1

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Атестаційна робота магістра

Дослідження моделей та засобів генерації діаграм об’єктів та класів для оцінювання якості розробки

Керівник: **проф. Єрохін А.Л.**

Виконав: студент групи ІПЗм-17-2, **Остапенко Денис Сергійович**

2019

Про машинне навчання

2

*"Кажуть, що комп'ютерна програма вчиться з досвіду **E** по відношенню до якогось класу задач **T** та міри продуктивності **P**, якщо її продуктивність у задачах з **T**, вимірювана за допомогою **P**, покращується з досвідом **E**»*

Том Мітчелл

Постановка задачі

1. Проаналізувати предметну область
2. Дослідити існуючі рішення
3. Надати власне рішення
4. Довести доцільність наданого рішення

Якісний програмний код

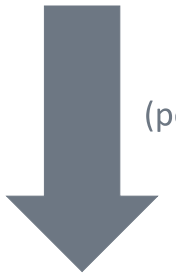
- **Низька складність -** малі функції, простий програмний потік
- **Надійність -** обробка виключних ситуацій
- **Стандартизованість –** відповідність деяким заданим стандартам

- **Ефективність -** використання шаблонів та практик написання коду
- **Швидкодія -** обробка виключних ситуацій
- **Оптимізація –** доцільне розпорядження програмними ресурсами

- **Читабельність –** доцільні назви для програмних компонентів
- **Документованість –** використання осмислених коментарів
- **Зв’язаність** – налаштування з’вязку між іншими компонентами

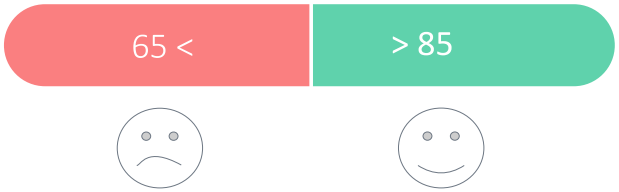
Математичні метрики оцінки якості коду

Цикломатична складність



(розширює)

