УДК 004.4, 681.3

П.В. Дударин, В.Г. Тронин, К.В. Святов, В.А. Белов, Р.А. Шакуров

ПОДХОД К ОЦЕНКЕ ТРУДОЕМКОСТИ ЗАДАЧ В ПРОЦЕССЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ[1]

Дударин Павел Владимирович, *окончил Ульяновский государственный университет, аспирант кафедры «Информационные системы» Ульяновского государственного технического университета. Имеет научные работы в области обработки данных с использованием нейронных сетей. [e-mail: p.dudarin@ulstu.ru].*

Тронин Вадим Георгиевич, *кандидат технических наук, окончил УлГТУ, доцент кафедры «Информационные системы» УлГТУ. Имеет научные работы в области экономики, финансов и информационных технологий. [e-mail: v.tronin@ulstu.ru].*

Святов Кирилл Валерьевич, *кандидат технических наук, окончил УлГТУ, декан факультета информационных систем и технологий УлГТУ. Имеет научные работы в области автоматизации процессов управления. [e-mail: k.svyatov@ulstu.ru].*

Белов Владимир Александрович, *окончил бакалавриат «Программная инженерия» УлГТУ, магистрант кафедры «Информационные системы» УлГТУ. Имеет статью по мониторингу работы компьютера. [e-mail: v.belov@ulstu.ru].*

Шакуров Роман Азатович, *окончил бакалавриат «Программная инженерия» УлГТУ, магистрант кафедры «Информационные системы» УлГТУ. Имеет статьи по мониторингу работы компьютера и разработке системы определения победителя киберсоревнований. [e-mail: r.shakurov@ulstu.ru].*

Аннотация

Процесс разработки программного обеспечения (ПО) активно изучается экспертами из различных областей науки с различных точек зрения. Тем не менее степень успешности проектов в области разработки ПО меняется незначительно и остается на уровне 50% соответствия первоначальным требованиям (бюджет, время и функциональность) для проектов среднего размера. Годовые финансовые потери в этой области в связи с этим исчисляются сотнями миллиардов долларов. Ввиду высокой сложности процесса разработки ПО, оценки трудоемкости возникающих задач содержат большие погрешности. При этом даже повсеместный переход к гибким и итеративным методикам разработки не решают указанную проблему. В данной работе показывается, что экспертная ретроспективная оценка трудоемкости решения задачи в процессе разработки ПО может быть аппроксимирована функцией, реализованной в виде нейронной сети, от определенного набора метрик сложности программного кода. Также описан процесс сбора необходимых метрик и обучения нейронной сети, позволяющий автоматизировать получение оценок трудоемкости задач, выполненных в ходе спринта в парадигме гибкой разработки ПО. Проведенные эксперименты на реальном функционирующем проекте по разработке ПО демонстрируют работоспособность и эффективность предлагаемого подхода.

Ключевые слова: процесс разработки программного обеспечения, нейронные сети, обогащение данных, метрика Холстеда, цикломатическая метрика.

doi: 10.35752/1991-2927-2019-3-57-65-72

AN APPROACH TO LABOR INTENSITY EVALUATION IN SOFTWARE DEVELOPMENT PROCESS BASED ON NEURAL NETWORKS

Pavel Vladimirovich Dudarin, *graduated from Ulyanovsk State University; Postgraduate Student at the Department of Information Systems of Ulyanovsk State Technical University; an author of scientific papers in the field of data processing by means of neural networks. e-mail: p.dudarin@ulstu.ru.*

1 Исследование выполнено при финансовой поддержке РФФИ и Правительства Ульяновской области в рамках научного проекта № 18-47-732005.

Vadim Georgievich Tronin, *Candidate of Science in Engineering; Associate Professor at the Department of Information Systems of UlSTU; an author of scientific papers in the field of economics, finance and information technologies. e-mail: v.tronin@ulstu.ru.*
Kirill Valerevich Sviatov, *Candidate of Science in Engineering; graduated from UlSTU; Dean of the Faculty Information Systems and Technologies of UlSTU; an author of scientific papers in the field of automation of management processes. e-mail: k.svyatov@ulstu.ru.*
Vladimir Aleksandrovich Belov, *graduated from UlSTU with a bachelor degree in Software Engineering; Master Student at the Department of Information Systems of UlSTU; an author of an article in the field of computer operation monitoring. e-mail: v.belov@ulstu.ru.*
Roman Azatovich Shakurov, *graduated from UlSTU with a bachelor degree in Software Engineering; Master Student at the Department of Information Systems of UlSTU; an author of articles in the field of computer operation monitoring and developing of system for determining the winner in cyber security competitions. e-mail: r.shakurov@ulstu.ru.*

