Об автоматической генерации сообщений к коммитам в системах контроля версий

И.А. Косьяненко, Р.Г. Болбаков

Аннотация — В современном мире ни один проект по разработке программного обеспечения не обходится без применения систем контроля версий для отслеживания изменений в данных. Особенно важна система контроля версий в командной работе, когда работа над разными частями одного проекта делегируется нескольким разработчикам. Необходимо релевантно описать внесенные изменения с их контекстом, при этом уместив их в ограничения, задаваемые самой системой контроля версий (например, git позволяет поместить в сообщение коммита всего 72 символа). Тем не менее, процесс описания, или комментирования изменений, все еще остается рутинной, не автоматизированной задачей, к которой разработчики часто относятся без должного внимания. Применение современных методов обработки естественного языка (или иных моделей и методов, позволяющих на основе исходного кода генерировать описания изменений на естественном языке) может помочь разработчикам сэкономить время на написание актуальных сообщений к коммитам и поддерживать корректность репозитория версий исходного кода.

Ключевые слова **— автоматическая генерация сообщений коммита, обработка исходного кода, системы контроля версий, сообщения коммита, сообщения об изменениях, git.**

I. Введение

Результатом труда разработчика чаще всего является программный код. В код вносятся изменения, он разрастается, затем изменения отменяются — и с каждой внесенной строкой кода контролировать разработку без специализированный программных средств становится все сложнее. А когда нескольким инженерам требуется получить доступ для работы с одним и тем же файлом, обеспечение такого режима работы становится большой проблемой [1]. Из-за отсутствия централизованного журнала изменений модификации, внесенные одним разработчиком, могут быть утеряны в результате работы другого разработчика.

При совместной работе в одном репозитории проблемой также становится передача с внесенными изменениями их контекста. Такой контекст предназначен, в первую очередь, для других разработчиков, которые работают над общей кодовой базой. Из этого следует, что передаваемый контекст должен быть описан с помощью естественного языка.

Такая концепция работы в команде привела к реализации инструмента, называемой сообщением об изменениях (**commit message**). Если дельта (список изменений исходного кода) отвечает на вопрос "*что изменено*?", то сообщения об изменениях призваны отвечать на вопрос "*почему внесены изменения*?".

При этом многие разработчики пренебрегают написанием качественного комментария к внесенным изменениям. Хорошее описание изменений позволит облегчить понимание изменений программного обеспечения и истории его развития, однако его написание — непростая задача. Она усложняется отсутствием единой методологии описания изменений. Развитие нейронных сетей позволило инженерам применять их для генерации сообщений об изменениях, а в последнее время стало появляться все больше исследований и инструментов для решения проблемы, описанных ниже в статье.

II. Анализ методов генерации коммитов глубиной в 10 лет

Для решения проблем, связанных с контролем версионирования кодовой базы программного продукта, инженеры спроектировали и разработали системы контроля версий. История выхода наиболее популярных систем контроля версий изображена на рис. 1 [2].

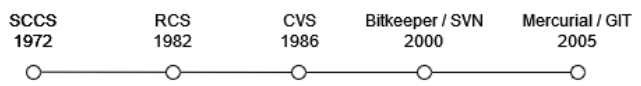

Рис.1. История выхода систем контроля версий.

Системы контроля версий принято делить на поколения [3]. В системах контроля версий первого поколения файлы редактировались только локально и одним пользователем за раз. Во втором поколении появляется поддержка сети, что приводит к появлению централизованных репозиториев хранения данных. Третье поколение состоит из распределённых систем контроля версий, где все копии репозитория считаются равными, отсутствует централизация. Это открывает

Статья получена 15 февраля 2022.

Иван Александрович Косьяненко, Кафедра инструментального и прикладного программного обеспечения, Российский технологический университет МИРЭА, Москва, Россия (e-mail: kosyanenko.edu@gmail.com).