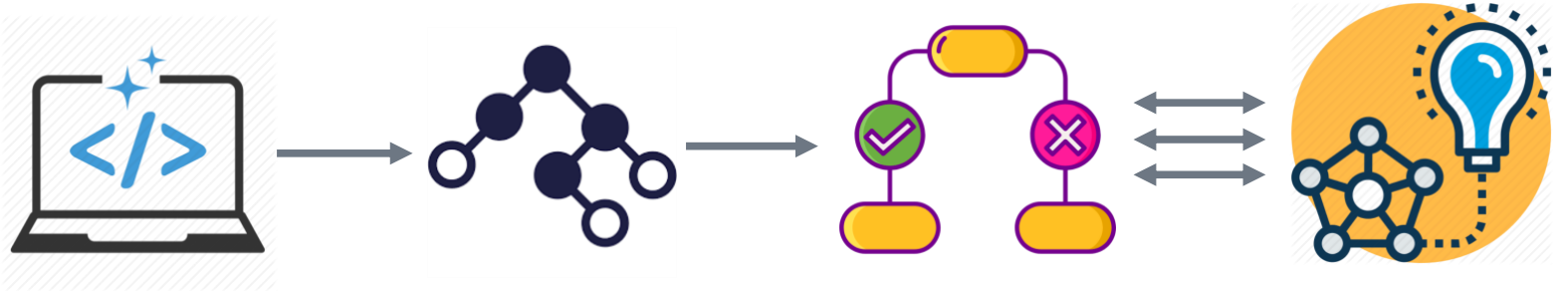
Індекс підтримки



Існуючі рішення

DEEPCODE

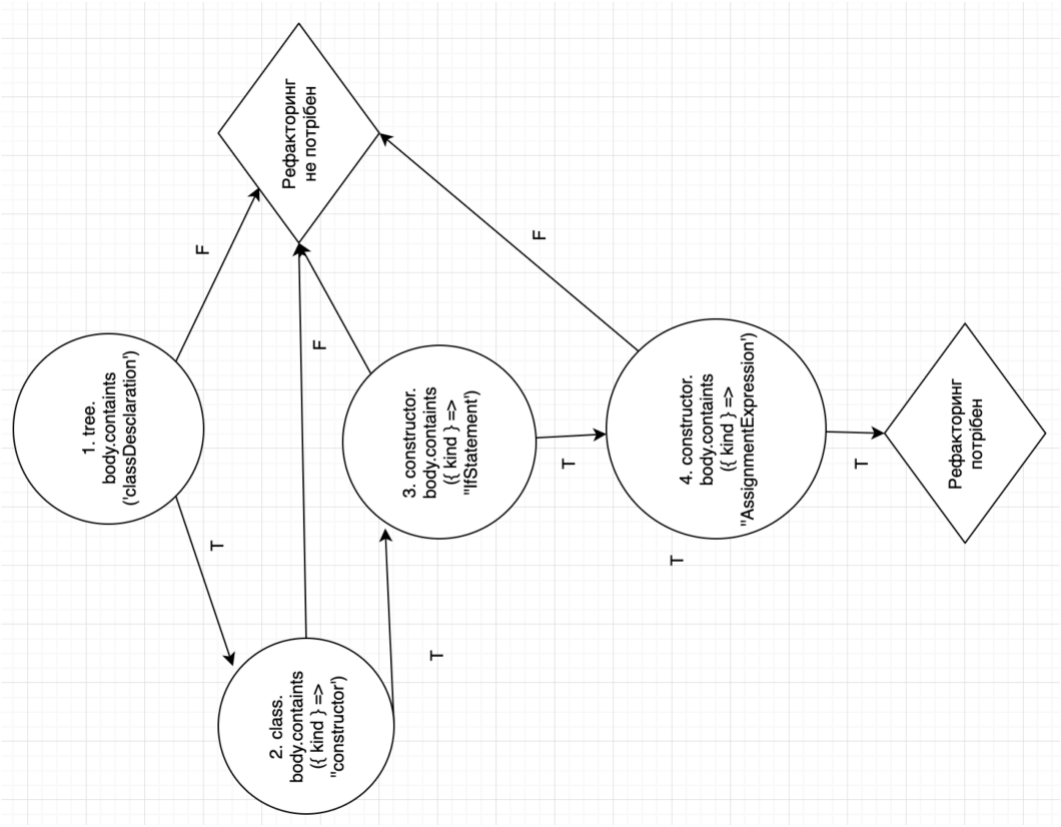
Система машинного навчання оцінювання якості розробки



Рефакторинг – шаблон «Будівельник»

```
1   class DummyClass {
2     constructor(prop1, prop2) {
3       if (prop1.length > 10) {
4         this.prop1 = prop1;
5       }
6
7       if (prop2 === true) {
8         this.prop2 = true;
9       }
10    }
11  }
12
13  const object = new DummyClass("test", true);
```

9


1. tree.
body.containts
('classDesclaration')
2. class.
body.containts
({ kind } =>
"constructor')
3. constructor.
body.containts
({ kind } =>
"IfStatement')
4. constructor.
body.containts
({ kind } =>
"AssignmentExpression')
Рефакторинг
не потрібен
Рефакторинг
потрібен
T
F
T
F
T
F
T
T

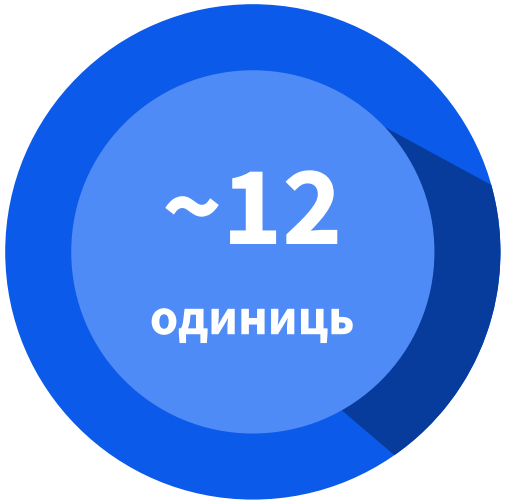
Рефакторинг – шаблон «Будівельник»


```
class RefactoredDummyClass {
  constructor() {
    this.prop1 = null;
    this.prop2 = false;
  }
}

class DummyClassBuilder {
  constructor() {
    this.instance = new RefactoredDummyClass();
  }

  buildProp1(prop1) {
    if (prop1.length > 10) {
      this.instance = prop1;
    }
    return this;
  }

  buildProp2(prop2) {
    this.prop2 = prop2;
    return this;
  }

  done() {
    return this.instance;
  }
}

const object = new DummyClassBuilder()
  .buildProp1("test")
  .buildProp2(true)
  .done();
```

Рефакторинг - паттерн «Будівельник»