Abstract
Software development process is actively studied by experts from different spheres of science and different viewpoints. However, the degree of success of projects in the development of software intensive systems (Software Intensive Systems, SIS) has changed insignificantly, remaining at the level of 50% inconsistency with the initial requirements (finance, time and functionality) for medium-sized projects. The annual financial losses in the world because of the total failures are counted by hundreds of billions of dollars. Its high complexity leads to constant mistakes in labor intensity evaluation, and even new agile development paradigm does not solve this problem. This paper shows that retrospective labor intensity estimation could be approximated by a function, implemented by neural network, with some amount of code complexity metrics as arguments. Also there is a described an approach of neural network training and data collection, which allows to automate a process of retrospective labor intensity evaluation in agile software developing process. Experiments performed on the real life software project show the effectiveness of proposed technique.

Key words: software developing process, neural network, data augmentation, Halstead metrics, Cyclomatic metric.

Введение

Процесс разработки программного обеспечения (ПО) активно изучается различными областями науки с различных точек зрения [1, 2]. ПО пронизывает все сферы нашей жизни [3]. Тем не менее, согласно исследованиям Standish Group в период 1992–2017 годов, степень успешности проектов в области разработки ПО изменилась незначительно и находится на уровне 50% соответствия первоначальным требованиям (финансовым, временным и функциональным) для проектов среднего размера. Ежегодные финансовые потери, связанные с ошибками в разработке ПО, исчисляются сотнями миллиардов долларов.

В связи с большой сложностью процесса создания ПО стандартный подход к процессу разработки, известный как «водопад», сменяется различными «гибкими» (agile) техниками, например «скрам» или «канбан». Эти техники позволяют разрабатывать большие и сложные системы шаг за шагом. Каждый шаг включает в себя полный цикл разработки: планирование, кодирование, тестирование и анализ. На этапе планирования задач (в рамках спринта) разработчики оценивают задачи в баллах трудоемкости (story points). Это экспертная оценка, и ее качество зависит от уровня знаний разработчика, сработанности команды, мотивации и ряда других факторов. Исходя из проставленных оценок трудоемкости и статистической мощности команды (количество баллов трудоемкости, выполняемой в среднем за спринт) задачи берутся в работу. Ошибки в оценке приводят к перегруженности или недозагруженности команды, к неравномерной нагрузке на отдельных членов команды, что, в конечном счете, сказывается на общем качестве разрабатываемой системы.

На последнем этапе – этапе анализа – в «гибких» методологиях разработки проводится ретроспективный анализ, включающий в себя ретроспективный анализ кода. Этот анализ проводится для определения качества кода, возможности его улучшения или оптимизации. Также в рамках этого анализа осуществляется сверка оценки трудоемкости, проставленной при планировании задачи, с фактической трудоемкостью разработки. Сравнение проводится с целью определения причин ошибки в оценке и улучшения качества оценки в будущем. Ретроспективный анализ должен проводиться экспертом, т. е. является, с одной стороны, дорогостоящей операцией, а с другой стороны – достаточно рутинным процессом, т. е. демотивирующим для специалиста высокой квалификации. В итоге на реальных крупных проектах тщательная работа по сравнению планируемой и фактической оценок практически не проводится.

В такой ситуации автоматизированный инструмент, выполняющий ретроспективную оценку, мог бы быть очень полезен. В данной работе показывается, что с определенными ограничениями экспертная ретроспективная оценка трудоемкости решения задачи может быть аппроксимирована функцией, реализованной в виде нейронной сети, от определенного набора метрик сложности программного кода. Для разработчиков данный инструмент может служить способом самопроверки и обучения для более корректной оценки будущих

задач, показывая недооцененные или переоцененные задачи. Такие задачи должны обсуждаться на ретроспективе спринта. Лидеру команды предложенный инструмент может помочь в индивидуальной оценке членов команды. А для руководителя проекта это инструмент оценки продуктивности команды в целом, без необходимости погружения в детали выполненных задач и особенностей процесса разработки. Также различные команды одного проекта могут быть сравнимы по предложенной оценке суммарной трудоемкости.

Дальнейшее содержание работы организовано следующим образом: вначале идет обзор существующих подходов к оценке сложности кода; далее в деталях описан предлагаемых подход; после чего представлены результаты экспериментов; и финальная секция заключает работу с выводами и вариантами дальнейшего развития исследования.

