Master Thesis

October 17, 2018

Changelyzer

Learning Change Type Classifications for Software Evolution from Big Code on GitHub

Sali Zumberi

of Zürich, Switzerland (13-790-951)

supervised by

Prof. Dr. Harald C. Gall

Dr. Juergen Cito and Carol Alexandru

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Master Thesis

Changelyzer

Learning Change Type Classifications for Software Evolution from Big Code on GitHub

Sali Zumberi

University of Zurich^UZH

s.e.a.l. software evolution & architecture lab

Master Thesis

Author: Sali Zumberi, sali.zumberi@live.de

Project period: 17.04.2018 - 17.10.2018

Software Evolution & Architecture Lab
Department of Informatics, University of Zurich

Acknowledgements

I would like to thank Dr. Harald C. Gall and the software evolution and architecture lab of the University of Zurich for the possibility of making it possible to delve into a versatile topic, leading to interesting insights. I would also like to thank my supervisor Jürgen Cito and Carol Alexandru very much for all inputs, hints and help at any time during the writing of the thesis.

Abstract

Software needs to be adapted to its rapidly changing environment. “A key issue in software evolution analysis is the identification of particular changes that occur across several versions of a program. “ [11]. In order to understand and analyse source code changes, it is crucial to make them tangible. While plain-text diffs are a straight-forward way of keeping track of changes in a software project, they are poorly suited for understanding those changes. Different semantic changes might be mixed together in a single diff and it is difficult to further process diffs using automated tools. Approaches like ChangeDistiller extract changes between two revisions based on abstract syntax trees (ASTs) instead of plain text source code. This allows them to recognize semantic changes. More specifically, whether certain elements (if conditions, classes, methods, etc.) have been added, removed, modified or even moved to other locations in the source code. However, change types identified by ChangeDistiller have been manually crafted by researchers. This thesis tries to extract common change types by applying big data analytics including clustering, word embeddings and topic modelling techniques on changes of over 500 projects (18.6 GB). We were able to find more than 70 common change types, use neural network to show similar changes, cluster similar changes into 55 clusters, then extract 35 topics with help of topic modelling and last but not least prove existence of change-groups within larger diffs by implementing a sophisticated algorithm. Finally, we propose tools and tasks based on provided data corpus.

Zusammenfassung

Software muss kontinuierlich an ihre sich schnell verändernde Umgebung angepasst werden. "Ein zentrales Thema in der Softwareevolutionsanalyse ist die Identifizierung bestimmter Änderungen, die über mehrere Versionen eines Programms hinweg auftreten." [11]. Um Software Veränderungen zu verstehen und zu analysieren, ist es entscheidend, sie greifbar zu machen. Während Plain-Text-Differenzen eine einfache weitverbreitete Möglichkeit sind, Änderungen in einem Softwareprojekt zu verfolgen, sind sie schlecht geeignet, diese Änderungen zu verstehen. Verschiedene semantische Änderungen können in einem einzigen Diff zusammengefasst werden, und es ist schwierig, Diffs mit automatisierten Werkzeugen weiterzuverarbeiten. Ansätze wie ChangeDistiller extrahieren Änderungen zwischen zwei Revisionen, die auf abstrakten Syntaxbäumen (ASTs) anstelle von Klartext-Quellcode basieren. So können sie semantische Veränderungen erkennen. Genauer gesagt, ob bestimmte Elemente (wenn Bedingungen, Klassen, Methoden usw.) hinzugefügt, entfernt, modifiziert oder sogar an andere Stellen im Quellcode verschoben wurden. Die vom ChangeDistiller identifizierten Änderungsarten wurden jedoch von den Forschern manuell erstellt. Diese Masterarbeit versucht, gängige Change-Typen mithilfe von Big Data Analyse zu extrahieren, unter anderem werden Clustering, Worteinbettungen und Themenmodellierungstechniken bei Änderungen von über 500 Projekten (18,6 GB) einsetzt. Wir konnten mehr als 70 gängige Change-Typen finden, neuronale Netze verwenden, um ähnliche Änderungen darzustellen, ähnliche Änderungen in 55 Clustern zusammenfassen, dann 35 Themen mit Hilfe von Themenmodellierung extrahieren und nicht zuletzt die Existenz von Change-Gruppen innerhalb größerer Diffs durch die Implementierung eines ausgeklügelten Algorithmus nachweisen. Schließlich schlagen wir Werkzeuge und Aufgaben vor, die auf dem bereitgestellten Datenkorpus basieren.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reasoning about changes in source code is a fundamental task for software engineers to enable understanding of the evolution of software [11]. Contemporary version control systems (e.g., Git) support line-based, plain-text differencing of source code to support this reasoning. While plain-text diffs are a straight-forward way of keeping track of changes in a software project, they are poorly suited for understanding those changes in a more structured manner. Different semantic changes might be mixed together in a single diff and it is difficult to further process diffs using automated tools.

Approaches like ChangeDistiller [11] extract changes between two revisions based on abstract syntax trees (ASTs) instead of plain text source code. This allows them to recognize syntactical changes. For instance, whether specific elements (if conditions, classes, methods, etc.) have been added, removed, modified or even moved to other locations in the source code. ChangeDistiller maintains a collection of over 40 manually classified changes.

This thesis can be regarded as fundamental approach of applying machine learning on fine grained source code changes, which are based on abstract syntax tree operations. Different from ChangeDistiller, our changes are not manually classified, instead we use the underlying raw JDT Elements. We present a data analytic approach where we make use of different machine learning techniques in order to find and extract source code change groups and patterns, similar changes by making use of neural networks and descriptive statistic to provide insights of source code changes on JAVA programming language.

Our evaluation is based on statistical measurements as well as on surveys based on the "human-in-the-loop" approach. To facilitate the evaluation we developed a webtool which allows users to explore and search over 500 000 source code changes. Finally, we present a list of tools and tasks which can be build on top of the provided data.

1.1 Motivation

Motivation of this master thesis is based on two existing problems.

- The change types identified by tools such as ChangeDistiller have been manually crafted by researchers and can seem arbitrary and biased. The provided taxonomy of changes includes 40 change types.

- Tools like this need to be implemented separately for each specific programming language.

Solving these tools will enable the possibility to create several tools, which could support software developers in the maintenance and evolution of source code. Empirical change groups and patterns could be integrated in repository search engines and support the lookup of changes (e.g. show latest re-factored for loops etc.), furthermore open source code maintainers could reduce effort by subscribing to fine grained changes or change patterns of a specific file (e.g. core functionality of a module, dependency change of a specific package.

1.2 Goal

The goal of this thesis is to develop a framework and tool chain that, given a large corpus of source code (transformed to an AST) in a language can cluster the most common change types and learn a model to classify these changes in an unsupervised manner.

1.3 Research Questions

Based on the goals of this thesis, there are three main research questions to be discussed:
RQ1: Is it possible to find change types from Big data analytic?
RQ2: Is it possible to learn similar source code changes with the help of unsupervised machine learning?
RQ2: Do change topics and groups exist in source code change?

1.4 Contribution

1. Over 890 empirical change types
2. Over 70 common change types have been extracted
3. 55 Cluster of similar changes
4. Changelyzer WebTool - Prototype to discover changes and higher order edit scripts
5. 33 Change Topics
6. Data corpus of 500k software changes (each has 5 files: File Before, File After, Plain Diff Result, GumTree Cluster, GumTree Diff)
7. 8 proposals for potential software maintenance and evolution tools

1.5 Outline

Chapter 2 Background provides the basic background of Abstract Tree Differencing, insights of Machine Learning and Text Classification

Chapter 3 Related Work gives a short and comprehensive view over previous research on this field

Chapter 4 Approach describes the methodology selection and data analytic

Chapter 5 Results show the main results of this thesis. Firstly it shows the 10 higher order edit scripts, and secondly, a list of new change types followed by ....

Chapter 6 Evaluation explains the procedures on how we evaluated the used models and threats to Validity of this thesis

Chapter 7 Future work presents a list of tools and tasks which can be build on top of this thesis

Chapter 2

Background

2.1 Abstract Syntax Trees

In a programming language, an Abstract Syntax Tree (AST) represents a piece of source code as a tree. In the source code, a construct is signified as a node of the tree. However, the syntax is abstract and does not demonstrate every aspect of the actual syntax, for example, the tree structure is indicative of grouping parentheses, and a node with two branches may signal a syntactic construct like an if-condition expression. This is how ASTs are different from parse trees. The parse trees are constructed through a parser during the translation and compilation process of source code. After its construction, the AST is filled up with further information through additional processing, for instance, contextual analysis.

The applications of an AST are immense, and also include program transformation systems and program analysis. In addition, ASTs are extensively utilized in compilers to represent the source code and its structure. The AST usually operates as an intermediary representation of the code through a variety of phases that the compiler entails, which influences the output of the compiler in a significant manner.

The AST assists the compiler in a number of ways. An AST may be customized and improved by providing it with information like annotations and properties for each of its component. This annotation and editing are not possible with the program code since an overall alteration would be required. In comparison to the source code, an AST is freed of any unnecessary delimiters and punctuation like parentheses, semicolons, braces, etc. In addition, an AST mainly consists of additional information about a particular program, for instance, the compiler may be able to print intelligible error messages because the position of each component in the program code would be stored by an AST.

2.1.1 Abstract Syntax Tree Differencing

The AST differencing is grounded on the concept of AST edit actions. In his research, [**?**] elucidate that the aim of AST differencing is to compute a sequence of edit actions that transmute a particular AST into another. This sequence is commonly labelled as an edit script. The edit actions of an AST are ***updateValue(t, v_n)*** , ***add(t, t_p, i, l, v)*** , ***delete(t)*** , and ***move(t, t_p, i)***. As is evident, the possibility of a number of edit scripts that performs the same transformation is immense. However, the quality of an edit script is heavily dependent on its length, that is, the longer the transformation, the worse the quality. If the move action is taken into account, the determination of the shortest transformation is NP-hard. If the actions pertaining to add node, update node, and delete node are taken into consideration, the problem of tree differencing becomes an area of important consideration [3]. In this case, [29] have discussed that the fastest algorithm runs in

O(n^3), however, it may result in higher computation times, especially for the source code files that are large in size. In addition, these algorithms are also plagued by their incapacity to discover moved nodes, which is a recurrent action in the files containing the source code. As a result, it becomes very difficult to comprehend these colossal edit scripts.

2.1.2 Change Distiller

According to Fluri et al. (2007), ChangeDistiller is one of the most renowned algorithms that work on the ASTs and was largely inspired by Chawathe's proposal. However, one of the main assumptions of ChangeDistiller is that the leaf nodes comprise a noteworthy volume of text. The ASTs are, therefore, simplified so that the code statements are carried by the leaves instead of the raw AST. Therefore, it is quite evident that the ChangeDistiller will not calculate fine-grained edit scripts on languages that boast a significant number of constituents in statements, for instance, JavaScript is an ideal example.

2.1.3 GumTree – Fine-grained and Accurate Source Code Differencing

The differencing algorithms of AST primarily work in two fundamental steps, that is, developing mappings and then identifying the edit scripts. In a GumTree algorithm, the mappings that are computed between two ASTs comprise of two consecutive stages: first, the isomorphic subtrees of decreasing height are identified through a greedy top-down algorithm; second, a bottom-up algorithm is introduced and performs containing mapping (that is, two nodes match) if their children (or descendants) consist of an array of common anchors [28]. In the first step, the nodes of the isomorphic subtrees are utilized to establish appropriate mappings. If the two nodes match, then an optimal algorithm is applied to explore recovery mappings (or additional mappings) among their descendants. The motivation behind this algorithm is the developers who manually review the changes that are made in the files. In this approach, the first search for those pieces of code that are biggest and unaltered. Next, those containers of code are identified that have the ability to be mapped together. In the final step, the differences are viewed with stark precision to identify the leftovers in every container. In summary, it may be established that the GumTree algorithm is based on three fundamental steps, which are, a greedy top-down algorithm, a bottom-up algorithm, and recovery mappings.

2.2 Machine Learning

Recently, the machine learning has become an integral component of the spectrum of information technology. Due to the emergence of Big Data, the notion of smart data analysis is expected to become more prevalent. Therefore, as the technological advancements make further progress, the machine learning will continue to occupy a significant position in the world of technology. Witten et al. [34] define machine learning as a field of information technology that makes use of statistical methods to provide the computer systems with an ability to learn with the data. In the last two decades, a number of algorithms have been introduced that assists the computers to learn. This has further allowed a number of commercial applications to establish their place in the digital market. If the problem of speech recognition is taken into consideration, it has been identified that the algorithms based specifically on machine learning have outperformed every other approach that has been adopted before that. The field of data mining has also benefitted from the machine learning algorithms by discovering valuable knowledge from excessively large databases comprising medical records, financial transactions, loan applications, equipment maintenance records, and the like. There is no doubt about the fact that as our comprehension of the machine learning algorithms continues to mature, it is quite imminent that the machine learning will have a crucial role to play in the computing technology.

2.3 Unsupervised Learning

The unsupervised learning refers to how the systems can learn to signify specific input patterns in a manner that echoes the statistical organization of these input patterns in a holistic manner [15]. In contrast to the supervised learning, this approach does not consist of environmental evaluations and explicit target outputs associated with every input. However, the significance of this approach is immense because this is exactly how the brain works. This allows the method of unsupervised learning to be used for synaptic adaptation as computational models. The Bayesian Networks are a good example of unsupervised learning that deduces how a created input relates to an underlying cause. Although the practice of conducting unsupervised learning may be ambiguous to a number of people because of its inability to acquire supervised target outputs, a formal framework can still be developed that can be effectively used for prediction of future inputs, decision making, and efficient communication of inputs to another machine, and the like. In similar words, this technique may be used to deduce patterns in a noise that is purely unstructured. The applications of unsupervised learning are tremendous, for instance, grouping the movies by ratings that are assigned by its viewers, characterizing the various groups of shoppers by their purchasing and browsing histories, and grouping the subgroups of cancer patients by the measurements of their gene expression.

2.3.1 K-Means

The K-means is a clustering technique that clusters the observations into disjoint clusters, usually of definite numbers [26]. To deduce which cluster needs to take in a particular observation, a number of distance measures are intelligibly utilized. The aim of this algorithm is to mitigate the distance between the given observation and the centroid of the cluster. This is accomplished by providing the clusters with appropriate observations in an iterative manner. When the lowest distance measure is accomplished, the algorithm is then terminated. Initially, the sample space is divided into K clusters. The observations are then randomly assigned to every cluster. Now, for every sample, the distance between the centroid of the cluster and the observation is computed. If the sample is calculated to lie closer to its existing cluster then it retains its position, otherwise, it is transferred to a different cluster. Both of these aforementioned steps are repeated until no further movement between the clusters is witnessed. Hence, an overall stability is attained. Usually, the Manhattan distance, Euclidean distance, and the Euclidean squared distance are used to compute the distances. However, in some applications, for instance, speech processing, the distance measure of Euclidean squared distance is mostly preferred.

The applications of K-means clustering are immense. It is extensively used to form clusters of the features that have been extracted from the speech signals. Therefore, the signals with similar spectral traits occupy similar positions in the codebook, limiting its size by an appreciable margin.

2.3.2 Topic Modelling (LDA

According to Jelodar et al. [17], the technique of topic modelling is one of the most significant and powerful approaches for deducing relationships between data or documents, discovering latent data, and data mining. In this regard, a large number of researchers have elaborated on the concept of topic modelling in various fields including linguistic science, medical, and science. Although a large number of methods exist for conducting topic modelling, the LDA (Latent Dirichlet allocation) is perceived as the most preferred one.

The LDA makes use of Dirichlet priors for the word-topic and document-topic distributions, allowing for better generalizability. In LDA, every document is perceived as a combination of numerous topics where every document is considered to contain an array of topics that are assigned to these documents through LDA. The practice of LDA is similar to that of the pLSA (probabilistic latent semantic analysis), however, LDA supposedly consists of a sparse Dirichlet prior, as already discussed. This allows for the encoding of intuition. The topics that are small and those that only utilize a limited set of words are covered. Hence, it may be established that an LDA is a generalized form of a pLSA model.

2.3.3 Embedding (word2vec)

In machine learning, word2vec learning is a collection of models that are utilized to construct word embedding. The purpose of these two-layered neural network-based models is to rebuild the linguistic contexts of words through training. A large chunk of structured text is given as an input to word2vec, which produces a vector space having substantial dimensions. In this vector space, every unique word in this chunk of structured text is assigned to a particular vector in the vector space. The position of word vectors is decided based on the proposition that they share common contexts with the words located in close proximity. To develop a distributed representation of the words, the word2vec can make use of two model architectures, that is, continuous skip-program or CBOW (continuous bag-of-words). Due to the fact that the training of word2vec has the tendency to be sensitive to parameterization, the parameters: training algorithm, subsampling, dimensionality, and context window are significant. The approach pertaining to word embedding has the ability to deduce varying extents of similarities between the words. In addition, Mikolov, Yih, and Zweig (2013) further elaborated that the syntactic and semantic patterns can be mimicked through vector arithmetic [24].

2.4 Text Classification

In computer science, text classification is an integral part of text mining which is defined by Korde, and Mahender (2012) as, “manually building automatic TC systems by means of knowledge-engineering techniques, that is, manually defining a set of logical rules that convert expert knowledge on how to classify documents under the given set of categories.” Taking the example of an array of stories, for example, consisting of topics like business, politics, or sports. The fundamental problem is to formulate a classification model that assigns an accurate class to every new document. However, it needs to be noted that the text classification supports two labels, which are, single label and multi-label. The main difference between these labels being their ability to support single or multiple classes. The process of text classification is composed of six main steps. In the first step, that is, document collection, the different types of documents are collected, for instance, documents having formats like .doc, .pdf, .html, and the like. The second step deals with preprocessing in which the documents are prepared for the subsequent steps using techniques such as tokenization and stemming word. In the third step, that is, indexing, usually the vector space model represents the documents as word vectors through a word document matrix. The purpose is to represent every entry as the occurrence of a specific word in a particular document, which is done through the determination of weight. This may be done through entropy, tf-IDF, frequency weighting, and the like. The fourth step deals with feature selection in which a vector space is constructed to enhance the accuracy, efficiency, and scalability of the text classifier. This step is closely followed by the fifth step of classification where the documents are then automatically classified either through supervised, unsupervised, or semi-supervised methods. Lastly, the final step of text classification evaluates the text classifiers through comprehensive experimentation.

2.5 Similaritiy Measures

In the field of text classification, the measurement of similarity between various documents is considered to be an imperative function. However, because the dimensionality of the majority of the documents is quite large, resulting in a sparse vector, a large part of these feature values in the vector are zero [23]. Therefore, the presence of this high dimensionality is one of the biggest challenges that the algorithms of text classification have to face. A large number of methods have previously been introduced to compute the similarity between the two vectors. The method of Kullback-Leibler divergence computes the differences in probability distributions between two vectors. In the Euclidean geometry field, the similarity metric of the Euclidean distance is quite famous and is the usual choice when similarity-based measures are taken into view, for instance, K-means algorithms. In addition to the Euclidean distance, the Manhattan distance is also one well-known metric. The Bray-Curtis, Jaccard Coefficient, and Dice Coefficient are other methods that are popular for computing similarity measures. However, in their study, Dhillon and Modha (2001) concluded that the similarity measure of cosine for text classification and clustering demonstrated the best solutions [8].

Chapter 3

Related Work

3.1 Software Evolution

After an empirical study within IBM by Lehman [21], which aimed to improve the effectiveness of the company's software development, a new prolific research field was introduced rather than changing the development process within the company itself [16]. He proposed laws for software maintenance and evolution such as software evolution can be studied using statistical methods and changes increase the complexity of software. Software evolution is all programming activity that is intended to generate a new software version from an earlier operational version [5] [7]. Thomas ball et al. illustrated ways to extract software history informations from version control systems and thus better understand programs development evolution [1].

3.2 Software Maintenance

Equivalent to Software evolution, Software maintenance are historically dated from the 1960th [4]. Software must needs continious changes, otherwise it will become progressively less useful [20] [12] .Software Maintenance can be classified into four classes, namely adaptive: keeping software usable in changing software environment, secondly perfective: implementation of new user requirements, thirdly corrective: fixing discovered errors in software code and finally preventive: prevent problems in the future before they occour [22]. Notably the first two classes make up more than 70% of the overall software maintenance activities. Bernett and Rajlich presented a roadmap for software maintenance and evolution in their paper and also discussed different aspects, including the research topic of raising abstraction level in which evolution is expressed [30] [2]. An edit script is a collection of tree operations needed to transform T1 (AST of File before change) to T2 (AST of File after change).

3.3 Change classification

Thomas Zimmerman et al developed a tool, namely ROSE, which aims to guide programmers along related changes, furthermore it can warn developers about missing changes. He stated that, "the more there is to learn from history, the more and better suggestions can be made" [37]. Source code changes are needed in order to help increasing the change impact awareness [32] [33]. Later on, Beat Fluri and Harald C Gall presented a taxonomy of source code changes, by deriving them from tree edit operations. Furthermore they classify each change type with a significance level, which expresses how strong a change may impact the source code [10]. A more advanced tool, GumTree, developed by Falleri et al. detects also move operations and produces raw edit scripts, instead of manually classified changes. Moreover it is able to process edit scripts for C, C++, JavaScript etc. next to JAVA [9].

3.4 Change Anlysis

Gall et al analysed changes with Evolizer and ChangeDistiller. They investigated commenting behaviours, discovered change type patterns and changes that fixed bugs. Change patterns are defined as source code changes, which are mostly applied together (e.g parameter renaming impacts all statements that access the parameter inside the method body). Advantages of source code are two fold, first one can perform analysis of source code change patterns to discover violations of principles after shifting paradigms, secondly [13] programmers can be supported in adopting software projects and correct usage of certain guidelines. Sunghun et al specialized in finding bug fixing change patterns [18]. Emanuel Giger et al. performed several experiments to evaluate if source code changes perform better in bug prediction than the metric of LM [1] in source code. Their results clearly show that software source changes outperform LM [14]. Cito et al did an empirical analysis of the docker container ecosystem. In order to facilitate data analytic they created change types and came to the conclusion that most changes deal with dependencies, that are stored in an unstructured manner, furthermore he proposed to introduce an abstraction that would make it easier to deal with package management. This example shows us how software source changes can help through evolution by extrapolating the software history [6] [31].

[1] lines modified

Chapter 4

Approach

This chapter describes the end-to-end process of the overall approach, which supports the mining, processing, analysing and extracting phase of this thesis.

First of all, various methodologies and tools have been conducted and examined in order to find the appropriate data (e.g. simple plain diff Classification with use of Topic modeling algorithms such as Latent Semantic Analysis, Latent Dirichlet allocation and Hierarchical Direchlet Allocation, and more advanced tree differencing approaches such as ChangeDistiller and GumTree).

After choosing the most suitable methodology, the data selection and mining workflow has been set up. This phase includes the data source filtering and selection. In order to allow for flawless scaling, a low-level script has been developed, which can be used for similar mining approaches. Afterwards the data got preprocessed by integrating, cleaning and aggregating it through different other scripts. Subsequently the data got vectorized into numbers, in order to reduce the complexity and to transform the data in a more machine-friendly way.
Thereupon the data got processed and different unsupervised statistical and machine-learning algorithms such as Clustering (K-Means), Word Embeddings (word2vec), Document Embeddings(doc2vec) and Topic Modeling(LDA) have been applied on top of it in order to represent knowledge and thus find patterns and similar software changes.

A proprietary algorithm (DCFSG) provides more information about the structures within a diff and assumes that large diffs which do not occur frequently consist of more than one smaller and more frequent diff. At the end, implementation details are presented and explained in order to make everything reproducible.

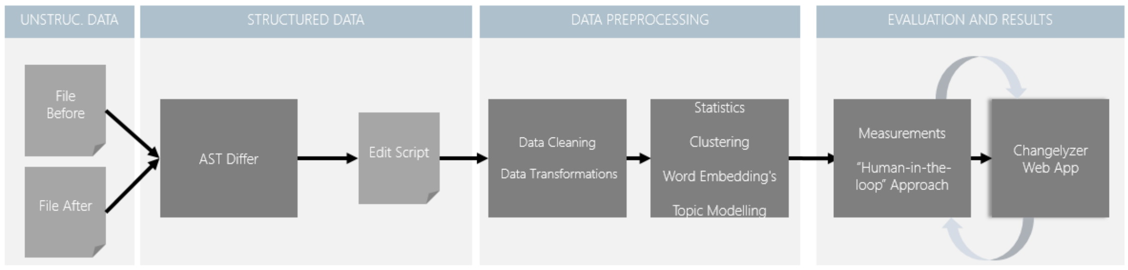


Figure 4.1: Approach overview

4.1 Methodology Selection

Source Code is stored in CVS, which tracks changes on text files by comparing added and deleted lines, thus informations about software changes have very low quality and do not consider structural changes at all. This section aims to find the most suitable data in order to answer RQ1, whose purpose is to find new software change types by applying different methods on big data rather than manually discover change types. The used data sample includes 10 000 diffs from over 500 top rated java projects.

4.1.1 Plain Diff LDA

Idea

As mentioned in chapter 2.6 Background a topic modelling is a type of statistical model for discovering abstract topics that occur in a set of documents. There are different algorithms implementing several heuristics for a maximum likelihood of the fit. This thesis considers and test the following algorithms:

Latent Semantic Analysis is the initial implementation of topic modeling. It creates a matrix of the input and decomposes it into a new document-topic matrix by counting the frequency of a word appeared in a document and thereby it forms a sparse matrix whose rows correspond to terms and whose columns refer to documents.

Latent Dirichlet allocation is a generative topic bag of words model that calculates probabilities that a word belongs to a specific topic. Compared to the normal distribution, which is a probability distribution over all real numbers, LDA is also a probability distribution over probabilities over K distinct categories rather than numbers. The difficulty here is to find the optimal size K which maximizes the likelihood fit.

Hierarchical Dirichlet allocation The HDP mixture model is a natural nonparametric generalization of Latent Dirichlet allocation, where the number of topics can be unbounded and learnt from data rather than specified in advance. Since it is an extension of LDA, one can manually iterate through different K's with LDA to find the best fit.

```
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

bigram = gensim.models.Phrases(data_words, min_count=2, threshold=100)
bigram_mod = gensim.models.phrases.Phraser(bigram)
def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

data_words_bigram = make_bigrams(data_words)
id2word_bigram = corpora.Dictionary(data_words_bigram)
corpus = [id2word_bigram.doc2bow(text) for text in texts]
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=100,
                                            alpha='auto',eta='auto',
                                            iterations=1000)
```

```
doc_lda = lda_model[corpus]
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=texts, dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

Listing 4.1: Plain Text LDA

This thesis assumes that there are multiple change types between two source code files. Therefore a topic is defined as a change type and a diff between two files consists of multiple topics. The idea is to label the different topics manually or with help of the crowd (active learning)

Input

Input is generated by taking two successive commits and getting the changes between them via git diff. For this approach the following two parameters have been used:

1. git diff <commit-before> <commit-after>.

2. git diff –word-diff=plain <commit-before> <commit-after>

The first one shows line by line whereas as the second one considers word diffs, delimited by whitespace. The algorithms above take a collection of documents as input. LDA and LSA takes an additional parameter K (number of topics). More paramaters such as alpha, beta have been set to 'auto'. Number of iterations is set up to 1000.

Preprocessing

In order to avoid overfitting and to increase the model accuracy, only the diff context is used for the input. The following regex was used to remove the surrounding software source code: REGEX CODE HERE. Furthermore, all symbols have been replaced with spelling words (e.g. { to CURLYBRACKETOPEN)

Output

We started our first attemp with the most used topic modelling algorithm LDA:

By iterating through a range of 10-200 topics and measuring the perplexity and coherence of the model, an optimal topic number of 100 was found.

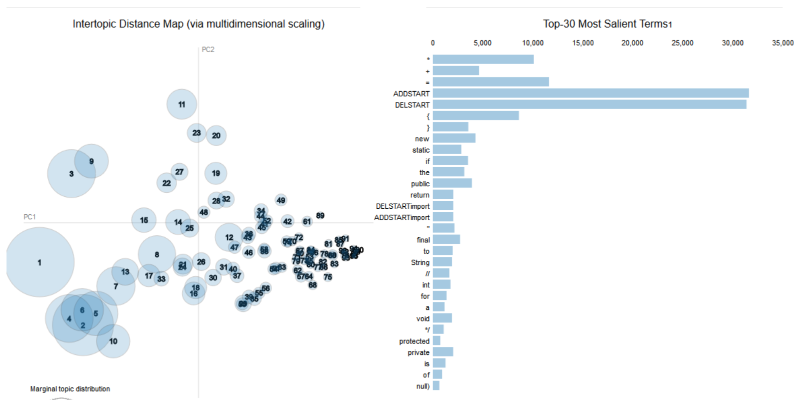


Figure 4.2: Plain Diff PCA

Conclusion

Indeed it clustered some similar code fragments (e.g. modifiers, expressions, comments, import statements). However the topics where not coherent enough and thus we quickly decided to continue with this approach. Noteworthy the output clearly shows that unsupervised clustering works well and similar code snippets can be found, but the results are not sufficient to answer the RQ1. The Goal is to identify and learn change types rather finding similar semantic source code snippets. This problem is well described by Beat Fluri et al., which brings us to the more advanced approaches: Abstract Syntax Tree Differencing.

4.1.2 ChangeDistiller

Idea

Whereas in the previous attempt we tried to extract informations from plain diffs, in this section a more advanced technique is used where the underlying structure of source code, namely Abstract Syntax Tree are differenced. ChangeDistiller is the very first tool to extract fine grained software changes between subsequent revision of Java classes and was implemented 2007. The manually defined taxonomy supports more than 40. [1] change types.