78.73

Початковий Індекс підтримки

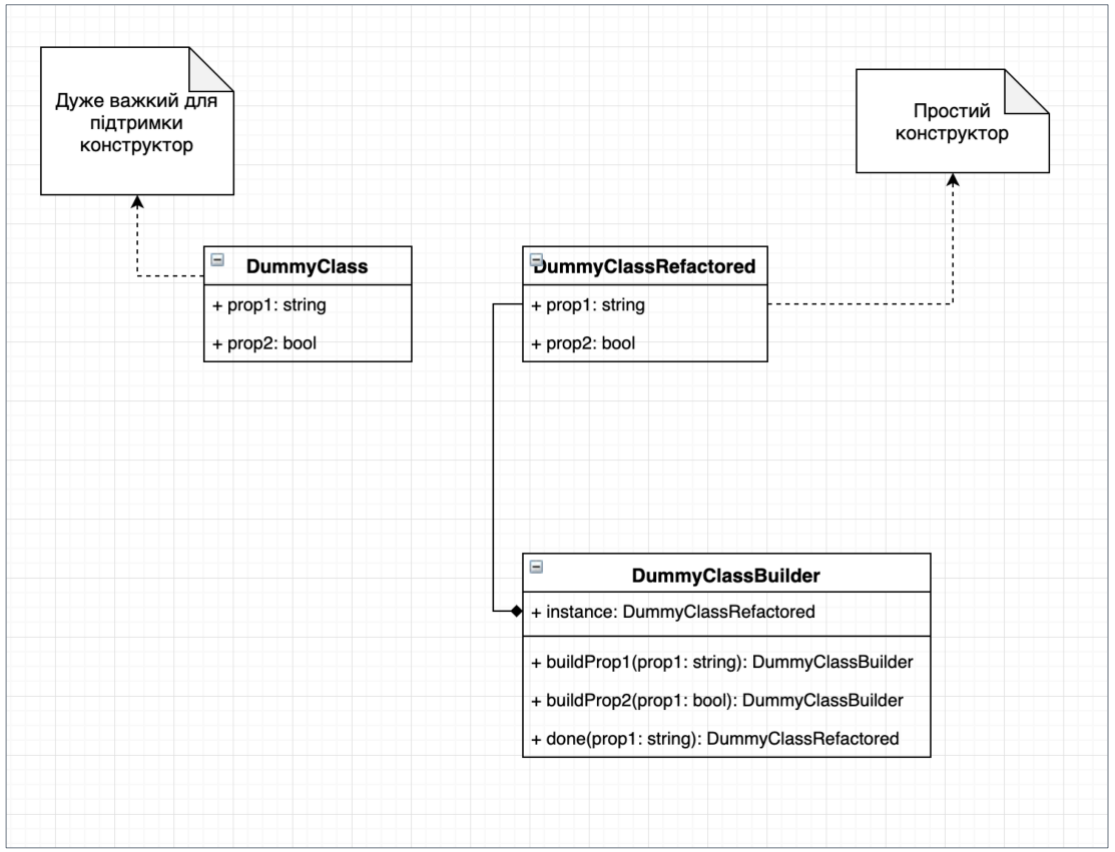


Зміна індексу

90.06

Індекс підтримки Після реорганізації

Рефакторинг – шаблон «Будівельник»



Рефакторинг – усунення дуплікацій у коді

```
1   const array1 = [];
2   const array2 = [];
3
4   let sum1 = 0;
5   let sum2 = 0;
6   let average1;
7   let average2;
8
9   for (let i = 0; i < 4; ++i)  {
10      sum1 += array1[i];
11  }
12
13  average1 = sum1 / 4;
14
15  for (let i = 0; i < 4; ++i)  {
16      sum2 += array2[i];
17  }
18
19  average2 = sum2 / 4;
20
```

Значення індексу підтримки – **58**

```
1   const calcAverage = array => {
2     let sum = 0;
3
4     for (let i = 0; i < 4; ++i) sum += array[i];
5
6     return sum / 4;
7   };
8
9   const array1 = [];
10  const array2 = [];
11
12  const average1 = calcAverage(array1);
13  const average2 = calcAverage(array2);
14
```

Значення індексу підтримки – **73**

15 одиниць

Висновки

1. Оцінка якості програмного коду може бути перенесена у математично площину 
2. Проблема автоматичного рефакторингу та оцінки якості програмного коду не є вирішеною на 100%
3. Метод дерев рішення може бути успішно використаний для оцінки якості програмного коду та автоматичного рефакторингу

