Гибридный подход к анализу сложности программного кода с учетом метрик и методов искусственного интеллекта

М.А. Морозова, А.В. Морозов

Астраханский государственный технический университет, Астрахань, Россия

Аннотация **– Представлен гибридный метод оценки сложности программного кода, объединяющий классические метрики (цикломатическая сложность, метрики Холстеда, сцепление, дублирование, покрытие кода) и методы искусственного интеллекта. Разработан механизм нормализации и весовой оценки метрик с учетом специфики критически важного программного обеспечения. Предложены агрегированные показатели и пороговые значения для интерпретации качества. Обоснована необходимость дальнейшей интеграции инструментов машинного обучения в процесс метрикового анализа.**

Ключевые слова **– метрики, анализ сложности, методы ИИ.**

I. ВВЕДЕНИЕ

В работе [1] был сделан вывод, что комбинированный метод полезен для анализа сложности кода, однако подвержен быстрому устареванию в связи с появлением новых инструментов на основе искусственного интеллекта. Таким образом, появляется необходимость в конструировании нового метода, содержащего в себе привычные метрики Холстеда, цикломатическую сложность и менее используемые метрики как покрытие кода, дублирование кода, сложность классов, сцепление. Рассмотрим один из методов объединения данных метрик с методами искусственного интеллекта.

II. ОПИСАНИЕ ГИБРИДНОГО МЕТОДА АНАЛИЗА ПРОГРАММНОГО КОДА

Сначала определим цели в области качества относительно группировки метрик:

- Ремонтопригодность: цикломатическая сложность (сложность потока управления), метрики Холстеда (когнитивная сложность), сложность класса [2];
- Эффективность тестов: покрытие кода и цикломатическая сложность для выявления непроверенных путей с высоким риском [3];
- Риск дублирования: дублирование кода и сцепление – выявление повторно используемых, тесно связанных компонентов [4];
- Модульность: сцепление и когезия – оценка взаимосвязи и модульного дизайна [4].

Далее необходимо нормализовать и назначить весовые коэффициенты метрикам. Для начала приведем шкалы для каждой из необходимых метрик. Данные о шкалировании представлены в Табл. I.

ТАБЛИЦА I
ШКАЛЫ НОРМАЛИЗАЦИИ ПОКАЗАТЕЛЕЙ МЕТРИК

Метрика	**Способ нормализации**	**Шкала**	**Пороговые значения / Обоснование**
Цикломатическая сложность	Отображение на основе пороговых значений: 0 (1–10), 0,5 (11–20), 1 (21+)	0 – 1	Значения больше 20 указывают на невозможность поддержки кода в критически важных системах [5]
Метрики Холстеда	Масштабирование по минимальному и максимальному значениям для объёма (V) и усилий (E) на основе исторических данных по проекту	0 – 1	Усилия больше 300 → 1 (высокая когнитивная нагрузка) [6]
Покрытие кода	Прямое линейное масштабирование (например, 80% покрытия → 0,8)	0 – 1	Меньше 70% → высокий риск; Больше или равно 90% → оптимально для встроенных систем [5]
Дублирование кода	Процентное → линейное масштабирование (например, 30% дублирования → 0,3)	0 – 1	Больше 10% дублирования вызывает рефакторинг встроенного кода [7]

ПРОДОЛЖЕНИЕ ТАБЛ. I

Метрика	**Способ нормализации**	**Шкала**	**Пороговые значения / Обоснование**
Сложность класса	Нормализация Z-показателей для взвешенных методов по классам (WMC)	-∞ – ∞+	Z-оценка больше 2 → классы выбросов, требующие проверки [8]
Сцепление (CBO)	Масштабирование по минимуму и максимуму с пороговыми значениями: CBO меньше или равно 5 → 0–0,5, больше 5 → 0,5–1	0 – 1	CBO больше 5 указывает на чрезмерную взаимозависимость в модульных системах [9]

Далее необходимо расставить приоритеты метрик в зависимости от потребностей проекта. В данном случае будем ориентироваться на программное обеспечение, критически важное для безопасности. Данные о распределении весов представлены в Табл. II.

