УДК 658.8:004.9

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРИМЕНЕНИЯ ГРАФОВЫХ МОДЕЛЕЙ ПРИ ОЦЕНКЕ КАЧЕСТВА ПРОГРАММНЫХ СИСТЕМ НА ЯЗЫКЕ GO И ДРУГИХ ЯЗЫКАХ ПРОГРАМИРОВАНИЯ

Д.А. Чернов
Программист, аспирант
chbedbro@gmail.com

С.Н. Шульженко
Профессор ТУ им. А.А. Леонова (филиал) МИИГАиК, доктор технических наук
shulzh79@mail.ru

Д.А. Чернов
Окончил Технологический университет имени А.А. Леонова.. Область научных интересов связана с ракетокосмической областью и разработкой информационно-прогностических систем.

С.Н. Шульженко
Окончил Тульский государственный университет. Область научных интересов связана с разработкой и совершенствованием интеллектуальных информационных систем и их применением в производственных задачах.

Аннотация. В работе исследуется применение графовых моделей для оценки качества программных систем, разработанных на языке Go, и проводится их сравнительный анализ с программами на других языках программирования. Целью исследования является выявление структурных особенностей кода, влияющих на надёжность и сопровождаемость программных приложений. Программная система представляется в виде ориентированного графа зависимостей, на основе которого рассчитываются показатели структурной сложности и связанности. В результате выявлены различия в плотности связей и характере зависимостей, отражающие специфику архитектурных решений языка Go.

Ключевые слова: графовые модели, анализ программного кода, качество программного обеспечения, язык Golang, структурная сложность, графы зависимостей, сравнительный анализ, метрики связанности, архитектура программных систем.

Введение. Современные программные системы характеризуются высокой степенью структурной сложности и распределённости. Рост масштабов проектов и увеличение числа

взаимосвязанных компонентов приводит к усложнению процессов анализа качества и обеспечения надёжности программного обеспечения.

Вопросы надёжности и структурной устойчивости программных систем подробно рассматриваются в работах Липаева[8] и Шубинского [14], где подчёркивается влияние архитектурной организации кода на эксплуатационные характеристики программ. Традиционные методы оценки качества основаны на использовании структурных и количественных метрик, включая показатели сложности и анализа зависимостей [1, 11]. Однако такие метрики преимущественно ориентированы на локальные характеристики кода и не всегда позволяют адекватно оценить архитектурные особенности программной системы.

В последние годы всё большее внимание уделяется применению формальных моделей представления программ, в том числе графовых структур [6, 12].

Графо ориентированный подход позволяет рассматривать программную систему как совокупность взаимосвязанных сущностей, что даёт возможность анализировать её глобальные структурные свойства.

Программная система может быть формально представлена в виде ориентированного графа (таблица 1):

$$G=(V,E),$$

где V – множество программных сущностей,
E – множество зависимостей между ними.

Таблица 1. Пример ориентированного графа зависимостей программной системы

Показатель	**Обозначение**	**Формула**	**Интерпретация**
Число вершин	$\|V\|$	–	Количество программных сущностей
Число рёбер	$\|E\|$	–	Количество зависимостей
Плотность	ρ	$\rho = \|E\| / (\|V\|(\|V\|-1))$	Степень насыщенности зависимостями
Средняя степень	$\bar{k}$	$\bar{k} = 2\|E\| / \|V\|$	Среднее число связей на вершину
Коэффициент кластеризации	C	$C = 3T/\tau$	Плотность локальных подсистем
Диаметр	D	$\max d(v_i,v_j)$	Глубина транзитивных зависимостей

В рамках графовой модели особое значение имеет показатель плотности зависимостей:

$$\rho= 2EV(V-1),$$

который отражает степень насыщенности системы внутренними связями и может рассматриваться как индикатор архитектурной сложности. Особенности построения графовых моделей в программной инженерии также рассматриваются в работах, посвящённых графо ориентированным методам организации вычислений и архитектурных решений [3, 12].

Это создаёт теоретическую основу для сравнительного анализа структур программных систем, реализованных на различных языках программирования.

Графовая модель представления программной системы. Для формального анализа структурных характеристик программных систем в работе используется их представление в виде ориентированного графа зависимостей. Такой подход позволяет рассматривать программный код не как совокупность отдельных файлов или функций, а как целостную систему взаимосвязанных элементов. Вершины графа соответствуют программным сущностям (функциям, модулям, типам данных), а рёбра отражают отношения между ними – вызовы, импортируемые зависимости и другие формы взаимодействия. Использование графовой модели обеспечивает возможность количественной оценки архитектурной организации системы. В отличие от локальных метрик сложности, ориентированных на отдельные фрагменты кода, графовое представление позволяет анализировать структуру на

уровне всей программной системы. Подобные подходы применяются при исследовании взаимосвязанных вычислительных компонентов и сложных программных комплексов [3, 12].

