Генерация высокоуровневых представлений изменений программного кода на основе абстрактных синтаксических деревьев

Абрамов Дмитрий Павлович

Научный руководитель: к.т.н. Т. А. Брыксин, JetBrains Research

Санкт-Петербургская школа физико-математических и компьютерных наук

НИУ ВШЭ — Санкт-Петербург

9 Июня 2020

Задача сравнения версий исходного кода

Редакционный сценарий

Набор действий превращающий версию кода "до изменений" в версию кода "после изменений".

```
public void execute(Runnable command) {
    super.execute(taskDecorator.decorate(command));
    Runnable decorated = taskDecorator.decorate(command);
    if (decorated != command) {
        decoratedTaskMap.put(decorated, command);
    }
    super.execute(decorated);
}
```

Описание проблемы

Для таких задач, как ревью кода, исследования разработки ПО, разрешение конфликтов слияния, полезно представление изменений отражающее структуру и взаимосвязи между изменениями.

- Построчная разница не отражает информацию о сложной иерархической структуре кода
- Низкоуровневые подходы к сравнению AST, такие как GumTree[1], оперирующие действиями с отдельными вершинами, не отражают взаимосвязи между изменениями
- Подходы к генерации высокоуровневых редакционных сценариев специфичны для языка Java, что ограничивает их практическую и академическую применимость

[1] Falleri и др., “Fine-grained and accurate source code differencing”, 2014.

Цель и задачи

Цель

Разработать минимально зависящий от языка программирования инструмент для представления изменений в программном коде отражающих его структуру на уровне абстракции близком к тому, на котором были произведены эти изменения.

Задачи:

- Разработать минимально зависящий от языка алгоритм построения высокоуровневых редакционных сценариев
- Создать расширяемый инструмент сравнения версий кода на основе GumTree и предложенного алгоритма
- Создать инструмент визуализации таких высокоуровневых изменений
- Протестировать инструмент на различных изменениях на Python и Java

Редакционный сценарий — существующие подходы I

Построчная разница[2]

- \+ Универсальный подход
- \+ Быстро работает
- – Не имеет информации о структуре кода

Сравнение AST[3,4]

- \+ Полностью отражает структуру кода
- – Низкоуровневый подход, не отражает взаимосвязи между изменениями и близких разработчику манипуляций с кодом

---

[2] Myers, “An O(ND) Difference Algorithm and Its Variations.”, 1986.

[3] Chawathe и др., “Change Detection in Hierarchically Structured Information”, 1996.

[4] Falleri и др., “Fine-grained and accurate source code differencing”, 2014.

Редакционный сценарий — существующие подходы II

Выделение рефакторингов[5]

- \+ Информация о структуре
- \+ Напрямую отражает намерения разработчиков
- – Далеко не все изменения являются рефакторингами

Высокоуровневая разница AST[6,7]

- \+ Информация о структуре
- \+ Отражает взаимосвязи между изменениями
- – Типы изменений определены вручную
- – Подход специфичен только для языка Java

---

[5] Tsantalis и др., “Accurate and Efficient Refactoring Detection in Commit History”, 2018.

[6] Huang и др., “ClDiff: Generating Concise Linked Code Differences”, 2018.

[7] Fluri и др., “Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction”, 2007.

Редакционный сценарий — алгоритм

- В качестве информации о языке используется информация о типе вершины / синтаксической категории (атомарные и вложенные операции, выражения, блоки)
- Идея алгоритма: группировка изменений из низкоуровневой разницы AST
- 3 этапа алгоритма:
  - Обработка действий перемещения вершины
  - Обработка удалений и добавлений вложенных операций
  - Обработка действий с атомарными операциями и выражениями (локальные изменения)
- Алгоритм старается выделять изменения на том уровне абстракции, на котором они могли быть произведены разработчиком

Инструмент сравнения версий HLDiff

Подход:

- Информация о языке содержится в конфигурационных файлах
- Алгоитм из предыдущего пункта применяется к низкоуровневому редакционному сценарию и соответствиям полученным с помощью GumTree

Чтобы поддержать новый язык программирования нужно:

- Реализовать обертку над парсером, которая преобразует результат парсинга в AST — дерево из вершин, хранящих информацию о типе, текстовом вхождениии и представлении
- Разметить типы вершин в файле конфигурации языка (вложенные и атомарные операции, выражения, блоки)