ТАБЛИЦА II
РАСПРЕДЕЛЕНИЕ ВЕСОВ МЕТРИК

Метрика	**Вес**	**Обоснование**
Цикломатическая сложность	0,3	Напрямую коррелирует с плотностью ошибок; значения больше 20 представляют критический риск [5]
Покрытие кода	0,25	Высокий уровень покрытия (более 90%) гарантирует, что в критическом коде будет сведено к минимуму количество непроверенных путей [5]
Метрики Холстеда (Усилия, E)	0,2	Измеряет когнитивную сложность; большие усилия повышают вероятность ошибки [6]
Сцепление (CBO)	0,15	Обеспечивает модульность для изоляции неисправностей в строго регулируемых системах [9]
Дублирование кода	0,07	Малый вес из-за меньшего влияние на показатели надежности по сравнению со сложностью
Класс сложности	0,03	Вторичен по отношению к цикломатической сложности в системах с большим потоком управления

Таким образом, сводный показатель качества для структурной единиц программного кода может выглядеть следующим образом:

$$Оценка = \sum \left(Нормализ.метр_i \cdot Вес_i \right) \quad (1)$$

Результаты расчета можно интерпретировать следующим образом:

- более 0,7: приемлемо для развертывания;
- от 0,5 до 0,7: требуется проверка;
- менее 0,5: обязательный рефакторинг.

Данный метод применим для различных типов программ, для каждой из которой будут свои весовые коэффициенты, например, для веб-приложения необходимо увеличить вес дублирования (0,15) и связанности (0,2) и снижение цикломатической сложности (0,15); для устаревающих систем стоит отдавать предпочтение дублированию кода (0,25) и метрикам Холстеда (0,2), что в первую очередь направлено на повышение удобства обслуживания.

На следующем этапе необходимо получить агрегированные показатели для получения целостной информации. Описание составных баллов представлено в Табл. III.

ТАБЛИЦА III
СОСТАВНЫЕ БАЛЛЫ

Измерение качества	**Используемые показатели**	**Пример формулы**
Ремонтопригодность	Цикломатическая сложность, метрики Холстеда (Усилия, E), сложность класса	Оценка = 0,3 × × МакКейб + + 0,2 × Холстед + + 0,03 × Сложность класса [10]
Качество тестов	Покрытие кода, цикломатическая сложность	Оценка = = Покрытие кода × ×(1-(МакКейб/Порог)) [11]
Влияние дублирования	Дублирование кода, сцепление	Риск = Дублирование × Сцепление [4]

По результатам расчета можно сделать следующие выводы, на основании комбинаций результатов. Данные комбинации выявляют системные проблемы с высоким уровнем риска:

- Цикломатическая сложность и покрытие кода:

Высокая сложность (более 15) с низким покрытием ветвей (менее 70%) указывает на непроверенные пути управления, повышая вероятность возникновения дефектов [12].

Пример: модуль со сложностью 18 и 60%-ным покрытием требует немедленного расширения тестирования.

- Усилие Холстеда и Сцепление (CBO):

высокие когнитивные усилия (E более 300) в сочетании с чрезмерной связью (CBO более 5) указывают на хрупкие, подверженные ошибкам компоненты, требующие рефакторинга [13];

- Дублирование кода и сложность класса (WMC):

Дублированный код (более 15%) в классах с высокой WMC (более 10 методов) увеличивает затраты на обслуживание и риски распространения дефектов [14].

Для выявления дополнительных стратегических приоритетов в рефакторинге стоит обращать внимание на данные комбинации параметров, представленные в Табл. IV.

ТАБЛИЦА IV
СТРАТЕГИЧЕСКИЕ ПРИОРИТЕТЫ РЕФАКТОРИНГА

Метрическая пара	**Пороговый уровень риска**	**Практическое понимание**
CBO более 5 + LCOM менее 2	Низкое сцепление + высокое сцепление	Переработать интерфейсы классов для улучшения модульности [14]
WMC более 12 + Покрытие менее 80%	Сложные, непроверенные классы	Разделить классы или улучшите модульные тесты [12]
Дублирование более 20% + объем Halstead более 500	Повторяющийся, когнитивно насыщенный код	Извлечь повторно используемые утилиты или библиотеки [13]

III. ВЫВОДЫ И ЗАКЛЮЧЕНИЕ

В результате исследования гибридного метода анализа сложности программного кода с применением классических метрик можно выделить ключевые пункты и тенденции развития:

1. На современном этапе наблюдается устойчивая тенденция к трансформации традиционного метрико-ориентированного анализа программного обеспечения в сторону интеллектуальных предиктивных моделей. Повышенный интерес вызывают гибридные подходы, интегрирующие формализованные количественные метрики (такие как цикломатическая сложность, показатели Холстеда, уровень покрытия кода и др.) с алгоритмами машинного обучения, что обеспечивает более высокую точность в оценке качества программных компонентов и управлении рисками.