Роман Геннадьевич Болбаков, Кафедра инструментального и прикладного программного обеспечения, Российский технологический университет МИРЭА, Москва, Россия (e-mail: bolbakov@mirea.ru)

путь для коммитов, ветвей и слияний, которые создаются локально без доступа к сети и перемещаются в другие репозитории по мере необходимости.

Одной из первых успешных систем контроля версий считается **SCCS**. Как и в большинстве современных систем, в SCCS есть набор команд для работы с версиями файлов [4] (внесение, извлечение, извлечение конкретных версий для редактирования и т.д.). Одной из предоставляемых функций является комментирование внесенных изменений и последующее отображение лога изменений с комментариями.

Для фиксации изменений файла в SCCS применяется команда *delta* со следующей сигнатурой:

Листинг 1. Сигнатура фиксации изменений в SCCS

```
delta [-r SID][-s ][-n ][-g List][-p ][-m
ModificationRequestList][-y [Comment]]File
...
```

Важность осмысленности и точности подписи к изменениям (комментария) отмечалась еще в документации программного средства: "*Comments should be meaningful, since you may return to the file one day.*" [5]. С появлением сети в системах контроля версий проблема встала еще острее — теперь контекст изменений было необходимо передавать всем разработчикам, работающим с данным репозиторием.

В конце 00-х разработчики стали все больше задумываться о времени, затрачиваемом на разработку программных продуктов. Одним из перспективных направлений в снижении затрачиваемого времени была автоматическая генерация документирования внесенных в исходный код изменений [6]. Реймонд Бус и Уэстли Веймер из Виргинского университета предложили свой алгоритм, названный DeltaDoc для решения данного вопроса.

Входными данными алгоритма являются две версии исходного кода программы; на выходе алгоритма — текстовое описание (на естественном языке) изменения выполнения программы. Поток исполнения алгоритма представлен на рис.2.

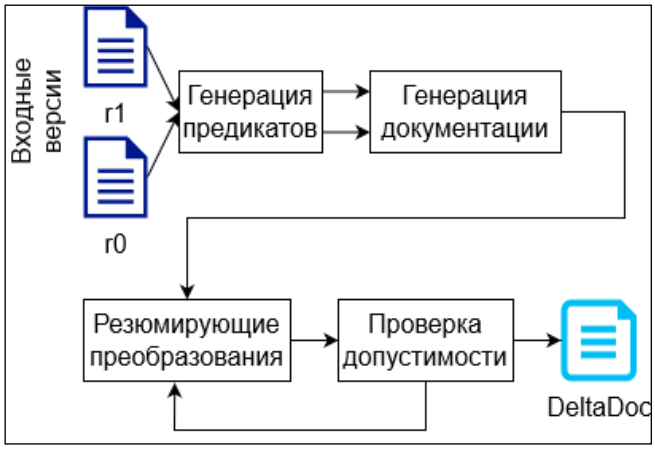

Рис.2. Последовательность исполнения DeltaDoc.

Первый шаг алгоритма — получение предикатов внутрипроцедурных путей, формул, которые описывают условия, при которых может быть выбран путь выполнения выражения. Сначала определяются пути потока управления без циклов, исходя из предположения, что каждый цикл либо не выполняется вообще, либо выполняется единожды. Отслеживаются также и условные операторы (if / else): оценка их защитных предикатов (guarding predicates) происходит с помощью символических значений переменных. Результирующие предикаты в совокупности образуют предикат пути для данного выражения.

В фазу генерации документации поступают два набора сопоставлений «выражение-предикат», представляющий исходный код до и после внесения изменений. Эти сопоставления используются для генерации документации вида "***If X, Do Y Instead of Z*** ". На рис. 3 представлен высокоуровневый псевдокод для реализации данной фазы.