Обзор метрик сложности кода

Стандартным и распространенным подходом к ретроспективной оценке трудоемкости задачи является экспертная оценка. В данном подходе один или несколько экспертов выдвигают свои предположения о трудоемкости разработки, опираясь на свои знания и опыт. Обычно данную работу выполняют руководители команд разработки (сотрудники с большим опытом работы в данной области).

Однако даже при использовании данной методики получаемая оценка является субъективной, даже при условии, что она получена в результате обсуждения группой специалистов. Так если бы оценка проводилась в какой-либо другой день или в ней бы принимали участие другие специалисты, то результат оценки будет различаться. При этом такая оценка стоит больших затрат для проекта, ведь вместо выполнения текущих задач группа ведущих специалистов потратит свое время на оценку.

Альтернативным подходом может быть автоматическая оценка, основанная на анализе реализованного программного кода. Разработанную программу можно оценить по формальным признакам – метрикам кода [4]. Это позволяет определить качество, сложность, трудоемкость написанного кода. Далее приведен набор наиболее распространённых классов метрик и их представителей [5, 6].

Количественные метрики – метрики, подсчитывающие количество различных компонентов программы. Поскольку данный класс является самым простым в реализации, данные метрики одними из первых были использованы для расчета трудоемкости проектов. К количественным метрикам относятся:

- SLOC – количество строк кода,
- количество комментариев,
- процент комментариев – отношение количества комментариев ко всему объему кода,
- среднее число строк в методах и классах,
- количество методов и полей класса.

Главный недостаток данных метрик в том, что одна и та же бизнес-функция может быть реализована различным количеством строк кода. Поэтому корреляция между сложностью кода и сложностью задачи слабая.

К количественным метрикам также можно отнести метрики Холстеда, которые вычисляются на основе количества операторов и операндов в исходном коде. На основе данной метрики можно провести следующие оценки:

- Halstead Vocabulary – словарь программы,
- Halstead Length – длинна программы,
- Halstead Volume – объем программы,
- Halstead Difficulty – сложность программы,
- Halstead Effort – умственные затраты программиста на создание кода,
- Halstead Operators Total / Distinct – количество операторов всего / уникальных,
- Halstead Operands Total / Distinct – количество операндов всего / уникальных.

Основным преимуществом метрик Холстеда перед остальными количественными метриками является относительная независимость к стилю написания кода. Но все операторы обрабатываются одинаково и сложность потока управления не оценивается.

Метрики сложности потока управления программы – данный класс метрик основан на анализе управляющего графа программы. Одной из самых распространенных оценок данного графа является цикломатическая сложность программы (Cyclomatic [7]). В метрике вычисляется количество независимых маршрутов, которые можно провести в графе, построенном по программному коду. Количество маршрутов увеличивается с каждым условным оператором или циклом. Главный недостаток данной оценки – это отсутствие различий между условными конструкциями и циклами. Также в метрике не учитывается количество и сложность предикатов в условии.

Метрики сложности потока управления данными – данный класс метрик основан на анализе потока ввода-вывода программы. К данному классу относится метрика Чепина в которой оценивается информационная прочность программного модуля или класса с помощью анализа характера использования списка переменных. Список переменных разбивается на следующие множества:

- P – переменные ввода-вывода, требуемые для расчетов;
- M – модифицируемые или создаваемые внутри кода переменные;
- C – переменные, управляющие состоянием программного модуля;
- T – не используемые переменные.

Итоговый результат метрики оценивается по следующей формуле: $Q = A1 * P + A2 * M + A3 * C + + A4 * T$, где $A1, A2, A3, A4$ – весовые коэффициенты. По мнению автора, эти коэффициенты равны: 1, 2, 3, 0,5 соответственно.

Объектно-ориентированные метрики – данный класс метрик применяется для проектов, в которых используется объектно-ориентированный подход:

• WMC (Weighted Methods Per Class) – взвешенная насыщенность класса. Данная метрика отражает сложность класса на основе цикломатической сложности его методов. Чем больше методов и сложность их реализации, тем выше значение метрики для класса;

• WMC2 (Weighted Methods Per Class) – взвешенная насыщенность класса 2. Сложность класса вычисляется в зависимости от количества методов в классе, а также количества аргументов в методе;

• DIT (Depth of inheritance tree) – глубина дерева наследования. Метрика DIT предусматривает, что каждый класс является мерой уровней наследования от объекта вершины иерархии;

• NOC (Number of children) – метрика определяет количество классов, которые являются наследниками текущего класса;

• CBO (Coupling between objects) – метрика определяет количество классов, которые используют текущий класс или используются в нем;

• RFC (Response For Class) – метрика определяет количество методов, которые могут быть вызваны объектом класса.