```
File left = new File("FileBefore.java");
File right = new File("FileAfter.java");
FileDistiller distiller = ChangeDistiller.createFileDistiller(Language.JAVA);
try {
    distiller.extractClassifiedSourceCodeChanges(left, right);
} catch(Exception e) {
    System.err.println("Warning: error while change distilling. " + e.getMessage());
}

List<SourceCodeChange> changes = distiller.getSourceCodeChanges();
if(changes != null) {
for(SourceCodeChange change : changes) {
    writeToFile(change.getChangeType())
    }
```

[1]https://github.com/sealuzh/tools-changedistiller/blob/master/src/main/java/ch/uzh/ifi/seal/changedistiller/model/classifiers/ChangeType.ja

```
}
```

Listing 4.2: ChangeDistiller Data Collection

Revive ChangeDistiller was very hard since its last update was in 2014, therefore it was time consuming to update dependencies and make the application runnable. An extension to export ChangeType property from a SourceCodeChange object was also necessary.

Input

The application needs a Java file before and after the change as input.

Preprocessing

As shown in the small source code snippet, exporting ChangeTypes in ChangeDistiller is very easy and straight forward.

Output

Example output looks the following:

```
ADDING_ATTRIBUTE_MODIFIABILITY
ATTRIBUTE_RENAMING
REMOVING_CLASS_DERIVABILITY
STATEMENT_ORDERING_CHANGE
```

Listing 4.3: ChangeDistiller Edit Script Example

Conclusion

A check of Gumtree with a set of 100 Diffs and the manual evaluation showed a precision of 100%. GumTree was and is still a very good tool. Very precise EditScripts, which are human readable and easy to understand.

4.1.3 GumTree

Idea

In 2014 Jean Remy et al developed Gumtree, a fine grained and accurate source code differencing tool, which is freely available. Java Files get parsed by the JDT Eclipse parser. There are different possible outputs feasible. Noteworthy is the "cluster" output type, which summarizes all edits scripts in a node and thus decreases the edit script size. Edit Scripts are build out of JDT Elements rather than manually classified Elements, thus GumTree is more language specific, where as Changedistiller is more aligned to AST operations.

Input

Output type as parameter and two subsequent java files:

```
docker run -v <path-to-files>:/diff gumtree cluster <file-before> <file-after> <output-path>.<output-name>
```

Listing 4.4: Dockerizing GumTree

Preprocessing

In order to use the application the creator of Gumtree had to update some dependencies and fix the dockerfile.

Output

Example output of 'cluster' looks the following:

```
New cluster:
DEL ImportDeclaration
------------
DEL QualifiedName: org.springframework.boot.actuate.metrics.CounterService
DEL ImportDeclaration
```

Listing 4.5: GumTree cluster Edit Script Example

Example output of 'diff' looks the following:

```
Delete QualifiedName: org.springframework.boot.actuate.metrics.CounterService(18)
Delete ImportDeclaration(19)
```

Listing 4.6: GumTree diff Edit Script Example

Conclusion

The output formatting is not machine friendly, but the results are very precise. Manual validation of 100 diffs lead to an accuracy of 99%. Only one diff was attached with a wrong Edit Script. Unfortunately, the Edit Script is not easy to understand, especially when not knowing the language behind, which comes in favor of ChangeDistiller, where manually crafted change types have been defined.

Methodology Selection

Previous sections briefly presented different approaches to generate the right data. The table below shows the summary criteria:

	Plain Text	**ChangeDistiller**	**GumTree**
Simplicity	High	Med	High
Maintenance	true	true	false
Accuracy	0.34 %	99%	100%
Data Quality	Very low	Very High	Medium
Classification	Generic	JDT Parser	Manual
Machine Friendly	true	true	false

Simplicity This criteria implies the effort.

Maintenance Current state of the tool and maintenance frequency.

Accuracy Accuracy of the data after manual validation of 100 Diffs.

Data Quality Information entropy output.

Classification The way data is generated.

Machine Friendly How easy can the output format be processed by a computer

Runtime Complexity This value will affect the scalability, thus the data mining scope.

GumTree's flexibility, language specific output and usability is chosen as the fundamental for the upcoming analysis. Unfortunately, it has no change impact significance level as ChangeDistiller. Since Gumtree does not use manually classified labels, it will allow us to find higher-order edit scripts with help of empirical research.

4.2 Data Analytics

After choosing the best suited methodology, this section focuses on the data analytics part, where firstly the data gets selected, mined, integrated, cleaned, and transformed in order to get the best knowledge representation.

4.2.1 Data Selection

First process step of the data analytics is the data selection, where the appropriate data source and type is determined. Github is a hosting service for version control. It includes more than 57 million repositories and thus is the biggest host of source code in the world. It also offers a well described REST API [2], which allows building tools on top of it. It is possible to search for projects through the API as well, which gives the opportunity to scan and pick projects in large scale. The following listing shows the data selection used in this thesis:

1. Main source: Github.com
2. Programming language: JAVA
3. Forked Projects: False
4. Archived projects: False
5. Min size: 1000 Kilobytes
6. Sorted: by stars in descending order[3]

4.2.2 Data Mining

The data mining process is portioned in six different components. Applying separation of concerns helps improving the maintainability, scalability and expandability to other languages. Bottle neck of the data mining worklflow is the AST Diffing component. It takes about 2 seconds per Diff, which makes it impossible to evade parallelization.

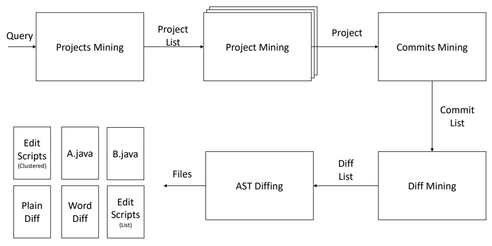


Figure 4.3: Data Mining Process Overview

[2]Application Interface (https://developer.github.com/v3)
[3]https://developer.github.com/v3/search/

Projects Mining

Firstly github gets queried with the parameters defined in the previous section. This thesis considers the first 500 projects. Output is a list of project repositories ending with .git file name extension.

The following code snippet shows our project mining script:

```bash
#!/bin/bash
outfile="$1"
url="https://api.github.com/search/repositories?q="
pages=10
per_page=100
needed=1000
query="\
language:Java\
+fork:false\
+archived:false\
+is:public\
+size:>=1000\
+NOT book in:description,readme\
+NOT cookbook in:name\
+NOT awesome in:name\
+NOT tutorial in:name\
+NOT manual in:name\
"

amount=$((pages*per_page))
rm -f "$outfile.tmp"
while true; do
    for (( i=0; i<$pages; i++ )); do
        searchUrl="$url""$query"'&sort=stars&order=desc&per_page='"$per_page"'&page='"$((i+1))"
        echo "getting $searchUrl"
        wget -q "$searchUrl" -O - \
            | jq '.items[].git_url' \
            | sed -e 's/^"//g;s/"$//g' \
            | head -n "$amount" \
            >> "$outfile.tmp"
        unique=$(sort "$outfile.tmp" | uniq | wc -l)
        echo "found $unique unique projects"
        if [[ "$unique" -ge "$needed" ]]; then break; fi
        sleep "6.1"
    done
    if [[ "$unique" -ge "$needed" ]]; then break; fi
done

sort "$outfile.tmp" | uniq | head -n "$needed" > "$outfile"
rm "$outfile.tmp"
```

Listing 4.7: Projects mining Script

Commits Mining

In order to get the software change artifacts, the commiEvery commit adds the latest changes to the repository. Instead of cloning all projects to the disk, a separate script takes the projects list as input and starts mining the commit shas directly from github, which significantly increases the mining speed. Per project a file named with its repository ID is created and all commits sha's are inserted, starting with the very first one. This way the transaction order is kept.

```
url="https://api.github.com/repos/"
per_page=100
pages=10000
commitFolder="commits/"

get_commits(){
    project=$1
    projectMetaUrl="$url$project"
    echo $projectMetaUrl
    projectId=$(wget -q "$projectMetaUrl" -O - | jq '.id')
    echo "$projectId"
    fileNameTemp="$commitFolder$projectId"'.tmp'
    fileName="commitFolder$projectId"'.txt'
    echo $fileNameTemp

    rm -f $fileNameTemp

    endOfPagination=true

    while $endOfPagination; do
        for (( i=0; i<$pages; i++ )); do
            searchUrl="$url""$1"'/commits?page='"$((i+1))"'&per_page=100'
            echo "getting $searchUrl"

            commitShas=$(curl "$searchUrl" | jq '.[].sha' | tr " " "\n")
            echo $commitShas

            if [ -z "$commitShas"]; then
            endOfPagination=false
            break
            else
            echo $commitShas >> "$fileNameTemp"
            echo $endOfPagination
            fi
        done
        echo "looooop got broken!! BOOM"
    done
}
filename='projects.txt'
echo Start
while read p; do
get_commits $p
done < $filename
```

Listing 4.8: Commits mining Script

Diff Mining

Since we have an ordered list of commits for every project, we can resolve the performance bottleneck by dividing the workload into batches and also use multi threading for every batch. This strategy reduced the mining time from 24 days to less than 24h. First it clones the repository, then it takes two subsequent commits and checks if the commit includes more than 10 files. Afterwards it iterates for every changed file within the commit and saves the file before (A) and after (B) the commit.

AST Diffing

With help of containerization GumTree can be easily used. Input are two subsequent Java Files and the Output locations: docker run –rm -v <MAINFOLDER/projects>:/diff gumtree cluster <nameOfFileA> <nameOfFileB> The output for cluster and diffs is not machine friendly, therefore a fork of the current implementation was unavoidable[4].

Output follows this structure:
{ Change Cluster };[1 ... n Change Pieces]

Whereas one Change consist of several Elements:
<AST Operation> <JDT Element (from)> <JDT Element (to)>

The following graphics shows an example:

from	**to**
New cluster:	
INS ImportDeclaration to CompilationUnit at 9	
————	{INS ImportDeclaration CompilationUnit};\n
INS QualifiedName: Map to ImportDeclaration at 0	[INS QualifiedName ImportDeclaration];\n
INS ImportDeclaration to CompilationUnit at 9	
	[INS ImportDeclaration CompilationUnit]\n
New cluster:	{INS MethodDeclaration TypeDeclaration};
INS MethodDeclaration to TypeDeclaration at 15	[INS MethodDeclaration TypeDeclaration];\n
————	[INS MarkerAnnotation MethodDeclaration];\n
INS MethodDeclaration to TypeDeclaration at 15	[INS SimpleName MethodInvocation];\n
INS MarkerAnnotation to MethodDeclaration at 0	[INS SimpleName MethodDeclaration];\n
INS SimpleName: map to MethodInvocation at 2	[INS SimpleName SimpleType];\n
INS SimpleName: list to MethodDeclaration at 3	[INS Modifier MethodDeclaration];\n
INS SimpleName: String to SimpleType: String at 0	[INS MethodInvocation ReturnStatement];\n
INS Modifier: public to MethodDeclaration at 1	
INS MethodInvocation to ReturnStatement at 0	

Table 4.1: AST Diffing Data Transformation

Mining Results

End Result of the mining pipeline are 5 files per commit shah:

1. **Before and After** <repository_id>_<commit_sha>_<(A | B).java>
2. **Cluster** <repository_id>_<commit_sha>_<file>_<# cluster>_<# inserts><# updates><# deletes><# moves>
3. **Diff** <repository_id>_<commit_sha>_<diff>
4. **Plain-Diff** <repository_id>_<commit_sha>_<plain_diff>
5. **Word-Diff** <repository_id>_<commit_sha>_<git_diff>

This naming convention enables querying on top of the filesystem.

[4]https://github.com/SaliZumberi/gumtree

4.2.3 Data Integration and Data Cleaning

Firstly data from various locations (10 servers) are aggregated into one. In order to get a machine-friendly output positioning numbers, redundant words such as "from, at, to" have been cleaned and removed.

4.2.4 Data Transformation

With help of abstraction, dimensionality reduction simplified the data processing. Three data transformation decisions have been made:

1. Identify granularity levels
2. Define data shape (two or three elements)
3. Index data for every level by transforming them from words to numbers

Transforming words into numbers made it more frugal to process the data as small units. There are 4 different granularity levels of a software change.

Elements

The first level consists of a JDT element, the smallest unit. Every JDT element has an id. This table shows some example Elements and the corresponding description from the official page[5]:

name	**description**
QualifiedType	AST node for a qualified name. A qualified name is defined recursively as a simple name preceded by a name, which qualifies it. Expressing it this way means that the qualifier and the simple name get their own AST nodes.
ForStatement	For statement AST node type.
ThisExpression	Simple or qualified "this" AST node type.
MethodInvocation	Method invocation expression AST node type.
SwitchCase	Switch case AST node type. A switch case is a special kind of node used only in switch statements. It is a Statement in name only.

Change-piece

Change piece consist of three elements, starting with the AST Diff operation (INS | UPD | MOV | DEL) followed by one or two elements. The First element is what has been changed and if AST Diff operation is INS then the insertion location is also specified.

This table shows some example change-pieces:

Change

Changes consist of one or more change-pieces. They summarize changes from an AST Node perspective rather than smaller actions within a change itself. Changes are described as "Clusters" in GumTree. Unlike in change-pieces the AST Diff Operation naming is written out (INSERT | UPDATE | MOVE | DELETE).
This table shows some example changes and the associated change-pieces :

[5]https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fd

name	**description**
DEL ImportDeclaration	Delete import statement
INS SimpleName MarkerAnnotation	Insert a identifier
INS Modifier FieldDeclaration	Insert a modifier (public/private/protected) to a variable
INS PrimitiveType VariableDeclarationStatement	Add a primitive type (int, String, Number etc.) to a variable
MOV Block to ForStatement at 3	Move code block into a for statement

example	type	vectorized
MOVE MethodInvocation	**Change**	2 3
MOV SimpleName MethodInvocation	*Change-piece 1*	14 1 3
MOV SimpleName MethodInvocation	*Change-piece 2*	14 1 3
DELETE ReturnStatement	**Change**	9 65
DEL ReturnStatement	*Change-piece 1*	6 65
DEL SimpleName	*Change-piece 2*	6 14
DEL SimpleName	*Change-piece 3*	6 14

Diff

A diff between two files is defined as collection of changes. Every change is indexed.

id	example	type	vectorized		
1	**MOVE MethodInvocation**	**Change**	2 3		
	MOV SimpleName MethodInvocation	*Change-piece 1*	14 1 3		
	MOV SimpleName MethodInvocation	*Change-piece 2*	14 1 3		
				=>	**Diff: 1 2**
2	**DELETE ReturnStatement**	**Change**	9 65		
	DEL ReturnStatement	*Changwe-piece 1*	6 65		
	DEL SimpleName	*Change-piece 2*	6 14		
	DEL SimpleName	*Change-piece 3*	6 14		

4.3 Implementation Details

Mining scripts have been implemented using shell scripting. For data selection, preprocessing and transformation we chose the programming language python, which is one of the most flexible languages. Notwithstanding it provides very mature machine learning libraries such as sklearn and gensim but also offers data analytic libraries like pandas and numpy. We used the containerization tool Docker to run GumTree independently from our script. The web-tool, which will be presented in the evaluation array, is built upon the Angular framework. In addition to that we use SSR rendering to increase the user experience. In order use the limited time as efficiently as possible, we decided to abandon a back-end implementation and use serverless functions instead. We made use of AWS Lambda, which processes our evaluation request and saves the item to NOSQL database, namely DynamoDB, which is well connected to Lambda through Amazon web-services infrastructure. We also used EC2 Instances for data mining, where we ran 10 c4.8xlarge instances. Within 24h we mined more than 500 projects, resulting in over 500 000 diffs (18.6 GB).

Chapter 5

Results

After mining, preprocessing and transforming the data, the next step is to extract informations by approaching different models. We start with very basic descriptive statistics, where we present how and what changes are applied on JAVA source code, proceeding with more complex applications machine learning techniques namely K-Mean Clustering, which furthers our understanding in clustering similar diffs. Next, we have been inspired by natural language processing techniques (NLP) in particular word embeddings and topic modelling to . Finally a self implemented techniques is used to split diffs into change groups.

5.1 Empirical Analysis

Descriptive statistics help to understand the basic features as well as the distribution of the data.

5.1.1 Elements

The table below shows the top 20 most used Elements:

nr	element	%	count
1	MethodInvocation	12.3	1006432
2	MethodDeclaration	10.2	832854
3	Block	9.6	786251
4	ImportDeclaration	7.1	585954
5	TypeDeclaration	6.5	532170
6	ExpressionStatement	5.6	458257
7	CompilationUnit	5.1	418926
8	SimpleName	5	411820
9	IfStatement	3.4	281818
10	VariableDeclarationStatement	3.2	266434
11	FieldDeclaration	3	248047
12	SimpleType	2.2	185289
13	InfixExpression	2.2	177264
14	ClassInstanceCreation	2.1	170892
15	SingleVariableDeclaration	2	166280
16	TagElement	1.7	146078
17	QualifiedName	1.3	107959
18	StringLiteral	1.3	107088
19	Modifier	1.2	100734
20	ParameterizedType	1.1	96041

Table 5.1: Top 20 most used Elements

Notably the top 20 occuring JDT elements sum up to 86% of all 91 found elements.
Furthermore the first two elements in the ranking are referring to methods. Firstly method invocations in applications and declaration of methods. Interestingly inserting, moving and deletion of dependencies is ranked fourth, which reflects high maintainability and modularity of JAVA. If

statements are changed more frequently than the creation of objects and modifiers such as public, private and protected. This table shows all AST Differencing operations:

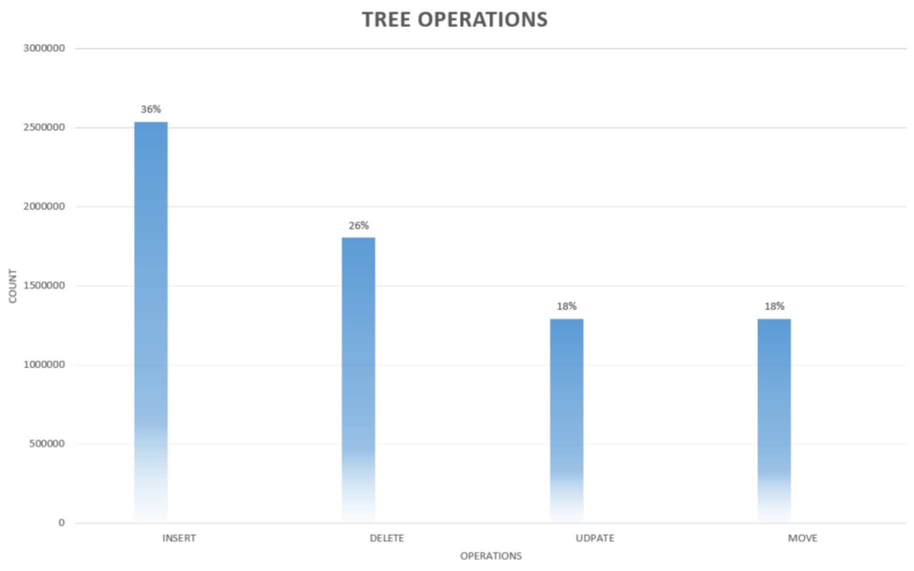


Figure 5.1: image1 caption

Figure 5.2: Tree Operation Distribution

More than one third of all AST Diff operations are Insertions. Impressively Java code is more likely to be deleted than updated and code is updated and moved equally, which shows how important movements of source code is in maintenance and evolution of an application.

5.1.2 Changes

20 most used changes (specific)

Top 20 changes make up 45.55% of the total 895 changes found. Update of identifiers such as methods, variables, classes and interfaces is the most used change in JAVA. Second place is covered by moving something within a method invocation. We can deduce refactoring of method parameters from this change. The diffs below show two examples of this change.

```
@@ -112,7 +112,7 @@ public class MnistInputProvider extends TrainingInputProviderImpl {
   images.seek(16 + size * indexes[i]);
   images.readFully(current);

   for (int j = 0; j < size; j++) {

- tempImages.set(j, i, current[j] & 0xFF);
+ tempImages.set(current[j] & 0xFF, j, i);
   }
}
} catch (IOException e) {
```

nr	Change	%	count
1	UPDATE	12	1048375
2	MOVE MethodInvocation	3.9	316225
3	INSERT ImportDeclaration CompilationUnit	3.3	271501
4	INSERT MethodDeclaration TypeDeclaration	2.9	238768
5	INSERT ExpressionStatement Block	2.45	200732
6	MOVE Block	2.3	187816
7	DELETE ImportDeclaration	2.2	181208
8	DELETE SimpleName	2.1	173312
9	DELETE MethodDeclaration	1.8	147292
10	DELETE ExpressionStatement	1.7	141090
11	DELETE MethodInvocation	1.5	127788
12	INSERT SimpleName MethodInvocation	1.5	121746
13	MOVE MethodDeclaration	1.4	118811
14	INSERT FieldDeclaration TypeDeclaration	1.2	99230
15	INSERT VariableDeclarationStatement Block	1.1	90753
16	INSERT MethodInvocation MethodInvocation	0.9	76084
17	MOVE IfStatement	0.9	74125
18	INSERT IfStatement Block	0.8	70681
19	DELETE VariableDeclarationStatement	0.8	70249
20	MOVE VariableDeclarationStatement	0.8	67277

Table 5.2: Top 20 most used changes

Listing 5.1: Code Snippet 1

```
@@ -12,7 +12,7 @@ public class MnistTargetSingleNeuronOutputConverter implements InputConv
Matrix m = new Matrix(1, input.length);
for (int i = 0; i < input.length; i++) {

- m.set(0, i, (int) input[i]);
+ m.set((int) input[i], 0, i);
}
return m;
```

Listing 5.2: Code Snippet 2

Notably rank 7 to 11 is covered by DELETE statements, starting with deletion of import statement, followed by deleting a SimpleType. The Diff below shows an example of a SimpleType deletion (String).

```
@@ -31,7 +31,7 @@ public final class IncorrectExample extends ExampleSentence {
private final List<String> corrections;