Додаток Б Лістинг коду дерева прийняття рішень

```
var dt = (function () {

    /**
     * Creates an instance of DecisionTree
     *
     * @constructor
     * @param builder – contains training set and
     *                  some configuration parameters
     */
    function DecisionTree(builder) {
        this.root = buildDecisionTree({
            trainingSet: builder.trainingSet,
            ignoredAttributes:
arrayToHashSet(builder.ignoredAttributes),
            categoryAttr: builder.categoryAttr || 'category',
            minItemsCount: builder.minItemsCount || 1,
            entropyThrehold: builder.entropyThrehold || 0.01,
            maxTreeDepth: builder.maxTreeDepth || 70
        });
    }

    DecisionTree.prototype.predict = function (item) {
        return predict(this.root, item);
    }
    /**
     * Creates an instance of RandomForest
     * with specific number of trees
     *
     * @constructor
     * @param builder – contains training set and some
     *                  configuration parameters for
     *                  building decision trees
     */
    function RandomForest(builder, treesNumber) {
        this.trees = buildRandomForest(builder, treesNumber);
    }

    RandomForest.prototype.predict = function (item) {
        return predictRandomForest(this.trees, item);
    }
```

```
    /**
     * Transforming array to object with such attributes
     * as elements of array (afterwards it can be used as
HashSet)
     */
    function arrayToHashSet(array) {
        var hashSet = {};
        if (array) {
            for(var i in array) {
                var attr = array[i];
                hashSet[attr] = true;
            }
        }
        return hashSet;
    }

    /**
     * Calculating how many objects have the same
     * values of specific attribute.
     *
     * @param items – array of objects
     *
     * @param attr  – variable with name of attribute,
     *                which embedded in each object
     */
    function countUniqueValues(items, attr) {
        var counter = {};

        // detecting different values of attribute
        for (var i = items.length – 1; i >= 0; i––) {
            // items[i][attr] – value of attribute
            counter[items[i][attr]] = 0;
        }

        // counting number of occurrences of each of values
        // of attribute
        for (var i = items.length – 1; i >= 0; i––) {
            counter[items[i][attr]] += 1;
        }

        return counter;
    }

    /**
```

```
     * Calculating entropy of array of objects
     * by specific attribute.
     *
     * @param items – array of objects
     *
     * @param attr  – variable with name of attribute,
     *                which embedded in each object
     */
    function entropy(items, attr) {
        // counting number of occurrences of each of values
        // of attribute
        var counter = countUniqueValues(items, attr);

        var entropy = 0;
        var p;
        for (var i in counter) {
            p = counter[i] / items.length;
            entropy += –p * Math.log(p);
        }

        return entropy;
    }

    /**
     * Splitting array of objects by value of specific
attribute,
     * using specific predicate and pivot.
     *
     * Items which matched by predicate will be copied to
     * the new array called 'match', and the rest of the items
     * will be copied to array with name 'notMatch'
     *
     * @param items – array of objects
     *
     * @param attr  – variable with name of attribute,
     *                which embedded in each object
     *
     * @param predicate – function(x, y)
     *                    which returns 'true' or 'false'
     *
     * @param pivot – used as the second argument when
     *                calling predicate function:
     *                e.g. predicate(item[attr], pivot)
     */
```

```
    function split(items, attr, predicate, pivot) {
        var match = [];
        var notMatch = [];

        var item,
            attrValue;

        for (var i = items.length - 1; i >= 0; i--) {
            item = items[i];
            attrValue = item[attr];

            if (predicate(attrValue, pivot)) {
                match.push(item);
            } else {
                notMatch.push(item);
            }
        };

        return {
            match: match,
            notMatch: notMatch
        };
    }

    /**
     * Finding value of specific attribute which is most
frequent
     * in given array of objects.
     *
     * @param items - array of objects
     *
     * @param attr  - variable with name of attribute,
     *                which embedded in each object
     */
    function mostFrequentValue(items, attr) {
        // counting number of occurrences of each of values
        // of attribute
        var counter = countUniqueValues(items, attr);

        var mostFrequentCount = 0;
        var mostFrequentValue;

        for (var value in counter) {
            if (counter[value] > mostFrequentCount) {
```

```
                mostFrequentCount = counter[value];
                mostFrequentValue = value;
            }
        };

        return mostFrequentValue;
    }

    var predicates = {
        '==': function (a, b) { return a == b },
        '>=': function (a, b) { return a >= b }
    };

    /**
     * Function for building decision tree
     */
    function buildDecisionTree(builder) {

        var trainingSet = builder.trainingSet;
        var minItemsCount = builder.minItemsCount;
        var categoryAttr = builder.categoryAttr;
        var entropyThrehold = builder.entropyThrehold;
        var maxTreeDepth = builder.maxTreeDepth;
        var ignoredAttributes = builder.ignoredAttributes;

        if ((maxTreeDepth == 0) || (trainingSet.length <=
minItemsCount)) {
            // restriction by maximal depth of tree
            // or size of training set is to small
            // so we have to terminate process of building tree
            return {
                category: mostFrequentValue(trainingSet,
categoryAttr)
            };
        }

        var initialEntropy = entropy(trainingSet, categoryAttr);

        if (initialEntropy <= entropyThrehold) {
            // entropy of training set too small