Представленные метрики отлично отражают сложность реализации объектно-ориентированного приложения. Однако совершенно не применимы в иных подходах.

Все эти метрики имеют свои преимущества и недостатки. Каждая метрика в отдельности не оценивает весь комплекс факторов, влияющих на трудоемкость решения задачи. К сожалению, в ходе подготовки и проведения исследования, не удалось найти работы, в которых бы рассматривалась связь между оценкой трудоемкости решения задачи и метриками сложности программного кода. Ряд исследований посвящен аппроксимации описанных выше метрик с помощью нейронных сетей [8, 9], для того чтобы в дальнейшем генерировать более сложные и комплексные метрики. Следует также заметить, что эти исследования использовались на небольших или искусственно созданных фрагментах программного кода, а не на реальных коммерческих проектах. В данной работе преследуется иная цель: не генерация новых метрик оценки сложности кода, а аппроксимация оценки трудоемкости задачи по коду, ее решающему. Существующие метрики сложности кода используются как вектор признаков для нейронной сети, как наиболее универсального нелинейного функционала.

В качестве данных для анализа трудоемкости был использован частный проект, который реализован на языке type Script с помощью библиотеки для языка Java Script – React JS. Поскольку для разработки на языке Type Script используются лишь некоторые из инструментов объектно-ориентированного подхода, для оценки проекта не подходят объектно-ориентированные метрики. Поскольку данный проект реализует клиентскую часть, то он не включает в себя сложные расчеты, преобразование и анализ данных, а служит для их запроса и отображения. В соответствии с представленными доводами было решено отказаться от метрик сложности управления данными. Для расчета входного вектора нейронной сети были выбраны следующие метрики: Cyclomatic, Cyclomatic Density, Param Count, Halstead Bugs, Halstead Difficulty, Halstead Effort, Halstead Length, Halstead Time, Halstead Vocabulary, Halstead Volume, Halstead Operators Total, Halstead Operators Distinct, Halstead Operands Total, Halstead Operands Distinct, Sloc Logical, Sloc Physical.

Методика оценки трудоёмкости

Как было показано выше, разные классы метрик учитывают разные аспекты сложности программного кода. Нелинейная функция от различных наборов формальных признаков, учитывающих разные стороны особенности кода, даст лучшие результаты по сравнению с экспертным и формальным подходами. Тем не менее, правильный выбор типа функции и ее коэффициентов не является тривиальной задачей, поскольку она зависит от многих факторов: языка программирования, стиля написания кода, принятого на проекте, специфики проекта и членов команды. Это приводит к необходимости обучения нейронной сети для каждого проекта. Для этого необходимо выполнить следующие шаги:

• извлечь историю по выполненным задачам с оцененной трудоемкостью из системы отслеживания задач;

• извлечь зависимые части программного кода;

• вычислить векторы метрик для всех частей кода, связанных с задачей;

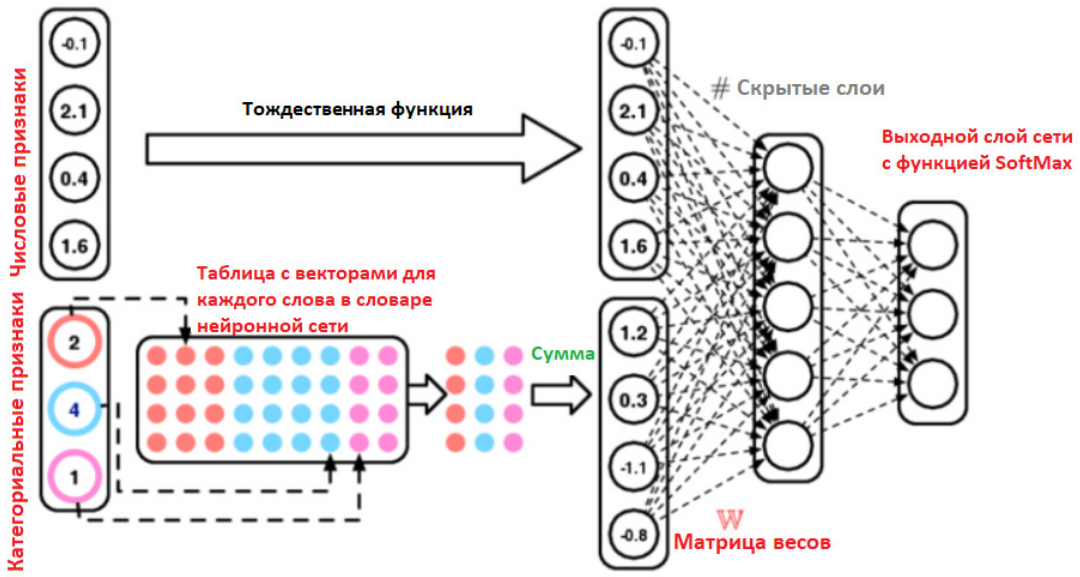
• выбрать архитектуру нейронной сети и построить модель;

• обучить нейронную сеть и проанализировать результат.