public IncorrectExample(String example) {
- this(example, Collections.<String>emptyList());
+ this(example, Collections.emptyList());
}
```

Listing 5.3: Code Snippet 3

Top 10 (AST Differencing Operations)

nr	INSERT (from)	INSERT (to)
1	ImportDeclaration (12.6%)	Block (19%)
2	MethodDeclaration (11.3%	TypeDeclaration (17.5%)
3	SimpleName (9.8%)	MethodInvocation (15.1%)
4	ExpressionStatement (9.7%)	CompilationUnit (12.7%)
5	MethodInvocation (7.5%)	MethodDeclaration (8.6%)
6	FieldDeclaration (4.6%)	TagElement (2.8%)
7	VariableDeclarationStatement (4.3%)	ClassInstanceCreation (2.5%)
8	IfStatement (3.6%)	IfStatement (2.5%)
9	Block (2.8%)	InfixExpression (2.1%)
10	SimpleType (2.7%)	VariableDeclarationFragment (2%)
Total	*68.9%*	*84.6%*

nr	MOVE	DELETE
1	MethodInvocation (25%)	ImportDeclaration (11.4%)
2	Block (15.2%)	SimpleName (10.9%
3	MethodDeclaration (9.6%)	MethodDeclaration (9.3%)
4	IfStatement (6%)	ExpressionStatement (8.9%)
5	VariableDeclarationStatement (5.4%)	MethodInvocation (8%)
6	SimpleType (4.2%)	VariableDeclarationStatement (4.4%)
7	CompilationUnit (4%)	FieldDeclaration (3.8%)
8	ClassInstanceCreation (3.9%)	Block (3.2%)
9	InfixExpression (3.2%)	IfStatement (3.1%)
10	VariableDeclarationFragment (3%)	SimpleType (3.1%)
Total	*79.5%*	*66.1%*

Table 5.3: Top 10 changes grouped by Tree operations

INSERT (what). Tree Insert operations consists of two components. First what Element has been inserted and secondly where it is inserted. Most inserted element are dependencies followed by implementing a method and defining simple name (identifier). Fourth rank is held by wrapping expressions (e.g. score = 25, number1 == number2) to a complete unit of execution (int score = number1 * number2).

INSERT (where). Not surprisingly the most prominent location for insertions is a Block of statements and declarations, followed by assigning something to a variable. Third place is held by CompilationUnit, which includes package-, import-, type- declaration-, enum- and annotation declarations. 2.5% of all insertions are added when instantiation a new object (ClassInstanceCreation). If statement and Infix expressions (expression (e) infix operator (io) expression (o) e.g. (number1 == number2 (o) & (io) (number1 > 10) (o))) are also important locations for insertions.

MOVE. Java developers mostly move method invocations (one quarter of all movements), followed by Blocks of statements and declarations. Interestingly movement of if statements are placed on the fourth rank. Noteworthy instantiated objects are also very likely to be moved within a class. In contrast to Insert locations, CompilationUnits are less likely to be moved than if statements and variable declaration, thus global constants, annotations are more sedentary.

DELETE. Most of deletions occur on import declarations, which indicates the high modularity of Java programming language. Methods and simple names are also very likely to be deleted within a class. Deletion of instantiated objects is not on the list, which could point out their significance.

5.1.3 Diffs

Most occurring Diffs (n=2)

nr	Changes	%
1	UPDATE UPDATE	17.2

2	INSERT ImportDeclaration CompilationUnit
	INSERT MethodDeclaration TypeDeclaration
		5.9

3	MOVE CompilationUnit UPDATE	4.2
4	INSERT MethodDeclaration TypeDeclaration INSERT MethodDeclaration TypeDeclaration	3.6

5	INSERT ImportDeclaration CompilationUnit
	INSERT ExpressionStatement Block
		2.2

6	DELETE ImportDeclaration DELETE MethodDeclaration	2

7	INSERT ImportDeclaration CompilationUnit
	UPDATE
		2

8	INSERT ExpressionStatement Block INSERT ExpressionStatement Block	1.6
9	DELETE MethodDeclaration DELETE MethodDeclaration	1.5
10	UPDATE DELETE ImportDeclaration	1.5
11	INSERT IfStatement Block MOVE Block	1.2
12	INSERT InfixExpression IfStatement MOVE InfixExpression	1.1
13	INSERT MethodDeclaration TypeDeclaration UPDATE	1.1
14	DELETE ImportDeclaration DELETE ExpressionStatement	1
15	INSERT MethodInvocation MethodInvocation MOVE MethodInvocation	0.8
16	DELETE ExpressionStatement DELETE ExpressionStatement	0.8
17	DELETE ImportDeclaration DELETE ImportDeclaration	0.8

18	INSERT ImportDeclaration CompilationUnit
	INSERT NormalAnnotation TypeDeclaration
		0.7

19	DELETE ImportDeclaration DELETE SingleMemberAnnotation	0.6
20	INSERT FieldDeclaration TypeDeclaration INSERT FieldDeclaration TypeDeclaration	0.6

Table 5.4: Most occurring Diffs (n=2)

1. Updating two identifiers within the class. The following combinations occour the most: (method name, method name), (import, class name),(import, inherited class),(import, method parameter).

2. Importing dependency and creating a new method

3. Updating an import declaration and moving it.

4. Implementation of two new methods.

5. Import dependency and add use it to create an expression, which gets added to a block of declarations and statements.

6. Opposite of 2

7. Import new dependency and use it to replace it on single or multiple places within the file (e.g. importing java.io.File and replace existing String type with File).

8. Add two expressions within the file. Inspecting 20 random Diffs show that in 7 cases the expression statements are the same and thus violated the 'DRY' principle.

9. Opposite of 4.

10. Delete dependency and update identifier where it was used.

11. Surround a block of declarations and statements with an if condition

12. Add new infix expression to if condition and move existing one next to it.

13. Add new method after updating an identifier. A very interesting pattern within this diffs is that an existing method gets split into two methods, where the existing one changes the method name, thus complexity gets reduced by applying separation of concerns.

14. Opposite of 5.

15. Add parameter to method invocation, additionally move existing ones.

16. Opposite of 8.

17. Delete two import statements.

18. Import dependency of multi-member annotation and use it to annotate the class.

19. Delete dependency of single-member annotation and its usage within class.

20. Declare two global constants.

21. Import dependency of single-member annotation and use it to annotate the class.

22. Opposite of 15.

23. Import dependency and add new if statement

24. Use interface or extend class.

26. Opposite of 15.

27. Delete moodier (public, private, protected) in multiple locations.

29. Import dependency of marker annotation (e.g. Test) and use it to annotate the class.

30. Extend if statement with else if condition.

31. Opposite of 29.

32. Implement and invoke new method.

34. Remove conditions from if statement.

35. Opposite of 17.

38. Import new Type and extend method parameter with this type.

41. Delete dependency of multi-member annotation and its usage within class.

43. Similar to 29, but here it annotates methods rather than class

45. Opposite of 24.

48. Update identifier and change name in comment description as well.

50. Update identifiers on multiple locations

54. Declare a variable, which is then used in a newly created if statement

57. Add method parameter and use new paramter within the method to call another method.

59. Clean-up import statements by deleting and moving them around.

61. Add javadoc elements to methods and classes

62. Import qualified name (e.g. enum) and use it within a method invocation

72. Tag method as deprecated and add javadoc comment as well.

78. Change mutiple method modifiers..

79. Add parameter to constructor and add it as input of super method invocation of parents constructor.

79. Instantiate generic type with actuall type (e.g. new ArrayList<>(this.expectations) –> new ArrayList<Expectation>(this.expectations))

94. Add nested condition to if condition

98. Import dependency and add to an Array.

99. Remove declaration within if condition.

100. Remove declared object and expression statement, where its method was called.

111. Remove marker annotation (e.g. @Override, @Deprecated etc.) on methods in multiple locations.

114. Delete object instantiation (new) and call method on static class.

117. Add javadoc comments on method.

124. Move block of statements and declarations into a try and catch statment.

125. Add new infix expression to return statement

127. Extend parameters of a (abstract) method in an interface

136. Change order of method modifiers (e.g. private final static –> private static final)

139. Import new dependency and add anonymous class.

140. Import new dependency and use it as constructor parameter when creating an object.

148. Import new dependency and use it as to throw exception.

150. Extend method with new parameter, and use this parameter as parameter within a constructor of a newly created object.

159. Add cast expression to return statement.

160. Remove multiple null literals.

161. Import new dependency and use it as parameter of a new method invocation, which is a parameter of a newly created object.

165. Surround content of enhanced for loop statement with if statement

165. Opposite of 15

167. Change inherited class.

179. Import Exception type and add catch clause, which makes use of imported exception

192. Remove infix expression in return statement.

197. Add new case in switch statement and also add a return statement.

199. Replace throw statement by return statement.

200. Add parametrized type to existing method paramter (e.g. public void setDict(DictionaryEntity dict, –> public void setDict(DictionaryEntity<? extends DictionaryItemEntity> dict,).

201. Exend return statement with conditional expression.

236. Add try catch statement within a if statement

Most occurring Diffs (n=3)

nr	Changes	%
1	UPDATE UPDATE UPDATE	9.8
2	UPDATE UPDATE MOVE CompilationUnit	4.2

3	INSERT MethodDeclaration TypeDeclaration
	INSERT ImportDeclaration CompilationUnit
	INSERT ImportDeclaration CompilationUnit
		3.1

4	INSERT MethodDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT ImportDeclaration CompilationUnit
		2

5	UPDATE
	UPDATE
	INSERT ImportDeclaration CompilationUnit
		1.9

6	INSERT FieldDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
		1.7

7	UPDATE UPDATE DELETE ImportDeclaration	1.4
8	INSERT MethodDeclaration TypeDeclaration INSERT MethodDeclaration TypeDeclaration INSERT MethodDeclaration TypeDeclaration	1.2
9	DELETE ImportDeclaration DELETE ImportDeclaration DELETE MethodDeclaration	1
10	INSERT Block MethodDeclaration DELETE Block MOVE Block	0.9

11	UPDATE
	INSERT MethodInvocation MethodInvocation
	MOVE MethodInvocation
		0.7

12	DELETE ImportDeclaration DELETE MethodDeclaration DELETE MethodDeclaration	0.7
13	INSERT ExpressionStatement Block INSERT ExpressionStatement Block INSERT ExpressionStatement Block	0.7
14	UPDATE DELETE MethodInvocation MOVE MethodInvocation	0.7
15	DELETE FieldDeclaration DELETE MethodDeclaration DELETE MethodDeclaration	0.7
16	DELETE MethodDeclaration DELETE MethodDeclaration DELETE MethodDeclaration	0.6

17	INSERT SingleMemberAnnotation TypeDeclaration
	INSERT ImportDeclaration CompilationUnit
	INSERT ImportDeclaration CompilationUnit
		0.6

18	INSERT ImportDeclaration CompilationUnit
	INSERT ExpressionStatement Block
	INSERT ExpressionStatement Block
		0.5

19	INSERT FieldDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT ExpressionStatement Block
		0.5

20	DELETE ImportDeclaration DELETE ImportDeclaration DELETE ImportDeclaration	0.5

Table 5.5: Most occurring Diffs (n=3)

1. Updates three identifiers within the class. Mostly same change is applied.

2. Updates dependency and changes new type in multiple locations.

3. Add two dependencies and implement a method which makes use of it.

6. Declare a global variable and add two methods, which use it.

8. Implement three methods.

9 . Opposite of 3

11. . Insert three expressions. Mostly one expression added to three different locations.

12. . Delete global variable together with corresponding getter and setter method.

14. . Import two dependencies, first one is used as annotation, second one as member of annotation

18. . Opposite of 11

Most occurring Diffs (n=4)

nr	Changes	%
1	UPDATE UPDATE UPDATE UPDATE	5.8
2	MOVE CompilationUnit UPDATE UPDATE UPDATE	2.5

3	INSERT ImportDeclaration CompilationUnit
	INSERT ImportDeclaration CompilationUnit
	INSERT ImportDeclaration CompilationUnit
	INSERT MethodDeclaration TypeDeclaration
		1.8

4	INSERT ImportDeclaration CompilationUnit
	INSERT ImportDeclaration CompilationUnit
	INSERT MethodDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
		1.5

5	INSERT MethodDeclaration TypeDeclaration INSERT MethodDeclaration TypeDeclaration INSERT MethodDeclaration TypeDeclaration INSERT MethodDeclaration TypeDeclaration	0.9

6	INSERT ImportDeclaration CompilationUnit
	INSERT MethodDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
		0.7

7	DELETE ImportDeclaration DELETE ImportDeclaration DELETE ImportDeclaration DELETE MethodDeclaration	0.6
8	DELETE ImportDeclaration DELETE ImportDeclaration DELETE MethodDeclaration DELETE MethodDeclaration	0.6

9	INSERT ImportDeclaration CompilationUnit
	UPDATE
	UPDATE
	UPDATE
		0.5

10	INSERT ImportDeclaration CompilationUnit
	INSERT FieldDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
		0.4

11	INSERT ExpressionStatement Block INSERT ExpressionStatement Block INSERT ExpressionStatement Block INSERT ExpressionStatement Block	0.3
12	UPDATE UPDATE UPDATE DELETE ImportDeclaration	0.3
13	DELETE MethodDeclaration DELETE MethodDeclaration DELETE MethodDeclaration DELETE MethodDeclaration	0.3

14	INSERT ImportDeclaration CompilationUnit
	INSERT ImportDeclaration CompilationUnit
	INSERT ExpressionStatement Block
	INSERT ExpressionStatement Block
		0.3

15	DELETE ImportDeclaration DELETE ImportDeclaration DELETE ImportDeclaration DELETE ImportDeclaration	0.3

16	INSERT FieldDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT ExpressionStatement Block
		0.3

17	UPDATE DELETE ImportDeclaration DELETE ImportDeclaration DELETE ImportDeclaration	0.3
18	UPDATE UPDATE DELETE ImportDeclaration DELETE ImportDeclaration	0.3

19	INSERT FieldDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
	INSERT SingleVariableDeclaration MethodDeclaration
	INSERT ExpressionStatement Block
		0.3

20	INSERT ImportDeclaration CompilationUnit
	INSERT Block MethodDeclaration
	MOVE Block
	DELETE Block
		0.3

Table 5.6: Most occurring Diffs (n=4)

1. . Exactly same pattern as we saw in n=2 and n=3. Four similar updates on identifiers have been applied.

2. Update import statements after package re-factoring.

11. Insert four expressions. Mostly one expression added to four different locations.

Top diffs with 4 changes are very similar to the previous table, with the difference that a change occurs several times.

5.2 Clustering (K-Means)

In order to cluster similar diffs, we first created a similarity matrix based on cosine similarity and started training the model with 1000 Iterations. Since we use cosine similarity we can not calculate the similarity of diffs with different lengths, because cosine similarity calculates vector angles, thus comparing short diffs with longer ones which leads to inaccurate results. We will inspect diffs of sizes three, four and five changes. Our goal is to find change patterns, which can be described by a sentence or even labelled with a word. After finding an appropriate number of clusters for every diff size, we calculate and plot the cosine similarity within a cluster and the number of diffs belonging to the cluster. This facilitates picking important and coherent clusters from the plotting. Obviously the more items in a cluster, the lower the cosine similarity within the cluster.Therefore, we try to maximize the cosine similarity and number of diffs.

5.2.1 Clusters(n=3)

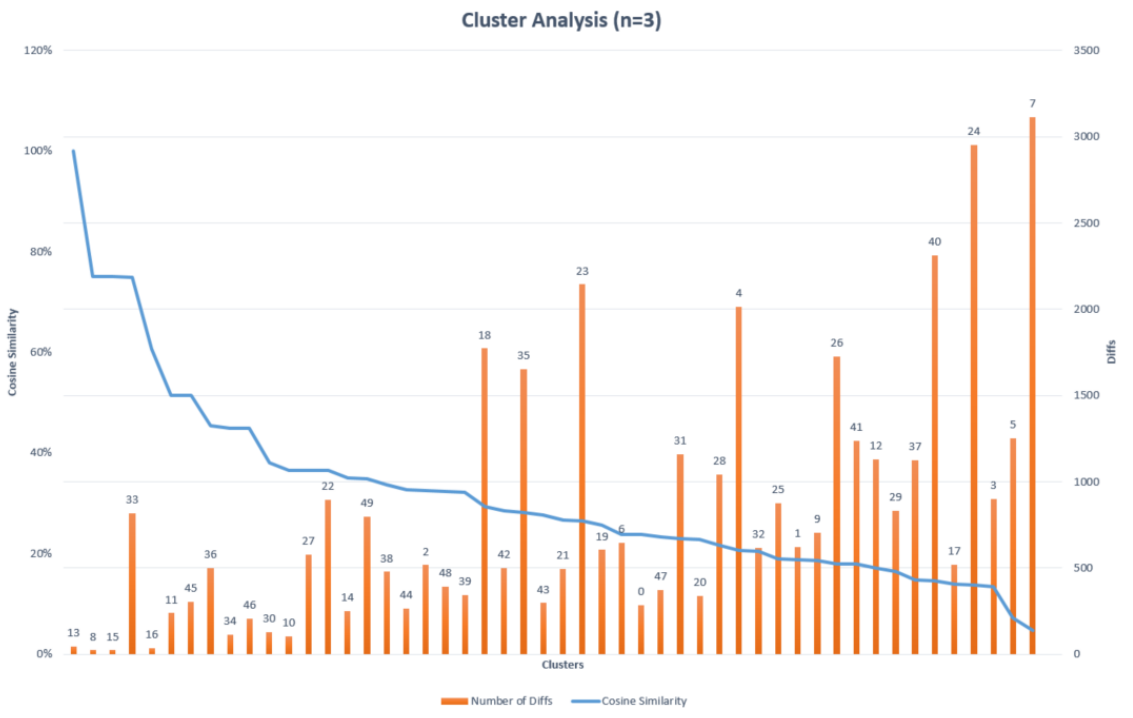


Figure 5.3: Cluster Analysis (n=3)

According to the evaluation, clusters with three unique changes reveal similar diffs. The following change clusters are noteworthy to describe briefly:

Cluster 33 Opposite of Cluster 13, method is now within the class.

Cluster 11. Chained method invocation is moved into the previous.

Cluster 45. Create an if condition which makes use of newly added dependency / global variable and implement a new method.

Cluster 36. Parameter of a returned method invocation gets changed, additionally if conditions are added

Cluster 46. Method gets removed and new dependency is being added and used as a parameter within a method invocation

Cluster 27. Dependency gets deleted and removed where it was used, mostly in constructor parameters and super method invocations.

Cluster 22. Import dependency and instantiate it and use its methods.

Cluster 49. Remove global variable and its usage within methods, sometimes also its dependency get deleted.

Cluster 10. A instantiated Object gets replaced by a internal object, finally the corresponding import gets deleted as well.

Cluster 12 Method invocation parameters are changed by a simple name.

Cluster 13 After removing import statements a method called within the Class is called from an object (e.g. assertXpathEvaluatesTo –> XMLAssert.assertXpathEvaluatesTo).

Cluster 14 Import of dependency which is then used in a newly created if and else statement.

Cluster 15 Deletion of dependency, addition of new dependency, which is then used as constant or variable within the class.

Cluster 16 + 43 Surround expression statement by if condition.

Cluster 44 String parameter in method invocation gets re-factored by newly added dependency

5.2.2 Clusters(n=4)

According to the evaluation, clusters with four unique changes exhibit slightly dissimilar diffs within it.
However, we tried to describe the ones occurring more than 100 times and with the highest cosine similarity.

Cluster 22 Replace return statement or simple expression with newly imported Class, additionally add if condition, where the new dependency is used.

Cluster 61 Remove dependency and add a new one, which is then used as type within a method parameter in an interface.

Cluster 54 Insert new method declaration and add new if condition or extend if condition.

Cluster 59 Replace usage of old dependency with new one in multiple places.

Cluster 44 Declare global variable and use this variable as parameter of a new method invocation, which is chained on a existing method invocation.

Cluster 34 Add conditions in a if and else condition).

Cluster 37 Replace method invocations, if conditions by surrounding some parts of them with a new return statement

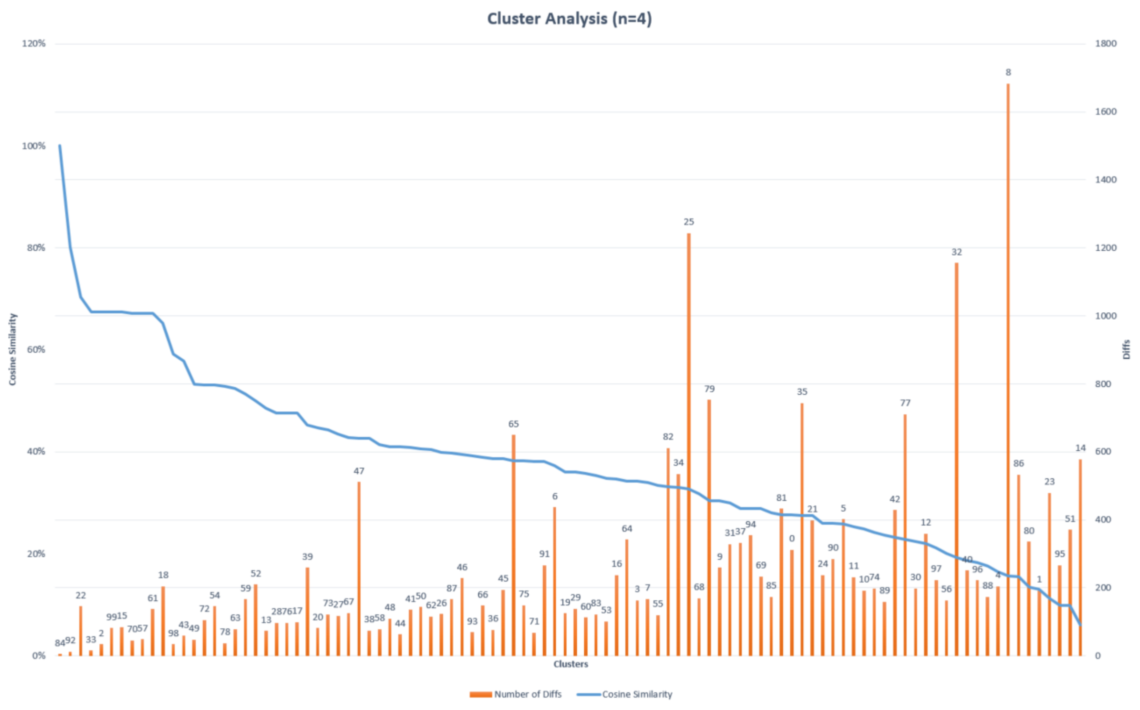


Figure 5.4: Cluster Analysis (n=4)

5.2.3 Clusters(n=5)

According to the evaluation, clusters with five unique changes display slightly dissimilar diffs within it.
However, we tried to describe the ones occurring more than 100 times and with the highest cosine similarity.

Cluster 68 Remove global variables and replace parameters in method invocations in multiple places.

Cluster 70 Replace dependency and re-factor its usage.

Cluster 38 Add several import statements and implement new methods, additionally also some global constants are declared.

Cluster 30 Similar to 70, but here same re-factoring applied on multiple locations.

Cluster 36 Remove method invocation within a parameter of a method invocation

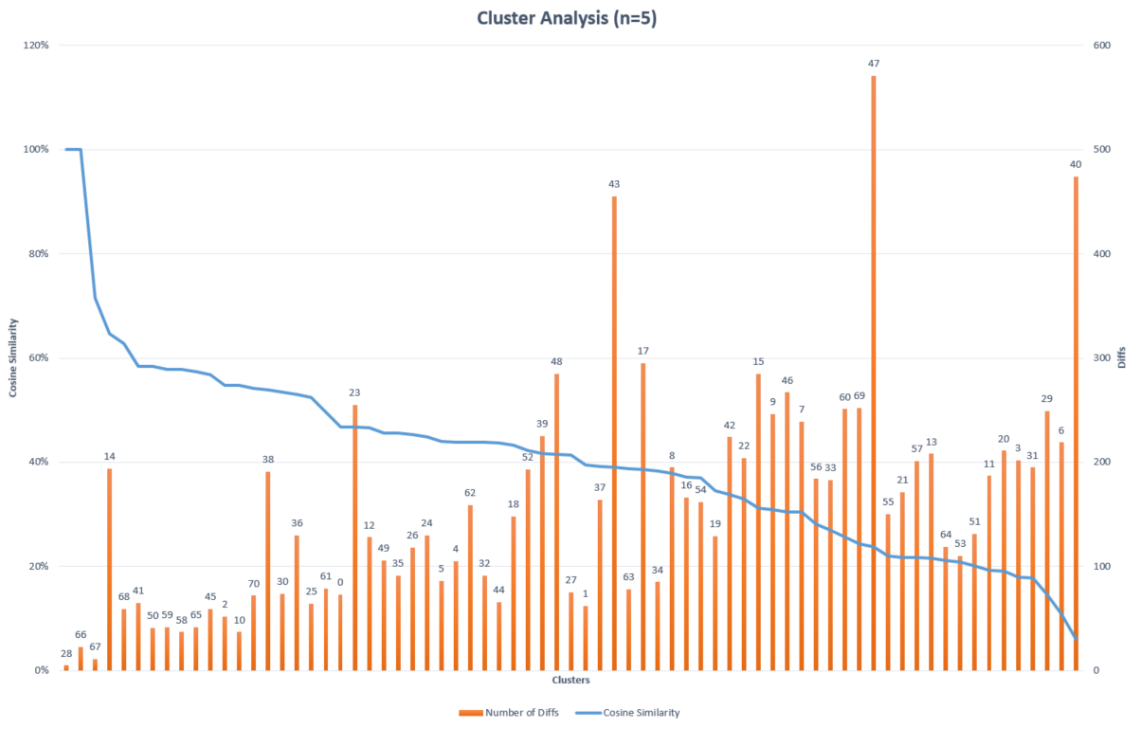


Figure 5.5: Cluster Analysis (n=5)

5.3 Word Embeddings (Word2Vec)

Word2vec is one the most used word embedding models. It consists of two layers neural net and basically is a technique to represent word as embeddings by transforming words into vectors (list of numbers). With help of an gradient decent, these entities are trained to be useful representations in their context. Since its release it has not only been used for text classification, but also in different applications. Output of word2vec goes beyond basic syntactic regularities, e.g KIN, Man, Queen example showed that is is possible to perform algebraic operations on vector("King") – vector("Man") + vector("Woman") ==> results in a vector Queen.

Since this thesis is not using words but changes, we can call it change2vec instead of word2vec. The model parameters need to be adjusted to the input characteristic. Two major differences exist when comparing diffs to text. Firstly diffs are very short (see statistic above) and secondly the used vocabulary consists of 895 unique changes. In this use case, the input is a diff (list of numeric labelled changes) (e.g: 21 4 13 5) rather than text, thus the diff dimension is set to 10 and window size to 2, because we want at least two changes. Size of context window determines how many neighbours should be included. Shorter window produces more related terms. Number of iterations over the corpus have been set to 1000. Since we are interested in similar changes we use the Continues-Bag-of-Words model (CBOW) to contextualize the changes.

The model took 5h to train 534116 diffs. Noteworthy the trained model knows only about the change indices, thus no informations about the JDT elements.

5.3.1 Top 10 Change Similarities

The table below shows the most similar changes in descending order. Word2vec computes cosine similarity between a simple mean of the projection weight vectors of the given words and vectors for each word in the model.

	Change	Most Similar 1	Most Similar 2
1	INSERT Block IfStatement	MOVE IfStatement (99.42%)	INSERT SimpleName IfStatement (94.95)
2	INSERT TagElement TagElement	INSERT TextElement TagElement (99.38)	MOVE TagElement (85.81)
3	INSERT QualifiedName EnumConstantDeclaration	INSERT StringLiteral EnumConstantDeclaration (99.34%)	MOVE EnumConstantDeclaration
4	INSERT SuperFieldAccess InfixExpression	INSERT ParenthesizedExpression SuperConstructorInvocation (98.63%)	INSERT BooleanLiteral SuperMethodInvocation (98.35)
5	INSERT MethodRef TagElement	INSERT MemberRef TagElement (98.25%)	INSERT SimpleName TagElement (95.64%)
6	INSERT BooleanLiteral AssertStatement	INSERT BooleanLiteral IfStatement (98.24%)	INSERT SimpleName DoStatement (98.11%)
7	INSERT ParameterizedType ParameterizedType	INSERT SimpleType ParameterizedType (97.62%)	MOVE ParameterizedType (92.50)
8	INSERT ForStatement SwitchStatement	INSERT SimpleName SwitchStatement (97.53 %)	INSERT EnhancedForStatement SwitchStatement (93.69%)
9	INSERT InfixExpression InfixExpression	INSERT ParenthesizedExpression InfixExpression (97.39 %)	INSERT PrefixExpression InfixExpression (95.93%)
10	MOVE TryStatement	INSERT Block TryStatement (97.18%)	INSERT CatchClause TryStatement (91.62%)

Table 5.7: Top 10 Change Similarities

1. (If statements Adding a block of declarations and statements into a if statement is very similar to moving the if statement or adding a simple name within it.

2. (TagElement) Indeed TagElements and TextElements are very similar according to JDT Documentation.

3. (Enums) In fact, inserting a qualified name (Enum from import statement) and string literal into enum declaration it's the same.

4. (Super) Inserting a super variable is very similar to add an expression to super constructor invocation or adding a boolean to a super method invocation. Inheritance was successfully determined here.

5. (TagElement) Adding a reference of a method, member or simple name to tag element.

6. (Booleans) Interestingly adding a boolean literal to an assert statement is very similar to adding a boolean literal into a if statement and a simple name into a do statement.

7. (Parameterized Types) Adding a parametrized type to a parametrized type is highly similar to adding a simple name to a parametrized type

8. (Switch) Remarkably inserting a for loop into a switch statement and adding a simple name is analogous, as well as adding an enhanced for loop.

9. (Infix) Noteworthy how accurate w2v equates the addition of an infix- and prefix expression into an if statement condition.

10. (Try Catch) Moving a try statement and adding a block of declaration is according to the model similar, furthermore also adding a catch-clause is very

By only providing numeric encoded diffs, word2vec was able to find similar change types. This shows us that there are patterns and syntactical structures within source code changes.

5.3.2 Top 10 Change Similarities (Average Cosine Similarity)

This table presents the top ten change similarity based on the calculated cosine similarity between similar changes. Overall mean cosine similarity is 82%.

	change	**Most Similar 1**	**Most Similar 2**	**Most Similar 3**	**Ø**
1	INSERT NullLiteral SuperConstructorInvocation	INSERT BooleanLiteral SuperConstructorInvocation	INSERT NumberLiteral SuperConstructorInvocation	INSERT StringLiteral SuperConstructorInvocation	0.972
2	INSERT AnnotationTypeMemberDeclaration AnnotationTypeDeclaration	INSERT MarkerAnnotation AnnotationTypeDeclaration	MOVE AnnotationTypeMemberDeclaration	MOVE AnnotationTypeDeclaration	0.967
3	INSERT SimpleType SingleVariableDeclaration	INSERT SimpleName SingleVariableDeclaration	INSERT ArrayType SingleVariableDeclaration	INSERT PrimitiveType SingleVariableDeclaration	0.966
4	INSERT ClassInstanceCreation ConstructorInvocation	INSERT ClassInstanceCreation SuperConstructorInvocation	INSERT CastExpression ConstructorInvocation	INSERT BooleanLiteral SuperConstructorInvocation	0.963
5	INSERT MethodDeclaration EnumDeclaration	INSERT EnumConstantDeclaration EnumDeclaration	INSERT FieldDeclaration EnumDeclaration	INSERT EnumDeclaration TypeDeclaration	0.963
6	INSERT MarkerAnnotation SingleVariableDeclaration	INSERT SingleMemberAnnotation SingleVariableDeclaration	INSERT SimpleName SingleVariableDeclaration	INSERT NormalAnnotation SingleVariableDeclaration	0.963
7	INSERT EnumDeclaration TypeDeclaration	INSERT MethodDeclaration EnumDeclaration	INSERT EnumConstantDeclaration EnumDeclaration	INSERT FieldDeclaration EnumDeclaration	0.960
8	INSERT ParenthesizedExpression ConditionalExpression	INSERT InfixExpression ConditionalExpression	INSERT ParenthesizedExpression PrefixExpression	INSERT PrefixExpression ConditionalExpression	0.960
9	INSERT ConditionalExpression ConstructorInvocation	INSERT PrefixExpression ConstructorInvocation	INSERT InfixExpression ConstructorInvocation	INSERT ConditionalExpression ConditionalExpression	0.958
10	INSERT TypeParameter TypeDeclaration	INSERT ParameterizedType TypeParameter	INSERT SimpleType TypeParameter	INSERT ParameterizedType TypeDeclaration	0.958

Table 5.8: Top 10 Change Similarities (Average Cosine Similarity)

5.3.3 Top 12 Cluster (Survey)

In order to create a cluster of changes, it is necessary to create a similarity matrix between the changes by using the built in word-similarity function of word2vec. A nested for loop through all changes does the job.

Survey Results. The table below shows the TOP 12 Clusters. The next section will present the corresponding evaluation. We tried to describe the cluster briefly, ideally this part would be defined by using the crowd with active learning.

It is very surprising how well word2vec identified similar changes, without knowing the elements behind the index, thus we can say that changes can be clustered into similar groups by only using their appearance in a diff without even exploiting the edit script.

Cluster 1
INSERT Modifier TypeDeclaration
INSERT SingleMemberAnnotation TypeDeclaration
INSERT ParameterizedType TypeDeclaration
INSERT SimpleName TypeDeclaration
INSERT TypeDeclaration TypeDeclaration
INSERT QualifiedName SimpleType
MOVE CompilationUnit
INSERT MarkerAnnotation TypeDeclaration
MOVE TypeDeclaration
INSERT NormalAnnotation TypeDeclaration
INSERT SimpleType TypeDeclaration
INSERT TypeDeclaration CompilationUnit
Changes related to types

Cluster 2
INSERT VariableDeclarationFragment FieldDeclaration
INSERT PrimitiveType FieldDeclaration
INSERT ParameterizedType FieldDeclaration
INSERT MarkerAnnotation FieldDeclaration
MOVE FieldDeclaration
INSERT Modifier FieldDeclaration
INSERT SimpleType FieldDeclaration
INSERT ArrayType FieldDeclaration
INSERT ArrayType MethodDeclaration
Changes related to variables

Cluster 3
INSERT NumberLiteral MethodInvocation
INSERT QualifiedName MethodInvocation
INSERT MethodInvocation ReturnStatement
INSERT NullLiteral MethodInvocation
MOVE VariableDeclarationFragment
INSERT MarkerAnnotation PackageDeclaration
INSERT StringLiteral MethodInvocation
INSERT ClassInstanceCreation MethodInvocation
INSERT SimpleName MethodInvocation
INSERT MethodInvocation ExpressionStatement
INSERT MethodInvocation MethodInvocation
INSERT ArrayCreation MethodInvocation
INSERT MethodInvocation Assignment
MOVE MethodInvocation
Changes related to method invocations

Cluster 4
INSERT SimpleName EnumConstantDeclaration
INSERT EnumConstantDeclaration EnumDeclaration
INSERT QualifiedName ConditionalExpression
INSERT SimpleType EnumDeclaration
INSERT MarkerAnnotation VariableDeclarationStatement
INSERT Modifier EnumDeclaration
INSERT Javadoc EnumConstantDeclaration
INSERT EnumDeclaration TypeDeclaration
INSERT ClassInstanceCreation EnumConstantDeclaration
INSERT TypeLiteral EnumConstantDeclaration
INSERT NumberLiteral EnumConstantDeclaration
INSERT QualifiedName EnumConstantDeclaration
INSERT CastExpression EnumConstantDeclaration
INSERT NumberLiteral ConstructorInvocation
INSERT StringLiteral EnumConstantDeclaration
INSERT NullLiteral EnumConstantDeclaration
INSERT FieldAccess SynchronizedStatement
MOVE EnumConstantDeclaration
INSERT BooleanLiteral EnumConstantDeclaration
INSERT FieldDeclaration EnumDeclaration
INSERT QualifiedName PostfixExpression
INSERT QualifiedName PrefixExpression
INSERT MethodDeclaration EnumDeclaration
Changes related to Enums

Cluster 5
INSERT EmptyStatement Block
INSERT ForStatement Block
INSERT VariableDeclarationStatement Block
INSERT WhileStatement Block
MOVE DoStatement
INSERT Block ForStatement
INSERT SynchronizedStatement Block
INSERT PrimitiveType VariableDeclarationStatement
INSERT IfStatement Block
MOVE WhileStatement
INSERT VariableDeclarationExpression ForStatement
DELETE DoStatement
INSERT NullLiteral VariableDeclarationFragment
MOVE ForStatement
INSERT EnhancedForStatement Block
INSERT DoStatement Block
INSERT ExpressionStatement Block
Changes related to code blocks

Cluster 6
INSERT SimpleName ArrayCreation
DELETE ArrayInitializer
INSERT PrimitiveType ArrayType
INSERT NumberLiteral ArrayCreation
MOVE ArrayCreation
INSERT ArrayAccess MethodInvocation
DELETE ArrayType
INSERT SimpleName SingleMemberAnnotation
INSERT MethodInvocation ArrayCreation
INSERT SimpleType ArrayType
INSERT ArrayInitializer ArrayCreation
DELETE ArrayCreation
INSERT ArrayAccess FieldAccess
INSERT ArrayInitializer VariableDeclarationFragment
INSERT ArrayType ArrayCreation
INSERT InfixExpression ArrayCreation
INSERT ArrayCreation VariableDeclarationFragment
INSERT Dimension ArrayType
INSERT PrimitiveType VariableDeclarationExpression
INSERT ArrayAccess VariableDeclarationFragment
INSERT Dimension VariableDeclarationFragment
Changes related to Arrays

Cluster 7
MOVE ConstructorInvocation
INSERT FieldAccess ClassInstanceCreation
INSERT MethodInvocation ConstructorInvocation
INSERT SimpleName SuperConstructorInvocation
INSERT ClassInstanceCreation ConditionalExpression
INSERT SimpleName ConstructorInvocation
INSERT NullLiteral SuperConstructorInvocation
INSERT ConstructorInvocation Block
INSERT ConditionalExpression ClassInstanceCreation
INSERT WhileStatement SwitchStatement
INSERT TypeDeclarationStatement SwitchStatement
INSERT SimpleName SuperMethodInvocation
INSERT PrefixExpression ClassInstanceCreation
INSERT ParenthesizedExpression ClassInstanceCreation
Changes related to calling classes in super-class, and adding elements to constructors as well

Cluster 8
DELETE FieldDeclaration
INSERT CastExpression LambdaExpression
DELETE TypeDeclaration
DELETE SingleVariableDeclaration
DELETE Initializer
DELETE MarkerAnnotation
DELETE TagElement
DELETE EnumDeclaration
DELETE TextElement
DELETE SimpleName
DELETE Modifier
DELETE MethodDeclaration
DELETE PrimitiveType
Changes related to deletion of simple elements such as elements, declarations, types

Cluster 9
DELETE TryStatement
DELETE Block
DELETE ReturnStatement
DELETE ForStatement
DELETE VariableDeclarationStatement
DELETE VariableDeclarationExpression
DELETE EnhancedForStatement
DELETE CastExpression
DELETE IfStatement
DELETE SynchronizedStatement
DELETE ExpressionStatement
DELETE CatchClause
DELETE InstanceofExpression
DELETE MethodInvocation
DELETE WhileStatement
DELETE ConditionalExpression
DELETE EmptyStatement
Changes related to deletion of more advanced code structs such as compound statements, expressions and method invocations

Cluster 10
INSERT ReturnStatement IfStatement
INSERT Block WhileStatement
INSERT Block IfStatement
MOVE PrefixExpression
INSERT InstanceofExpression IfStatement
INSERT MethodInvocation IfStatement
INSERT IfStatement IfStatement
MOVE IfStatement
INSERT Block SynchronizedStatement
INSERT PrefixExpression IfStatement
INSERT SimpleName IfStatement
MOVE SynchronizedStatement
INSERT BreakStatement Block
INSERT BooleanLiteral ReturnStatement
INSERT ContinueStatement Block
INSERT ExpressionStatement IfStatement
INSERT InfixExpression WhileStatement
INSERT InfixExpression IfStatement
Changes related to if statements

Cluster 11
INSERT PackageDeclaration CompilationUnit
DELETE AnonymousClassDeclaration
INSERT BooleanLiteral ClassInstanceCreation
INSERT InfixExpression ClassInstanceCreation
INSERT ClassInstanceCreation ClassInstanceCreation
INSERT CastExpression SwitchStatement
INSERT NullLiteral ClassInstanceCreation
INSERT ThisExpression ClassInstanceCreation
INSERT TypeLiteral ClassInstanceCreation
INSERT MethodInvocation ClassInstanceCreation
INSERT SimpleName ClassInstanceCreation
INSERT AnonymousClassDeclaration ClassInstanceCreation
INSERT StringLiteral ClassInstanceCreation
MOVE ClassInstanceCreation
INSERT NumberLiteral ClassInstanceCreation
INSERT SimpleType ClassInstanceCreation
INSERT CastExpression ClassInstanceCreation
INSERT MethodInvocation SuperConstructorInvocation
INSERT QualifiedName ClassInstanceCreation
Changes related to creating new objects

Cluster 12
INSERT QualifiedName EnhancedForStatement
INSERT StringLiteral SwitchCase
INSERT PostfixExpression ClassInstanceCreation
INSERT EnhancedForStatement SwitchStatement
INSERT ThrowStatement SwitchStatement
INSERT LabeledStatement Block
INSERT ReturnStatement SwitchStatement
INSERT MethodInvocation SwitchStatement
INSERT QualifiedName SwitchCase
INSERT SwitchCase SwitchStatement
DELETE EnumConstantDeclaration
INSERT SimpleName SwitchCase
INSERT AssertStatement SwitchStatement
INSERT ForStatement SwitchStatement
INSERT SimpleName SwitchStatement
INSERT ClassInstanceCreation ExpressionStatement
Changes related to switch statement

Table 5.9: Top 12 Most coherent Clusters

5.4 Document Embeddings (Doc2Vec)

Doc2Vec is an extension of word2vec, which aims to create a numeric representation of a document rather than words in word2vec. The idea is to get similarities among diffs, thus the provided input will be a list of diffs. Since can be considered as short text, we set a small alpha 0.005.