            // (it means that training set is almost
homogeneous),
            // so we have to terminate process of building tree
            return {
```

```
                category: mostFrequentValue(trainingSet,
categoryAttr)
            };
        }

        // used as hash-set for avoiding the checking of split
by rules
        // with the same 'attribute-predicate-pivot' more than
once
        var alreadyChecked = {};

        // this variable expected to contain rule, which splits
training set
        // into subsets with smaller values of entropy (produces
informational gain)
        var bestSplit = {gain: 0};

        for (var i = trainingSet.length - 1; i >= 0; i--) {
            var item = trainingSet[i];

            // iterating over all attributes of item
            for (var attr in item) {
                if ((attr == categoryAttr) ||
ignoredAttributes[attr]) {
                    continue;
                }

                // let the value of current attribute be the
pivot
                var pivot = item[attr];

                // pick the predicate
                // depending on the type of the attribute value
                var predicateName;
                if (typeof pivot == 'number') {
                    predicateName = '>=';
                } else {
                    // there is no sense to compare non-numeric
attributes
                    // so we will check only equality of such
attributes
                    predicateName = '==';
                }
```

```
                var attrPredPivot = attr + predicateName +
pivot;
                if (alreadyChecked[attrPredPivot]) {
                    // skip such pairs of 'attribute-predicate-
pivot',
                    // which been already checked
                    continue;
                }
                alreadyChecked[attrPredPivot] = true;

                var predicate = predicates[predicateName];

                // splitting training set by given 'attribute-
predicate-value'
                var currSplit = split(trainingSet, attr,
predicate, pivot);

                // calculating entropy of subsets
                var matchEntropy = entropy(currSplit.match,
categoryAttr);
                var notMatchEntropy =
entropy(currSplit.notMatch, categoryAttr);

                // calculating informational gain
                var newEntropy = 0;
                newEntropy += matchEntropy *
currSplit.match.length;
                newEntropy += notMatchEntropy *
currSplit.notMatch.length;
                newEntropy /= trainingSet.length;
                var currGain = initialEntropy - newEntropy;

                if (currGain > bestSplit.gain) {
                    // remember pairs 'attribute-predicate-
value'
                    // which provides informational gain
                    bestSplit = currSplit;
                    bestSplit.predicateName = predicateName;
                    bestSplit.predicate = predicate;
                    bestSplit.attribute = attr;
                    bestSplit.pivot = pivot;
                    bestSplit.gain = currGain;
                }
            }
```

```
        }

        if (!bestSplit.gain) {
            // can't find optimal split
            return { category: mostFrequentValue(trainingSet,
categoryAttr) };
        }

        // building subtrees

        builder.maxTreeDepth = maxTreeDepth - 1;

        builder.trainingSet = bestSplit.match;
        var matchSubTree = buildDecisionTree(builder);

        builder.trainingSet = bestSplit.notMatch;
        var notMatchSubTree = buildDecisionTree(builder);

        return {
            attribute: bestSplit.attribute,
            predicate: bestSplit.predicate,
            predicateName: bestSplit.predicateName,
            pivot: bestSplit.pivot,
            match: matchSubTree,
            notMatch: notMatchSubTree,
            matchedCount: bestSplit.match.length,
            notMatchedCount: bestSplit.notMatch.length
        };
    }

    /**
     * Classifying item, using decision tree
     */
    function predict(tree, item) {
        var attr,
            value,
            predicate,
            pivot;

        // Traversing tree from the root to leaf
        while(true) {

            if (tree.category) {
                // only leafs contains predicted category
```

```
            return tree.category;
        }

        attr = tree.attribute;
        value = item[attr];

        predicate = tree.predicate;
        pivot = tree.pivot;

        // move to one of subtrees
        if (predicate(value, pivot)) {
            tree = tree.match;
        } else {
            tree = tree.notMatch;
        }
    }
}

/**
 * Building array of decision trees
 */
function buildRandomForest(builder, treesNumber) {
    var items = builder.trainingSet;

    // creating training sets for each tree
    var trainingSets = [];
    for (var t = 0; t < treesNumber; t++) {
        trainingSets[t] = [];
    }
    for (var i = items.length - 1; i >= 0 ; i--) {
      // assigning items to training sets of each tree
      // using 'round-robin' strategy
      var correspondingTree = i % treesNumber;
      trainingSets[correspondingTree].push(items[i]);
    }

    // building decision trees
    var forest = [];
    for (var t = 0; t < treesNumber; t++) {
        builder.trainingSet = trainingSets[t];

        var tree = new DecisionTree(builder);
        forest.push(tree);
    }
```

```
        return forest;
    }

    /**
     * Each of decision tree classifying item
     * ('voting' that item corresponds to some class).
     *
     * This function returns hash, which contains
     * all classifying results, and number of votes
     * which were given for each of classifying results
     */
    function predictRandomForest(forest, item) {
        var result = {};
        for (var i in forest) {
            var tree = forest[i];
            var prediction = tree.predict(item);
            result[prediction] = result[prediction] ?
result[prediction] + 1 : 1;
        }
        return result;
    }

    var exports = {};
    exports.DecisionTree = DecisionTree;
    exports.RandomForest = RandomForest;
    return exports;
})();
```