Input: Method P_{old} : statements → path predicates.
Input: Method P_{new} : statements → path predicates.
Input: Mapping E : statements → symbolic statements.
Output: emitted structured documentation
Global: set *MustDoc* of statements = ∅
1: **let** *Inserted* = Domain(P_{new}) \ Domain(P_{old})
2: **let** *Deleted* = Domain(P_{old}) \ Domain(P_{new})
3: **let** *Changed* = ∅
4: **for all** statements $s \in$ Domain(P_{new})∩Domain(P_{old}) **do**
5: **if** $P_{new}(s) \neq P_{old}(s)$ **then**
6: *Changed* ← *Changed* ∪ {s}
7: **end if**
8: **end for**
9: *MustDoc* ← *Inserted* ∪ *Deleted* ∪ *Changed*
10: **let** *Predicates* = ∅ (a multi-set)
11: **for all** statements $s \in$ *MustDoc* **do**
12: **let** $C_1 \wedge C_2 \cdots \wedge C_n = P_{new}(s) \wedge P_{old}(s)$
13: *Predicates* ← *Predicates* ∪ {C_1} ∪ ⋯ ∪ {C_n}
14: **end for**
15: *Predicates* ← *Predicates* sorted by frequency
16: **call** HierarchicalDoc (∅, , P_{new}, P_{old}, *Predicates*, E)

Рис.3. Реализация фазы генерации документации на псевдокоде

Алгоритм сопоставляет выражения с одним и тем же байт-кодом инструкции и символьными операндами, помечая все состояния, которые имеют измененный предикат или присутствуют только в одной версии. Выражения группируются по предикатам, и сначала обрабатываются те выражения, что встречаются в исходном коде наиболее часто. Документация генерируется иерархическим образом — все выражения, “защищенные” выбранным предикатом сортируются по номерам строк. Выражения, которые строго соответствуют выбранному предикату, документируются согласно описанному выше правилу и исключаются из дальнейшего рассмотрения. Если остаются незадокументированные выражения, выбирается следующий наиболее распространенный предикат. Затем рекурсивно вызывается вспомогательная функция для документирования выражений, которые теперь могут быть полностью охвачены добавлением нового предиката. Процесс завершается тогда, когда все инструкции задокументированы. Сами выражения и предикаты отображаются с помощью функции «описания» (describe), которая выводит исходный язык с незначительными заменами естественного языка.

Вывод результата данного шага представлен на рис.4.

```
When calling sayHello(String name)
  If name != null AND System.Lang == ENG
    return "Hi " + name Instead of "Hello " + name
```

Рис.4. Результат шага генерации документации.

Шаг резюмирующих преобразований, по словам авторов алгоритма, является ключевым. Дело в том, что документация, генерируемая на предыдущем шаге, является слишком обширной, из-за чего понижается удобочитаемость работы с такой документацией. Преобразования являются синергетическими и могут применяться последовательно, подобно стандартным оптимизациям потоков данных в компиляторах. Однако, в отличие от стандартных оптимизаций компилятора, не все преобразования сохраняют семантику. Вместо этого многие преобразования приводят к потерям информации, жертвуя информационным контентом для экономии места. По мере увеличения количества фактов, подлежащих документированию, увеличивается и количество применяемых преобразований. Указанные выше авторы [6] приводят в пример следующие преобразования одиночных предикатов:

- отбросить вызов методов, уже появляющихся в предикате;
- отбросить вызов методов, встречающихся в операторах return, throw или assignment;
- отбросить условия, которые не содержат хотя бы одного операнда в документированных выражениях.

Результатом работы алгоритма является описание изменений исходного кода программы (на языке программирования Java) на естественном языке. Тем не менее, данный алгоритм все еще не передает контекст вносимых изменений.

III. Современное положение решения проблемы

Согласно ежегодному опросу портала для разработчиков Stack Overflow [7], в современной разработке наиболее популярной системой контроля версий является Git, спроектированный Линусом Торвальдсом для управления разработкой ядра Linux. Важность описания вносимых изменений в репозиторий подчеркивается в каждом руководстве по работе с системой [8]. Тем не менее, анализ репозиториев с открытым исходным кодом показывает, что многие разработчики пренебрегают этими рекомендациями. Так, исследование более 23 тысяч сообщений коммитов репозиториев проектов, написанных на языке Java показало, что 14% от всех сообщений были пустыми [9].