```
tagged_data = [TaggedDocument(words=_d, tags=[str(i)]) for i, _d in enumerate(diffs)]
max_epochs = 100, vec_size = 7, alpha = 0.005

doc2vecmodel= Doc2Vec(size=vec_size,
                        alpha=alpha,
                        min_alpha=0.00025,
                        min_count=1,
                        dimension=2,
                        dm =1, workers=8)

doc2vecmodel.build_vocab(tagged_data)

for epoch in range(max_epochs):
    print('iteration {0}'.format(epoch))
    doc2vecmodel.train(tagged_data,
    total_examples=doc2vecmodel.corpus_count,
    epochs=doc2vecmodel.iter)
    # decrease the learning rate
    doc2vecmodel.alpha −= 0.0002
    # fix the learning rate, no decay
    doc2vecmodel.min_alpha = doc2vecmodel.alpha
```

Listing 5.4: Doc2Vec Implementation

After training for more than 5h we presented the randomly chosen results and the Top 5 similar changes and asked 5 developers how similar the diffs are. Unfortunately the survey result is very disappointing. The next chapter shows the evaluation. Applying K-mean clustering on top of doc2vec embeddings leads sadly to the following result, which reflects the survey values:

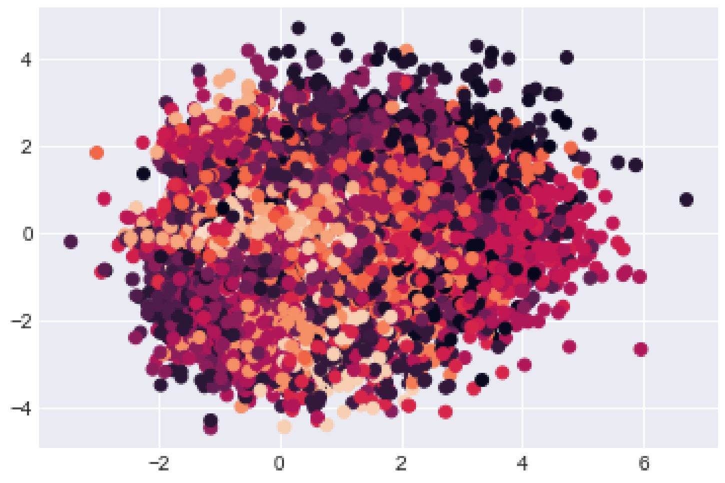


Figure 5.6: Clustering Doc2Vec

5.5 Topic Modelling (LDA)

With Topic modeling we are able to derive hidden patterns in diffs and extract change topics in a unsupervised manner, fully automatically. Topics can be seen as statistically repeated pattern of co-occuring terms (changes. LDA is a general purpose technique and thus can be applied in several applications such as tracking geographical location with geo-aware topics [19], extracting gene-phenotype relationships and biomarkers [35], duplicate bug report detection [27], comparing twitter and traditional media using topic modeling [36].

The difference between Topic Modeling and Word Embeddings is that LDA can derive higher correlations than two-elements, word2vec allows us to use vector geometry. LDA also shows statistical relationship of occurrences rather than real semantic information, which are embedded in words. In other words, LDA can be used to map a document (in our case diff) to vector and word2vec to word (change) to vector.

Before we start training the model we prepare the input by selecting diffs of length 4 to 13 changes, creating a dictionary and trigrams to infer the corpus (term document frequency).

With the help of different measurements we defined the optimal number of topics to 35.

```
bigram = gensim.models.Phrases(data_words, min_count=2, threshold=100) # higher threshold fewer phrases.
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)
trigram_mod = gensim.models.phrases.Phraser(trigram)
data_words_trigrams = make_trigrams(data_words)
# Create Dictionary
id2word_trigram = corpora.Dictionary(data_words_trigrams)
# Term Document Frequency
corpus = [id2word_trigram.doc2bow(text) for text in data_words_trigrams]

lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=55,
                                            alpha='auto',eta='auto',
                                            iterations=100)
doc_lda = lda_model[corpus]
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus)) # a measure of how good the model is. lower the better.
# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=texts, dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

Listing 5.5: Topic Modeling Implementation

In order to get a better understanding of the allocated topic, we make use of the principal component analysis, which is used for dimension reduction, thus squashing a multi-dimensional dataset into a proper lower-dimensional hyperplane. This enables us to visualize topics as bubbles. The larger the bubble, the more widespread it is. Overlapping topics may indicate too many topics.

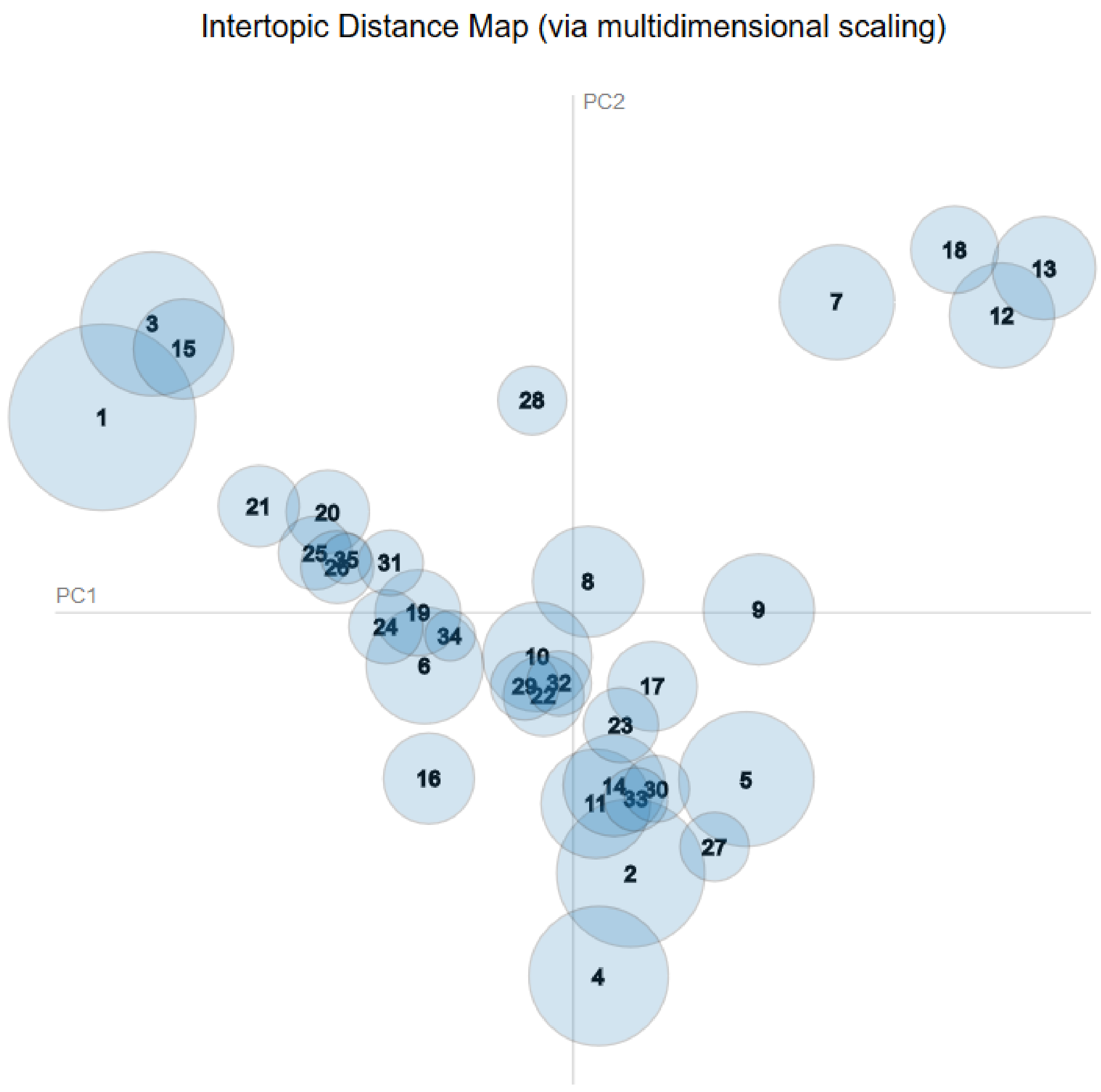


Figure 5.7: PCA Plot of Topic Modeling Results

A coherent topic can be labeled / described in a few words. The study participants could optionally label a topic. The following list shows the processed labels and descriptions

Topic 3 - method invocations

Topic 4 - type declarations

Topic 5 - variable declaration and code block

Topic 6 - imports

Topic 7 - source code block

Topic 8 - variable declaration

Topic 10 - simple names

Topic 11 - add something to a method invocation

Topic 13 - if statements and blocks

Topic 16 - method declarations

Topic 17 - condition in if statement

Topic 18 - move of if statment

Topic 19 - literals (boolean, String, Number)

Topic 20 - tags and annotation

Topic 24 - switch statement

Topic 26 - paramterized typs

Topic 28 - method invocations in express and returns

Topic 35 - variables in condition

17 out of 35 Topics have been labelled. Interestingly Topics with an uniform tag distribution are more likely to be labelled. Noteworthy, the trained model never saw the JDT Element tags, only the numeric labels. However, this shows us, that topics can be assigned to diffs in exactly the same way as to documents. Topic modelling on software changes, can speed up writing of commits, by showing the topic labels and distribution of local changes. Another use case would be adding topic labels to Pull Requests.

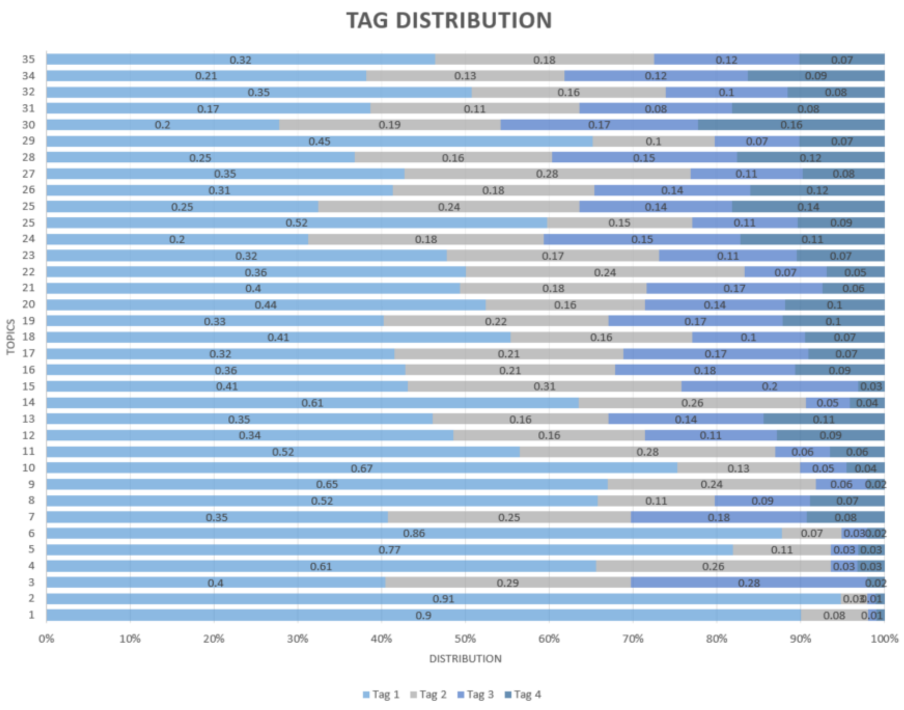


Figure 5.8: Topic Distribution

5.6 Diff Change Frequency Score Grouping

So far based on the defined data abstraction, a diff is a collection of changes. To get more informations about existing and occurring patterns in diffs, we had to make an assumption that diffs could be composed of change groups.

Example: Diff: 12 31 49 0 19 93 could be a group of: [(12 0), (49 31 19) (93)] whereas every tuple is a change group.

First of all for every diff, all possible splitting combinations are created, thus an exponential rise of combination for every additional change in a diff (e.g. diff with 2 changes –> n=2 –> $2^{(n-1)} = 2$ After some clean-ups a nested for loop through all combinations and the corresponding fragments checks if the fragment exists in the overall dataset. If this is the case, the following formula calculates the scoring of a combination:

$$\sum_{i=1}^{d=2^{(d-1)}} \left(\sum_{j=1}^{m} (d/c_j) \times f_j \right)_i \tag{5.1}$$

N is number of combinations, M is number of change groups within combination, d the amount of changes within the diff, c amount of changes within a change group and f the frequency of the change group within the overall dataset.

Advantages of this approach is:

1. Dimensionality and Complexity reduction
2. find change groups according to scores
3. use output to cluster change groups

Implementation:

```python
def generate_combinations(text):
    words = text.split()
    ns = range(1, len(words)+1) # n = 1..(n−1)
    for n in ns: # split into 2, 3, 4, ..., n parts.
        for idxs in itertools.combinations(ns, n):
            yield [' '.join(words[i:j]) for i, j in zip((0,) + idxs, idxs + (None,))]


def get_best_combined_cluster(diff, returntype):
    combination_cleaned = []
    for x in generate_combinations(diff):
        new_combi = []
        for element in x:
            if element != '' and element not in new_combi:
                new_combi.append(element)
        combination_cleaned.append(new_combi)

    combination_cleaned_unique = []
```

```
for x in combination_cleaned:
    if x not in combination_cleaned_unique:
    combination_cleaned_unique.append(x)
    print('Total combinations generated: ' + str(len(combination_cleaned_unique)))

    best_combo_vector = ' '
    best_combo_length = 10
    best_combo_score = 0
    treshold = 100
    for x in combination_cleaned_unique:
        score2 = 0
        vector = ''
        exists = False
        for element in x:
        try:
            unqique_item = diff_df_meta.loc[diff_df_meta['vectorized'] == element.strip()].index[0]
            if(len(element.split(' '))>1) or element == diff:
            score += (len(diff.split(' '))/len(element.split(' '))) * diff_df_meta['freq'][unqique_item]
            else:
            score += 0
            exists = True
            vector += ' ' + str(unqique_item)
        except Exception as e:
            exists = False
            break

        if exists == True:
        form_score = score2 / len(x)
        if best_combo_score3 < form_score3:
        best_combo = vector
        best_combo_score = form_score
        best_combo_length = len(vector.split(' '))

    return best_combo, best_combo_score
```

Listing 5.6: DCFSG Implementation

This table shows change groups with 2 changes

	change group	%

1	INSERT ImportDeclaration CompilationUnit
	INSERT ExpressionStatement Block
		2.98

2	MOVE MethodInvocation INSERT ExpressionStatement Block	2.28

3	UPDATE
	INSERT VariableDeclarationStatement Block
		2.27

4	DELETE FieldDeclaration UPDATE	1.81
5	UPDATE MOVE MethodDeclaration	1.8

6	UPDATE
	INSERT FieldDeclaration TypeDeclaration
		1.45

7	INSERT FieldDeclaration TypeDeclaration
	INSERT MethodDeclaration TypeDeclaration
		1.41

8	INSERT MethodInvocation MethodInvocation MOVE MethodInvocation	1.33
9	DELETE ExpressionStatement DELETE ImportDeclaration	1.21
10	DELETE SimpleName INSERT SimpleName MethodInvocation	1.18

Table 5.10: DCFSG - Change groups with 2 items

Indeed, almost all change groups are covered by the most occurring diffs (n=2). This means our model works accurate enough and splits diffs based on frequencies of our dataset.

This table shows change groups with 3 changes

	change group	%

1	INSERT MethodInvocation MethodInvocation
	INSERT ImportDeclaration CompilationUnit
	MOVE MethodInvocation
		5.43

2	MOVE InfixExpression MOVE Block INSERT InfixExpression IfStatement	1.97

3	INSERT Block MethodDeclaration
	INSERT FieldDeclaration TypeDeclaration
	DELETE Block
		1.86

4	UPDATE
	MOVE VariableDeclarationStatement
	INSERT VariableDeclarationStatement Block
		1.7

5	INSERT IfStatement Block DELETE IfStatement MOVE IfStatement	1.28
6	MOVE Block MOVE MethodInvocation INSERT ExpressionStatement Block	1.2

7	UPDATE
	MOVE MethodDeclaration
	INSERT SingleVariableDeclaration MethodDeclaration
		0.97

8	DELETE MethodInvocation DELETE ImportDeclaration MOVE MethodInvocation	0.87

9	INSERT Block MethodDeclaration
	INSERT SingleVariableDeclaration MethodDeclaration
	DELETE Block
		0.83

10	INSERT SimpleName MethodInvocation
	MOVE Block
	INSERT ImportDeclaration CompilationUnit
		0.77

Table 5.11: DCFSG - Change groups with 3 items

Similar for change groups of size 3, we have almost identical results as we presented in most occouring diffs (n=3).

5.6.1 Word Embeddings on Change groups

Since we gathered change-groups from diffs, we are now able to create embeddings between the groups and find similar change-groups same as we applied for changes in previous section. The table below shows the results after training the model on numeric labels.

nr	Change group	Similar group 1	Similar group 2

1	UPDATE
	INSERT Javadoc MethodDeclaration
		UPDATE
		INSERT Javadoc PackageDeclaration
			UPDATE
			INSERT MarkerAnnotation FieldDeclaration

2	INSERT BreakStatement SwitchStatement
	INSERT SwitchCase SwitchStatement
		INSERT ReturnStatement SwitchStatement
		INSERT SwitchCase SwitchStatement
			INSERT ReturnStatement SwitchStatement
			INSERT SwitchCase SwitchStatement
			INSERT SimpleName SwitchCase

3	INSERT FieldDeclaration TypeDeclaration
	INSERT SimpleName InfixExpression
		INSERT VariableDeclarationStatement Block
		INSERT SimpleName InfixExpression
			INSERT FieldDeclaration TypeDeclaration
			INSERT IfStatement Block

4	UPDATE
	DELETE InfixExpression
		UPDATE
		INSERT ParenthesizedExpression InfixExpression
			UPDATE
			INSERT VariableDeclarationStatement SwitchStatement

5	INSERT TryStatement Block
	INSERT VariableDeclarationStatement Block
		UPDATE
		INSERT AssertStatement Block
			INSERT VariableDeclarationStatement Block

6	INSERT VariableDeclarationStatement Block
	INSERT IfStatement Block
		INSERT FieldDeclaration TypeDeclaration
		INSERT IfStatement Block
			INSERT QualifiedName MethodInvocation
			INSERT IfStatement Block

7	UPDATE DELETE Block	UPDATE DELETE ReturnStatement	UPDATE DELETE SwitchStatement

8	UPDATE
	INSERT TryStatement Block
		UPDATE
		INSERT TryStatement Block
		MOVE TryStatement
			UPDATE
			INSERT Block IfStatement

9	INSERT TagElement TagElement
	INSERT TextElement TagElement
		INSERT TagElement TagElement
		MOVE TagElement
		INSERT TextElement TagElement
			INSERT TagElement TagElement
			MOVE TagElement

10	INSERT ClassInstanceCreation MethodInvocation
	INSERT ImportDeclaration CompilationUnit
		INSERT MethodInvocation ClassInstanceCreation
		INSERT ImportDeclaration CompilationUnit
			INSERT ClassInstanceCreation MethodInvocation
			MOVE ClassInstanceCreation
			INSERT ImportDeclaration CompilationUnit

Table 5.12: DCFSG - Similar Change Groups

Impressively our model performs very accurate in finding similar change groups. We commented four interesting change groups:

1. Update of identifier and adding javadoc comment to a method is similar to adding javadoc to a packagedeclaration. Interestingly these two are similar to updating an identifier and inserting a marker annotation (@Test) , reason for this could be the syntactical location of javadoc comments and annotations.

