Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Жуков Павел Николаевич

Построение вопросно-ответной системы на основе программного обеспечения и его артефактов

КУРСОВАЯ РАБОТА

Научный руководитель:
к.ф.-м.н., доцент Головин Игорь Геннадьевич

Москва, 2020

Содержание

1 Введение

В наши дни разработка программного обеспечения является ключевой активностью, необходимой большей части сфер жизни для существования и эффективного функционирования, будь то наука, медицина или коммерция. С увеличением сложности и роли программного обеспечения в нашей жизни развиваются и методы его разработки.

Во-первых, за последние десятилетия появилось множество программных средств, облегчающих ведение процесса разработки. Список включает системы управления версиями, позволяющие координировать работу разработчиков и отслеживать изменения в файлах; крупные веб-сервисы для хостинга проектов и совместной разработки; системы отслеживания ошибок, позволяющие учитывать и контролировать возникающие в программах ошибки и неполадки, пожелания пользователей, а также следить за процессом работы над этими ошибками и пожеланиями; различные средства для непрерывной интеграции, доставки и развёртывания, много из которых появилось за последние годы; и многое другое. В результате работы этих систем получаются т.н. артефакты — побочные продукты производства программного обеспечения.

Во-вторых, проникновение информационных технологий в большинство сфер жизни привело к появлению различных методологий разработки и, соответственно, профессий, не требующих понимание кода вообще или на уровне разработчика: специалисты по ручному и автоматизированному тестированию, функциональные и бизнес-аналитики, менеджеры проектов, DevOps-инженеры.

Существование таких профессий и систем, производящих полезную информацию о проекте, подразумевает, что должен быть способ для далёких от разработки людей получать знания о технических аспектах проекта, собранные в один источник информации, удобный в использовании. Такой способ также мог бы быть использован разработчиками, которые только присоединились к команде и ещё не знакомы со всеми деталями проекта.

2 Постановка задачи

Целью работы является разработка вопросно-ответной системы, которая бы позволяла узнавать информацию о программном обеспечении, изначально не доступную на естественном языке. В частности необходимо иметь возможность расширения такой системы для извлечения информации из артефактов программного обеспечения.

3 Обзор существующих решений

3.1 Широко используемые решения

Обычно в качестве способа получения информации о проекте в удобном виде выступают wiki-порталы с документацией, которую разработчики должны заполнять и поддерживать самостоятельно, или вышеупомянутые системы отслеживания ошибок. Также существуют программные средства, такие как Doxygen или javadoc, предоставляющие простой способ создавать документацию в формате html-страниц на основе специальных комментариев в коде.

Тем не менее, эти средства имеют свои недостатки, которые не позволяют удовлетворить все возникающие потребности пользователей.

Системы отслеживания ошибок работают по модели создания заявки, её обсуждения, прогресса и закрытия. Это означает, что в тот момент, когда заявка закрыта, уже нельзя с полной уверенностью сказать, актуальна ли та информацию, которую оттуда можно почерпнуть.

С автоматически генерируемой документацией проблема другая. В формате html-страниц она не подразумевает каких-либо правок, обсуждений и интерактива в целом, что необходимо для координации различных участников проекта. Её возможности строго ограничены тем функционалом, который предоставляется соответствующей утилитой. Таким образом, её не получится расширить с использованием других артефактов, кроме программного кода, а также нетехническому человеку всё так же непросто с ней работать.

Основным средством, которое потенциально могло бы решить задачу предоставления актуальной технической информации о проекте в понятной форме, являются вики-системы. Как правило, они имеют большинство возможностей, необходимых для комфортной работы — поисковый движок, комментарии, историю версий, средства для представления информации в удобочитаемом виде и другие. Но есть и коренные недостатки самой концепции таких систем.

Так как предполагается, что страницы создаются и заполняются вручную, это сразу приводит к проблеме поддержки такой документации. Информация об активно развивающемся программном проекте имеет свойство быстро терять свою актуальность, а время, расходуемое на поддержку внутренней документации, может стоить нескольких невыпущенных технических улучшений самого проекта.