В рамках исследования анализируются такие характеристики, как степень связанности компонентов, насыщенность системы внутренними зависимостями и распределение связей между программными элементами. Рост числа межмодульных зависимостей, как отмечается в исследованиях, посвящённых надёжности программных систем [1, 11], приводит к увеличению вероятности дефектов и усложнению сопровождения. Графовые показатели могут рассматриваться как индикаторы архитектурной сложности и потенциальных рисков качества. Особый интерес представляет применение данного подхода к программам, реализованным на языке Go. Специфика организации кода в Go – строгая модульность пакетов, отсутствие классического наследования, использование композиции и встроенных механизмов конкурентности – формирует отличающуюся структуру зависимостей. Это создаёт предпосылки для сравнительного анализа архитектурных характеристик программ, реализованных на различных языках программирования.

Структурные особенности программных систем на языке Go. Язык Go изначально проектировался как средство разработки масштабируемых и распределённых систем, что повлияло на его архитектурные принципы и организацию кода. В отличие от классических объектно-ориентированных языков, Go не поддерживает наследование в традиционном виде, отдавая предпочтение композиции и интерфейсному взаимодействию. Это приводит к иной структуре зависимостей между программными сущностями. Модульность в Go реализуется через пакеты, каждый из которых формирует самостоятельную единицу компиляции. Импорт пакетов задаёт явные зависимости, что делает граф межмодульных связей более прозрачным и формально определённым. Отсутствие глубокой иерархии наследования, характерной для языков с развитой объектной моделью, снижает вероятность формирования сильно вложенных зависимостей и уменьшает глубину графа. Существенное влияние на структуру программ оказывает механизм конкурентности – использование goroutines и каналов. Хотя конкурентные взаимодействия не всегда отражаются напрямую в статическом графе зависимостей, они формируют дополнительные логические связи между компонентами. Это приводит к появлению функционально связанных, но структурно разнесённых элементов системы.

В сравнении с языками, использующими классическую объектно-ориентированную парадигму, структура зависимостей в Go, как правило, характеризуется (таблица 2):

- меньшей глубиной иерархий;
- более явными границами модулей;
- сниженной плотностью межклассовых связей;
- меньшим количеством транзитивных зависимостей.

Таблица 2. Сравнение типовых архитектурных структур зависимостей

Характеристика	**Go**	**Языки с наследованием**
Иерархия классов	Отсутствует	Выраженная
Транзитивные зависимости	Ограниченные	Глубокие
Централизация структуры	Низкая	Возможна высокая
Плотность графа	Умеренная	Может быть повышенной
Распределение степеней	Более равномерное	С наличием концентраторов

Исследования, посвящённые анализу архитектурной сложности программных систем [1, 11], показывают, что увеличение числа транзитивных связей и скрытых зависимостей коррелирует с ростом дефектности и усложнением сопровождения. С этой точки зрения архитектурные особенности Go потенциально способствуют формированию более предсказуемой структуры графа.Для сравнительного анализа были рассмотрены программные проекты, реализованные на языке Go и на языках с развитой объектной моделью. Оценка

проводилась на основе показателей связанности, плотности зависимостей и распределения степеней вершин графа. Анализ показал, что для приложений на Go характерна более равномерная структура распределения связей без выраженных «центров притяжения», тогда как в системах с наследованием чаще наблюдаются узлы с высокой степенью связности, играющие роль архитектурных концентраторов. Специфика языка программирования оказывает влияние на формирование графовой структуры программной системы, что необходимо учитывать при оценке её качества и архитектурной устойчивости.

Сравнительный анализ графовых характеристик программных систем. Сравнительный анализ был направлен на выявление различий в структурной организации программных систем, реализованных на языке Go и на языках с развитой объектно-ориентированной моделью. В качестве критериев оценки использовались показатели плотности зависимостей, средней степени вершин и характер распределения связей внутри графа. Полученные результаты показывают, что для проектов на языке Go характерна более модульная структура с чётко выраженными границами пакетов и относительно равномерным распределением связей между компонентами. Средняя степень вершин в графах зависимостей, как правило, не демонстрирует резких пиков, что свидетельствует об отсутствии чрезмерно централизованных элементов архитектуры. В программных системах, построенных на основе классического наследования, наблюдается иная картина. Формирование иерархий классов и использование базовых абстракций приводит к появлению узлов с высокой степенью связности. Такие элементы играют роль архитектурных центров и при изменении могут оказывать влияние на значительное число зависимых компонентов.

Согласно исследованиям в области оценки структурной сложности [1, 11], высокая концентрация связей в отдельных узлах повышает чувствительность системы к изменениям и увеличивает риск каскадных дефектов. Дополнительным фактором, влияющим на графовую структуру, является способ организации конкурентного взаимодействия. В Go логические связи между компонентами, возникающие при использовании goroutines и каналов, зачастую не увеличивают плотность статического графа зависимостей, поскольку реализуются через явные интерфейсы взаимодействия. В целом анализ показывает, что архитектурные принципы языка программирования оказывают непосредственное влияние на формирование графовой структуры программной системы. Для языка Go характерна тенденция к более плоской и модульной архитектуре, тогда как системы с выраженной иерархической моделью демонстрируют более сложную структуру зависимостей.