2. Inserting a break statemnt and switch case is similar to adding a return statement and switch case to swtich statement. Indeed from a syntax perspective it is very similar.

3. Highly similar change groups. Adding a declaration of variable, field and then extending a condition in if statement

10. Importing new class and create object of new class into a method invocation is similar to importing new class, instantiate object of it, and add a method invocation.

Chapter 6

Evaluation

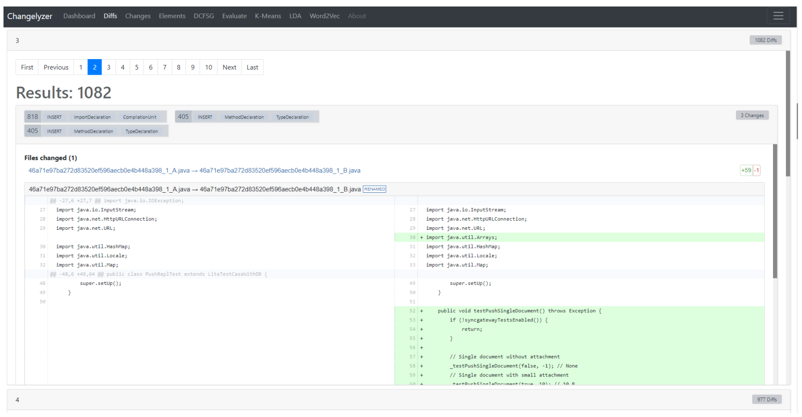
This section presents the evaluation of the machine learning technique used in section 4.

6.1 Approach

Since all used techniques are unsupervised, no cross-validation is possible. Therefore we have to optimize the model with measurements (e.g. perplexity, coherence) and methods (e.g perplexity, coherence score) on one side, and on the other side we make use of the "Human-in-the-loop" Validation Apporach. We roll out different surveys, where we present the topics and clusters to domain specialists.

6.1.1 Changelyzer Web Application

In order to visualize the labelled software changes a web based prototype was implemented. The Main motivation of this tool was to support the selected study participants on the evaluation. Later on functions like searching and navigating have been implemented to facilitate exploration of source code changes.



Results: 1082

Files changed (1)

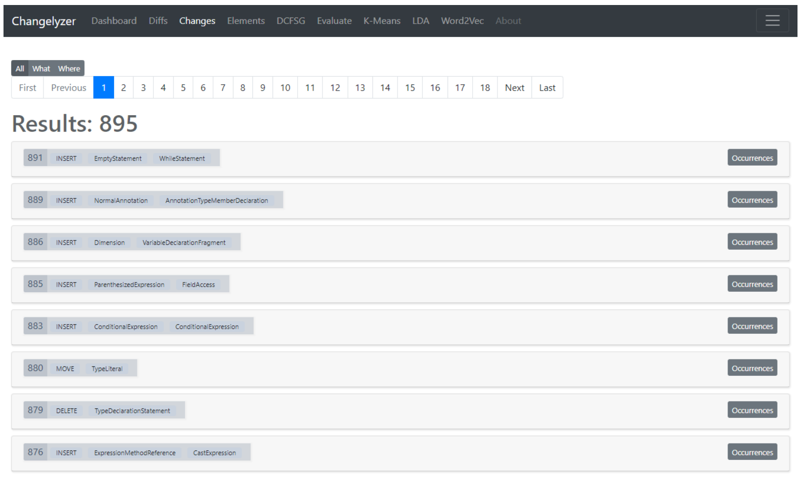
46a71e97ba272d83520ef596aecb0e4b448a398_1_A.java → 46a71e97ba272d83520ef596aecb0e4b448a398_1_B.java

46a71e97ba272d83520ef596aecb0e4b448a398_1_A.java → 46a71e97ba272d83520ef596aecb0e4b448a398_1_B.java

```
@@ -27,6 +27,7 @@ import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

import java.util.HashMap;
import java.util.Locale;
import java.util.Map;
@@ -48,6 +49,64 @@ public class PushReplTest extends LiteTestCaseWithDB {
        super.setUp();
    }

```

```
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
+ import java.util.Arrays;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;

        super.setUp();
    }

+     public void testPushSingleDocument() throws Exception {
+         if (!syncgatewayTestsEnabled()) {
+             return;
+         }
+
+         // Single document without attachment
+         _testPushSingleDocument(false, -1); // None
+         // Single document with small attachment
```

Figure 6.1: Changelyzer WebApp - Diffs



Results: 895

```
891 INSERT EmptyStatement WhileStatement
889 INSERT NormalAnnotation AnnotationTypeMemberDeclaration
886 INSERT Dimension VariableDeclarationFragment
885 INSERT ParenthesizedExpression FieldAccess
883 INSERT ConditionalExpression ConditionalExpression
880 MOVE TypeLiteral
879 DELETE TypeDeclarationStatement
876 INSERT ExpressionMethodReference CastExpression
```

Figure 6.2: Changelyzer WebApp - Changes

6.1.2 Study

Study participants

For this study we selected one junior software developer, two professional software developers and two seniors. They work full time in a software house in Zurich, where they provide solutions for different clients. Java is used on daily basis. Due resource limitations, surveys were not conducted at the same time, but over and over again during 3 weeks. We have asked the developers not to discuss it until the survey is complete, otherwise the survey could be biased.

6.2 Model Evaluation

6.2.1 K-Mean Clusters

Measurements

We use two measurements for our K-Means clustering evaluation. Firstly we need to determine the number of right clusters. One way to do it is by using the elbow method, which shows us how much the error rate decreases when an additional cluster is added. Ideally the plot should form an elbow. Number of clusters is therefore on the bottom left side. Secondly we make use of the silhouette measurement, which basically measures the space between clusters, and we therefore try to maximize this value. Combining those two values will support us finding the right number of clusters.

50 Clusters for diffs with 3 unique changes.

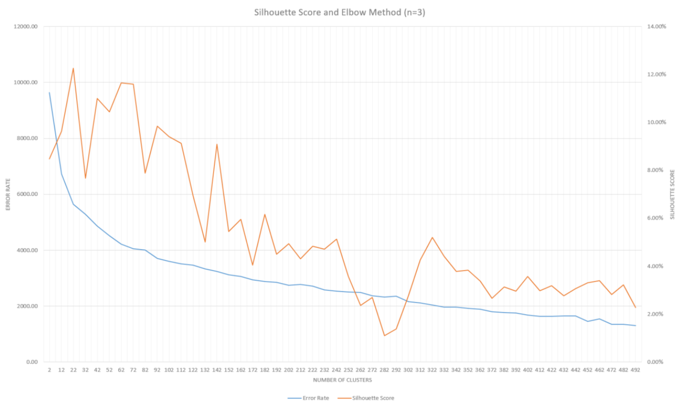


Figure 6.3: Silhouette Score and Elbow Method (n=3)

100 Clusters for diffs with 4 unique changes.

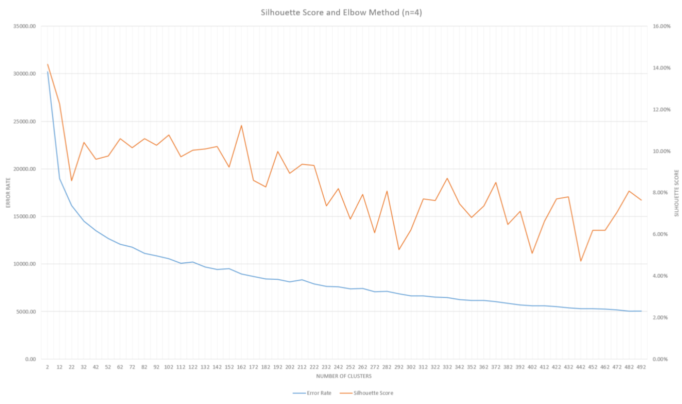


Figure 6.4: Silhouette Score and Elbow Method (n=4)

71 Clusters for diffs with 5 unique changes.

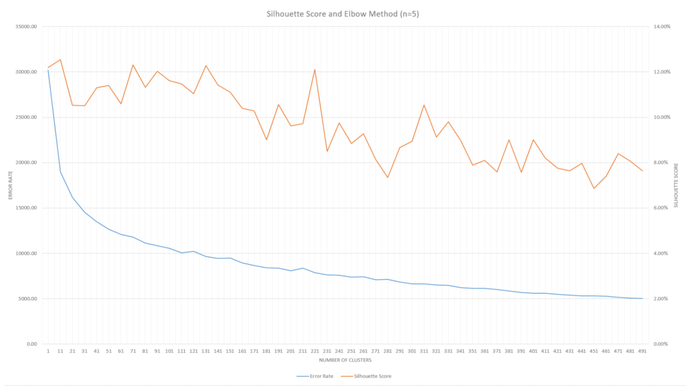
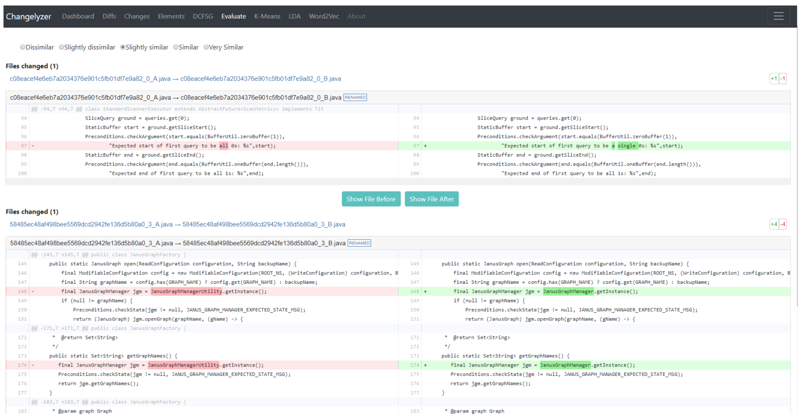


Figure 6.5: Silhouette Score and Elbow Method (n=5)

Survey



Dissimilar Slightly dissimilar [✓ Slightly similar] Similar Very Similar

Files changed (1)

c08eacef4e6eb7a2034376e901c5fb01df7e9a82_0_A.java → c08eacef4e6eb7a2034376e901c5fb01df7e9a82_0_B.java +1 -1

c08eacef4e6eb7a2034376e901c5fb01df7e9a82_0_A.java → c08eacef4e6eb7a2034376e901c5fb01df7e9a82_0_B.java RENAMED

```
@@ -94,7 +94,7 @@ class StandardScannerExecutor extends AbstractFuture<ScanMetrics> implements Tit
                SliceQuery ground = queries.get(0);
                StaticBuffer start = ground.getSliceStart();
                Preconditions.checkArgument(start.equals(BufferUtil.zeroBuffer(1)),
-                       "Expected start of first query to be all 0s: %s",start);
                StaticBuffer end = ground.getSliceEnd();
                Preconditions.checkArgument(end.equals(BufferUtil.oneBuffer(end.length())),
                        "Expected end of first query to be all 1s: %s",end);
```

```
                SliceQuery ground = queries.get(0);
                StaticBuffer start = ground.getSliceStart();
                Preconditions.checkArgument(start.equals(BufferUtil.zeroBuffer(1)),
+                       "Expected start of first query to be a single 0s: %s",start);
                StaticBuffer end = ground.getSliceEnd();
                Preconditions.checkArgument(end.equals(BufferUtil.oneBuffer(end.length())),
                        "Expected end of first query to be all 1s: %s",end);
```

Files changed (1)

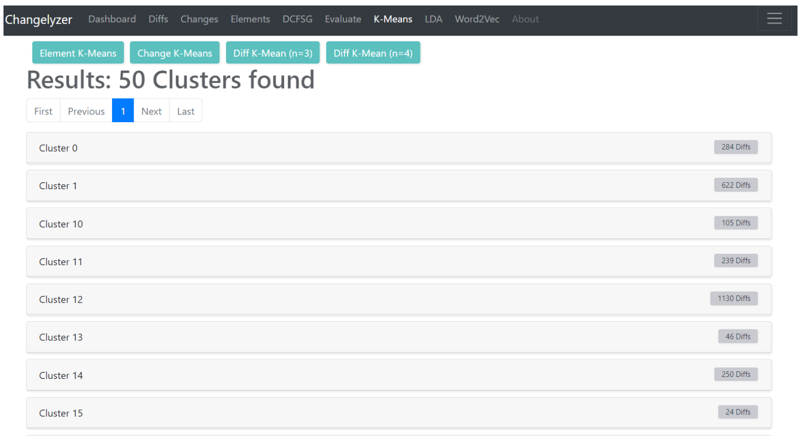
58485ec48af498bee5569dcd2942fe136d5b80a0_3_A.java → 58485ec48af498bee5569dcd2942fe136d5b80a0_3_B.java +4 -4

58485ec48af498bee5569dcd2942fe136d5b80a0_3_A.java → 58485ec48af498bee5569dcd2942fe136d5b80a0_3_B.java RENAMED

```
@@ -145,7 +145,7 @@ public class JanusGraphFactory {
    public static JanusGraph open(ReadConfiguration configuration, String backupName) {
        final ModifiableConfiguration config = new ModifiableConfiguration(ROOT_NS, (WriteConfiguration) configuration, B
        final String graphName = config.has(GRAPH_NAME) ? config.get(GRAPH_NAME) : backupName;
-       final JanusGraphManager jgm = JanusGraphManagerUtility.getInstance();
        if (null != graphName) {
            Preconditions.checkState(jgm != null, JANUS_GRAPH_MANAGER_EXPECTED_STATE_MSG);
            return (JanusGraph) jgm.openGraph(graphName, (gName) -> {
@@ -171,7 +171,7 @@ public class JanusGraphFactory {
     *  @return Set<String>
     */
    public static Set<String> getGraphNames() {
-      final JanusGraphManager jgm = JanusGraphManagerUtility.getInstance();
       Preconditions.checkState(jgm != null, JANUS_GRAPH_MANAGER_EXPECTED_STATE_MSG);
       return jgm.getGraphNames();
    }
@@ -183,7 +183,7 @@ public class JanusGraphFactory {
     * @param graph Graph
```

```
    public static JanusGraph open(ReadConfiguration configuration, String backupName) {
        final ModifiableConfiguration config = new ModifiableConfiguration(ROOT_NS, (WriteConfiguration) configuration, B
        final String graphName = config.has(GRAPH_NAME) ? config.get(GRAPH_NAME) : backupName;
+       final JanusGraphManager jgm = JanusGraphManager.getInstance();
        if (null != graphName) {
            Preconditions.checkState(jgm != null, JANUS_GRAPH_MANAGER_EXPECTED_STATE_MSG);
            return (JanusGraph) jgm.openGraph(graphName, (gName) -> {

     *  @return Set<String>
     */
    public static Set<String> getGraphNames() {
+      final JanusGraphManager jgm = JanusGraphManager.getInstance();
       Preconditions.checkState(jgm != null, JANUS_GRAPH_MANAGER_EXPECTED_STATE_MSG);
       return jgm.getGraphNames();
    }

     * @param graph Graph
```

Figure 6.6: Changelyzer - Cluster Evaluation



Results: 50 Clusters found

Cluster 0	284 Diffs
Cluster 1	622 Diffs
Cluster 10	105 Diffs
Cluster 11	239 Diffs
Cluster 12	1130 Diffs
Cluster 13	46 Diffs
Cluster 14	250 Diffs
Cluster 15	24 Diffs

Figure 6.7: Changelyzer - Cluster Overview

Cluster Coherence Survey (n=3)

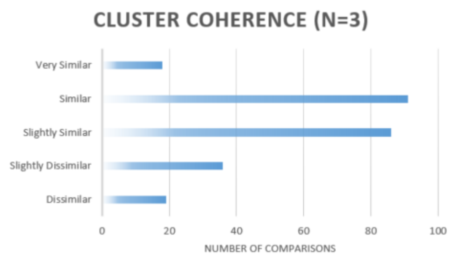


Figure 6.8: Cluster Coherence Survey (n=3)

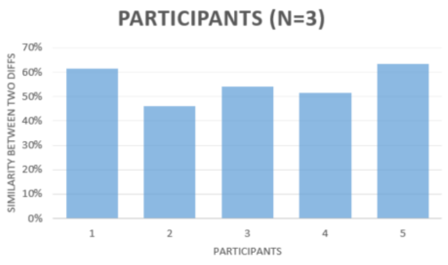


Figure 6.9: Cluster Participants(n=3)

Cluster Coherence Survey (n=4)

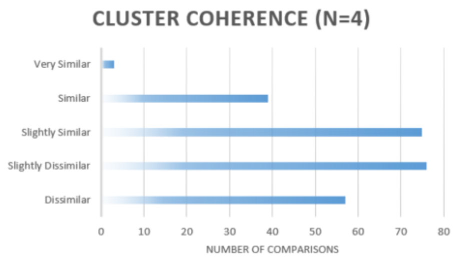


Figure 6.10: Cluster Coherence Survey (n=4)

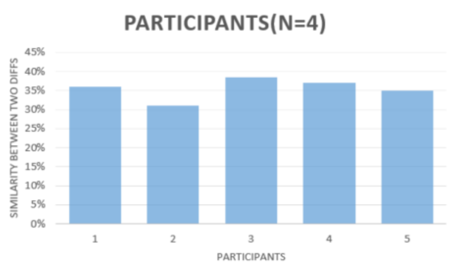


Figure 6.11: Cluster Participants (n=4)

Cluster Coherence Survey (n=5)

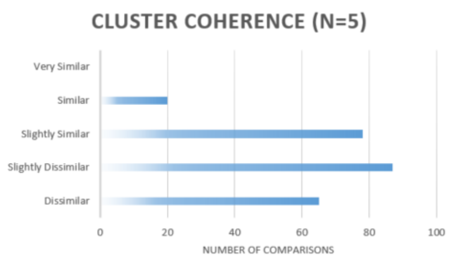


Figure 6.12: Cluster Coherence Survey (n=5)

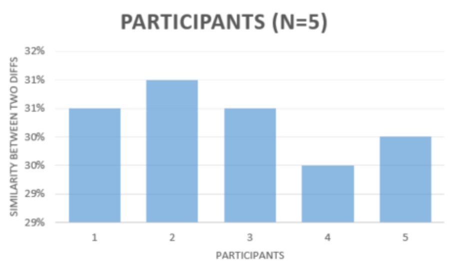


Figure 6.13: Cluster Participants (n=5)

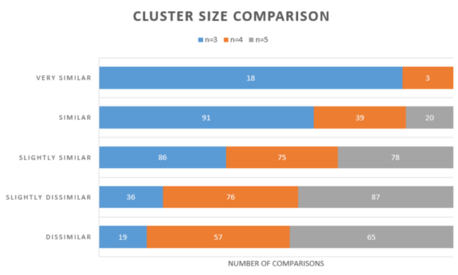


Figure 6.14: Cluster size comparison

6.2.2 Word Embeddings

Measurements

Change2Vec. Since we now have the JDT element behind the label, we use the average character based cosine similarity to check the similarity within the 3 most similar changes of a change.

total elements	min	max	mean	variance
895	0.37	0.972	0.82	0.0087

Mean of 82% and a very low variance shows us the effectiveness of word2vec. To be recalled again, the trained word2vec model has never seen the JDT Elements, only numeric labels.

Clustering. By using the Elbow method we can find the appropriate number of clusters within the dataset. The table below shows the error rate on the y axis and the number of clusters on the x axis. In other words, one should choose a number of clusters in order that adding another cluster doesn't give much better (less delta of error rate than before) modelling results of the data.

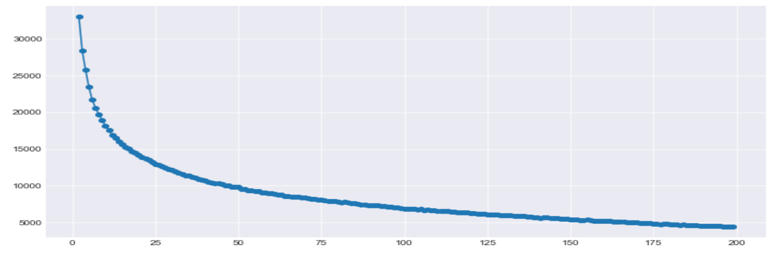


Figure 6.15: Word Embeddings - Elbow Method

The graphic shows that the error rate reduction rate starts to decrease (inflection point) from 30 clusters to 60. With help of the silhouette method, it is possible to find the right cluster. By definition the silhouette value is a measure of how similar an item (change) is to its own cluster (cohesion) compared to other clusters (separation).

number of clusters	Silhouette Coefficient
36	0.1622
37	0.1595
38	0.1561
39	0.1543
40	0.1591
41	0.1624
42	0.1604
43	0.1595
44	0.1595
45	0.1606
46	0.1634
47	0.1556
48	0.1620
49	0.1594
50	0.1539
51	0.1606
52	0.1604
53	0.1622
54	0.1626
55	0.1744
56	0.1605
57	0.1566
58	0.1629
59	0.1601

We select 55 as our cluster number, because it by far the highest silhouette coefficient in the considered frame.

Survey

All five survey participants went through 55 clusters. For each cluster five randomly selected subsets of changes have been presented. Participants had to select if the changes within the subset are similar (1) or not (0) similar. We define 0-20% as dissimilar, 20-40% as slightly dissimilar, 40-60% as slightly similar, 60-80% similar and 80%-100% as very similar.

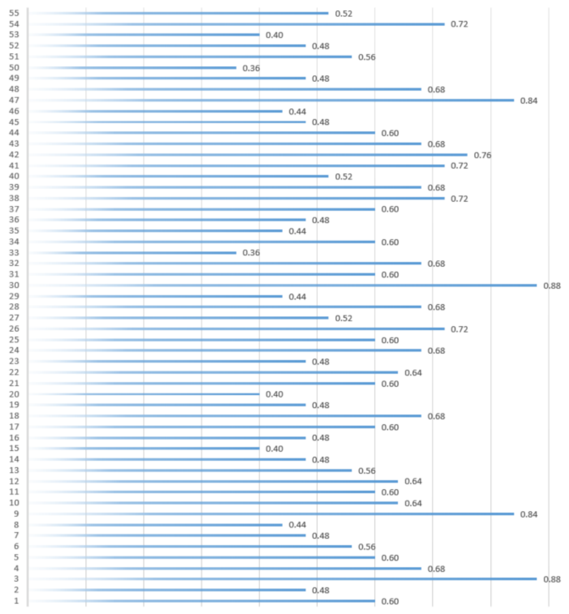


Figure 6.16: Word Embeddings Survey Result 1

54% of all clusters are categorized as similar and very similar, whereas 0 clusters have been classified as dissimilar. These results shows that our approach to find change clusters works very well.

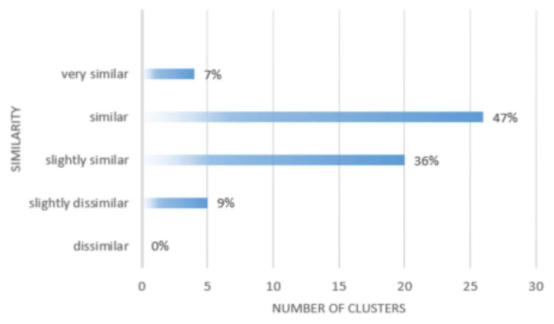


Figure 6.17: Word Embeddings Survey Result 2

6.2.3 Topic Modelling

Measurements

In order to find a appropriate number of topics there are two metrics to consider. Firstly, we use perplexity Score, which is used by convention in language modelling and is monotonically decreasing in the likelihood of the test data. And is algebraical equivalent to the inverse of the geometric average per-word likelihood. A lower perplexity score indicates better generalization performance. Our second metric, namely the coherence score which indicates how meaningful and interpretable the topics are. An indication for a too high number of topic is when some keywords occur in multiple topics.

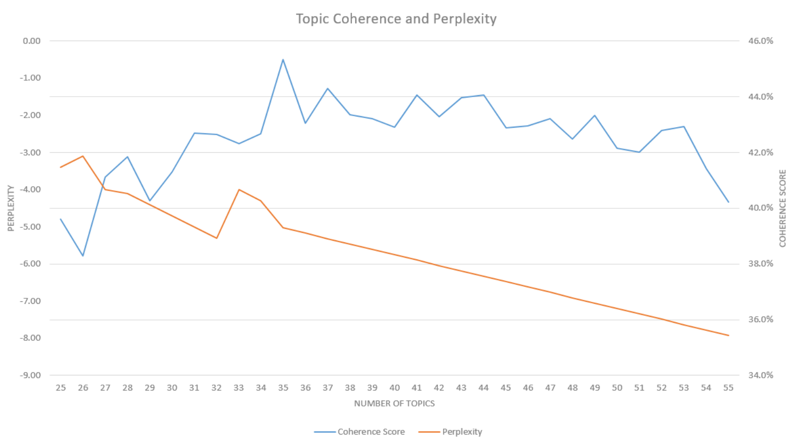


Figure 6.18: Topic Cohereson and Perplexity

Survey

To find out if a trained topic is good, we make use of the word intrusion evaluation method. Chang et al. show how it outperforms traditional topics modelling evaluation methods [**?**]. Word intrusion works the following way: for each topic, three real tags of the topic are shown and one randomly selected tag is added. A human (in our case study participant) then has to show which one it is. Additionally we asked the participant to describe the topic optionally.

The figure below shows the word intrusion method:

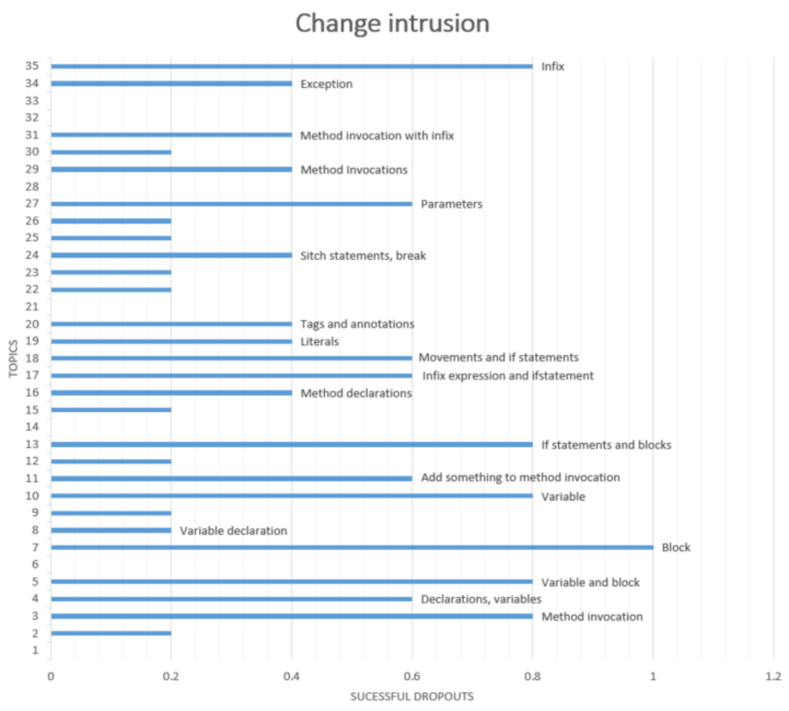


Figure 6.19: Topic Modeling - Change Intrusion

Interestingly, topics where the random tag was found have been most likely described. This proves the self-expressiveness of the topic.

6.2.4 Doc2vec

Survey

In our case, every doc is a diff and is thus indexed with its id. In order to validate how well the trained model performs, we presented 20 randomly selected diffs and asked the software developers if the most similar diff is similar to the presented diff (1) or not (0). We explicitly showed only similar diffs with over 90% similarity score, to ensure that only very highly similar documents are shown.

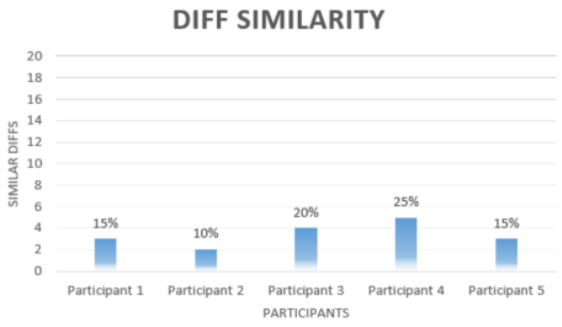


Figure 6.20: Doc2Vec Similarity Survey Result

The results clearly shows that doc2vec does not perform well for source code changes. Only 17% of all presented similar diffs are actually similar. A reason could be the small vocabulary (895 changes) and small size of diffs.