Для решения этой проблемы было разработано множество инструментов — описанный выше алгоритм DeltaDoc, ChangeScribe [10]. Они показывают, что было изменено, и в какой части исходного кода произошли эти изменения. Но сгенерированные этими инструментами сообщения являются подробными и не раскрывают обоснование изменений.

Сгенерировать качественное сообщение о внесенных изменениях сложно, поскольку для ответа на вопрос, почему произошло изменение, обычно требуется синтез различных видов знаний и контекста. Тем не менее, современные исследования [11, 12] показали, что сообщения об изменениях следуют некоторым паттернам, и в больших наборах данных эти паттерны можно выявить и изучить [13]. С решением указанной выше проблемы помогает развитие нейронных сетей.

Один из предложенных подходов к генерации качественных сообщений об изменениях основан на **нейронном машинном переводе** (НМП) — нейронных сетях, которые моделируют процесс трансляции последовательности исходного языка $x = (x_1, \dots, x_n)$ в последовательность целевого языка $y = (y_1, \dots, y_n)$ с условной вероятностью $p = (y|x)$ [14, 15]. В своей работе под названием "Automatically Generating Commit Messages from Diffs using Neural Machine Translation" [14] авторы Амир Армали и Коллин МакМиллан применили несколько алгоритмов НМП для трансляции естественного языка путем обучения нейронной сети на парах *изменения исходного кода – сообщение об изменении.* Набор данных включал в себя 2 миллиона коммитов.

Основой технологии, предложенной упомянутыми выше авторами, является модель *RNN Encoder-Decoder*. Модель состоит из двух рекуррентных нейронных сетей (РНН). Первая РНН транслирует последовательности на исходном языке в векторное представление. Она называется кодировщиком (encoder). Вторая РНН преобразует векторное представление в последовательности на целевом языке, и называется декодером (decoder).

Входные данные кодировщика — последовательности переменной длины $x = (x_1, \dots, x_T)$. Кодировщик обрабатывает последовательность посимвольно. Как рекуррентная нейронная сеть, кодировщик имеет скрытое состояние h, являющееся вектором фиксированной длинны. За время t кодировщик вычисляет скрытое состояние h_t по формуле (1):

$$h_t = f(h_{t-1}, x_t) \quad (1)$$

где f — нелинейная функция (долгая краткосрочная память [16] или управляемый рекуррентный блок [17]). Конечным символов последовательности *x* должен являться символ окончания последовательности (<eos>), уведомляющий кодировщик об окончании процесса перевода и выводе финального скрытого состояния h_T, являющегося векторным представлением *x*.

Выходными данными для декодера является целевая последовательность $y = (y_1, \dots, y_{T'})$. Одним из входных сигналов декодера является символ <start>, обозначающий начало целевой последовательности. За время *t* декодер вычисляет скрытое состояние $h_t^{`}$ и условное распределение следующего символа y_t по формулам (2) (3):

$$h_t^{`} = f(h_{t-1}^{`}, y_{t-1}, h_T) \quad (2)$$

$$\mathrm{p}(y_t|y_{t-1}, \dots, y_1, h_T) = g(h_t^{`}, y_{t-1}, h_T) \quad (3)$$

где h_T — вектор контекста, генерируемый кодировщиком; f и g — нелинейные функции, причем f — та же функция, что и в кодировщике. В качестве функции g выступает функция Softmax [18].

Кодировщик и декодер обучаются совместно, чтобы максимизировать условную логарифмическую вероятность (4):

$$\max_\theta \frac{1}{N}\sum_{i=1}^{N} \log p(y_i|x_i;\theta) \quad (4)$$

где θ — набор параметров модели; N — размер тренировочного множества; (x_i, y_i) — пары исходных и целевых последовательностей в тренировочном множестве.