Визуализация

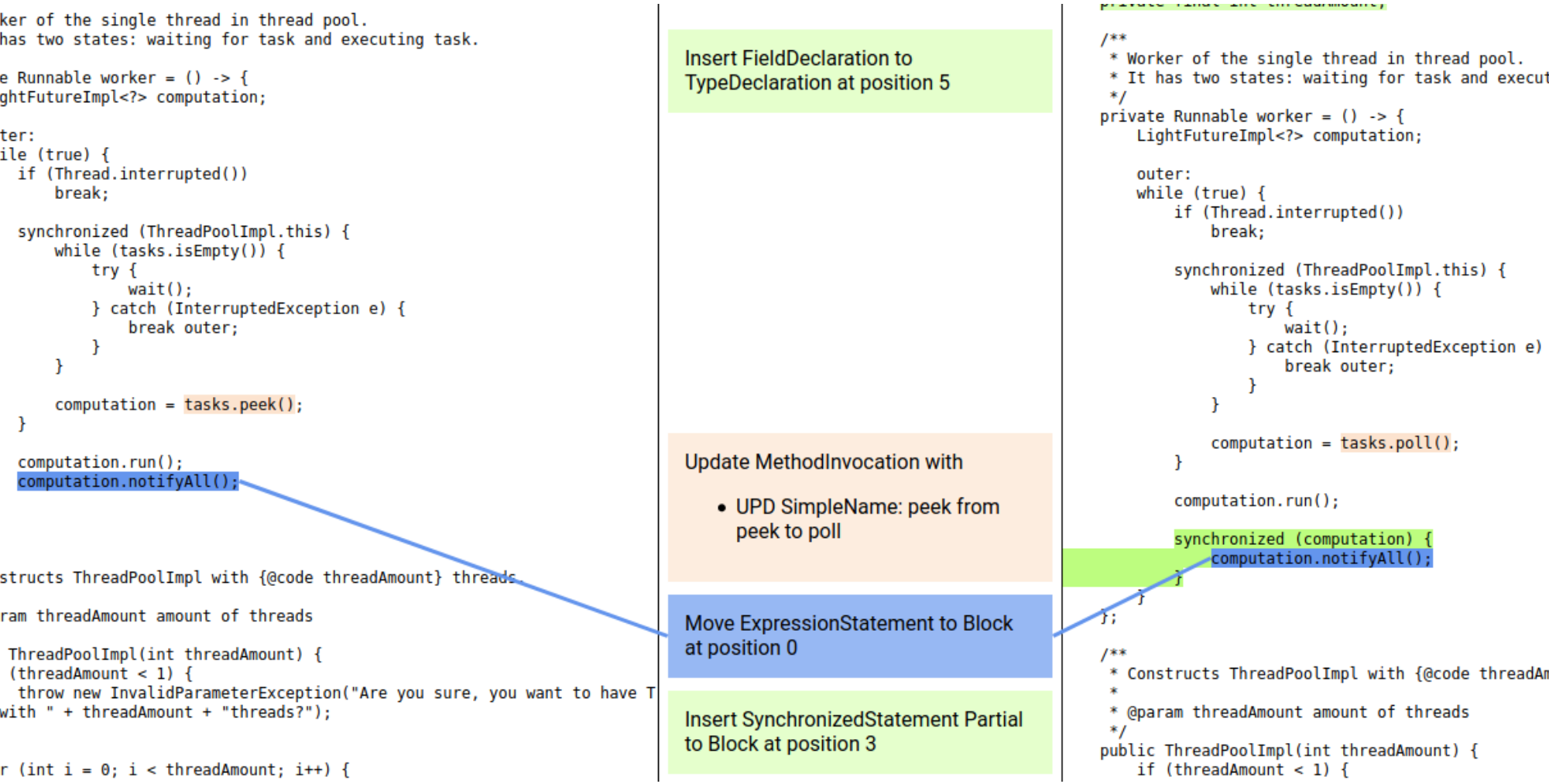
Цели:

- Тестирование подхода и оценка получаемых изменений
- Прототип инструмента ревью кода на базе предлагаемого подхода

Рассмотренные альтернативы:

- Различные инструменты визуализации построчной разницы, текстовый формат
  - – Не позволяет ясно отобразить и список действий и код
  - – Почти нечего переиспользовать и нужно многое добавлять
  - – Отсутствие интерактивности
- Визуализация сравнения версий платформы IntelliJ
  - \+ Не только уровень строк, интерактивность
  - – Высокая сложность интеграции