Также, как уже было упомянуто выше, современный процесс разработки часто связан с использованием множества вспомогательных средств, каждое из которых создаёт свои артефакты. Интеграция таких побочных продуктов, например, данных git-

репозитория, в вики-системы обычно не представляется возможной и нецелесообразна с точки зрения потраченных усилий и получаемого результата. Это препятствует созданию единой базы знаний, в которой можно было бы найти актуальную информацию о разрабатываемом программном обеспечении.

3.2 Система Hipikat

Наиболее похожим на поставленную тему исследований по предназначению является проект Hipikat [1], разработанный в 2005 г. Его целью было создание «памяти проекта», как называли её авторы, заключающейся в объединении данных нескольких типов артефактов разработки: кода, документации, переписки, отчётов об ошибках и планах по тестированию. Далее Hipikat должен был рекомендовать разработчику релевантные его задаче артефакты.

Несомненно, создателями этой системы была проделана внушительная работа. Разработанный прототип позволял по существующей заявке в системе контроля ошибок или по файлу с кодом получить список релевантных артефактов. Авторы приводят показательный пример, в котором разработчик, решая задачу исправления дефекта в коде, запускает поиск по номеру решаемой заявки, а система рекомендует ему другие заявки об ошибках, наиболее релевантные решаемой. В одной из них похожая проблема затрагивает некоторый файл, и в результате дальнейшего исследования проблемы выясняется, что ошибка снова находится в этом файле с кодом. Таким образом система Hipikat помогает сузить диапазон поиска ошибки. Хоть приведённый пример использования вряд ли бы действительно помог опытному разработчику, которому такие задачи целесообразнее решать чтением кода, в остальном система действительно может быть полезной, рекомендуя к исследованию старые баги и обсуждения. Всё это интегрируется в среду разработки и использует удалённый сервер из нескольких модулей для обработки запросов.

Тем не менее, предложенное решение не получилось бы применить в исследуемой задаче, даже если бы оно было в открытом доступе. Во-первых, концепция системы заключается в том, чтобы по одному артефакту найти связанные с ним другие, т.е. пользователю изначально нужно иметь на руках некоторую сущность для возможности поиска, а затем самостоятельно, на основе набора предоставленных источников, искать ответы на свои вопросы. Во-вторых, система Hipikat полностью ориентирована на новых в проекте разработчиков, подразумевая, что у пользователя немалый технический уровень — именно поэтому авторами не ставилась задача создания подобия вопросно-ответной системы. В-третьих, система не пытается осуществлять глубокий анализ имеющихся артефактов для самостоятельного извлечения из них различной информации. То есть для большей части проблем понадобится отдельный, углублённый анализ да-

же в рамках одного предоставленного в качестве ответа релевантного артефакта. И, наконец, в-четвёртых, эта система слишком завязана на использование нескольких сервисов и программных средств, чего хотелось бы избежать для создания общедоступного решения.

3.3 Выводы

Таким образом, ни одно из используемых в жизни или предложенных в прошлом решений не позволяет и не ставит своей целью эффективно решить проблему поиска актуальной информации о проекте с помощью вопросов на естественном языке. Они либо сложны в поддержке, не гибкие и не подходят к задаче, либо требуют высокий технический уровень при использовании, что в рамках текущей задачи нивелирует их остальные достоинства.

4 Исследование и построение решения задачи

4.1 Нейронная сеть BERT

В результате анализа современных достижений в области вопросно-ответных систем было выявлено, что за последние годы со значительным отрывом от своих предшественников в задачах распознавания естественного языка побеждает нейронная сеть BERT [2] от компании Google. Именно она в данный момент применяется в соответствующей поисковой системе для улучшения понимания 10% запросов на английском языке. В конце 2018 г. компания Google открыла исходный код BERT и опубликовала несколько моделей [3], предобученных на стенфордском наборе данных для вопросов и ответов [4] (англ. The Stanford Question Answering Dataset, SQuAD), составленном на основе большого количества вопросов к статьям с ресурса Wikipedia. Преимуществом этой нейронной сети перед остальными является то, что она использует двунаправленное обучение для языковых моделей, т.е. позволяет понимать контекст слова на основе окружающих его слов, а не только на основе слов с одной стороны предложения.

Так как это решение является лидирующим, было решено в данной задаче попытаться использовать предобученную модель BERT в связке с информацией, извлекаемой из артефактов программного обеспечения.