Додаток В Наукові публікації

В.1 Стаття у міжнародному науковій конференції «Вестник.Наука и Практика»

MONOGRAFIA POKONFERENCYJNA

СИСТЕМА МАШИННОГО НАВЧАННЯ ОЦІНЮВАННЯ ЯКОСТІ КОДУ

Остапенко Д. С.
Бакалавр
Харківський Національний Університет Радіоелектроніки
Кафедра Програмної Інженерії

Ключові слова:
Машинне навчання, якість, програмний код, абстрактні синтаксичні дерева.
Keywords:
Machine learning, quality, software code, abstract syntax trees.

американський вчений Том Мітчелл запровадив широко цитоване формальніше визначення алгоритмів, що досліджують у галузі машинного навчання: «Кажуть, що комп'ютерна програма вчиться з досвіду *E* по відношенню до якогось класу задач *T* та міри продуктивності *P*, якщо її продуктивність у задачах з *T*, вимірювана за допомогою *P*, покращується з досвідом *E*.» [1], виходячи з визначення Тома, можна сказати, що будь який вид діяльності, що можна перекласти у площину «Математичного досвіду» буде мати потенційне місце для використання алгоритмів, методів та практик машинного навчання для автоматизації процесів або/та отримання кращих результатів.

Якість розробки програмного забезпечення залежить від багатьох аспектів, але через деякий час існування та розвитку програмного продукту, одним з найголовнішим критеріїв якості програмного забезпечення є його програмний код написаний командою розробників. Якість коду – це математична та обчислювальна величина, так як є багато правил, технік, шаблонів, тощо, які описують стандарти написання гарного програмного коду, обчислення складності, виявлення слабких місць, то проблему написання гарного програмного коду можна виділити у окремий клас завдань *T*, кожна з під задач несе у собі деяку міру продуктивності P_i (у контексті якості програмного коду – це будуть загальні правила, ознаки та метрики чистого легкого у підтримуваності коду), виходячи з цього, різні комбінації вирішення задач будуть давати різні величини досвіду *E*.

У якості джерела навчання використовуються заздалегідь підготовлені абстрактні синтаксичні дерева, які описують ті чи інші шаблони або практики написання програмного коду, а на далі використовувати вже натреновані моделі при аналізуванні кодової бази кінцевих користувачей. Досить вагомим плюсом систем машинного навчання є гнучкість, тому користувачі можуть з легкістю запровадити свої практики та шаблони стандартизованого коду.

Основним методом виконання ана-

Рисунок В.1 – Сторінка 1

лізу у системі машинного навчання використовується підхід «дерев рішень». Гнучкість та швидкість дерев рішення дозволяє створювати дуже складні ланцюжки аналізу з можливістю повторного використання та розширення. Листками цих дерев є предикати, або набір предикатів, які базуючись на вхідному абстрактному дереві в змозі визначити порушення ти чи інших шаблонів, правил чи стандартів написання стандартизованного підтримуваного програмного коду і створити набір необхідних кроків для рефакторингу.

Данна система машинного навчання може бути використана в наступних галузях:

Динамічні аналізатори коду. Ця система може бути представлена як CLI утиліта, яка скануватиме проект та виводитиме знайдені помилки у розгорнутому форматі на веб сторінці;

Інтегрування у розподілені системи контролю версій, такі як Atlassian Bitbucket, GitHub, GitLab, тощо. Цю систему можна використовувати під час огляду коду командою, що може значно спростити та скоротити сам процес, а також суттєво зменшити кількість помилок, допущенних людиною;

Як вже було зазначено, ця система досить гнучка і може «підлаштовуватись» під стандарти, практики, шаблони, тощо, задані конкретною командою розробників для конкретного продукту, тому це відкриває ще більше дверей до написання чистого та стандартизованого коду.