Все задачи, постановки, оценка по задачам в данном проекте находятся в системе отслеживания ошибок и управления проектом – JIRA. Оценка задач в данном проекте происходит в баллах трудоемкости (story points) – оценки объёма работы в задаче.

Для получения номеров задач, их описания, оценки, исполнителем задачи был реализован сервис интеграции с Jira. Сервис был реализован на языке Java с использованием библиотеки jira-rest-java-client. Далее потребовалось получить исходный код, реализованный для данной задачи.

Исходный код по проекту обычно хранится в системе контроля версий Git (например, на хостинге Bitbucket). Разработанный сервис был дополнен функционалом интеграции с репозиториями на Bitbucket. С помощью данного функционала были загружены все хэш-значения изменений в коде (коммитов) с их описанием из репозитория. Команда разработки придерживалась следующей модели описания изменений: «Номер задачи – что сделано». Соответственно хэш-значение изменения и задачи из Jira были сопоставлены по номеру задачи. Данный функционал был реализован с помощью




```
(0): ModuleList()
(1): Dropout(p=0.0)
(2): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(3): Linear(in_features=16, out_features=200, bias=True)
(4): ReLU(inplace)
(5): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(6): Linear(in_features=200, out_features=100, bias=True)
(7): ReLU(inplace)
(8): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(9): Linear(in_features=100, out_features=1, bias=True)
```

Рис. 1. Архитектура нейронной сети и вывод описания слоев в Jupyter Notebook

регулярных выражений и поиска по подстроке. Таким образом, был загружен исходный код, расположенный в хранилище Git.

Для каждой задачи все связанные файлы были оценены с помощью сервиса оценки сложности кода, написанного на Node JS с использованием библиотеки typhonjs-escomplex [10]. Выделение данного сервиса из общего приложения было необходимо как с архитектурной, так и с точки зрения производительности. В будущем данный подход позволит расширить список поддерживаемых языков программирования без изменения основного приложения. Кроме того, это позволит использовать более подходящие библиотеки и реализации, написанные для конкретного языка, с учетом его особенностей.

Для анализа частичных изменений в коде сначала оценивается сложность исходного кода (до внесения изменений). Затем анализируется сложность после внесенных изменений. Подобным образом обрабатываются все файлы, измененные в рамках задачи. Накопленное увеличение сложности является результатом оценки сложности программного кода для конкретной задачи.

В среднем разработчик решает от 1 до 3 задач в неделю, что означает, что стандартная команда из 5–7 человек даже после года работы будет иметь небольшое количество задач, подходящих для автоматической оценки. Таким образом, возникает необходимость в искусственном увеличении количества данных. Другой причиной увеличения размера выборки является низкое качество экспертных оценок. Это связано с тем, что оценка проводится во время рабочего процесса, когда команда загружена задачами разработки. В таких случаях используют методики обогащения данных, т. е. генерируют новые данные. Дополнительные объекты выборки с точной оценкой могут быть сгенерированы следующим образом: во-первых, эксперты оценивают некоторое количество образцов кода (10–20); затем новые задачи создаются в виде случайных наборов из оцененных образцов кода, а общая сложность кода является суммой оценок входящих в набор образцов. Такой подход позволяет генерировать столько образцов, сколько нужно, без необходимости в искусственной генерации кода.

Опыт разработки и эксперименты показывают, что задача с большим количеством баллов не может быть оценена правильно, поэтому такие задачи следует исключить из оценки и избегать в процессе разработки. Задача с большим количеством баллов должна быть разбита на множество небольших задач.

С точки зрения математического моделирования, задача оценки может рассматриваться как задача прогнозирования. Таким образом, из всего многообразия архитектур нейронных сетей, таких как RNN, CNN, GAN и др., была выбрана архитектура, предоставляемая фреймворком fastai [11] для прогнозирования продаж продуктов [12]. На рисунке 1 можно увидеть архитектуру

нейронной сети. Эта модель нейронной сети состоит из встроенных слоев для категориальных переменных, за которыми следуют Dropout и BatchNorm для непрерывных переменных. Результаты объединяются и сопровождаются блоками BatchNorm, Dropout, Linear и ReLU (в первом блоке отсутствуют BatchNorm и Dropout, в последнем блоке отсутствует ReLU). Используется функция активации – сигмоида. Сеть включает 200 нейронов на первом и 100 нейронов на втором внутреннем слое соответственно.