6.3 Threats to Validity

Our data quality relies on the accuracy of underlying AST Diff tool GumTree. Indeed when we explored source code changes with our web tool, we've also seen mistakes (e.g. new import statement was added, but instead a movement was stated).

According to the presented distribution, a dataset consisting out of 500 project, could be not sufficiently significant, since changes that do not occur often are difficult to detect.

In order to prevent local optimization on changes on identifiers (e.g. getProject –> getProjects) we forked GumTree and replaced all updates on identifiers with UPDATE, that also explains the frequent appearances of Update. However, that does not mean that updates on identifier are unimportant, on the contrary, the analysis will be even better by including them. This thesis starts with a profound analysis on syntactical changes rather than semantic interpretations. In a next step, the results obtained could be used as the basis for a semantic analysis.

Due to resource limitations we did not select equally skilled participants for the survey, which may influence the results (e.g. senior software developer may see connections,while a junior doesn't. In addition five participants may not be enough for significant results. We actually wanted to use active learning for evaluation, but for time reasons we didn't get the chance to do so.

Abstract syntax tree diff-calculations need a lot of computer power, so we couldn't mine any more data. The results had been be more expressive if we would have mined more than 1000 projects.

Chapter 7

Future Work

This section presents different solutions which can be build upon the data analysis of this thesis.

7.1 Tools

7.1.1 EvoSearch - Maintenance Search Tool

Searching through repositories is already possible in the largest hosting providers such as GitHub, BitBucket, GitLab. It enables exploring projects and also finding specific or similar ones using the wide range of filter options. Software maintenance and evolution is a very important topic in software engineering. Unfortunately this aspect is not covered in search engines.

A possible approach could be to add a pre-commit hook which labels the changed files with changes. This basic prototyp would already enable search for maintenance. Adding similar functionality to the default search function such as stars on a diff, number of comments on a diff, and impact level (as implemented in ChangeDistiller) would enrich the search functionality. There are millions of open source projects on github which are watched by prospective software developers. A Search functionality would help beginners to understand software engineering better by concatenating subsequent diffs with their labels (e.g. end-to-end implementation of a feature)

Advantages of a maintenance and evolution search function could be:

1. search for common re factorings after a breaking change
2. search for bug fixes
3. search for similar evolution paths
4. find important and notable diffs (according to stars , comments and impact level)
5. support beginners with labelled diffs

7.1.2 InstaChange - Subscription System to support maintenance of open Source projects

Github offers the functionality to watch repositories and receive notifications for new pull requests and issues that are created and other activities in a repository or organization. Since diffs are represented only in plain text, it is impossible to follow specific changes within a file, folder or repository. InstaChange could extend the subscription functionality and increase the user experience by providing change subscription or label subscriptions on source code.

With the provided dataset it is possible to subscribe to specific changes (e.g when a method has been added or when an import statement has been removed) but also semantic labelled diffs on top of changes can be subscribed.

Advantages of an enhanced change subscription system are:

1. follow specific changes on files and repositories
2. improve user experience in subscription menu, by showing only relevant changes
3. support owners of modules in open source projects understand scope and impact of pull request according to their subscriptions (e.g. subscription to a very important core functionality)
4. support open source projects by connecting specific changes to the code owner

7.1.3 DeepES - Learn Edit Scripts with Supervised Learning

The applications above would need AST Differencing for every diff on a file, which is very costly from an computing perspective. A solution to speed up this process would be to learn the edit scripts with deep learning an predict the changes. The provided data corpus can be used to train the model. Text would be the line or word based diff and label would be the change index.

Diff	**Label**

`@@ -19,7 +19,7 @@ package com.androidquery.util;`
`public interface Constants {`
`public static final String VERSION = [-"0.23.6";-]{+"0.23.7";+}`
`public static final int LAYER_TYPE_SOFTWARE = 1;`
`public static final int LAYER_TYPE_HARDWARE = 2;`
	31

Listing 7.1: Insert code directly in your document

Table 7.1: DeepES - Example Tagging

Advantages of learning edit script with deep learning:

1. reduce computation time
2. combine model with other models (e.g model that identifies topic of the software source code)
3. remove AST Differencing Tool as dependency

7.1.4 Changer - Management Tool for better software life cycle reporting

Visualizations about the software development life cycle could help software development teams and product owners, understand better the characteristic, culture and frequency of their code. Github only provides a simple graph so far, which shows a time-line on how much the developers contributed to the source code. Timeline informations about implementation of new features, solving bug fixes, cleaning code or re-factoring are not shown. The idea behind Changer is to extend the current "Insights" of Github with more informations about software evolution and maintenance. In a first step a crowd would add (pre-defined) labels such as REFACTOR, NEW FEATURE, BUG FIX, CLEAN UP to the changes, until all existing changes are labeled. This information would already allow a more sophisticated contribution visualization

Advantages of more sophisticated software life cycle visualization:

1. understand the software development culture of your team (e.g. fast in building new features, but later on lot of re-factor)
2. know your developers better (e.g. some developers might produce more new features, whereas others do mostly re-factorings and others are lazier and do more clean-ups)
3. increase software quality by extrapolating through sophisticated software maintenance and evolution visualization

7.2 Tasks

7.2.1 Store data in a structured manner

Implementing Changelyzer without proper backend and database implementation was not an easy task. In retrospect, it would have saved us a lot of time in the long run.

Advantages of database:

1. facilitate development of new applications
2. reduce data redundancy
3. more advanced queries with SQL rather than high level libraries
4. faster data retrieval
5. single source of truth, especially when reproducing and sharing results
6. first of this kind

7.2.2 Mine and support different Languages and map similar changes

This thesis focused only on the JAVA programming language. GumTree supports in addition to Java also C, C++, C#, JavaScript, CSS, Matlab, PHP, Python, R, Ruby and XML which allows to extend the data analysis on other languages as well. With help of active learning, it could be possible to map changes over different languages to g̈eneralc̈hanges. Since different languages types and programming paradigms are covered by GumTree, it is also possible to do an empirical analysis about software maintenance over different languages. The dataset could be integrated in the EvoSearch search functionality. Advantages of a mining and mapping changes of different programming languages:

1. create taxonomy of language independent software changes
2. compare maintenance and evolution frequency and other metrics across languages
3. understand consequences of impact levels on different languages
4. open source diff corpus to other maintenance and evolution researchers

7.2.3 Semantic ChangeLabeling

Audris Mockus et al. [25] identified three primary reasons for change: adding new feature (adaptive), fixing faults (corrective) and restructuring code to accommodate future changes (perfective). One could use this labels and add map them to changes provided within this thesis. It is also possible to generate more sophisticated labels.

7.2.4 Add change significance level to new changes

Unfortunately our changes do not have a change significance level as ChangeDistiller provides. According to their definition it expresses the possible impact a change type may have on other source code entities and whether it may be altering the functionality. Change significance levels are used to measure the relevance of each particular source code change.

7.2.5 Support Commit Message Generation

Commit messages help speed up the reviewing process, write good release notes and improve maintainability. Adding labels to the changes we discovered empirically could increase the commit message quality, by automatically tagging commit messages with it. Example:

Template	**Example**
(LABEL) - Commit Header	(NEW FEATURE) - Implementation of search bar
- Commit message	- SERVICE ADDED : SearchBarService
	- METHOD ADDED: filterResults()
Commit Log	- METHOD CHANGED: search(String keyWord)
	- SWITCH STATEMENT EXTENDED: filterCriteria

Chapter 8

Conclusions

Descriptive Statistic. Firstly we presented descriptive statistics in order to understand the data, starting from describing how JDT elements are used, distribution of tree operations in JAVA programming language, furthermore showing 20 most used changes, which make up 45% of all 895 changes. We also discussed briefly top 10 changes for all tree operations, last but not least we briefly described more than 70 common diffs (n=2), thus it is possible to find meaningful source code changes with help of big data. Using this simple data analytic method allowed us already to gain a lot of informations regarding source code changes.

Clustering. Secondly we used machine learning to cluster similar diffs of different sizes (starting from three changes to 5). We came to the conclusion that clustering works well for small diffs, but the larger the diff size gets, the meaningless the purpose gets. We evaluated that by onducting a survey with software developer and also by measuring heuristics for cluster coherence such as silhoutte score and elbow method. A possible reason for the unreasonable purpose for cluster with larger diffs could be that Kmeans only checks the cosine similarity, rather than the underlying structures.

Word Embeddings. Thirdly we applied word embeddings on source code changes in order to get similarities of changes. Our trained change2vec model performed very well and was able to find very similar changes, our survey results prove this. Since the results were very well, we created a similarity matrix, where we used to apply K-Mean clustering on top of it. With measurements we found a justified number of clusters. In a second study we presented the clusters and asked the participants how coherent the clusters are. We finally presented the 12 most coherent clusters. Therefore we successfully answered RQ2, actually it is possible to find similar source code changes with help of neural networks. Remarkably, our trained word2vec model only saw numeric encodings (e.g 312 = INSERT ELEMENT METHODINVOCATION).

Document Embeddings. Motivated by word embeddings, we tried to use the same technique for diffs by using word2vecs extension doc2vec. Unfortunately this method did not work out. Potential reasong could be that our diffs are very rather small compared to texts like for example news. Another reason is our limited vocabulary of 895 changes and its dense distribution.

Topic Modeling. Moreover we applied topic modelling on top of diffs, where we utilized Latent Dirichlet allocation. In a study we presented the topics and asked the participants to label or describe a topic in few words, but only if the topic is self-descriptive. 17 out of 35 Topics have been labeled. By inspecting the labelled topics, we figured out that their distribution is more uniformly distributed across the labels.

DCFSG. After examining al sorts of approaches, we noticed that none of these approaches could find groups of changes within longer diffs. Therefore we designed an algorithm, which tries to split diffs in change groups by scoring different possible combinations. Finally we tested the algorithm by applying word embeddings on the output. In conclusion we were able to identify similar meaningful change groups, which answers our last RQ.

Proposal for Tools and Tasks. Finally we presented four different maintenance and evolution tools, which could help software developers in their daily life.Firstly EVOSearch, a maintenance search tool, which enables software developers finding and exploring source code changes within repository hosting providers. Second tool, namly INSTAChange aims to offer a better notifications system for changes within a repository. At the moment users will be informed for every little issue, excluding source code changes. Such a subscription system would simplify the management of pull requests within open source projects. Lastly we present CHANGER, which objective is to reduce information asymmetry between between product owners and software developers, by providing them structured data about product evolution, team activities and also informations about individual team members (e.g. how many features have been implemented, how much time on refactoring, cleaning up code etc.).

Reproducibility Data, Scripts, Survey Results and everything related to this thesis can be found in the following Github Repository: SaliZumberi/master-thesis

Bibliography

[1] T. Ball, J.-m. Kim, A. Porter, and H. Siy. If your version control system could talk ... 10 1997.

[2] L. A. Belady and M. M. Lehman. A model of large program development. *IBM Syst. J.*, 15(3):225–252, Sept. 1976.

[3] P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, June 2005.

[4] N. Chapin, J. E. Hale, K. M. Kham, J. F. Ramil, and W.-G. Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance*, 13(1):3–30, Jan. 2001.

[5] B. W. Chatters, M. M. Lehman, J. F. Ramil, and P. Wernick. Modelling a software evolution process: a long-term case study. *Software Process: Improvement and Practice*, 5(2-3):91–102.

[6] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall. An empirical analysis of the docker container ecosystem on github. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 323–333, May 2017.

[7] A. Dearle. Software deployment, past, present and future. In *2007 Future of Software Engineering*, FOSE '07, pages 269–284, Washington, DC, USA, 2007. IEEE Computer Society.

[8] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Mach. Learn.*, 42(1-2):143–175, Jan. 2001.

[9] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus. Fine-grained and accurate source code differencing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 313–324, New York, NY, USA, 2014. ACM.

[10] B. Fluri and H. C. Classifying change types for qualifying change couplings. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 35–45, June 2006.

[11] B. Fluri, M. Wuersch, M. PInzger, and H. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Trans. Softw. Eng.*, 33(11):725–743, Nov. 2007.

[12] H. C. Gall, B. Fluri, and M. Pinzger. Change analysis with evolizer and changedistiller. *IEEE Software*, 26(1):26–33, Jan 2009.

[13] H. C. Gall, B. Fluri, and M. Pinzger. comparing-fine-grained-source. *IEEE Software*, 26(1):26–33, Jan 2009.

[14] E. Giger, M. D'Ambros, M. Pinzger, and H. C. Gall. Method-level bug prediction. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pages 171–180, New York, NY, USA, 2012. ACM.

[15] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning – data mining, inference, and prediction.

[16] I. Herraiz, D. Rodriguez, G. Robles, and J. M. Gonzalez-Barahona. The evolution of the laws of software evolution: A discussion based on a systematic literature review. *ACM Comput. Surv.*, 46(2):28:1–28:28, Dec. 2013.

[17] H. Jelodar, Y. Wang, C. Yuan, and X. Feng. Latent dirichlet allocation (LDA) and topic modeling: models, applications, a survey. *CoRR*, abs/1711.04305, 2017.

[18] S. Kim, K. Pan, and E. E. J. Whitehead, Jr. Memories of bug fixes. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, pages 35–45, New York, NY, USA, 2006. ACM.

[19] T. Kurashima, T. Iwata, T. Hoshide, N. Takaya, and K. Fujimura. Geo topic model: Joint modeling of user's activity area and interests for location recommendation. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 375–384, New York, NY, USA, 2013. ACM.

[20] M. M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, Sept 1980.

[21] M. M. Lehman and L. A. Belady, editors. *Program Evolution: Processes of Software Change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[22] B. Lientz, E. Burton Swanson, and G. E. Tompkins. Characteristics of application software maintenance. 21:466–471, 06 1978.

[23] Y. Lin, J. Jiang, and S. Lee. A similarity measure for text classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 26(7):1575–1590, July 2014.

[24] T. Mikolov, W. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751, 2013.

[25] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, ICSM '00, pages 120–, Washington, DC, USA, 2000. IEEE Computer Society.

[26] S. Na, L. Xumin, and G. Yong. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, pages 63–67, April 2010.

[27] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 70–79, New York, NY, USA, 2012. ACM.

[28] N. Palix, J. Falleri, and J. Lawall. Improving pattern tracking with a language-aware tree differencing algorithm. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 43–52, March 2015.

[29] M. Pawlik and N. Augsten. Rted: A robust algorithm for the tree edit distance. *Proc. VLDB Endow.*, 5(4):334–345, Dec. 2011.

[30] V. Rajlich. Software evolution and maintenance. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 133–144, New York, NY, USA, 2014. ACM.

[31] G. Schermann, S. Zumberi, and J. Cito. Structured information on state and evolution of dockerfiles on github. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 26–29, New York, NY, USA, 2018. ACM.

[32] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.

[33] Y. Sun, Q. Wang, and Y. Yang. Frlink. *Inf. Softw. Technol.*, 84(C):33–47, Apr. 2017.

[34] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[35] W. Zhao, J. J. Chen, R. Perkins, Y. Wang, Z. Liu, H. Hong, W. Tong, and W. Zou. A novel procedure on next generation sequencing data analysis using text mining algorithm. *BMC Bioinformatics*, 17(1):213, May 2016.

[36] W. X. Zhao, J. Jiang, J. Weng, J. He, E.-P. Lim, H. Yan, and X. Li. Comparing twitter and traditional media using topic models. In *Proceedings of the 33rd European Conference on Advances in Information Retrieval*, ECIR'11, pages 338–349, Berlin, Heidelberg, 2011. Springer-Verlag.

[37] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society.

Appendix A

Appendix

A.1 Source code snippets

```
CORES=32 # or 16, or whatever
MAINFOLDER=$(pwd)
function get_all_diffs(){
#echo "3. Start getting all diffs for repo: $1"
rm −rf ../projects/$1
awk ’{gsub(/\"/,"")};1’ "../$1.tmp" > "../$1_no_quotes"
tr ’ \t’ ’\n’ <"../$1_no_quotes" >"../$1.ready"
shaArray=($(<../$1.ready))
arraylength=${#shaArray[@]}

#echo "3.1 This file contains $arraylength commit sha’s "
projectId=$1
fileNameTemp="$projectId"
for i in "${!shaArray[@]}"; do
get_diff "${shaArray[$i]}" "${shaArray[$i+1]}" $fileNameTemp $i
done

mkdir −p ../projects/$1
cat ../$fileNameTemp | grep ’\S’ >> ../diffs
mv ../$fileNameTemp ../projects/$1
mv ../$1.ready ../projects/$1
mv ../$1.tmp ../projects/$1
mv ../$1_no_quotes ../projects/$1
}

function clone_repository(){
#echo "2. Cloning Project.."
projectId=$1
url="https://api.github.com/repositories/"
gitCloneUrl=$(curl −−user "juice_457@hotmail.com:Zim−Beri1" $url$projectId | jq ’.clone_url’ | xargs)
clone=‘git clone $gitCloneUrl $projectId‘
}

function delete_repository(){
projectId= "$MAINFOLDER/$1/"
rm −rf $projectId
}
```

```
function get_diff(){
commitAfter=$1
commitBefore=$2
fileNameTemp=$3
index=$4
#echo "4. Get Diff for Commit: $index"

revision="$commitAfter^!"
local IFS=$'\n'
local changed_files=(‘git diff −−word−diff=plain $commitBefore $commitAfter −−name−only‘)
local i
echo "This diff has ${#changed_files[@]} Changed Files!"
if (( ${#changed_files[@]} < 100 )); then
echo "Less than 100 changed Files"
for (( i=0; i<${#changed_files[@]}; i++ )) ; do
pathToFile=${changed_files[$i]}
if [ ${pathToFile: −5} == ".java" ]; then
# echo "$fileNameTemp − $i: ${changed_files[$i]}"
#git show "$commitBefore:${changed_files[$i]}" > "../$fileNameTemp/A_$i.java"
#git show "$commitAfter:${changed_files[$i]}" > "../$fileNameTemp/B_$i.java"
diff_output=‘git diff −−word−diff=plain $revision ${changed_files[$i]}‘
#diff_output=‘echo $diff_output | tr '\n' ' '‘
#diff_output=‘echo $diff_output | grep −oP '(\[−(.*?)\−]{+(.*?)\+})'‘
diff_output_B=‘git show "$commitAfter:${changed_files[$i]}"‘
diff_output_A=‘git show "$commitBefore:${changed_files[$i]}"‘

nameOfFileB=$fileNameTemp'_'$commitAfter'_'$i'_'"B.java"
pathToFileB="$MAINFOLDER/projects/"$nameOfFileB
nameOfFileA=$fileNameTemp'_'$commitAfter'_'$i'_'"A.java"
pathToFileA="$MAINFOLDER/projects/"$nameOfFileA

minimumsize=80
diff_output_B_size=${#diff_output_B}
diff_output_A_size=${#diff_output_A}
if ([ $diff_output_B_size −ge $minimumsize ] && [ $diff_output_A_size −ge $minimumsize ]); then
#echo size is over $minimumsize bytes
echo "$diff_output_B" > $pathToFileB
echo "$diff_output_A" > $pathToFileA
echo "RUN DOCKER"
diff_output_cluster=‘docker run −−rm −v $MAINFOLDER/projects:/diff gumtree cluster $nameOfFileA $nameOfFileB‘
diff_output_diff=‘docker run −−rm −v $MAINFOLDER/projects:/diff gumtree diff $nameOfFileA $nameOfFileB‘

#echo "$diff_output_diff"

number_of_matches=‘echo $diff_output_diff | grep −o '\<Match\>' | wc −l‘
number_of_inserts=‘echo $diff_output_diff | grep −o '\<Insert\>' | wc −l‘
number_of_deletes=‘echo $diff_output_diff | grep −o '\<Delete\>' | wc −l‘
number_of_updates=‘echo $diff_output_diff | grep −o '\<Update\>' | wc −l‘
number_of_moves=‘echo $diff_output_diff | grep −o '\<Move\>' | wc −l‘
number_total=‘echo "$diff_output_diff" | wc −l‘
number_diffs=$(($number_total − $number_of_matches))
diff_stat=$number_diffs'_'$number_of_inserts'_'$number_of_deletes'_'$number_of_updates'_'$number_of_moves

number_cluster_inserts=‘echo $diff_output_cluster | grep −o '\<INSERT\>' | wc −l‘
number_cluster_deletes=‘echo $diff_output_cluster | grep −o ’\<DELETE\>’ | wc −l‘
number_cluster_updates=‘echo $diff_output_cluster | grep −o ’\<UPDATE\>’ | wc −l‘
number_cluster_moves=‘echo $diff_output_cluster | grep −o ’\<MOVE\>’ | wc −l‘
number_cluster=$(($number_cluster_inserts + $number_cluster_deletes + $number_cluster_updates + $number_cluster_moves))
cluster_stat=$number_cluster’_’$number_cluster_inserts’_’$number_cluster_deletes’_’ \
$number_cluster_updates’_’$number_cluster_moves

#echo "Number of matches: $number_of_matches"
#echo "Number of Total: $number_total"
#echo "Number of diffs: $number_diffs"

pathToCluster="$MAINFOLDER/projects/"$fileNameTemp’_’$commitAfter’_’$i’_’"cluster"’_’$cluster_stat
pathToDiff="$MAINFOLDER/projects/"$fileNameTemp’_’$commitAfter’_’$i’_’"diff"’_’$diff_stat
pathToPlainDiff="$MAINFOLDER/projects/"$fileNameTemp’_’$commitAfter’_’$i’_’"plain"’_’"diff"

echo "$diff_output" > $pathToPlainDiff
echo "$diff_output_cluster" > $pathToCluster
echo "$diff_output_diff" > $pathToDiff
sed −i ’/^Match/d’ $pathToDiff
else
echo size is under $minimumsize bytes
fi
fi
done
else
echo "Too many files!"
fi
}

function start_diff_mining(){
cd $MAINFOLDER
file_name=$1
if [[ $file_name != [0−9]∗ ]]; then
echo $file_name is not a repo Id
else
repo_id_with_ending=$file_name
repo_id=${repo_id_with_ending%.∗}
echo "1. Start mining for repo id: $repo_id"
clone_repository $repo_id
cd $repo_id
get_all_diffs $repo_id
cd ..
echo "1. Finished mining for repo id: $repo_id"
#delete_repository $repo_id
#echo "Finished Project Number: $i: ${files[$i]}"
fi
}

files=(∗)
echo "Found ${#files[@]} files in this Folder"
for file_with_commits in "${files[@]}" ; do
start_mining $file_with_commits &
background=( $(jobs −p) )
if (( ${#background[@]} == $CORES )); then
sleep 5s
```

```
echo "Waiting until $CORES"
wait −n
fi
done
wait
echo "All done"
```

Listing A.1: bash version

A.2 Changes

Id	Change	Count	%
619	DELETE AnnotationTypeDeclaration	136	0.0000227
664	DELETE AnnotationTypeMemberDeclaration	317	0.0000528
138	DELETE AnonymousClassDeclaration	756	0.0001260
577	DELETE ArrayAccess	4121	0.0006866
532	DELETE ArrayCreation	2290	0.0003815
91	DELETE ArrayInitializer	493	0.0000821
394	DELETE ArrayType	3304	0.0005505
402	DELETE AssertStatement	1603	0.0002671
838	DELETE Assignment	9595	0.0015986
208	DELETE Block	51642	0.0086039
456	DELETE BooleanLiteral	12269	0.0020441
646	DELETE BreakStatement	3211	0.0005350
478	DELETE CastExpression	9062	0.0015098
533	DELETE CatchClause	1617	0.0002694
543	DELETE CharacterLiteral	1211	0.0002018
445	DELETE ClassInstanceCreation	27493	0.0045805
804	DELETE ConditionalExpression	3501	0.0005833
185	DELETE ConstructorInvocation	450	0.0000750
20	DELETE ContinueStatement	453	0.0000755
152	DELETE CreationReference	78	0.0000130
307	DELETE Dimension	344	0.0000573
431	DELETE DoStatement	125	0.0000208
845	DELETE EmptyStatement	505	0.0000841
401	DELETE EnhancedForStatement	4310	0.0007181
622	DELETE EnumConstantDeclaration	2590	0.0004315
469	DELETE EnumDeclaration	725	0.0001208
418	DELETE ExpressionMethodReference	828	0.0001380
503	DELETE ExpressionStatement	141090	0.0235065
258	DELETE FieldAccess	7294	0.0012152
0	DELETE FieldDeclaration	61545	0.0102538
306	DELETE ForStatement	2508	0.0004178
480	DELETE IfStatement	49080	0.0081771
625	DELETE ImportDeclaration	181208	0.0301905
472	DELETE InfixExpression	33779	0.0056278
131	DELETE Initializer	808	0.0001346
578	DELETE InstanceofExpression	914	0.0001523
379	DELETE Javadoc	11390	0.0018976
82	DELETE LabeledStatement	41	0.0000068
197	DELETE LambdaExpression	2572	0.0004285
141	DELETE MarkerAnnotation	15628	0.0026037
403	DELETE MemberRef	294	0.0000490
704	DELETE MemberValuePair	1101	0.0001834
835	DELETE MethodDeclaration	147292	0.0245398
591	DELETE MethodInvocation	127788	0.0212903
606	DELETE MethodRef	353	0.0000588
753	DELETE MethodRefParameter	259	0.0000432
721	DELETE Modifier	36764	0.0061251
771	DELETE NormalAnnotation	3752	0.0006251
685	DELETE NullLiteral	12978	0.0021622

617	DELETE NumberLiteral	21190	0.0035304
385	DELETE PackageDeclaration	103	0.0000172
454	DELETE ParameterizedType	18524	0.0030862
144	DELETE ParenthesizedExpression	5006	0.0008340
748	DELETE PostfixExpression	625	0.0001041
412	DELETE PrefixExpression	5294	0.0008820
837	DELETE PrimitiveType	15931	0.0026542
387	DELETE QualifiedName	45309	0.0075488
50	DELETE QualifiedType	5	0.0000008
234	DELETE ReturnStatement	18008	0.0030003
634	DELETE SimpleName	173312	0.0288749
190	DELETE SimpleType	49048	0.0081717
640	DELETE SingleMemberAnnotation	6232	0.0010383
73	DELETE SingleVariableDeclaration	37192	0.0061964
827	DELETE StringLiteral	46326	0.0077182
671	DELETE SuperConstructorInvocation	1396	0.0002326
677	DELETE SuperFieldAccess	16	0.0000027
223	DELETE SuperMethodInvocation	1149	0.0001914
715	DELETE SwitchCase	3216	0.0005358
299	DELETE SwitchStatement	1012	0.0001686
499	DELETE SynchronizedStatement	715	0.0001191
341	DELETE TagElement	20149	0.0033570
548	DELETE TextElement	25237	0.0042047
117	DELETE ThisExpression	4516	0.0007524
481	DELETE ThrowStatement	2245	0.0003740
22	DELETE TryStatement	4724	0.0007870
67	DELETE TypeDeclaration	8212	0.0013682
879	DELETE TypeDeclarationStatement	40	0.0000067
739	DELETE TypeLiteral	3999	0.0006663
42	DELETE TypeMethodReference	1	0.0000002
604	DELETE TypeParameter	1894	0.0003156
382	DELETE UnionType	209	0.0000348
344	DELETE VariableDeclarationExpression	456	0.0000760
630	DELETE VariableDeclarationFragment	3402	0.0005668
308	DELETE VariableDeclarationStatement	70249	0.0117040
710	DELETE WhileStatement	1265	0.0002108
72	DELETE WildcardType	1656	0.0002759
815	INSERT AnnotationTypeDeclaration AnnotationTypeDeclaration	65	0.0000108
205	INSERT AnnotationTypeDeclaration CompilationUnit	7	0.0000012
801	INSERT AnnotationTypeDeclaration TypeDeclaration	167	0.0000278
557	INSERT AnnotationTypeMemberDeclaration AnnotationTypeDeclar	735	0.0001225
650	INSERT AnonymousClassDeclaration ClassInstanceCreation	503	0.0000838
890	INSERT AnonymousClassDeclaration EnumConstantDeclaration	79	0.0000132
204	INSERT ArrayAccess ArrayAccess	30	0.0000050
489	INSERT ArrayAccess ArrayInitializer	12	0.0000020
541	INSERT ArrayAccess Assignment	756	0.0001260
18	INSERT ArrayAccess CastExpression	45	0.0000075
346	INSERT ArrayAccess ClassInstanceCreation	143	0.0000238
484	INSERT ArrayAccess ConditionalExpression	4	0.0000007
564	INSERT ArrayAccess FieldAccess	1	0.0000002