Також слід зазначити, що використання абстрактних синтаксичних дерев та робота з об'єктно-орієнтованим середовищем відкриває деякі можливості для покращення загального кінцевого досвіду користування, наприклад, генерування діаграм класів та об'єктів, на яких будуть зазначені зв'язки між компонентами та детально описані порушення деяких правил або шаблонів.

Ще одним досить цікавим варіантом покращення цієї системи є «Прогнозування», аналізуючи кодові бази багатьох проектів та обчислюючи їх загальний рівень якості, можна навчити цю систему заздалегідь прогнозувати деякі потенційні зміні у структурі програмних компонентів та своєчасно попереджати власників кодових баз, щодо тих змін, надаючи корисні поради на основі вже проаналізованих проектів, тим самим значно мінімізувати ризик під час розробки та підвищити загальну безпеку та надійність усього програмного продукту і його компонентів.

Роблячи висновок, слід зазначити, що написання якісного програмного коду є математичною задачею, тому може бути перенесено у математичну площину, виходячи з цього, є можливість покращення та спрощення процесу написання за допомогою практик та методів, які пропонує галузь машинного навчання.

Література

1. Nils J. Nilsson, Introduction to Machine Learning, Stanford University Stanford, 2005, 180 c.

Рисунок В.1 – Сторінка 2

В.2 Доповідь на конференцію "Verification and AI"

CODE QUALITY ANALYSIS SYSTEM BASED ON GENERAL UML DIAGRAMS WITH SUPPORT OF MACHINE LEARNING

Ostapenko D.S., master of PE department, e-mail: denys.ostapenko@nure.ua
Academic adviser: Dean of the Faculty of Computer Science, D. Sc., Professor Andriy L. Yerokhin
Kharkiv National University of Radio Electronics

The suggested system provides an ability to evaluate the quality of given codebase drawing on well-known rules and principles like S.O.L.I.D, Clean Architecture, etc. A performed analysis is visualized in generated main UML diagrams which indicate clearly whether any rule was violated, thereby the amount of badly written code should be reduced dramatically.

Application development involves creating a computer program, or set of programs to perform tasks, from keeping track of inventory and billing customers to maintain bank accounts, speeding up business process and, in fact, even improving application effectiveness. Unlike vanilla programming, application development involves higher levels of responsibility (particularly for requirement capturing and testing) [1]. But the main parts of each software development phase process are design and coding.

The design represents the process of discussing the software architecture and documenting of it. During a design phase, a team headed by an architect clarifies software requirements, chooses appropriate technology and builds the basic architecture using different diagrams, flowcharts, and other auxiliary resources.

The coding phase follows the design one and usually continues much longer time. Basically, the coding is the process of writing a program code by developers in order implement some features or prepare the ground for implementing other functionality. Here are the main criteria which affect the code quality:

1. Time. As the time flies, each day new technologies, frameworks, development practices, and approaches are coming so it becomes quite important to maintain existing applications;

Рисунок В2 – Сторінка 1

2. Team members experience and skills. The more skillful a team is the better code quality is fair to expect. Overall team qualification is the crucial attribute;
3. Established and adjusted the coding process. Usually, this derives from the item # 2. Customized coding practices, regular detailed code reviews, and knowledge sharing sessions help to improve an overall codebase quality.

In fact, the reality is far from ideal and as a result developers don't have much time for adapting existing code base to some modern technologies, development teams are not staffed properly (lack of seniors) and inside them there is no precise development process which would cover the code quality issue. Mostly, software developers can not do pretty much in this situation and that is why the code quality remains the same or even is worsening over the time.

Basing on the stated above, it's possible to emphasize the main motivation of the system - providing an ability to control and evaluate the code quality of given codebase so that aforementioned risks related to code quality could become negligible. Here is the detailed explanation how this system is solving the issues

1. Time. Let's say there is a version A of some language and assume that version B has drastic improvements related to Open-Closed principle, so it does make sense to create rules which would simplify the migration process and it would be possible to fed them into Machine Learning software so that respective diagrams will be generated and team can start to plan their further refactoring process accordingly;
2. Probably, this software will not turn junior developers into seniors instantly, but it definitely helps with seeking and identifying potential problems. As a bonus it can help junior developers to learn and understand new practices a bit faster;
3. Established and adjusted the coding process. Since the better part of code quality issues could be covered programmatically a senior developers are able to spend less time reviewing a regular code changes and focus on challenging tasks or global improvements. Also, Machine Learning

Рисунок В2 – Сторінка 2

system could be expanded with some custom rules and standards, this should help teams to stay on the same page and experiment with new coding ways.

Hopefully, Machine learning technologies had a terrific development boost in past few years and it become possible to make sophisticated analysis and predictions.