Результаты эксперимента

В качестве экспериментальных данных использовался код одного коммерческого проекта. В этом проекте один балл трудоёмкости примерно равен 3 часам работы разработчика. С помощью разработанного вспомогательного сервиса было загружено 1177 задач, но только 704 из них были оценены с помощью балльной оценки.

В процессе сравнения данных JIRA и BitBucket выборка была уменьшена до 303 задач, которые были оценены группой экспертов и имеют соответствующие изменения (коммиты) в хранилище кода (репозитории). В данном примере оценка задач проводилась в баллах в диапазоне от 0,5 до 46 единиц. Выборка содержала некоторое количество неправильно оцененных задач, которые необходимо было исключить для улучшения качества модели. Было решено исключить задачи с оценкой более 20 единиц, поскольку их количество было слишком маленьким, а разница в показателях между объектами аналогичной оценки варьировалась в довольно большом диапазоне. Также были исключены задачи с нулевой оценкой. После фильтрации выборки её размер был уменьшен до 281 объекта.

Для увеличения выборки был использован предложенный подход, с помощью которого было сгенерировано 1000 новых объектов.

Нейронная сеть была обучена с использованием сервиса Google Colab. В результате обучения нейронной сети функция потерь (сумма квадратов ошибок) в тестовой части выборки достигла значений менее 5 единиц. На рисунке 2 показаны тренировочные эпохи.

В качестве наглядного сравнения с результатами нейронной сети на рисунке 3 представлен прогноз трудоемкости, основанный только на цикломатической метрике. По оси X отложена экспертная оценка трудоемкости, по оси Y приведен усредненный прогноз, основанный на цикломатической метрике. В случае идеального прогноза на графике была бы линия y=x. Квадратичная ошибка этого прогноза равна 9,5.

На рисунке 4 показан график прогнозируемых значений баллов трудоёмкости с помощью нейронной сети, который является более точным в сравнении с предыдущим. В диапазоне [0,6] график близок к линии y = x, что означает идеальный прогноз. После отметки 12 баллов график стал нестабильным, как на рисунке 4. Причиной этого явления являются ошибки в работе экспертной группы при оценке больших и сложных задач. Эксперты наблюдаемой команды склонны переоценивать большие задачи и недооценивать огромные.

epoch	train_loss	valid_loss	epoch	train_loss	valid_loss	
0	45.247143	26.154900	25	12.417384	7.146481	00:00
1	42.605099	22.374788	26	12.173486	6.835426	00:00
2	40.612766	17.865685	27	11.935629	7.749971	00:00
3	37.896023	12.954546	28	11.729676	5.519372	00:00
4	35.434860	9.990437	29	11.420516	4.679532	00:00
5	32.487514	8.639280	30	11.226760	4.719009	00:00
6	29.802570	6.890329	31	11.083948	4.998270	00:00
7	27.047462	7.728254	32	11.068912	5.917696	00:00
8	24.737619	11.083056	33	10.913160	7.892910	00:00
9	22.838421	11.333658	34	10.802450	8.165093	00:00
10	21.319412	8.851075	35	10.614226	6.211092	00:00
11	20.144808	8.983871	36	10.525489	5.501328	00:00
12	19.075329	7.845511	37	10.413771	5.010170	00:00
13	18.146326	6.960416	38	10.300986	5.977239	00:00
14	17.273411	6.572458	39	10.200490	7.082429	00:00
15	16.644100	6.538685	40	10.131946	7.225183	00:00