Качество генерируемых сообщений рассматриваемого инструмента напрямую зависит от качества тренировочного множества. В результате тестирования инструмента путем оценки алгоритмом BLEU [19] и опроса экспертов о схожести сгенерированного сообщения с реальным сообщением в тренировочном множестве был сделан вывод о необходимости доработки модели: из 983 сообщений, сгенерированных моделью, по шкале от нуля до семи 248 сообщений было оценено в 0 баллов, а 234 сообщения — в 7 баллов. Результатом тестирования стало добавление в модель фильтра контроля качества, который валидирует генерируемые сообщения. По словам авторов инструмента [14], такой фильтр снизил количество “плохих” сообщений на 44% ценой снижения количества “хороших” сообщений на 11%.

В последнее время набирает популярность языковая модель BERT. Разработанная компанией Google в 2018 году, модель отлично справляется с обработкой исходного кода программ [20], решая такие задачи, как локализация и устранение неправильно использованных переменных, и генерация комментариев к методам. По результатам проведенных оценок, модель BERT называют “state-of-the-art” — то есть моделью, показывающей лучшие результаты на всех популярных бенчмарках обработки естественного языка [21].

Свежие исследования также показывают, что модель справляется с генерацией сообщений об изменениях [22]. По мнению исследователей, метод, основанный на нейронной машинной трансляции, плохо справляется с поставленной задачей по причине различий в **контекстуальном представлении** естественных языков и языков программирования, и для решения проблемы автоматической генерации сообщений об изменениях разрыв между этим доменами необходимо снизить.

Для этих целей было принято решение использовать модель codeBERT [23] — вариант модели BERT, обученный и настроенный для работы с исходным кодом на языках Python, Java, JavaScript, Go, Ruby, PHP). Причем в архитектуре кодировщик-декодер (encoder-decoder) модель codeBERT заняла место кодировщика.

Также авторы [14] решили, что изменению подлежат еще и входные данные модели. Дело в том, что результат команды git diff (который применялся в качестве входных данных в описанной выше работе) генерирует много лишней информации в контексте натуральных языков. Было принято решение сократить эту информацию до последовательностей вида $X = ((add_1, del_1), \dots, (add_n, del_n)$, то есть только до наборов информации о удаленных и внесенных изменениях.

При обучении модели такие последовательности изменений передавались на вход кодировщика. На вход декодера, следуя модели NMT, передавались сообщения коммитов. Архитектура модели представлена на рис.5.

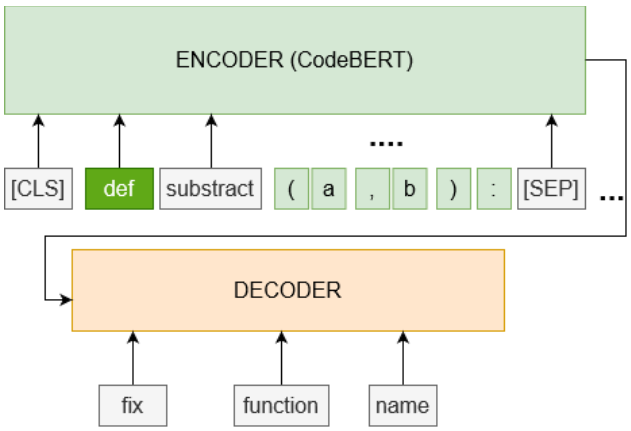

Рис.5. Архитектура описываемой модели с входами кодировщика и декодера.

В результате авторы сделали вывод [14], что применение модели codeBERT в качестве кодировщика в архитектуре NLP, а также обучение модели на корректном наборе данных, снижает разрыв в контекстуальном представлении между естественными языками и языками программирования. Оценка методов по алгоритму BLEU также показала повышение эффективности решения задачи генерации сообщений об изменениях.