4.2 Предлагаемый способ представления извлечённой из артефактов информации

Вышеупомянутая нейронная сеть для распознавания естественного языка работает с текстами, из чего логически следует, что если в неё нужно передать некоторую информацию, требуется составить некоторый связный текст. Проблема здесь заключается в том, что данные, извлекаемые из артефактов программного обеспечения, в основном представляются с помощью каких-либо ключевых слов или значений. Таким образом, необходимо определиться с тем, как представлять те или иные извлекаемые сущности таким образом, чтобы текст, полученный в результате, был «понятен» нейронной сети.

4.2.1 Извлекаемые сущности

В качестве двух главных источников информации были выбраны непосредственно программный код и git-репозиторий, используемый в его разработке. Решение мотивировано тем, что именно в них содержится труднодоступная информация, в то время как остальные артефакты, такие как отчёты об ошибках и другие, в основном связаны с информацией на естественном языке, для поиска которой часто можно исполь-

зовать встроенный поисковый движок соответствующей системы. В случае кода и git-репозитория неподготовленный человек не имеет никакой возможности извлечь информацию о них, так как для этого необходимо иметь навыки чтения кода или работы с утилитами git.

В качестве целевого языка программирования для извлечения информации был выбран язык Java. Можно выделить несколько основных сущностей, которые образуют программный код на этом языке, — это пакет, файл, класс, поле, метод. Соответственно, генерируемый текст должен основываться именно на их описании.

Для этих сущностей можно извлечь большое количество полезной информации, которой нет в доступном виде. Например, можно использовать сообщения коммитов, в которых были добавлены те или иные классы или методы, потому что скорее всего они несут релевантную информацию. Можно также извлекать имя, почтовый адрес создателя или дату создания и множество других параметров. Если говорить о самом программном коде, то в нём было бы интересно анализировать константы и внешние параметры конфигурации, потому что они часто несут в себе различные ограничения или значения, которые играют большую роль. Документация в коде, при её наличии, также будет чрезвычайно полезна в данной задаче, так как она изначально представлена на почти естественном языке, а это значит, что её можно добавить в генерируемый текст почти как есть, при этом получив много полезной информации и предложений, по которым нейронная сеть может искать ответ. Аналогичное можно сказать про сообщения исключений, поднимаемых в коде. Часто это те места программного кода, которые имеют свойство доходить до пользователя системы в виде ошибки, а значит, теоретически могут его заинтересовать и навести на какой-либо вопрос.

Было решено начать с предоставления пользователю возможности на естественном языке спросить о том, кто реализовал метод, отвечающий за ту или иную функцию. То есть название метода не должно фигурировать как таковое, вследствие его неизвестности пользователю системы. Например, подразумевается, что в результате вопроса пользователя: «Who did implement a method for sending nickname after it is changed?» (рус. «Кто реализовал метод для посылки имени пользователя после того, как оно изменено?») ответом должен служить создатель метода под названием «sendChangedNicknameMessage». Эта задача позволяет возвести каркас не только для разбора файлов с исходным кодом, но и для извлечения данных git-репозитория. Также эту информацию без сомнений можно назвать труднодоступной, потому что для её получения необходимо найти метод, реализующий запрашиваемый функционал, а затем пройти по дереву git-коммитов и найти тот, в котором он создаётся, что вручную делать трудозатратно даже подкованному пользователю.

4.2.2 Представление имён методов

В результате множества экспериментов был придуман способ преобразования метода в описывающий его текст на естественном языке. Сначала описываются характеристики метода, которые бы хотелось извлекать с помощью вопросов. Затем название метода делится на составные части, приводится к виду обычного предложения и вставляется после ключевой фразы «Its purpose is» (рус. «Его цель заключается в»). Таким образом, если метод осмысленно назван, это позволит нейронной сети сопоставить вопрос на естественном языке с представлением этого метода на естественном языке.

К примеру, для метода из прошлого подраздела, будет создан следующий текст: «Method «sendChangedNicknameMessage» was implemented by Dmitry Lyukov. Its purpose is send changed nickname message».

