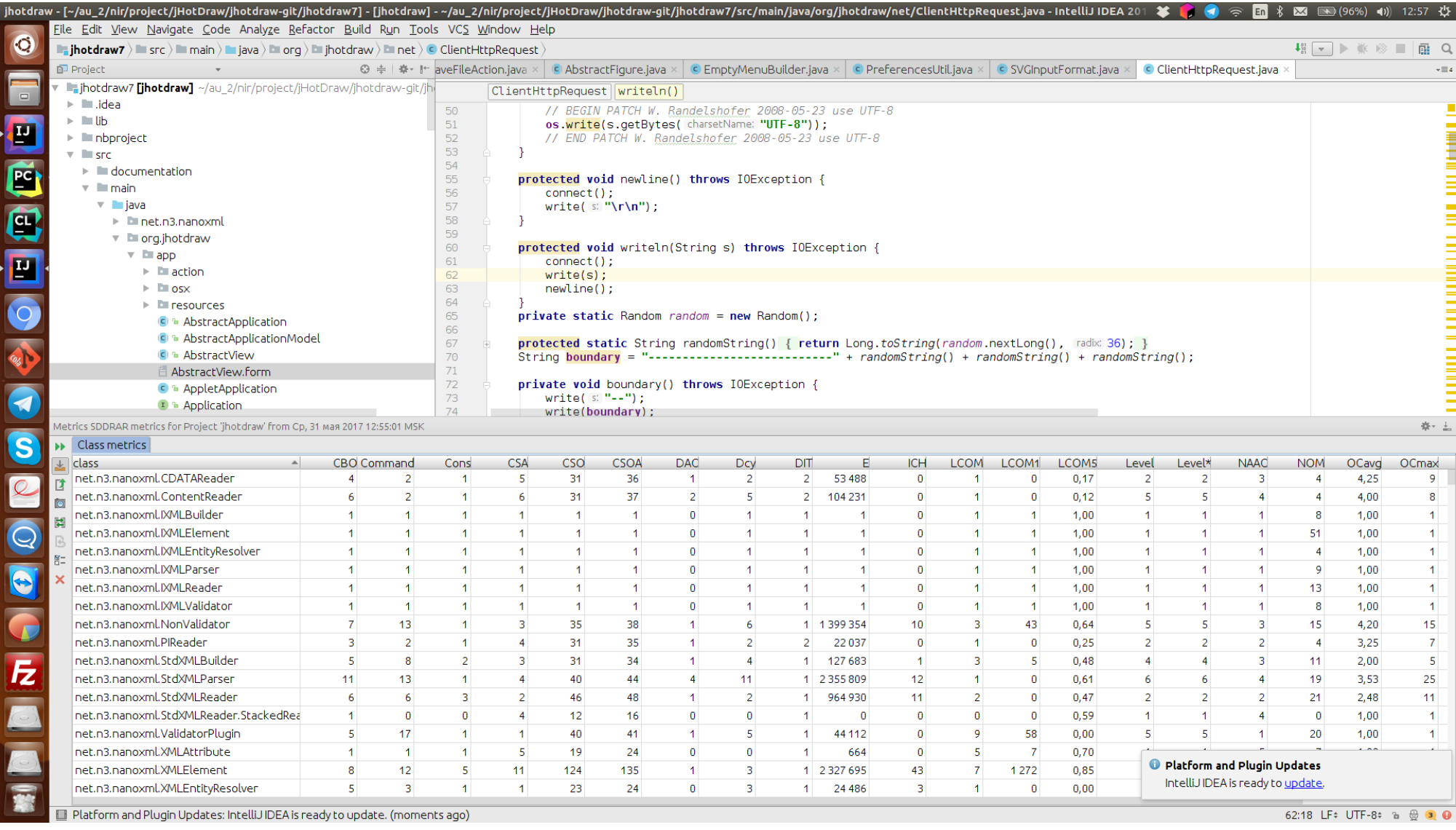
Использование



Использование



Использование



Результаты

Результаты

Интересные метрики:

- NumOperationsAdded
- NumTransitiveDependencies
- DepthOfInheritance
- CouplingBetweenObjects
- NumAttributesAdded
- LackOfCohesionOfMethods
- NumCommands
- MaximumOperationComplexity
- LevelOrder
- AverageOperationParameters
- NumOperationsOverridden
- NumInterfacesImplemented
- ClassSizeAttributes
- NumDependencies
- NumDependents
- ClassSizeOperations
- AdjustedLevelOrder
- HalsteadEffort
- ClassSizeOperationsAttributes
- NumOperationsInherited

Результаты

	FTP4J	ISO8583	WinRun4J
MySDDRAR	FTPClient(God) FTPFile(God) Base64	MessageFactory(God)	Closure(one_big_method) FileVerb Launcher NativeBinder(God) NativeHelper Native(interesting_case_only_native_methods) PInvoke,PInvoke.NativeStruct Registry.FILETIME,Registry.QUERY_INFO(public_not_final_data_fields) RegistryKey
SDDRAR	FTPClient	MessageFactory	NativeBinder
iPlasma	FTPClient FTPFile	MessageFactory	Closure FileVerb NativeBinder
JDeodorant	FTPClient NVTASCIIWriter NVTASCIIReader FTPProxyConnector FTPCommunicationChannel	MessageFactory ISOMessage	Launcher FileAssociation RegistryKey

Результаты

Что еще хотелось:

- Анализировать, почему помеченные классы оказались помеченными, и классифицировать по типичным архитектурным промахам
- Оформить в виде отдельного плагина