Заключение. В работе рассмотрено применение графовых моделей для анализа структурной организации программных систем и проведён сравнительный анализ их характеристик для приложений, реализованных на языке Go и на других распространённых языках программирования. Показано, что представление программной системы в виде ориентированного графа зависимостей позволяет формализовать архитектурные особенности кода и перейти от качественного описания структуры к количественной оценке её сложности и связанности. Результаты анализа свидетельствуют о том, что архитектурные принципы языка Go способствуют формированию более модульной и менее иерархически перегруженной структуры зависимостей по сравнению с языками, использующими классическую модель наследования. Для систем на Go характерно более равномерное распределение связей между компонентами и меньшая концентрация зависимостей в отдельных узлах графа, что потенциально снижает чувствительность архитектуры к локальным изменениям. Полученные выводы подтверждают целесообразность применения графовых метрик в качестве инструмента оценки качества программных систем и могут быть использованы при разработке средств автоматизированного анализа кода, а также при проектировании масштабируемых и распределённых приложений.

Перспективным направлением дальнейших исследований является расширение набора анализируемых метрик и применение методов интеллектуального анализа данных для прогнозирования дефектности на основе графовых характеристик программных систем.

Список литературы

[1] Бобрышев А.Н., Жарин Д.Е., Шафигуллин Л.Н., Гумеров М.И. Метрики структурной сложности программных систем. Информационные технологии. 2017;23(5):321–327.

[2] Дмитренко Е.Н. Методы анализа надежности сложных программных систем. Программная инженерия. 2019;10(4):150–158.

[3] Димитриенко Ю.И., Соколов А.П. Графоориентированный подход к организации распределённых вычислений. Известия вузов. 2008;4:33–40.

[4] Иванников В.П., Першин А.Ю. Инструментальные средства анализа качества программного обеспечения. Информатика и её применения. 2014;8(1):15–23.

[5] Климов С.М., Сосновский Ю.В., Чачиев Д.Р. Методика оценки функциональной надежности компонент программно-аппаратной встраиваемой микропроцессорной системы управления. Надежность. 2025;25(1):58–66. DOI:10.21683/1729-2646-2025-25-1-58-66.

[6] Кравченко Ю.А. Применение графовых моделей при анализе программных систем. Вестник МГТУ. 2015;18(2):45–52.

[7] Кузнецов В.В. Методы статического анализа исходного кода. Программные продукты и системы. 2021;34(2):89–97.

[8] Липаев В.В. Надежность программного обеспечения. М.: СИНТЕГ; 2008.

[9] Нетес В.А. Надежность программных средств: терминология и проблемы стандартизации. Надежность. 2016;16(3):3–10.

[10] Русаков А.М., Юшкова Н.А., Селиванова Е.А. Современные средства визуализации графов и графовых моделей для Python. Наукосфера. 2022;11(2):258–266. DOI:10.5281/zenodo.7372018.

[11] Сидоров А.А. Модели прогнозирования дефектов программных систем. Программная инженерия. 2020;11(6):255–262.

[12] Соколов А.П., Першин А.Ю. Система автоматизированного проектирования композиционных материалов. Часть 1. Концепции, архитектура и платформа разработки. Известия СПбГЭТУ «ЛЭТИ». 2020;8–9:72–83.

[13] Чаругин В.В., Чаругин В.В., Чесалин А.Н. Метрики оценки качества локализации уязвимостей и автоматического исправления исходного кода. Инновационные, информационные и коммуникационные технологии. 2025;22:399–403.

[14] Шубинский И.Б. Функциональная надежность информационных систем. М.: Горячая линия–Телеком; 2012.

[15] Громов А.А. Применение теории графов в задачах анализа архитектуры программного обеспечения. Вестник компьютерных и информационных технологий. 2018;15(6):27–35.

Авторский вклад

Чернов Данила Андреевич – постановка задачи исследования, формирование репрезентативной выборки телеметрических данных, программная реализация алгоритмов фильтрации и масштабирования признаков, проведение рангового корреляционного анализа и расчет метрик информационного прироста.

Шульженко Сергей Николаевич – постановка задачи исследования, концептуальное обоснование применения аппарата теории информации, верификация предложенной методики оптимизации признакового пространства и интерпретация полученных статистических зависимостей.

COMPARATIVE ANALYSIS OF THE APPLICATION OF GRAPH MODELS IN ASSESSING THE QUALITY OF SOFTWARE SYSTEMS WRITTEN IN GO AND OTHER PROGRAMMING LANGUAGES

D.A. Chernov
Programmer, PhD Student

S.N. Shuzhenko
Professor of the Department of ITUS A.A. Leonov Technological University (Branch of MIIGAiK), Doctor of Technical Sciences

Abstract. The paper investigates the application of graph-based models for assessing the quality of software systems developed in the Go programming language and provides a comparative analysis with programs written in other programming languages. The objective of the study is to identify structural code characteristics that affect the reliability and maintainability of software applications. The software system is represented as a directed dependency graph, on the basis of which structural complexity and connectivity metrics are calculated. The results reveal differences in link density and dependency patterns that reflect the architectural specifics of the Go language.

Keywords: graph models, source code analysis, software quality, Go (Golang) language, structural complexity, dependency graphs, comparative analysis, connectivity metrics, software architecture.