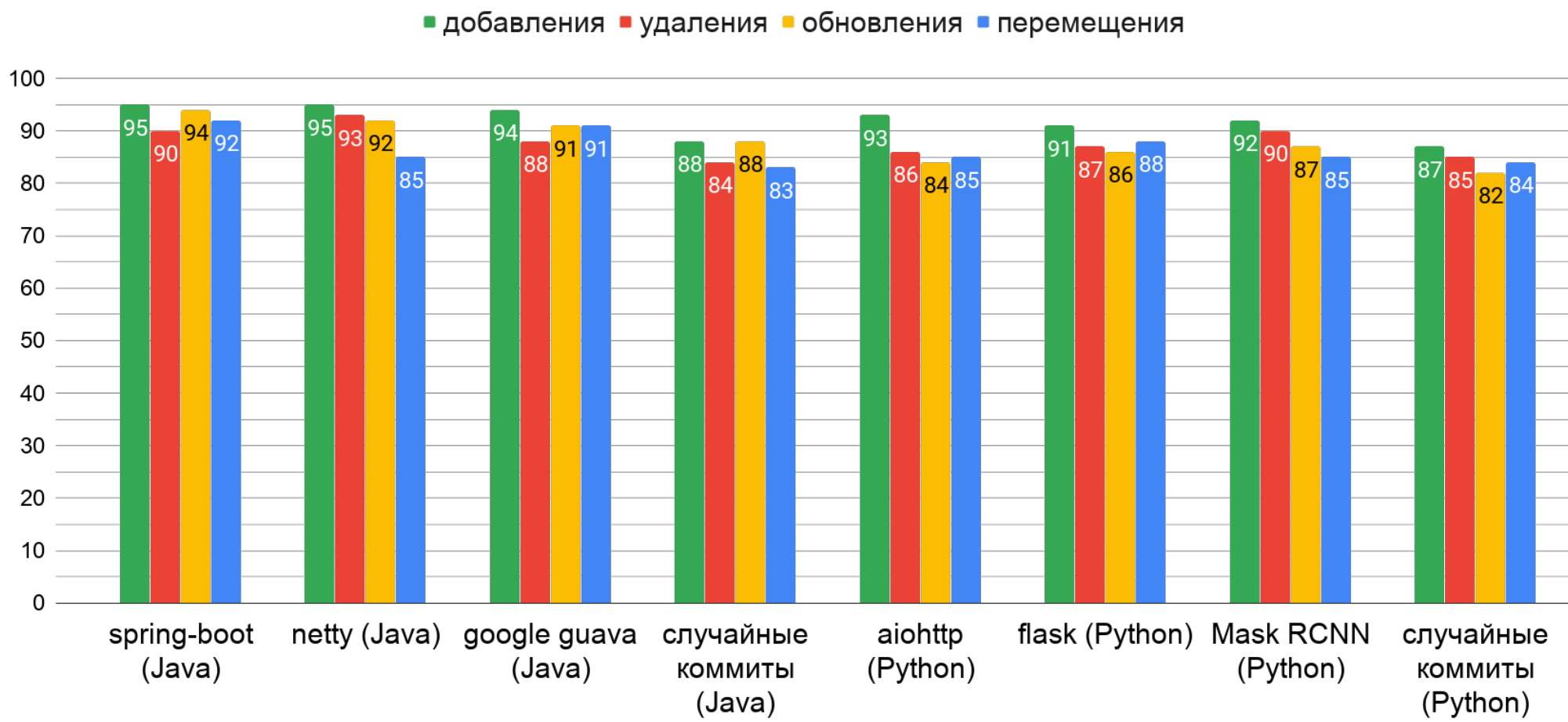
Визуализация



Тестирование — постановка эксперимента

- Экспертная оценка
- Каждое изменение оценивается: успешно или не успешно выделено
- Оценка производилась разработчиками и студентами старших курсов дисциплин в области программной инженерии
- Оценено 1307 изменений из 102 коммитов.
- Данные: проекты различной направленности на Java и Python, случайные коммиты

Результаты оценки



Процент успешно выделенных высокоуровневых изменений

Результаты

- Создан алгоритм генерации высокоуровневых изменений, сохраняющий информацию об иерархической структуре кода и использующий минимум информации о языке программирования
- Реализован инструмент HLDiff[8] на основе данного алгоритма; предусмотрена возможность расширения на новые языки программирования, реализована поддержка Java и Python
- Реализован инструмент для визуализации высокоуровневых изменений
- Предложенное решение показало себя, как обобщаемый на различные языки программирования подход к представлению изменений, отражающему взаимосвязи между ними.
  Его оценка продемонстрировала достаточно высокую долю удачно выделенных высокоуровневых изменений на разных языках программирования (Java — 90%, Python — 87%).