2. Применение гибридного метода анализа кода демонстрирует значительную практическую результативность за счёт использования нормализованных показателей и системы весовых коэффициентов. Это позволяет локализовать участки программного кода с высокой вероятностью дефектности (например, высокая структурная сложность при недостаточном уровне тестового покрытия) и целенаправленно формировать стратегии рефакторинга и повышения сопровождаемости.

3. В процессе исследования были выявлены устойчивые структурные зависимости между различными метриками, позволяющие идентифицировать типовые конфигурации программных модулей, характеризующиеся повышенным риском. Такие паттерны (например, сочетание высокой связности компонентов — CBO, и высоких когнитивных затрат по Холстеду) формируют теоретическую и практическую основу для построения автоматизированных экспертных систем, способных генерировать рекомендации по оптимизации архитектуры программного обеспечения.

ЛИТЕРАТУРА

[1] Морозова М.А. Искусственный интеллект для оценки программного кода на основе нелинейных метрик// 75-я Международная студенческая научно-техническая конференция, 14-19 апреля 2025г., Астрахань, 14.04. Астрахань : Изд-во АГТУ, 2025. С. 1 CD-диск [Электронный ресурс]: http://astu.org/ Content/Page/5833

[2] Complexity Reports – [Электронный ресурс] – Режим доступа – URL: https://www.ibm.com/docs/en/addi/6.1.3?topic=reports-complexity (дата обращения: 30.04.2025).

[3] Using Code Quality Metrics in Management of Outsourced Development and Maintenance – [Электронный ресурс] – Режим доступа – URL: http://www.mccabe.com/pdf/McCabeCodeQuality Metrics-OutsourcedDev.pdf (дата обращения: 30.04.2025).

[4] Architectural Quality Assessment with CodeMR – [Электронный ресурс] – Режим доступа – URL: https://www.codemr.co.uk/blog/architectural-quality-assessment-with-codemr/ (дата обращения: 01.05.2025).

[5] Cyclomatic Complexity Ranges – [Электронный ресурс] – Режим доступа – URL: https://softwareengineering.stackexchange.com/questions/194061/cyclomatic-complexity-ranges (дата обращения: 05.05.2025).

[6] Metric: Halstead Complexity. User Description– [Электронный ресурс] – Режим доступа – URL: https://product-help.schneider-electric.com/Machine%20Expert/V2.1/zh/CodeAnly/CodeAnly/D-SE-0095969.html (дата обращения: 01.05.2025).

[7] Effectiveness of Software Metrics on Reliability for Safety Critical Real-Time Software – [Электронный ресурс] – Режим доступа – URL: https://easychair.org/publications/preprint/BMMl/open (дата обращения: 06.05.2025).

[8] Data Normalization Explained: Types, Examples, & Methods– [Электронный ресурс] – Режим доступа – URL: https://estuary.dev/blog/data-normalization/ (дата обращения: 06.05.2025).

[9] Key Embedded Metrics for Assessing Project Quality and Performance – [Электронный ресурс] – Режим доступа – URL: https://www.linkedin.com/pulse/key-embedded-metrics-assessing-project-quality-semyon-veber-wnjrc (дата обращения: 06.05.2025).

[10] Code Complexity Metrics: Writing Clean, Maintainable Software– [Электронный ресурс] – Режим доступа – URL: https://www.iteratorshq.com/blog/code-complexity-metrics-writing-clean-maintainable-software/ (дата обращения: 09.05.2025).

[11] Is Code Quality Related to Test Coverage? – [Электронный ресурс] – Режим доступа – URL: https://pure.manchester.ac.uk/ws/portalfiles/portal/32382808/FULL_TEXT.PDF (дата обращения: 09.05.2025).

[12] An Empirical Study of Software Metrics Diversity for Cross-Project Defect Prediction– [Электронный ресурс] – Режим доступа – URL: https://onlinelibrary.wiley.com/doi/10.1155/2021/3135702 (дата обращения: 11.05.2025).

[13] What are Software Metrics? How to Measure Them? – [Электронный ресурс] – Режим доступа – URL: https://www.browserstack.com/guide/what-is-software-metrics (дата обращения: 11.05.2025).