4.3 Описание работы программы

Этап 1. Создаётся «слепок» текущего функционала, присутствующего в коде. Программа проходит по всем файлам с исходным кодом проекта и анализирует их с помощью построения синтаксических деревьев и извлечения необходимых данных. Полученная информация преобразуется в удобные для задачи представления файлов, методов и т.д.

Этап 2. Анализируются данные git-репозитория методом прохода по коммитам в порядке удаления даты создания в прошлое. В результате, на основе анализа изменений каждого из файлов, затронутых в коммитах, собирается список добавленных методов и соответствующих им коммитов.

Этап 3. Данные, полученные в результате двух предыдущих этапов, объединяются. Подразумевается, что основными служат данные о текущем состоянии программного проекта, а то, что получено на более поздних этапах, «обогощает» их. Таким образом, при добавлении возможности извлечения данных из других типов артефактов для объединения этой информации с исходным слепком потребуется лишь создать ещё один «обогощающий» метод.

Этап 4. Объединённые данные представляются в виде связного текста на английском языке — в формате абзаца на каждый файл по описанному в прошлом разделе способу.

Этап 5. Полученный текст вместе с вопросом пользователя передаётся на вход нейронной сети, которая предоставляет ответ с некоторой соответствующей ему точностью.

Программа была реализована на языке Python 3.7 для возможности удобной интеграции с обученной моделью. Для извлечения данных git-репозитория использовалась библиотека pydriller [5]. Для построения синтаксических деревьев Java-исходников использовалась библиотека javalang [6].

5 Эксперименты

Эксперименты проводились на учебном Java-проекте в котором принимало участие более десяти человек. В нём содержится 37 файлов с кодом, исключая те, в которых присутствуют тесты. Из них 25 файлов содержит реализацию классов, т.е. описываемые сущности не являются интерфейсами. Из этих файлов было извлечено 56 методов. У 26 из этих 56 методов названия несли некоторую полезную информацию, т.е. методы не назывались «main», «run» и аналогично.

В итоге на естественном языке удалось узнать 85% создателей таких методов, т.е. 22 из 26 штук. Проблемы обнаружились в довольно очевидных местах. Во-первых, так как не учитывались имена файлов и классов, в которых содержались методы, в некоторых случаях произошли коллизии имён. Во-вторых, при недостаточно развёрнутом вопросе происходили ошибки с выбором похожего по названию метода.

Обе эти проблемы возникли в результате того, что для описания методов использовалось малое количество информации. Так как в будущем планируется добавление других признаков, по которым можно будет дополнить описания методов и других сущностей, ожидается, что эта проблема будет устранена. Также имеет смысл предоставлять не просто односложный ответ на поставленный вопрос, но и некоторую дополнительную информацию, связанную с найденной сущностью. Это позволило бы избежать мелких уточняющих вопросов пользователя в его надежде на то, что вопросно-ответная система хранит контекст его запросов. В целом же, полученные результаты вселяют надежду на то, что идея интеграции BERT и артефактов разработки программного обеспечения имеет смысл.

6 Заключение

В данной работе впервые был предложен и апробирован метод автоматического построения вопросно-ответной системы на основе артефактов разработки программного обеспечения, в частности на основе файлов с программным кодом и git-репозитория. Также было предложено направление развития такой системы в будущем.

Список литературы

[1] Hipikat: a project memory for software development / D. Cubranic, G. C. Murphy, J. Singer, K. S. Booth // *IEEE Transactions on Software Engineering*. — 2005. — Vol. 31, no. 6. — Pp. 446–465.

[2] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova // *CoRR*. — 2018. — Vol. abs/1810.04805. http://arxiv.org/abs/1810.04805.

[3] *Devlin, Jacob*. Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing. — 2018. [HTML] (https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html).

[4] SQuAD: 100, 000+ Questions for Machine Comprehension of Text / Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang // *CoRR*. — 2016. — Vol. abs/1606.05250. http://arxiv.org/abs/1606.05250.

[5] *Spadini, Davide*. Python Framework to analyse Git repositories. — 2018. [HTML] (https://pydriller.readthedocs.io/en/latest/tutorial.html).

[6] *Thunes, Chris*. Pure Python Java parser and tools. — 2018. [HTML] (https://github.com/c2nes/javalang).