781	INSERT ArrayAccess IfStatement	9	0.0000015
491	INSERT ArrayAccess InfixExpression	337	0.0000561
756	INSERT ArrayAccess InstanceofExpression	12	0.0000020
99	INSERT ArrayAccess LambdaExpression	1	0.0000002
378	INSERT ArrayAccess MethodInvocation	1405	0.0002341
353	INSERT ArrayAccess PrefixExpression	8	0.0000013
109	INSERT ArrayAccess ReturnStatement	101	0.0000168
276	INSERT ArrayAccess SuperMethodInvocation	1	0.0000002
148	INSERT ArrayAccess SwitchStatement	1	0.0000002
866	INSERT ArrayAccess VariableDeclarationFragment	325	0.0000541
47	INSERT ArrayCreation ArrayInitializer	19	0.0000032
691	INSERT ArrayCreation Assignment	176	0.0000293
246	INSERT ArrayCreation ClassInstanceCreation	78	0.0000130
158	INSERT ArrayCreation ConditionalExpression	2	0.0000003
689	INSERT ArrayCreation ConstructorInvocation	5	0.0000008
280	INSERT ArrayCreation EnhancedForStatement	6	0.0000010
368	INSERT ArrayCreation EnumConstantDeclaration	2	0.0000003
329	INSERT ArrayCreation LambdaExpression	8	0.0000013
777	INSERT ArrayCreation MethodInvocation	809	0.0001348
187	INSERT ArrayCreation ReturnStatement	66	0.0000110
736	INSERT ArrayCreation SuperConstructorInvocation	2	0.0000003
181	INSERT ArrayCreation SuperMethodInvocation	1	0.0000002
832	INSERT ArrayCreation VariableDeclarationFragment	495	0.0000825
107	INSERT ArrayInitializer AnnotationTypeMemberDeclaration	47	0.0000078
530	INSERT ArrayInitializer ArrayCreation	70	0.0000117
479	INSERT ArrayInitializer ArrayInitializer	219	0.0000365
523	INSERT ArrayInitializer MemberValuePair	91	0.0000152
51	INSERT ArrayInitializer SingleMemberAnnotation	275	0.0000458
765	INSERT ArrayInitializer VariableDeclarationFragment	130	0.0000217
52	INSERT ArrayType AnnotationTypeMemberDeclaration	60	0.0000100
784	INSERT ArrayType ArrayCreation	118	0.0000197
35	INSERT ArrayType CastExpression	13	0.0000022
725	INSERT ArrayType FieldDeclaration	516	0.0000860
432	INSERT ArrayType InstanceofExpression	1	0.0000002
731	INSERT ArrayType MethodDeclaration	545	0.0000908
251	INSERT ArrayType MethodInvocation	2	0.0000003
97	INSERT ArrayType MethodRefParameter	9	0.0000015
676	INSERT ArrayType ParameterizedType	206	0.0000343
364	INSERT ArrayType SingleVariableDeclaration	630	0.0001050
189	INSERT ArrayType TypeLiteral	13	0.0000022
96	INSERT ArrayType VariableDeclarationStatement	1053	0.0001754
118	INSERT AssertStatement Block	1918	0.0003196
150	INSERT AssertStatement IfStatement	3	0.0000005
699	INSERT AssertStatement SwitchStatement	21	0.0000035
123	INSERT Assignment Assignment	47	0.0000078
714	INSERT Assignment ClassInstanceCreation	6	0.0000010
262	INSERT Assignment ExpressionStatement	8693	0.0014483
839	INSERT Assignment ForStatement	64	0.0000107
586	INSERT Assignment IfStatement	13	0.0000022
264	INSERT Assignment LambdaExpression	1	0.0000002

314	INSERT Assignment MethodInvocation	91	0.0000152
398	INSERT Assignment ParenthesizedExpression	4	0.0000007
795	INSERT Assignment ReturnStatement	8	0.0000013
350	INSERT Assignment SwitchStatement	1	0.0000002
623	INSERT Assignment VariableDeclarationFragment	12	0.0000020
450	INSERT Assignment WhileStatement	1	0.0000002
132	INSERT Block Block	339	0.0000565
143	INSERT Block CatchClause	712	0.0001186
75	INSERT Block DoStatement	20	0.0000033
844	INSERT Block EnhancedForStatement	2200	0.0003665
182	INSERT Block ForStatement	834	0.0001389
79	INSERT Block IfStatement	19902	0.0033158
515	INSERT Block Initializer	87	0.0000145
94	INSERT Block LambdaExpression	654	0.0001090
105	INSERT Block MethodDeclaration	33304	0.0055487
751	INSERT Block SwitchStatement	396	0.0000660
551	INSERT Block SynchronizedStatement	221	0.0000368
437	INSERT Block TryStatement	3015	0.0005023
57	INSERT Block WhileStatement	400	0.0000666
800	INSERT BooleanLiteral AnnotationTypeMemberDeclaration	9	0.0000015
572	INSERT BooleanLiteral ArrayInitializer	50	0.0000083
475	INSERT BooleanLiteral AssertStatement	3	0.0000005
40	INSERT BooleanLiteral Assignment	503	0.0000838
139	INSERT BooleanLiteral ClassInstanceCreation	1771	0.0002951
497	INSERT BooleanLiteral ConditionalExpression	20	0.0000033
65	INSERT BooleanLiteral ConstructorInvocation	114	0.0000190
80	INSERT BooleanLiteral DoStatement	2	0.0000003
662	INSERT BooleanLiteral EnumConstantDeclaration	107	0.0000178
720	INSERT BooleanLiteral ForStatement	2	0.0000003
627	INSERT BooleanLiteral IfStatement	7	0.0000012
435	INSERT BooleanLiteral InfixExpression	78	0.0000130
71	INSERT BooleanLiteral LambdaExpression	1	0.0000002
583	INSERT BooleanLiteral MemberValuePair	40	0.0000067
237	INSERT BooleanLiteral MethodInvocation	9023	0.0015033
680	INSERT BooleanLiteral ReturnStatement	1345	0.0002241
19	INSERT BooleanLiteral SuperConstructorInvocation	98	0.0000163
874	INSERT BooleanLiteral SuperMethodInvocation	3	0.0000005
474	INSERT BooleanLiteral VariableDeclarationFragment	737	0.0001228
325	INSERT BooleanLiteral WhileStatement	64	0.0000107
647	INSERT BreakStatement Block	670	0.0001116
156	INSERT BreakStatement IfStatement	90	0.0000150
216	INSERT BreakStatement SwitchStatement	4114	0.0006854
89	INSERT CastExpression ArrayAccess	9	0.0000015
651	INSERT CastExpression ArrayCreation	6	0.0000010
388	INSERT CastExpression ArrayInitializer	13	0.0000022
884	INSERT CastExpression Assignment	1277	0.0002128
296	INSERT CastExpression CastExpression	10	0.0000017
851	INSERT CastExpression ClassInstanceCreation	265	0.0000442
498	INSERT CastExpression ConditionalExpression	31	0.0000052
588	INSERT CastExpression ConstructorInvocation	34	0.0000057

575	INSERT CastExpression EnhancedForStatement	20	0.0000033
538	INSERT CastExpression EnumConstantDeclaration	95	0.0000158
278	INSERT CastExpression IfStatement	1	0.0000002
774	INSERT CastExpression InfixExpression	323	0.0000538
63	INSERT CastExpression LambdaExpression	18	0.0000030
708	INSERT CastExpression MethodInvocation	2690	0.0004482
376	INSERT CastExpression ParenthesizedExpression	23	0.0000038
811	INSERT CastExpression PrefixExpression	10	0.0000017
536	INSERT CastExpression ReturnStatement	714	0.0001190
286	INSERT CastExpression SuperConstructorInvocation	10	0.0000017
524	INSERT CastExpression SuperMethodInvocation	10	0.0000017
93	INSERT CastExpression SwitchCase	8	0.0000013
233	INSERT CastExpression SwitchStatement	1	0.0000002
285	INSERT CastExpression ThrowStatement	6	0.0000010
184	INSERT CastExpression VariableDeclarationFragment	1929	0.0003214
462	INSERT CatchClause TryStatement	1770	0.0002949
506	INSERT CharacterLiteral ArrayInitializer	4	0.0000007
727	INSERT CharacterLiteral Assignment	113	0.0000188
686	INSERT CharacterLiteral CastExpression	36	0.0000060
455	INSERT CharacterLiteral ClassInstanceCreation	9	0.0000015
652	INSERT CharacterLiteral EnumConstantDeclaration	14	0.0000023
5	INSERT CharacterLiteral InfixExpression	414	0.0000690
561	INSERT CharacterLiteral MethodInvocation	771	0.0001285
743	INSERT CharacterLiteral SwitchCase	57	0.0000095
310	INSERT CharacterLiteral VariableDeclarationFragment	22	0.0000037
212	INSERT ClassInstanceCreation ArrayInitializer	200	0.0000333
267	INSERT ClassInstanceCreation Assignment	2617	0.0004360
357	INSERT ClassInstanceCreation CastExpression	3	0.0000005
168	INSERT ClassInstanceCreation ClassInstanceCreation	2354	0.0003922
290	INSERT ClassInstanceCreation ConditionalExpression	61	0.0000102
399	INSERT ClassInstanceCreation ConstructorInvocation	125	0.0000208
521	INSERT ClassInstanceCreation EnhancedForStatement	23	0.0000038
335	INSERT ClassInstanceCreation EnumConstantDeclaration	149	0.0000248
615	INSERT ClassInstanceCreation ExpressionMethodReference	1	0.0000002
842	INSERT ClassInstanceCreation ExpressionStatement	82	0.0000137
635	INSERT ClassInstanceCreation InfixExpression	10	0.0000017
100	INSERT ClassInstanceCreation LambdaExpression	19	0.0000032
600	INSERT ClassInstanceCreation MethodInvocation	11547	0.0019238
590	INSERT ClassInstanceCreation ReturnStatement	1915	0.0003191
806	INSERT ClassInstanceCreation SuperConstructorInvocation	91	0.0000152
323	INSERT ClassInstanceCreation SuperMethodInvocation	10	0.0000017
298	INSERT ClassInstanceCreation ThrowStatement	191	0.0000318
355	INSERT ClassInstanceCreation VariableDeclarationFragment	6410	0.0010679
259	INSERT ConditionalExpression ArrayAccess	4	0.0000007
250	INSERT ConditionalExpression ArrayCreation	6	0.0000010
694	INSERT ConditionalExpression ArrayInitializer	15	0.0000025
873	INSERT ConditionalExpression AssertStatement	2	0.0000003
766	INSERT ConditionalExpression Assignment	671	0.0001118
494	INSERT ConditionalExpression ClassInstanceCreation	295	0.0000491
883	INSERT ConditionalExpression ConditionalExpression	28	0.0000047

339	INSERT ConditionalExpression ConstructorInvocation	5	0.0000008
36	INSERT ConditionalExpression EnumConstantDeclaration	4	0.0000007
823	INSERT ConditionalExpression IfStatement	21	0.0000035
594	INSERT ConditionalExpression LambdaExpression	22	0.0000037
892	INSERT ConditionalExpression MethodInvocation	1802	0.0003002
566	INSERT ConditionalExpression ParenthesizedExpression	28	0.0000047
84	INSERT ConditionalExpression ReturnStatement	1026	0.0001709
542	INSERT ConditionalExpression SuperConstructorInvocation	15	0.0000025
550	INSERT ConditionalExpression SuperMethodInvocation	2	0.0000003
605	INSERT ConditionalExpression SwitchStatement	18	0.0000030
843	INSERT ConditionalExpression ThrowStatement	4	0.0000007
452	INSERT ConditionalExpression VariableDeclarationFragment	844	0.0001406
552	INSERT ConditionalExpression WhileStatement	2	0.0000003
446	INSERT ConstructorInvocation Block	629	0.0001048
692	INSERT ContinueStatement Block	208	0.0000347
28	INSERT ContinueStatement IfStatement	81	0.0000135
442	INSERT ContinueStatement SwitchStatement	2	0.0000003
64	INSERT CreationReference ClassInstanceCreation	58	0.0000097
228	INSERT CreationReference MethodInvocation	107	0.0000178
759	INSERT CreationReference ReturnStatement	3	0.0000005
490	INSERT CreationReference VariableDeclarationFragment	9	0.0000015
840	INSERT Dimension ArrayType	88	0.0000147
485	INSERT Dimension SingleVariableDeclaration	23	0.0000038
886	INSERT Dimension VariableDeclarationFragment	132	0.0000220
846	INSERT DoStatement Block	169	0.0000282
854	INSERT DoStatement IfStatement	2	0.0000003
21	INSERT EmptyStatement Block	331	0.0000551
570	INSERT EmptyStatement ForStatement	2	0.0000003
861	INSERT EmptyStatement IfStatement	1	0.0000002
453	INSERT EmptyStatement SwitchStatement	23	0.0000038
891	INSERT EmptyStatement WhileStatement	5	0.0000008
822	INSERT EnhancedForStatement Block	5823	0.0009702
722	INSERT EnhancedForStatement IfStatement	15	0.0000025
127	INSERT EnhancedForStatement SwitchStatement	62	0.0000103
8	INSERT EnumConstantDeclaration EnumDeclaration	4563	0.0007602
742	INSERT EnumDeclaration AnnotationTypeDeclaration	26	0.0000043
673	INSERT EnumDeclaration CompilationUnit	27	0.0000045
712	INSERT EnumDeclaration EnumDeclaration	1	0.0000002
295	INSERT EnumDeclaration TypeDeclaration	960	0.0001599
702	INSERT ExpressionMethodReference Assignment	6	0.0000010
876	INSERT ExpressionMethodReference CastExpression	1	0.0000002
13	INSERT ExpressionMethodReference ClassInstanceCreation	80	0.0000133
188	INSERT ExpressionMethodReference ConstructorInvocation	4	0.0000007
482	INSERT ExpressionMethodReference MethodInvocation	1219	0.0002031
439	INSERT ExpressionMethodReference ReturnStatement	12	0.0000020
49	INSERT ExpressionMethodReference SuperConstructorInvocation	1	0.0000002
633	INSERT ExpressionMethodReference VariableDeclarationFragment	16	0.0000027
881	INSERT ExpressionStatement Block	200732	0.0334433
361	INSERT ExpressionStatement DoStatement	1	0.0000002
803	INSERT ExpressionStatement EnhancedForStatement	67	0.0000112

669	INSERT ExpressionStatement ForStatement	22	0.0000037
711	INSERT ExpressionStatement IfStatement	1039	0.0001731
872	INSERT ExpressionStatement SwitchStatement	7083	0.0011801
281	INSERT ExpressionStatement WhileStatement	12	0.0000020
9	INSERT FieldAccess ArrayAccess	30	0.0000050
865	INSERT FieldAccess ArrayCreation	11	0.0000018
15	INSERT FieldAccess ArrayInitializer	4	0.0000007
373	INSERT FieldAccess Assignment	3054	0.0005088
327	INSERT FieldAccess CastExpression	21	0.0000035
62	INSERT FieldAccess ClassInstanceCreation	359	0.0000598
667	INSERT FieldAccess ConditionalExpression	17	0.0000028
130	INSERT FieldAccess EnhancedForStatement	45	0.0000075
629	INSERT FieldAccess ExpressionMethodReference	8	0.0000013
421	INSERT FieldAccess FieldAccess	85	0.0000142
377	INSERT FieldAccess IfStatement	45	0.0000075
632	INSERT FieldAccess InfixExpression	525	0.0000875
587	INSERT FieldAccess InstanceofExpression	6	0.0000010
322	INSERT FieldAccess MethodInvocation	3578	0.0005961
787	INSERT FieldAccess PostfixExpression	6	0.0000010
145	INSERT FieldAccess PrefixExpression	27	0.0000045
241	INSERT FieldAccess ReturnStatement	390	0.0000650
847	INSERT FieldAccess SuperMethodInvocation	10	0.0000017
858	INSERT FieldAccess SwitchStatement	1	0.0000002
596	INSERT FieldAccess SynchronizedStatement	17	0.0000028
88	INSERT FieldAccess VariableDeclarationFragment	254	0.0000423
210	INSERT FieldDeclaration AnnotationTypeDeclaration	23	0.0000038
128	INSERT FieldDeclaration AnonymousClassDeclaration	390	0.0000650
678	INSERT FieldDeclaration EnumDeclaration	298	0.0000496
155	INSERT FieldDeclaration TypeDeclaration	99230	0.0165324
59	INSERT ForStatement Block	2654	0.0004422
535	INSERT ForStatement IfStatement	9	0.0000015
853	INSERT ForStatement LabeledStatement	1	0.0000002
723	INSERT ForStatement SwitchStatement	24	0.0000040
321	INSERT IfStatement Block	70681	0.0117759
637	INSERT IfStatement DoStatement	3	0.0000005
29	INSERT IfStatement EnhancedForStatement	22	0.0000037
824	INSERT IfStatement ForStatement	20	0.0000033
501	INSERT IfStatement IfStatement	5096	0.0008490
476	INSERT IfStatement SwitchStatement	2343	0.0003904
458	INSERT IfStatement WhileStatement	3	0.0000005
818	INSERT ImportDeclaration CompilationUnit	271501	0.0452339
581	INSERT InfixExpression ArrayAccess	389	0.0000648
797	INSERT InfixExpression ArrayCreation	60	0.0000100
101	INSERT InfixExpression ArrayInitializer	651	0.0001085
644	INSERT InfixExpression AssertStatement	175	0.0000292
95	INSERT InfixExpression Assignment	1249	0.0002081
162	INSERT InfixExpression ClassInstanceCreation	1642	0.0002736
582	INSERT InfixExpression ConditionalExpression	336	0.0000560
292	INSERT InfixExpression ConstructorInvocation	4	0.0000007
741	INSERT InfixExpression DoStatement	13	0.0000022

383	INSERT InfixExpression EnumConstantDeclaration	14	0.0000023
140	INSERT InfixExpression ForStatement	303	0.0000505
808	INSERT InfixExpression IfStatement	16426	0.0027367
887	INSERT InfixExpression InfixExpression	3275	0.0005456
726	INSERT InfixExpression LambdaExpression	49	0.0000082
493	INSERT InfixExpression MemberValuePair	30	0.0000050
599	INSERT InfixExpression MethodInvocation	9874	0.0016451
90	INSERT InfixExpression ParenthesizedExpression	284	0.0000473
684	INSERT InfixExpression ReturnStatement	2186	0.0003642
191	INSERT InfixExpression SingleMemberAnnotation	13	0.0000022
438	INSERT InfixExpression SuperConstructorInvocation	28	0.0000047
404	INSERT InfixExpression SuperMethodInvocation	11	0.0000018
505	INSERT InfixExpression SwitchStatement	10	0.0000017
336	INSERT InfixExpression VariableDeclarationFragment	1961	0.0003267
793	INSERT InfixExpression WhileStatement	241	0.0000402
687	INSERT Initializer AnonymousClassDeclaration	9	0.0000015
207	INSERT Initializer EnumDeclaration	19	0.0000032
103	INSERT Initializer TypeDeclaration	987	0.0001644
519	INSERT InstanceofExpression AssertStatement	1	0.0000002
116	INSERT InstanceofExpression ClassInstanceCreation	4	0.0000007
315	INSERT InstanceofExpression ConditionalExpression	1	0.0000002
252	INSERT InstanceofExpression IfStatement	669	0.0001115
508	INSERT InstanceofExpression InfixExpression	216	0.0000360
805	INSERT InstanceofExpression MethodInvocation	68	0.0000113
277	INSERT InstanceofExpression ParenthesizedExpression	18	0.0000030
802	INSERT InstanceofExpression ReturnStatement	26	0.0000043
122	INSERT InstanceofExpression VariableDeclarationFragment	4	0.0000007
147	INSERT Javadoc AnnotationTypeDeclaration	68	0.0000113
206	INSERT Javadoc AnnotationTypeMemberDeclaration	199	0.0000332
271	INSERT Javadoc EnumConstantDeclaration	311	0.0000518
613	INSERT Javadoc EnumDeclaration	156	0.0000260
164	INSERT Javadoc FieldDeclaration	2640	0.0004398
612	INSERT Javadoc Initializer	4	0.0000007
345	INSERT Javadoc MethodDeclaration	17978	0.0029953
163	INSERT Javadoc PackageDeclaration	1592	0.0002652
411	INSERT Javadoc TypeDeclaration	3855	0.0006423
224	INSERT LabeledStatement Block	59	0.0000098
701	INSERT LabeledStatement SwitchStatement	8	0.0000013
370	INSERT LambdaExpression Assignment	20	0.0000033
178	INSERT LambdaExpression CastExpression	2	0.0000003
794	INSERT LambdaExpression ClassInstanceCreation	179	0.0000298
372	INSERT LambdaExpression ConditionalExpression	1	0.0000002
558	INSERT LambdaExpression ConstructorInvocation	12	0.0000020
384	INSERT LambdaExpression EnumConstantDeclaration	6	0.0000010
697	INSERT LambdaExpression MethodInvocation	3387	0.0005643
666	INSERT LambdaExpression ReturnStatement	171	0.0000285
782	INSERT LambdaExpression SuperConstructorInvocation	4	0.0000007
761	INSERT LambdaExpression SuperMethodInvocation	2	0.0000003
733	INSERT LambdaExpression VariableDeclarationFragment	175	0.0000292
422	INSERT MarkerAnnotation AnnotationTypeDeclaration	125	0.0000208

447	INSERT MarkerAnnotation AnnotationTypeMemberDeclaration	66	0.0000110
745	INSERT MarkerAnnotation EnumConstantDeclaration	5	0.0000008
269	INSERT MarkerAnnotation EnumDeclaration	27	0.0000045
261	INSERT MarkerAnnotation FieldDeclaration	2648	0.0004412
273	INSERT MarkerAnnotation MemberValuePair	1	0.0000002
333	INSERT MarkerAnnotation MethodDeclaration	19339	0.0032220
526	INSERT MarkerAnnotation PackageDeclaration	2	0.0000003
813	INSERT MarkerAnnotation SimpleType	2	0.0000003
562	INSERT MarkerAnnotation SingleVariableDeclaration	6865	0.0011438
471	INSERT MarkerAnnotation TypeDeclaration	2274	0.0003789
169	INSERT MarkerAnnotation VariableDeclarationStatement	33	0.0000055
636	INSERT MemberRef TagElement	367	0.0000611
695	INSERT MemberValuePair NormalAnnotation	1481	0.0002467
304	INSERT MethodDeclaration AnonymousClassDeclaration	3143	0.0005236
798	INSERT MethodDeclaration EnumDeclaration	866	0.0001443
405	INSERT MethodDeclaration TypeDeclaration	238768	0.0397803
713	INSERT MethodInvocation ArrayAccess	138	0.0000230
500	INSERT MethodInvocation ArrayCreation	172	0.0000287
38	INSERT MethodInvocation ArrayInitializer	756	0.0001260
618	INSERT MethodInvocation AssertStatement	35	0.0000058
810	INSERT MethodInvocation Assignment	7810	0.0013012
213	INSERT MethodInvocation CastExpression	690	0.0001150
597	INSERT MethodInvocation ClassInstanceCreation	9768	0.0016274
218	INSERT MethodInvocation ConditionalExpression	664	0.0001106
151	INSERT MethodInvocation ConstructorInvocation	129	0.0000215
83	INSERT MethodInvocation DoStatement	6	0.0000010
522	INSERT MethodInvocation EnhancedForStatement	885	0.0001474
643	INSERT MethodInvocation EnumConstantDeclaration	61	0.0000102
608	INSERT MethodInvocation ExpressionMethodReference	6	0.0000010
696	INSERT MethodInvocation ExpressionStatement	20966	0.0034931
436	INSERT MethodInvocation FieldAccess	46	0.0000077
869	INSERT MethodInvocation ForStatement	7	0.0000012
311	INSERT MethodInvocation IfStatement	5218	0.0008694
425	INSERT MethodInvocation InfixExpression	10040	0.0016727
236	INSERT MethodInvocation InstanceofExpression	81	0.0000135
790	INSERT MethodInvocation LambdaExpression	418	0.0000696
746	INSERT MethodInvocation MethodInvocation	76084	0.0126761
268	INSERT MethodInvocation ParenthesizedExpression	71	0.0000118
348	INSERT MethodInvocation PrefixExpression	516	0.0000860
186	INSERT MethodInvocation ReturnStatement	8752	0.0014581
882	INSERT MethodInvocation SuperConstructorInvocation	296	0.0000493
199	INSERT MethodInvocation SuperMethodInvocation	36	0.0000060
427	INSERT MethodInvocation SwitchStatement	115	0.0000192
179	INSERT MethodInvocation SynchronizedStatement	15	0.0000025
351	INSERT MethodInvocation ThrowStatement	231	0.0000385
48	INSERT MethodInvocation VariableDeclarationFragment	17478	0.0029120
395	INSERT MethodInvocation WhileStatement	31	0.0000052
166	INSERT MethodRef TagElement	494	0.0000823
180	INSERT MethodRefParameter MethodRef	335	0.0000558
449	INSERT Modifier AnnotationTypeDeclaration	29	0.0000048

855	INSERT Modifier AnnotationTypeMemberDeclaration	8	0.0000013
183	INSERT Modifier EnumDeclaration	68	0.0000113
470	INSERT Modifier FieldDeclaration	14451	0.0024076
602	INSERT Modifier Initializer	2	0.0000003
859	INSERT Modifier MethodDeclaration	12050	0.0020076
867	INSERT Modifier SingleVariableDeclaration	5352	0.0008917
24	INSERT Modifier TypeDeclaration	4189	0.0006979
457	INSERT Modifier VariableDeclarationExpression	66	0.0000110
54	INSERT Modifier VariableDeclarationStatement	7622	0.0012699
272	INSERT NormalAnnotation AnnotationTypeDeclaration	31	0.0000052
889	INSERT NormalAnnotation AnnotationTypeMemberDeclaration	1	0.0000002
546	INSERT NormalAnnotation ArrayInitializer	133	0.0000222
406	INSERT NormalAnnotation EnumConstantDeclaration	1	0.0000002
217	INSERT NormalAnnotation EnumDeclaration	2	0.0000003
639	INSERT NormalAnnotation FieldDeclaration	489	0.0000815
428	INSERT NormalAnnotation MemberValuePair	1	0.0000002
654	INSERT NormalAnnotation MethodDeclaration	1375	0.0002291
856	INSERT NormalAnnotation SingleVariableDeclaration	1488	0.0002479
628	INSERT NormalAnnotation TypeDeclaration	1424	0.0002372
6	INSERT NullLiteral ArrayInitializer	31	0.0000052
579	INSERT NullLiteral Assignment	331	0.0000551
300	INSERT NullLiteral ClassInstanceCreation	2498	0.0004162
779	INSERT NullLiteral ConditionalExpression	108	0.0000180
443	INSERT NullLiteral ConstructorInvocation	228	0.0000380
584	INSERT NullLiteral EnumConstantDeclaration	87	0.0000145
78	INSERT NullLiteral InfixExpression	1208	0.0002013
434	INSERT NullLiteral LambdaExpression	2	0.0000003
461	INSERT NullLiteral MethodInvocation	6634	0.0011053
248	INSERT NullLiteral ReturnStatement	771	0.0001285
441	INSERT NullLiteral SuperConstructorInvocation	126	0.0000210
773	INSERT NullLiteral SuperMethodInvocation	15	0.0000025
718	INSERT NullLiteral VariableDeclarationFragment	2094	0.0003489
11	INSERT NumberLiteral AnnotationTypeMemberDeclaration	8	0.0000013
749	INSERT NumberLiteral ArrayAccess	85	0.0000142
249	INSERT NumberLiteral ArrayCreation	191	0.0000318
483	INSERT NumberLiteral ArrayInitializer	972	0.0001619
366	INSERT NumberLiteral Assignment	473	0.0000788
778	INSERT NumberLiteral CastExpression	3	0.0000005
679	INSERT NumberLiteral ClassInstanceCreation	1615	0.0002691
30	INSERT NumberLiteral ConditionalExpression	85	0.0000142
567	INSERT NumberLiteral ConstructorInvocation	51	0.0000085
448	INSERT NumberLiteral EnumConstantDeclaration	91	0.0000152
863	INSERT NumberLiteral InfixExpression	1898	0.0003162
716	INSERT NumberLiteral LambdaExpression	1	0.0000002
374	INSERT NumberLiteral MemberValuePair	14	0.0000023
108	INSERT NumberLiteral MethodInvocation	11147	0.0018572
459	INSERT NumberLiteral ParenthesizedExpression	8	0.0000013
719	INSERT NumberLiteral PrefixExpression	15	0.0000025
352	INSERT NumberLiteral ReturnStatement	247	0.0000412
670	INSERT NumberLiteral SingleMemberAnnotation	10	0.0000017