[8] Инструмент HLDiff: https://github.com/JetBrains-Research/hldiff

Библиография I

- Chawathe, Sudarshan S. и др. “Change Detection in Hierarchically Structured Information”. В: *In Proceedings of the ACM SIGMOD International Conference on Management of Data*. 1996, с. 493—504.
- Falleri, Jean-Rémy и др. “Fine-grained and accurate source code differencing”. В: *ACM/IEEE International Conference on Automated Software Engineering, ASE ’14, Vasteras, Sweden - September 15 - 19, 2014*. 2014, с. 313—324. DOI: 10.1145/2642937.2642982. URL: http://doi.acm.org/10.1145/2642937.2642982.
- Fluri, Beat и др. “Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction”. В: *IEEE Trans. Softw. Eng.* 33.11 (нояб. 2007), с. 725—743. ISSN: 0098-5589. DOI: 10.1109/TSE.2007.70731. URL: https://doi.org/10.1109/TSE.2007.70731.

Библиография II

Huang, Kaifeng и др. “ClDiff: Generating Concise Linked Code Differences”. В: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ASE 2018. Montpellier, France: ACM, 2018, с. 679—690. ISBN: 978-1-4503-5937-5. DOI: 10.1145/3238147.3238219. URL: http://doi.acm.org/10.1145/3238147.3238219.

Myers, Eugene W. “An O(ND) Difference Algorithm and Its Variations.”. В: *Algorithmica* 1.2 (1986), с. 251—266. URL: http://dblp.uni-trier.de/db/journals/algorithmica/algorithmica1.html#Meyers86.

Библиография III

Tsantalis, Nikolaos и др. “Accurate and Efficient Refactoring Detection in Commit History”. В: *Proceedings of the 40th International Conference on Software Engineering*. ICSE ’18. Gothenburg, Sweden: Association for Computing Machinery, 2018, с. 483—494. ISBN: 9781450356381. DOI: 10.1145/3180155.3180206. URL: https://doi.org/10.1145/3180155.3180206.

Конфигурация — Пример

```
{
  "language": "Java",
  "extensions": [".java"],
  "statements" : [
    {"id": 6, "type": "atomic", "name": "AssertStatement"},
    {"id": 7, "type": "nested", "name": "Assignment"},
    {"id": 8, "type": "block", "name": "Block"}
    {"id": 22, "type": "expr", "name": "FieldAccess"},
    ...
  ]
}
```

Пример конфигурации языка в инструменте HLDiff.

Алгоритм — Пример

Рассмотрим алгоритм на примере изменения из коммита 3c1adf7 из репозиория spring-framework.

```
public void execute(Runnable command) {
    super.execute(taskDecorator.decorate(command));
    Runnable decorated = taskDecorator.decorate(command);
    if (decorated != command) {
        decoratedTaskMap.put(decorated, command);
    }
    super.execute(decorated);
}
```

Алгоритм — Пример

```
|-Block (n1)
  |-ExpressionStatement (n2)
    |-SuperMethodInvocation (n3)
      |-SimpleName:execute (n4)
      |-MethodInvocation (n5)
        |-SimpleName:taskDecorator (n6)
        |-SimpleName:decorate (n7)
        |-SimpleName:command (n8)
```

```
|-Block (n9)
  |-VariableDeclarationStatement (n10)
  | |-SimpleType:Runnable (n11)
  | | |-SimpleName:Runnable (n12)
  | |-VariableDeclarationFragment (n13)
  |   |-SimpleName:decorated (n14)
  |   |-MethodInvocation (n15)
  |     |-SimpleName:taskDecorator (n16)
  |     |-SimpleName:decorate (n17)
  |     |-SimpleName:command (n18)
  |-IfStatement (n19)
  | |-InfixExpression:!= (n20)
  | | |-SimpleName:decorated (n21)
  | | |-SimpleName:command (n22)
  | |-Block (n23)
  |   |-ExpressionStatement (n24)
  |     |-MethodInvocation (n25)
  |       |-SimpleName:decTaskMap (n26)
  |       |-SimpleName:put (n27)
  |       |-SimpleName:decorated (n28)
  |       |-SimpleName:command (n29)
  |-ExpressionStatement (n30)
   |-SuperMethodInvocation (n31)
     |-SimpleName:execute (n32)
     |-SimpleName:decorated (n33)
```

add(n10,n1,1)
add(n19,n1,2)
add(n11,n10,1)
add(n13,n10,2)
add(n20,n19,1)
add(n23,n19,2)
add(n12,n11,1)
add(n14,n13,1)
mov(n5,n13,2)

add(n21,n20,1)
add(n22,n20,2)
add(n24,n23,1)
add(n33,n3,2)
add(n25,n24,1)
add(n26,n25,1)
add(n27,n25,2)
add(n28,n25,3)
add(n29,n25,4)

---

addVarDeclaration(n10,n1,1) PARTIAL
addIfStatement(n19,n1,2)
updateExpressionStatement(n2) with
  addSimpleName(n33,n3,2)
moveMethodInvocation(n5,n13,2)