Рис. 2. Эпохи обучения нейронной сети

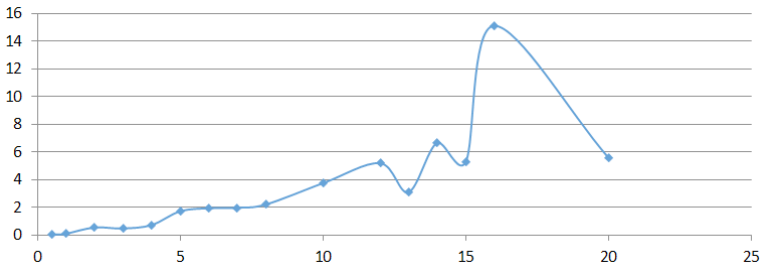


Рис. 3. Прогноз на основе цикломатической метрики

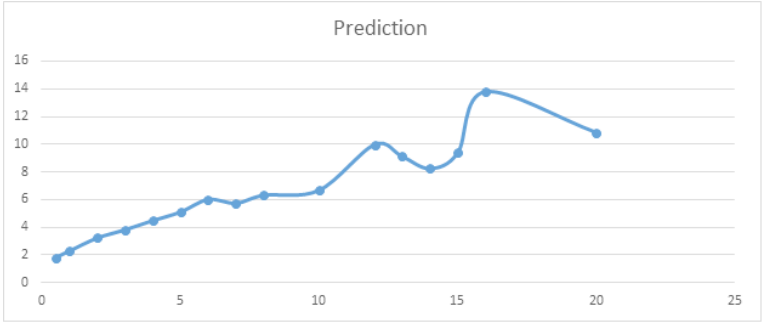


Рис. 4. Результаты прогнозирования с помощью нейронной сети

Заключение

В данной работе представлен подход к ретроспективной оценке трудоемкости решения задач, выполняемых в рамках процесса разработки программного обеспечения. Показано, что экспертная ретроспективная оценка трудоемкости решения задачи может быть аппроксимирована нейронной сетью от набора метрик сложности программного кода, среди которых использованы метрики Холстеда и метрика цикломатической сложности кода. На базе представленного подхода может быть разработан инструмент, полезный всем участникам команды разработки. Для разработчиков данный инструмент может служить способом самопроверки и обучения для более корректной оценки будущих задач, показывая недооцененные или переоцененные задачи. Лидеру команды предложенный инструмент может помочь в индивидуальной оценке членов команды. А для руководителя проекта это инструмент оценки продуктивности команды в целом, без необходимости погружения в детали выполненных задач и особенностей процесса разработки. Также различные команды одного проекта могут быть сравнимы по предложенной оценке суммарной трудоемкости.

Также предложенный подход помогает определить допустимые границы в оценке задачи экспертом, показывая какие задачи чаще всего недооценены или переоценены. Например, очевидно, что задачи, оцененные в высокую трудоемкость, чаще всего не достаточно декомпозированы и велик риск некорректной оценки. В проведенных экспериментах графики показали, что для данной команды максимальной разумной оценкой должна являться оценка в 12 баллов. Задачи с большей трудоемкостью имеют высокую погрешность в оценке.

В дальнейших работах предлагается обратить внимание на изменение оценок, выставляемых командой во времени. Команда проекта эволюционирует: приходят новые члены команды, уходят старые, происходит профессиональный рост членов команды. Качество оценки кода может характеризовать саму команду с точки зрения сработанности, профессионализма и т. п.

СПИСОК ЛИТЕРАТУРЫ

1. An Approach to Similar Software Projects Searching and Architecture Analysis Based on Artifcial Intelligence Methods / Nadezhda Yarushkina, Gleb Guskov, Pavel Dudarin, Vladimir Stuchebnikov // Proceedings of the Third International Scientifc Conference Intelligent Information Technologies for Industry (IITI'18). Advances in Intelligent Systems and Computing. Springer, Cham, 2018. Vol. 1. pp. 341–352.

2. Власов И.В., Дударин П.В., Юсупов А.Д. Опыт разработки программного обеспечения для интеграции информационных систем при производственной деятельности эксплуатанта авиационной техники // Известия Самарского научного центра Российской академии наук. – 2012. – Т. 14, № 4-2. – С. 577–582.

3. Дударин П.В., Пинков А.П., Ярушкина Н.Г. Методика и алгоритм кластеризации объектов экономической аналитики // Автоматизация процессов управления. – 2017. – № 1 (47). – С. 85–93.

4. Kamaljit Kaur, Kirti Minhas, Neha Mehan, Namita Kakkar. Static and Dynamic Complexity Analysis of Software Metrics // World Academy of Science, Engineering and Technology International Journal of Computer, Electrical, Automation, Control and Information Engineering. 2009. Vol. 3, no 8. pp. 1936–1938.

5. Ali Athar Khan, Amjad Mahmood, Sajeda M. Amralla and Tahera H. Mirza. Comparison of Software Complexity Metrics // International Journal of Computing and Network Technology. 2016. Vol. 4, iss. 1. pp. 19–25.

6. Esther Isola, Stephen Olabiyisi, Elijah Omidiora, Rafu Ganiyu. Performance evaluation of improved cognitive complexity metric and other code based complexity metrics // International Journal of Scientific and Engineering Research. 2012. Vol. 3, iss. 9.