IV. Заключение

Автоматическое создание высококачественных комментариев к вносимым изменениям является сложной задачей. Современные достижения в области нейронных сетей, а именно нейронном машинном переводе, позволяют автоматизировать эту задачу, что подтверждают описанные в работе исследования. Тем не менее, качество генерируемых таким образом сообщений позволяет желать лучшего, что открывает путь потенциальным исследованиям.

Библиография

[1] Otte S. Version control systems //Computer Systems and Telematics. – 2009. – С. 11-13;
[2] Ruparelia N. B. The history of version control //ACM SIGSOFT Software Engineering Notes. – 2010. – Т. 35. – №. 1. – С. 5-9
[3] История систем управления версиями // Хабр URL: https://habr.com/ru/post/478752/ (дата обращения: 07.12.2021);
[4] Rochkind M. J. The source code control system //IEEE transactions on Software Engineering. – 1975. – №. 4. – С. 364-370;
[5] Chapter 5 SCCS Source Code Control System // Oracle docs URL: https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dhp/index.html (дата обращения: 07.12.2021).
[6] Buse R. P. L., Weimer W. R. Automatically documenting program changes //Proceedings of the IEEE/ACM international conference on Automated software engineering. – 2010. – С. 33-42.
[7] 2021 Developer Survey // Stack Overflow URL: https://insights.stackoverflow.com/survey/2021#overview (дата обращения: 13.12.2021).
[8] Tsitoara M. Git Best Practices //Beginning Git and GitHub. – Apress, Berkeley, CA, 2020. – С. 79-86.
[9] Dyer R. et al. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories //2013 35th International Conference on Software Engineering (ICSE). – IEEE, 2013. – С. 422-431.

[10] Cortés-Coy L. F. et al. On automatically generating commit messages via summarization of source code changes //2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation. – IEEE, 2014. – С. 275-284.
[11] Jiang S., McMillan C. Towards automatic generation of short summaries of commits //2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). – IEEE, 2017. – С. 320-323.
[12] Mockus A., Votta L. G. Identifying Reasons for Software Changes using Historic Databases //icsm. – 2000. – С. 120-130.
[13] Abram H. et al. On the naturalness of software //Proceedings of the 34th International Conference on Software Engineering. – 2012. – С. 837-847.
[14] Jiang S., Armaly A., McMillan C. Automatically generating commit messages from diffs using neural machine translation //2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). – IEEE, 2017. – С. 135-146.
[15] Alexandru C. V., Panichella S., Gall H. C. Replicating parser behavior using neural machine translation //2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). – IEEE, 2017. – С. 316-319.
[16] Sutskever I., Vinyals O., Le Q. V. Sequence to sequence learning with neural networks //Advances in neural information processing systems. – 2014. – С. 3104-3112.
[17] Cho K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation //arXiv preprint arXiv:1406.1078. – 2014.
[18] Hinton G. E., Salakhutdinov R. R. Replicated softmax: an undirected topic model //Advances in neural information processing systems. – 2009. – Т. 22. – С. 1607-1614.
[19] Papineni K. et al. Bleu: a method for automatic evaluation of machine translation //Proceedings of the 40th annual meeting of the Association for Computational Linguistics. – 2002. – С. 311-318.
[20] Feng Z. et al. Codebert: A pre-trained model for programming and natural languages //arXiv preprint arXiv:2002.08155. – 2020.
[21] Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing // Google AI Blog URL: https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html (дата обращения: 15.12.2021).
[22] Jung T. H. CommitBERT: Commit Message Generation Using Pre-Trained Programming Language Model //arXiv preprint arXiv:2105.14242. – 2021.
[23] Feng Z. et al. Codebert: A pre-trained model for programming and natural languages //arXiv preprint arXiv:2002.08155. – 2020.