The algorithm how this system evaluates the code quality and builds diagram is illustrated in figure 1.

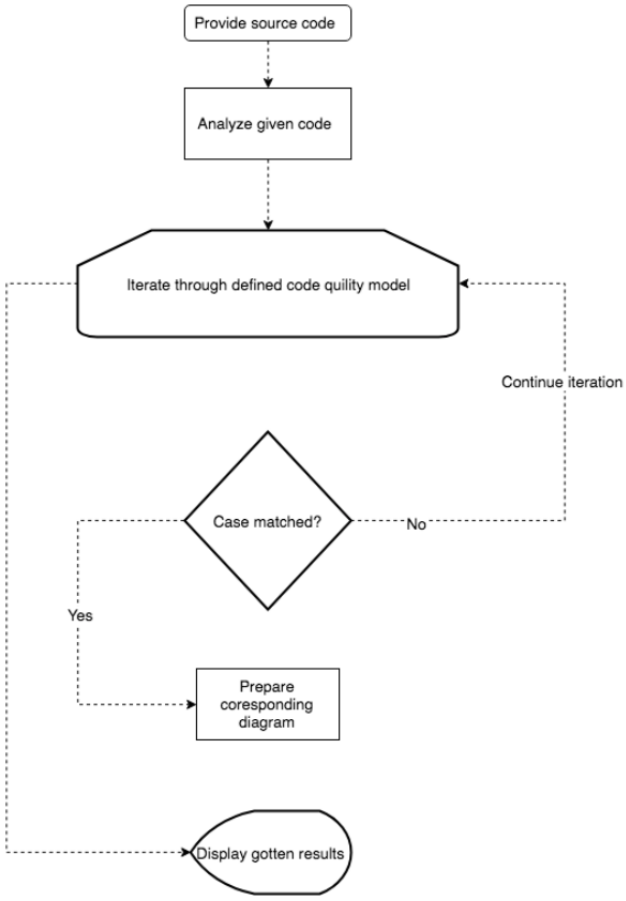


Figure 1 - Code quality evaluation simplified algorithm

Рисунок В2 – Сторінка 3

Let’s look step by step at provided flowchart in more detail.

1. Ideally, analysis shouldn’t be applied to the whole project because that can be prone to delays and low efficiency, so taking that fact into account, the preferable way of usage of this system is handling of moderate changes and that is making this system a very good candidate for embedding into web-based hosting services for version control Git like Github or Bitbucket. The command language interface tool will be supported as well.
2. Once a user provided the code the analysis phase starts. Basically, analysis consists of parsing given codebase in order to get abstract syntax trees and then the matching the received trees with already defined models which represent either good practices or bad practices (antipatterns). All good or bad patterns and practices are represented as sets of abstract syntax trees and relations between them. In order to find the violation of some architectural principle or usage of an antipattern the system will be using decisions tree technique[2]. The system contains metadata for each case so that it will choose an appropriate diagram to show found code issue. Supported diagrams are: Class diagram, objects diagram and calls graph[3].
3. After the analysis is completed and data is gathered the system will output generated diagrams, mark there all founded issues so that developers will be able to investigate generated charts and clearly understand the code issues.

Also, there is a pivotal feature which will be available in future releases, this is an auto fixing. Once the system has found some problem in given codebase it can try to refactor the code according to ‘good models’ so that identified problem will be fixed. That feature should drastically improve the overall code quality and significantly reduce the number of delays in software development.

Since any Machine Learning based system is not able to give precise answers out of the box[4] and must be trained on some real-world data this system can complement already existing collections of defined models with

Рисунок В2 – Сторінка 4

slightly altered versions of users models, as a result, the system will catch more issues and diagrams will show more detailed and articulated messages and at the end of the day exactly this capability will make possible to provide auto fixing.

Making a conclusion it's worth to note that currently there are no analogs on the market which could provide close functionality so there is a leaves a huge potential for this project.

Sources:

1. Kit, Edward (1992). Software Testing in The Real World. Addison-Wesley Professional. ISBN 0201877562.
2. Kamiński, B.; Jakubczyk, M.; Szufel, P. (2017). "A framework for sensitivity analysis of decision trees". Central European Journal of Operations Research
3. Ryder, B.G., "Constructing the Call Graph of a Program," Software Engineering, IEEE Transactions on, vol. SE-5, no.3pp. 216– 226, May 1979
4. Trevor Hastie, Robert Tibshirani and Jerome H. Friedman (2001). The Elements of Statistical Learning, Springer. ISBN 0-387-95284-5.

Рисунок В2 – Сторінка 5

Додаток Г Електронні матеріали (CD)