7. Kemerer F., Gill K. Cyclomatic complexity density and software maintenance productivity // IEEE Transactions on Software Engineering. 1992. Vol. 17, iss. 12. pp. 1284–1288.

8. Sibel Senan, Selcuk Sevgen. Measuring software complexity using neural networks // Journal of Electrical and Electronics Engineering. 2017. Vol. 17(2). pp. 3503–3508.

9. Boetticher Gary K., Srinivas David, Eichmann. A Neural net-based Approach to Software Metrics // 4th International Conference on Tools With Artificial Intelligence, Arlington, Virginia, 1992. – URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930003204.pdf (дата обращения: 01.08.2019).

10. Coleman Don, Ash Dan, Lowther Bruce, Oman Paul. Using Metrics to Evaluate Software System Maintainability // Computer. IEEE Computer Society Press, Los Alamitos, CA, USA. 1994. Vol. 27, iss. 8. pp. 44–49.

11. FastaAI Neural Network Framework. – URL: https://www.fast.ai/about/ (дата обращения: 01.08.2019).

12. Rossmann Store Sales – Kaggle contest. – URL: https://www.kaggle.com/c/rossmann-store-sales (дата обращения: 01.08.2019).

REFERENCES

1. Yarushkina N., G. Guskov, P. Dudarin, V. Stuchebnikov. An Approach to Similar Software Projects Searching and Architecture Analysis Based on Artificial Intelligence Methods. *Proceedings of the Third International Scientific Conference Intelligent Information Technologies for Industry (IITI'18). Advances in Intelligent Systems and Computing.* Springer, Cham, 2018, vol. 1, pp. 341–352.

2. Vlasov I.V., Dudarin P.V., Iusupov A.D. Opyt razrabotki programmnogo obespecheniia dlia integratsii informatsionnykh sistem pri proizvodstvennoi deiatelnosti ekspluatanta aviatsionnoi tekhniki [Experience of Software Development for the Integration of Information Systems in the Production of the Aircraft Operator]. *Izvestiia Samarskogo nauchnogo tsentra Rossiiskoi akademii nauk* [Academic Journal Izvestia of Samara Scientific Center of

the Russian Academy of Sciences], 2012, vol. 14, no. 4-2, pp. 577–582.

3. Dudarin P.V., Pinkov A.P., Yarushkina N.G. Metodika i algoritm klasterizatsii obektov ekonomicheskoi analitiki [Methodology and the Algorithm for Clustering Economic Analytics Object]. *Avtomatizatsiia protsessov upravleniia* [Automation of Control Processes], 2017, no. 1 (47), pp. 85–93.

4. Kamaljit Kaur, Kirti Minhas, Neha Mehan, Namita Kakkar. Static and Dynamic Complexity Analysis of Software Metrics. *World Academy of Science. Engineering and Technology International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 2009, vol. 3, no. 8, pp. 1936–1938.

5. Ali Athar Khan, Amjad Mahmood, Sajeda M. Amralla and Tahera H. Mirza. Comparison of Software Complexity Metrics. *International Journal of Computing and Network Technology*, 2016, vol. 4, iss. 1.

6. Esther Isola, Stephen Olabiyisi, Elijah Omidiora, Rafu Ganiyu. Performance Evaluation of Improved Cognitive Complexity Metric and Other Code Based Complexity Metrics. *International Journal of Scientific and Engineering Research*, 2012, vol. 3, iss. 9.

7. Kemerer F., Gill K. Cyclomatic Complexity Density and Software Maintenance Productivity. *IEEE Transactions on Software Engineering*, 1992, vol. 17, iss. 12. pp. 1284–1288.

8. Sibel Senan, Selcuk Sevgen, Measuring Software Complexity Using Neural Networks. *Journal of Electrical and Electronics Engineering*, 2017, vol. 17, iss. 2, pp. 3503–3508.

9. Boetticher Gary K., Srinivas David, Eichmann. A Neural Net-based Approach to Software Metrics. *4th International Conference on Tools with Artificial Intelligence.* Arlington, Virginia. 1992. Available at: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930003204.pdf (accessed: 01.08.2019).

10. Coleman Don, Ash Dan, Lowther Bruce, Oman Paul. Using Metrics to Evaluate Software System Maintainability. *Computer.* IEEE Computer Society Press, Los Alamitos, CA, USA. 1994, vol. 27, iss. 8, pp. 44–49.

11. *FastaAI Neural Network Framework.* Available at: https://www.fast.ai/about/. (accessed: 01.08.2019).

12. *Rossmann Store Sales – Kaggle Contest.* Available at: https://www.kaggle.com/c/rossmann-store-sales (accessed: 01.08.2019).