[14] Software Metrics – [Электронный ресурс] – Режим доступа – URL: http://people.scs.carleton.ca/~jeanpier/4004F18/T4-%20About %20Refactoring/6%20Software%20Metrics.pptx (дата обращения: 11.05.2025).

Информация об авторах

Морозов Александр Васильевич, к.т.н., доцент кафедры «Автоматизированные системы обработки информации и управления» Астраханского

государственного технического университета, г. Астрахань, Россия, morozov@ilabsltd.com

Морозова Мария Артемовна, магистрант кафедры «Автоматизированные системы обработки информации и управления» Астраханского государственного технического университета, г. Астрахань, Россия, morozovy2010@gmail.com

Hybrid approach to Analysis of program code complexity taking into account metrics and methods of artificial intelligence artificial intelligence

Maria Morozova, Alexander Morozov

Astrakhan State Technical University, Astrakhan, Russia

***Abstract* - A hybrid method for estimating software code complexity is presented, combining classical metrics (cyclomatic complexity, Holstead metrics, coupling, duplication, code coverage) and artificial intelligence methods. A mechanism for normalization and weighting of metrics taking into account the specifics of critical software is developed. Aggregated metrics and thresholds for quality interpretation are proposed. The necessity of further integration of AI-tools into the process of metrics analysis is substantiated.**

***Keywords* - metrics, complexity analysis, AI methods.**

REFERENCES

[1] Morozova M.A. Artificial intelligence for program code evaluation on the basis of nonlinear metrics// 75th International Student Scientific and Technical Conference, April 14-19, 2025, Astrakhan, 14.04. Astrakhan : Publishing house of ASTU, 2025. C. . 1 CD-disk – [Online]. Available: http://astu.org/Content/Page/5833

[2] Complexity Reports – [Online]. Available: https://www.ibm.com/docs/en/addi/6.1.3?topic=reports-complexity (дата обращения: 30.04.2025).

[3] Using Code Quality Metrics in Management of Outsourced Development and Maintenance – [Online]. Available: http://www.mccabe.com/pdf/McCabeCodeQualityMetrics-OutsourcedDev.pdf (дата обращения: 30.04.2025).

[4] Architectural Quality Assessment with CodeMR – [Online]. Available: https://www.codemr.co.uk/blog/architectural-quality-assessment-with-codemr/ (date of request: 01.05.2025).

[5] Cyclomatic Complexity Ranges – [Online]. Available: https://softwareengineering.stackexchange.com/questions/194061/cyclomatic-complexity-ranges (date of request: 05.05.2025).

[6] Metric: Halstead Complexity. User Description– [Online]. Available: https://product-help.schneider-electric.com/Machine%20Expert/V2.1/zh/CodeAnly/CodeAnly/D-SE-0095969.html (date of request: 01.05.2025).

[7] Effectiveness of Software Metrics on Reliability for Safety Critical Real-Time Software – [Online]. Available: https://easychair.org/publications/preprint/BMMl/open (date of request: 06.05.2025).

[8] Data Normalization Explained: Types, Examples, & Methods– [Online]. Available: https://estuary.dev/blog/data-normalization/ (date of request: 06.05.2025).

[9] Key Embedded Metrics for Assessing Project Quality and Performance – [Online]. Available: https://www.linkedin.com/pulse/key-embedded-metrics-assessing-project-quality-semyon-veber-wnjrc (date of request: 06.05.2025).

[10] Code Complexity Metrics: Writing Clean, Maintainable Software– [Online]. Available: https://www.iteratorshq.com/blog/code-complexity-metrics-writing-clean-maintainable-software/ (date of request: 09.05.2025).

[11] Is Code Quality Related to Test Coverage? – [Online]. Available: https://pure.manchester.ac.uk/ws/portalfiles/portal/32382808/FULL_TEXT.PDF (date of request: 09.05.2025).

[12] An Empirical Study of Software Metrics Diversity for Cross-Project Defect Prediction– [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1155/2021/3135702 (date of request: 11.05.2025).

[13] What are Software Metrics? How to Measure Them? – [Online]. Available: https://www.browserstack.com/guide/what-is-software-metrics (date of request: 11.05.2025).

[14] Software Metrics – [Online]. Available: http://people.scs.carleton.ca/~jeanpier/4004F18/T4-%20About%20Refactoring/6%20Software%20Metrics.pptx (date of request: 11.05.2025).