466	INSERT NumberLiteral SuperConstructorInvocation	57	0.0000095
275	INSERT NumberLiteral SuperMethodInvocation	1	0.0000002
831	INSERT NumberLiteral SwitchCase	311	0.0000518
115	INSERT NumberLiteral VariableDeclarationFragment	1872	0.0003119
134	INSERT PackageDeclaration CompilationUnit	127	0.0000212
607	INSERT ParameterizedType AnnotationTypeMemberDeclaration	29	0.0000048
279	INSERT ParameterizedType ArrayType	72	0.0000120
410	INSERT ParameterizedType CastExpression	233	0.0000388
653	INSERT ParameterizedType ClassInstanceCreation	2709	0.0004513
576	INSERT ParameterizedType EnumDeclaration	10	0.0000017
142	INSERT ParameterizedType FieldDeclaration	2454	0.0004089
400	INSERT ParameterizedType InstanceofExpression	2	0.0000003
104	INSERT ParameterizedType MethodDeclaration	4027	0.0006709
31	INSERT ParameterizedType MethodInvocation	108	0.0000180
87	INSERT ParameterizedType ParameterizedType	2696	0.0004492
56	INSERT ParameterizedType SingleVariableDeclaration	4892	0.0008150
214	INSERT ParameterizedType TypeDeclaration	1913	0.0003187
870	INSERT ParameterizedType TypeParameter	90	0.0000150
674	INSERT ParameterizedType VariableDeclarationExpression	16	0.0000027
167	INSERT ParameterizedType VariableDeclarationStatement	4804	0.0008004
369	INSERT ParameterizedType WildcardType	42	0.0000070
297	INSERT ParenthesizedExpression ArrayAccess	4	0.0000007
603	INSERT ParenthesizedExpression ArrayInitializer	1	0.0000002
365	INSERT ParenthesizedExpression AssertStatement	2	0.0000003
174	INSERT ParenthesizedExpression Assignment	115	0.0000192
232	INSERT ParenthesizedExpression CastExpression	144	0.0000240
834	INSERT ParenthesizedExpression ClassInstanceCreation	153	0.0000255
114	INSERT ParenthesizedExpression ConditionalExpression	172	0.0000287
265	INSERT ParenthesizedExpression ConstructorInvocation	3	0.0000005
359	INSERT ParenthesizedExpression EnhancedForStatement	4	0.0000007
885	INSERT ParenthesizedExpression FieldAccess	6	0.0000010
244	INSERT ParenthesizedExpression ForStatement	1	0.0000002
2	INSERT ParenthesizedExpression IfStatement	22	0.0000037
396	INSERT ParenthesizedExpression InfixExpression	2437	0.0004060
681	INSERT ParenthesizedExpression LambdaExpression	7	0.0000012
510	INSERT ParenthesizedExpression MemberValuePair	6	0.0000010
81	INSERT ParenthesizedExpression MethodInvocation	1930	0.0003216
495	INSERT ParenthesizedExpression PrefixExpression	99	0.0000165
331	INSERT ParenthesizedExpression ReturnStatement	172	0.0000287
848	INSERT ParenthesizedExpression SuperConstructorInvocation	1	0.0000002
626	INSERT ParenthesizedExpression SwitchCase	2	0.0000003
165	INSERT ParenthesizedExpression ThrowStatement	6	0.0000010
381	INSERT ParenthesizedExpression VariableDeclarationFragment	197	0.0000328
565	INSERT ParenthesizedExpression WhileStatement	1	0.0000002
755	INSERT PostfixExpression ArrayAccess	96	0.0000160
563	INSERT PostfixExpression Assignment	3	0.0000005
113	INSERT PostfixExpression ClassInstanceCreation	22	0.0000037
338	INSERT PostfixExpression ExpressionStatement	483	0.0000805
37	INSERT PostfixExpression ForStatement	289	0.0000481
724	INSERT PostfixExpression InfixExpression	16	0.0000027

706	INSERT PostfixExpression MethodInvocation	113	0.0000188
763	INSERT PostfixExpression ParenthesizedExpression	1	0.0000002
649	INSERT PostfixExpression ReturnStatement	2	0.0000003
221	INSERT PostfixExpression VariableDeclarationFragment	2	0.0000003
416	INSERT PrefixExpression AnnotationTypeMemberDeclaration	5	0.0000008
32	INSERT PrefixExpression ArrayAccess	7	0.0000012
125	INSERT PrefixExpression ArrayInitializer	122	0.0000203
270	INSERT PrefixExpression AssertStatement	10	0.0000017
316	INSERT PrefixExpression Assignment	98	0.0000163
830	INSERT PrefixExpression CastExpression	1	0.0000002
705	INSERT PrefixExpression ClassInstanceCreation	157	0.0000262
465	INSERT PrefixExpression ConditionalExpression	60	0.0000100
41	INSERT PrefixExpression ConstructorInvocation	10	0.0000017
358	INSERT PrefixExpression DoStatement	2	0.0000003
817	INSERT PrefixExpression EnumConstantDeclaration	6	0.0000010
512	INSERT PrefixExpression ExpressionStatement	11	0.0000018
12	INSERT PrefixExpression ForStatement	18	0.0000030
571	INSERT PrefixExpression IfStatement	2406	0.0004009
775	INSERT PrefixExpression InfixExpression	1527	0.0002544
66	INSERT PrefixExpression LambdaExpression	9	0.0000015
786	INSERT PrefixExpression MethodInvocation	915	0.0001524
367	INSERT PrefixExpression ParenthesizedExpression	10	0.0000017
320	INSERT PrefixExpression ReturnStatement	214	0.0000357
149	INSERT PrefixExpression SuperConstructorInvocation	10	0.0000017
106	INSERT PrefixExpression SwitchCase	5	0.0000008
317	INSERT PrefixExpression VariableDeclarationFragment	401	0.0000668
492	INSERT PrefixExpression WhileStatement	71	0.0000118
661	INSERT PrimitiveType AnnotationTypeMemberDeclaration	6	0.0000010
136	INSERT PrimitiveType ArrayType	163	0.0000272
841	INSERT PrimitiveType CastExpression	19	0.0000032
25	INSERT PrimitiveType FieldDeclaration	1560	0.0002599
511	INSERT PrimitiveType MethodDeclaration	6043	0.0010068
862	INSERT PrimitiveType MethodRefParameter	14	0.0000023
177	INSERT PrimitiveType SingleVariableDeclaration	2391	0.0003984
219	INSERT PrimitiveType TypeLiteral	22	0.0000037
860	INSERT PrimitiveType VariableDeclarationExpression	15	0.0000025
201	INSERT PrimitiveType VariableDeclarationStatement	1858	0.0003096
129	INSERT QualifiedName AnnotationTypeMemberDeclaration	19	0.0000032
326	INSERT QualifiedName ArrayAccess	106	0.0000177
343	INSERT QualifiedName ArrayCreation	47	0.0000078
424	INSERT QualifiedName ArrayInitializer	775	0.0001291
539	INSERT QualifiedName AssertStatement	1	0.0000002
393	INSERT QualifiedName Assignment	2684	0.0004472
293	INSERT QualifiedName CastExpression	44	0.0000073
894	INSERT QualifiedName ClassInstanceCreation	3686	0.0006141
112	INSERT QualifiedName ConditionalExpression	177	0.0000295
735	INSERT QualifiedName ConstructorInvocation	98	0.0000163
53	INSERT QualifiedName EnhancedForStatement	147	0.0000245
468	INSERT QualifiedName EnumConstantDeclaration	449	0.0000748
294	INSERT QualifiedName ExpressionMethodReference	3	0.0000005

703	INSERT QualifiedName IfStatement	254	0.0000423
419	INSERT QualifiedName ImportDeclaration	13	0.0000022
239	INSERT QualifiedName InfixExpression	4250	0.0007081
792	INSERT QualifiedName InstanceofExpression	17	0.0000028
61	INSERT QualifiedName LambdaExpression	4	0.0000007
318	INSERT QualifiedName MarkerAnnotation	202	0.0000337
574	INSERT QualifiedName MemberRef	79	0.0000132
820	INSERT QualifiedName MemberValuePair	98	0.0000163
137	INSERT QualifiedName MethodInvocation	28154	0.0046906
46	INSERT QualifiedName MethodRef	155	0.0000258
641	INSERT QualifiedName NormalAnnotation	46	0.0000077
386	INSERT QualifiedName PackageDeclaration	1	0.0000002
709	INSERT QualifiedName ParenthesizedExpression	9	0.0000015
730	INSERT QualifiedName PostfixExpression	16	0.0000027
740	INSERT QualifiedName PrefixExpression	60	0.0000100
319	INSERT QualifiedName ReturnStatement	979	0.0001631
417	INSERT QualifiedName SimpleType	4752	0.0007917
408	INSERT QualifiedName SingleMemberAnnotation	108	0.0000180
717	INSERT QualifiedName SuperConstructorInvocation	225	0.0000375
814	INSERT QualifiedName SuperMethodInvocation	11	0.0000018
593	INSERT QualifiedName SwitchCase	576	0.0000960
375	INSERT QualifiedName SwitchStatement	14	0.0000023
487	INSERT QualifiedName SynchronizedStatement	59	0.0000098
888	INSERT QualifiedName TagElement	630	0.0001050
332	INSERT QualifiedName ThisExpression	1	0.0000002
215	INSERT QualifiedName ThrowStatement	1	0.0000002
645	INSERT QualifiedName VariableDeclarationFragment	1549	0.0002581
783	INSERT QualifiedType FieldDeclaration	1	0.0000002
86	INSERT QualifiedType MethodDeclaration	1	0.0000002
305	INSERT QualifiedType SingleVariableDeclaration	4	0.0000007
852	INSERT QualifiedType TypeDeclaration	1	0.0000002
752	INSERT QualifiedType VariableDeclarationStatement	7	0.0000012
738	INSERT ReturnStatement Block	18776	0.0031282
34	INSERT ReturnStatement IfStatement	912	0.0001519
284	INSERT ReturnStatement SwitchStatement	3028	0.0005045
324	INSERT SimpleName AnnotationTypeDeclaration	5	0.0000008
195	INSERT SimpleName AnnotationTypeMemberDeclaration	63	0.0000105
614	INSERT SimpleName ArrayAccess	761	0.0001268
68	INSERT SimpleName ArrayCreation	151	0.0000252
360	INSERT SimpleName ArrayInitializer	774	0.0001290
796	INSERT SimpleName AssertStatement	65	0.0000108
229	INSERT SimpleName Assignment	8768	0.0014608
98	INSERT SimpleName BreakStatement	4	0.0000007
291	INSERT SimpleName CastExpression	449	0.0000748
621	INSERT SimpleName ClassInstanceCreation	18633	0.0031044
23	INSERT SimpleName ConditionalExpression	481	0.0000801
433	INSERT SimpleName ConstructorInvocation	1242	0.0002069
255	INSERT SimpleName ContinueStatement	3	0.0000005
764	INSERT SimpleName DoStatement	1	0.0000002
407	INSERT SimpleName EnhancedForStatement	773	0.0001288

7	INSERT SimpleName EnumConstantDeclaration	110	0.0000183
672	INSERT SimpleName EnumDeclaration	8	0.0000013
451	INSERT SimpleName ExpressionMethodReference	29	0.0000048
585	INSERT SimpleName FieldAccess	18	0.0000030
610	INSERT SimpleName IfStatement	1827	0.0003044
488	INSERT SimpleName ImportDeclaration	26	0.0000043
274	INSERT SimpleName InfixExpression	12255	0.0020418
525	INSERT SimpleName InstanceofExpression	179	0.0000298
371	INSERT SimpleName LambdaExpression	19	0.0000032
760	INSERT SimpleName MarkerAnnotation	263	0.0000438
220	INSERT SimpleName MemberRef	130	0.0000217
92	INSERT SimpleName MemberValuePair	38	0.0000063
496	INSERT SimpleName MethodDeclaration	10991	0.0018312
682	INSERT SimpleName MethodInvocation	121746	0.0202837
245	INSERT SimpleName MethodRef	246	0.0000410
531	INSERT SimpleName MethodRefParameter	4	0.0000007
266	INSERT SimpleName NormalAnnotation	61	0.0000102
342	INSERT SimpleName ParenthesizedExpression	70	0.0000117
39	INSERT SimpleName PostfixExpression	6	0.0000010
243	INSERT SimpleName PrefixExpression	237	0.0000395
754	INSERT SimpleName ReturnStatement	5701	0.0009498
55	INSERT SimpleName SimpleType	7813	0.0013017
473	INSERT SimpleName SingleMemberAnnotation	129	0.0000215
254	INSERT SimpleName SingleVariableDeclaration	5621	0.0009365
202	INSERT SimpleName SuperConstructorInvocation	2000	0.0003332
663	INSERT SimpleName SuperMethodInvocation	467	0.0000778
668	INSERT SimpleName SwitchCase	1054	0.0001756
762	INSERT SimpleName SwitchStatement	77	0.0000128
620	INSERT SimpleName SynchronizedStatement	95	0.0000158
313	INSERT SimpleName TagElement	1524	0.0002539
312	INSERT SimpleName ThisExpression	248	0.0000413
789	INSERT SimpleName ThrowStatement	61	0.0000102
230	INSERT SimpleName TypeDeclaration	1437	0.0002394
238	INSERT SimpleName TypeParameter	5	0.0000008
785	INSERT SimpleName VariableDeclarationFragment	4458	0.0007427
423	INSERT SimpleName WhileStatement	12	0.0000020
829	INSERT SimpleType AnnotationTypeMemberDeclaration	16	0.0000027
509	INSERT SimpleType ArrayType	339	0.0000565
389	INSERT SimpleType CastExpression	456	0.0000760
700	INSERT SimpleType ClassInstanceCreation	3412	0.0005685
750	INSERT SimpleType CreationReference	6	0.0000010
154	INSERT SimpleType EnumDeclaration	56	0.0000093
547	INSERT SimpleType FieldDeclaration	4245	0.0007072
568	INSERT SimpleType InstanceofExpression	34	0.0000057
253	INSERT SimpleType MethodDeclaration	17591	0.0029308
263	INSERT SimpleType MethodInvocation	1516	0.0002526
120	INSERT SimpleType MethodRefParameter	45	0.0000075
257	INSERT SimpleType ParameterizedType	7572	0.0012615
744	INSERT SimpleType SingleVariableDeclaration	7864	0.0013102
776	INSERT SimpleType TypeDeclaration	5881	0.0009798

309	INSERT SimpleType TypeLiteral	143	0.0000238
799	INSERT SimpleType TypeParameter	87	0.0000145
193	INSERT SimpleType UnionType	79	0.0000132
77	INSERT SimpleType VariableDeclarationExpression	33	0.0000055
356	INSERT SimpleType VariableDeclarationStatement	9517	0.0015856
135	INSERT SimpleType WildcardType	114	0.0000190
102	INSERT SingleMemberAnnotation AnnotationTypeDeclaration	62	0.0000103
347	INSERT SingleMemberAnnotation AnnotationTypeMemberDeclarat	3	0.0000005
518	INSERT SingleMemberAnnotation ArrayInitializer	1	0.0000002
772	INSERT SingleMemberAnnotation EnumConstantDeclaration	1	0.0000002
734	INSERT SingleMemberAnnotation EnumDeclaration	9	0.0000015
809	INSERT SingleMemberAnnotation FieldDeclaration	1599	0.0002664
728	INSERT SingleMemberAnnotation MethodDeclaration	4271	0.0007116
514	INSERT SingleMemberAnnotation PackageDeclaration	32	0.0000053
231	INSERT SingleMemberAnnotation SingleVariableDeclaration	1070	0.0001783
172	INSERT SingleMemberAnnotation TypeDeclaration	2484	0.0004139
288	INSERT SingleMemberAnnotation VariableDeclarationExpression	20	0.0000033
45	INSERT SingleMemberAnnotation VariableDeclarationStatement	221	0.0000368
464	INSERT SingleVariableDeclaration CatchClause	406	0.0000676
302	INSERT SingleVariableDeclaration EnhancedForStatement	427	0.0000711
638	INSERT SingleVariableDeclaration LambdaExpression	75	0.0000125
119	INSERT SingleVariableDeclaration MethodDeclaration	55217	0.0091995
819	INSERT StringLiteral AnnotationTypeMemberDeclaration	45	0.0000075
44	INSERT StringLiteral ArrayInitializer	2175	0.0003624
850	INSERT StringLiteral AssertStatement	12	0.0000020
209	INSERT StringLiteral Assignment	228	0.0000380
656	INSERT StringLiteral ClassInstanceCreation	2979	0.0004963
675	INSERT StringLiteral ConditionalExpression	54	0.0000090
737	INSERT StringLiteral ConstructorInvocation	15	0.0000025
580	INSERT StringLiteral EnumConstantDeclaration	242	0.0000403
420	INSERT StringLiteral InfixExpression	8100	0.0013495
528	INSERT StringLiteral LambdaExpression	1	0.0000002
111	INSERT StringLiteral MemberValuePair	136	0.0000227
595	INSERT StringLiteral MethodInvocation	23231	0.0038704
549	INSERT StringLiteral ReturnStatement	581	0.0000968
247	INSERT StringLiteral SingleMemberAnnotation	88	0.0000147
175	INSERT StringLiteral SuperConstructorInvocation	54	0.0000090
875	INSERT StringLiteral SuperMethodInvocation	1	0.0000002
58	INSERT StringLiteral SwitchCase	206	0.0000343
226	INSERT StringLiteral VariableDeclarationFragment	1668	0.0002779
871	INSERT SuperConstructorInvocation Block	1866	0.0003109
227	INSERT SuperFieldAccess Assignment	5	0.0000008
429	INSERT SuperFieldAccess EnhancedForStatement	3	0.0000005
43	INSERT SuperFieldAccess InfixExpression	1	0.0000002
235	INSERT SuperFieldAccess MethodInvocation	9	0.0000015
560	INSERT SuperMethodInvocation Assignment	5	0.0000008
171	INSERT SuperMethodInvocation CastExpression	2	0.0000003
683	INSERT SuperMethodInvocation ClassInstanceCreation	3	0.0000005
812	INSERT SuperMethodInvocation ConditionalExpression	5	0.0000008
780	INSERT SuperMethodInvocation EnhancedForStatement	1	0.0000002

693	INSERT SuperMethodInvocation ExpressionStatement	675	0.0001125
527	INSERT SuperMethodInvocation IfStatement	2	0.0000003
589	INSERT SuperMethodInvocation InfixExpression	38	0.0000063
256	INSERT SuperMethodInvocation LambdaExpression	1	0.0000002
287	INSERT SuperMethodInvocation MethodInvocation	39	0.0000065
517	INSERT SuperMethodInvocation PrefixExpression	2	0.0000003
878	INSERT SuperMethodInvocation ReturnStatement	163	0.0000272
409	INSERT SuperMethodInvocation VariableDeclarationFragment	66	0.0000110
601	INSERT SwitchCase SwitchStatement	7095	0.0011821
301	INSERT SwitchStatement Block	1390	0.0002316
555	INSERT SwitchStatement IfStatement	3	0.0000005
660	INSERT SwitchStatement SwitchStatement	58	0.0000097
196	INSERT SynchronizedStatement Block	748	0.0001246
413	INSERT SynchronizedStatement IfStatement	3	0.0000005
414	INSERT SynchronizedStatement SwitchStatement	6	0.0000010
642	INSERT TagElement Javadoc	20911	0.0034839
554	INSERT TagElement TagElement	9936	0.0016554
893	INSERT TextElement TagElement	47340	0.0078872
390	INSERT ThisExpression AssertStatement	4	0.0000007
203	INSERT ThisExpression Assignment	15	0.0000025
10	INSERT ThisExpression CastExpression	3	0.0000005
537	INSERT ThisExpression ClassInstanceCreation	756	0.0001260
69	INSERT ThisExpression ConditionalExpression	3	0.0000005
349	INSERT ThisExpression EnhancedForStatement	29	0.0000048
397	INSERT ThisExpression ExpressionMethodReference	16	0.0000027
769	INSERT ThisExpression FieldAccess	14	0.0000023
391	INSERT ThisExpression InfixExpression	30	0.0000050
70	INSERT ThisExpression MethodInvocation	3361	0.0005600
609	INSERT ThisExpression ReturnStatement	190	0.0000317
758	INSERT ThisExpression SuperConstructorInvocation	1	0.0000002
260	INSERT ThisExpression SwitchStatement	1	0.0000002
611	INSERT ThisExpression SynchronizedStatement	68	0.0000113
545	INSERT ThisExpression VariableDeclarationFragment	8	0.0000013
857	INSERT ThrowStatement Block	2300	0.0003832
146	INSERT ThrowStatement IfStatement	188	0.0000313
200	INSERT ThrowStatement SwitchStatement	135	0.0000225
14	INSERT TryStatement Block	6997	0.0011657
616	INSERT TryStatement IfStatement	17	0.0000028
477	INSERT TryStatement SwitchStatement	85	0.0000142
516	INSERT TryStatement WhileStatement	3	0.0000005
126	INSERT TypeDeclaration AnnotationTypeDeclaration	19	0.0000032
833	INSERT TypeDeclaration CompilationUnit	909	0.0001514
242	INSERT TypeDeclaration EnumDeclaration	6	0.0000010
240	INSERT TypeDeclaration TypeDeclaration	10990	0.0018310
161	INSERT TypeDeclarationStatement Block	11	0.0000018
655	INSERT TypeDeclarationStatement SwitchStatement	16	0.0000027
821	INSERT TypeLiteral AnnotationTypeMemberDeclaration	40	0.0000067
877	INSERT TypeLiteral ArrayInitializer	732	0.0001220
529	INSERT TypeLiteral Assignment	8	0.0000013
544	INSERT TypeLiteral ClassInstanceCreation	342	0.0000570

76	INSERT TypeLiteral ConditionalExpression	1	0.0000002
337	INSERT TypeLiteral EnumConstantDeclaration	15	0.0000025
110	INSERT TypeLiteral InfixExpression	17	0.0000028
160	INSERT TypeLiteral MemberValuePair	24	0.0000040
380	INSERT TypeLiteral MethodInvocation	3269	0.0005446
826	INSERT TypeLiteral ReturnStatement	16	0.0000027
559	INSERT TypeLiteral SingleMemberAnnotation	11	0.0000018
283	INSERT TypeLiteral SuperConstructorInvocation	55	0.0000092
707	INSERT TypeLiteral SynchronizedStatement	13	0.0000022
467	INSERT TypeLiteral VariableDeclarationFragment	12	0.0000020
176	INSERT TypeMethodReference MethodInvocation	2	0.0000003
486	INSERT TypeParameter MethodDeclaration	1571	0.0002617
330	INSERT TypeParameter TypeDeclaration	896	0.0001493
194	INSERT UnionType SingleVariableDeclaration	373	0.0000621
363	INSERT VariableDeclarationExpression ForStatement	264	0.0000440
573	INSERT VariableDeclarationExpression TryStatement	468	0.0000780
4	INSERT VariableDeclarationFragment FieldDeclaration	1824	0.0003039
303	INSERT VariableDeclarationFragment LambdaExpression	431	0.0000718
74	INSERT VariableDeclarationFragment VariableDeclarationExpressio	53	0.0000088
198	INSERT VariableDeclarationFragment VariableDeclarationStatemen	2030	0.0003382
124	INSERT VariableDeclarationStatement Block	90753	0.0151201
460	INSERT VariableDeclarationStatement SwitchStatement	1441	0.0002401
133	INSERT WhileStatement Block	1150	0.0001916
502	INSERT WhileStatement SwitchStatement	8	0.0000013
659	INSERT WildcardType ParameterizedType	1805	0.0003007
556	MOVE AnnotationTypeDeclaration	158	0.0000263
534	MOVE AnnotationTypeMemberDeclaration	139	0.0000232
289	MOVE AnonymousClassDeclaration	2021	0.0003367
153	MOVE ArrayAccess	1139	0.0001898
362	MOVE ArrayCreation	2949	0.0004913
688	MOVE ArrayInitializer	842	0.0001403
440	MOVE ArrayType	1934	0.0003222
732	MOVE AssertStatement	280	0.0000466
569	MOVE Assignment	18658	0.0031085
757	MOVE Block	187816	0.0312914
553	MOVE BreakStatement	1	0.0000002
767	MOVE CastExpression	11366	0.0018937
211	MOVE CatchClause	3427	0.0005710
665	MOVE ClassInstanceCreation	48881	0.0081439
426	MOVE CompilationUnit	49437	0.0082365
121	MOVE ConditionalExpression	4571	0.0007616
27	MOVE ConstructorInvocation	541	0.0000901
825	MOVE CreationReference	156	0.0000260
173	MOVE DoStatement	211	0.0000352
3	MOVE EnhancedForStatement	6206	0.0010340
648	MOVE EnumConstantDeclaration	414	0.0000690
768	MOVE EnumDeclaration	267	0.0000445
354	MOVE ExpressionMethodReference	23	0.0000038
159	MOVE ExpressionStatement	14666	0.0024435
513	MOVE FieldAccess	1531	0.0002551

334	MOVE FieldDeclaration	20614	0.0034344
791	MOVE ForStatement	3431	0.0005716
540	MOVE IfStatement	74125	0.0123497
747	MOVE InfixExpression	40189	0.0066958
849	MOVE Initializer	147	0.0000245
836	MOVE InstanceofExpression	1025	0.0001708
26	MOVE Javadoc	5418	0.0009027
415	MOVE LabeledStatement	24	0.0000040
698	MOVE LambdaExpression	4949	0.0008245
598	MOVE MarkerAnnotation	492	0.0000820
282	MOVE MemberValuePair	82	0.0000137
85	MOVE MethodDeclaration	118811	0.0197947
868	MOVE MethodInvocation	316225	0.0526852
192	MOVE MethodRef	412	0.0000686
225	MOVE MethodRefParameter	481	0.0000801
157	MOVE NormalAnnotation	527	0.0000878
33	MOVE PackageDeclaration	2	0.0000003
807	MOVE ParameterizedType	36892	0.0061465
631	MOVE ParenthesizedExpression	2899	0.0004830
507	MOVE PostfixExpression	95	0.0000158
170	MOVE PrefixExpression	2306	0.0003842
222	MOVE QualifiedType	10	0.0000017
463	MOVE ReturnStatement	9336	0.0015554
504	MOVE SimpleType	52656	0.0087728
340	MOVE SingleMemberAnnotation	133	0.0000222
788	MOVE SingleVariableDeclaration	23937	0.0039881
828	MOVE SuperConstructorInvocation	569	0.0000948
729	MOVE SuperMethodInvocation	213	0.0000355
816	MOVE SwitchCase	117	0.0000195
690	MOVE SwitchStatement	7317	0.0012191
624	MOVE SynchronizedStatement	721	0.0001201
658	MOVE TagElement	5880	0.0009796
17	MOVE ThisExpression	8	0.0000013
430	MOVE ThrowStatement	429	0.0000715
60	MOVE TryStatement	8471	0.0014113
592	MOVE TypeDeclaration	25284	0.0042125
392	MOVE TypeDeclarationStatement	2	0.0000003
880	MOVE TypeLiteral	1695	0.0002824
770	MOVE TypeParameter	323	0.0000538
657	MOVE UnionType	581	0.0000968
444	MOVE VariableDeclarationExpression	1480	0.0002466
520	MOVE VariableDeclarationFragment	36966	0.0061588
16	MOVE VariableDeclarationStatement	67277	0.0112088
328	MOVE WhileStatement	910	0.0001516
864	MOVE WildcardType	687	0.0001145
1	UPDATE	1048375	0.1746663