About automatic generation of commit messages in version control systems

I.A. Kosyanenko, R.G. Bolbakov

Abstract — In the modern world, no software development project is complete without the use of version control systems to track changes in data. The version control system is especially important in teamwork, when work on different parts of the same project is delegated to several developers. It is necessary to describe the changes made with their context in a relevant way, while fitting them into the restrictions set by the version control system itself (for example, git allows you to put only 72 characters in a commit message). Nevertheless, the process of describing or commenting on changes is still a routine, non-automated task, which developers often do not pay due attention to. The use of modern methods of natural language processing (or other models and methods that allow generating descriptions of changes in natural language based on the source code) can help developers save time on writing up-to-date messages to commits and maintain the correctness of the repository of source code versions.

Keywords - automatic generation of commit messages, source code processing, version control systems, commit messages, change messages, git.

REFERENCES

[1] Otte S. Version Control Systems //Computer systems and telematics. - 2009. - pp. 11-13.

[2] Ruparelia N. B. The history of version control //ACM SIGSOFT Software Engineering Notes. - 2010. - Vol. 35. - No. 1. - pp. 5-9;

[3] History of version control systems // Habr URL: https://habr.com/ru/post/478752 / (accessed: 07.12.2021).

[4] Rochkind M. J. The source code control system //IEEE transactions on Software Engineering. - 1975. - No. 4. - pp. 364-370

[5] Chapter 5 SCCS Source Code Control System // Oracle docs URL: https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dhp/index.html (accessed: 07.12.2021).

[6] Buse R. P. L., Weimer W. R. Automatically documenting program changes //Proceedings of the IEEE/ACM international conference on Automated software engineering. - 2010. - pp. 33-42.

[7] 2021 Developer Survey // Stack Overflow URL: https://insights.stackoverflow.com/survey/2021#overview (acessed: 13.12.2021).

[8] Tsitoara M. Git Best Practices //Beginning Git and GitHub. – Apress, Berkeley, CA, 2020. – pp. 79-86.

[9] Dyer R. et al. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories //2013 35th International Conference on Software Engineering (ICSE). – IEEE, 2013. – pp. 422-431.

[10] Cortés-Coy L. F. et al. On automatically generating commit messages via summarization of source code changes //2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation. – IEEE, 2014. – pp. 275-284.

[11] Jiang S., McMillan C. Towards automatic generation of short summaries of commits //2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). – IEEE, 2017. – pp. 320-323.

[12] Mockus A., Votta L. G. Identifying Reasons for Software Changes using Historic Databases //icsm. – 2000. – pp. 120-130.

[13] Abram H. et al. On the naturalness of software //Proceedings of the 34th International Conference on Software Engineering. – 2012. – pp. 837-847.

[14] Jiang S., Armaly A., McMillan C. Automatically generating commit messages from diffs using neural machine translation //2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). – IEEE, 2017. – pp. 135-146.

[15] Alexandru C. V., Panichella S., Gall H. C. Replicating parser behavior using neural machine translation //2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). – IEEE, 2017. – pp. 316-319.

[16] Sutskever I., Vinyals O., Le Q. V. Sequence to sequence learning with neural networks //Advances in neural information processing systems. – 2014. – pp. 3104-3112.

[17] Cho K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation //arXiv preprint arXiv:1406.1078. – 2014.

[18] Hinton G. E., Salakhutdinov R. R. Replicated softmax: an undirected topic model //Advances in neural information processing systems. – 2009. – Vol. 22. – pp. 1607-1614.

[19] Papineni K. et al. Bleu: a method for automatic evaluation of machine translation //Proceedings of the 40th annual meeting of the Association for Computational Linguistics. – 2002. – pp. 311-318.

[20] Feng Z. et al. Codebert: A pre-trained model for programming and natural languages //arXiv preprint arXiv:2002.08155. – 2020.

[21] Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing // Google AI Blog URL: https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html (accessed: 15.12.2021).

[22] Jung T. H. CommitBERT: Commit Message Generation Using Pre-Trained Programming Language Model //arXiv preprint arXiv:2105.14242. – 2021.

[23] Feng Z. et al. Codebert: A pre-trained model for programming and natural languages //arXiv preprint arXiv:2002.08155. – 2020.