DOI 10.37882/2223-2966.2025.05.08

ЭРГОНОМИЧЕСКИЙ СЕРТИФИКАТ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, БАЗИРУЮЩИЙСЯ НА ИЕРАРХИЧНОСТИ

ERGONOMIC SOFTWARE CERTIFICATE BASED ON HIERARCHY

B. Goryachkin
N. Klyukin

Summary. *Problem statement*. In the context of modern programming, where software support and development are becoming priorities, it is important to take into account that the lack of ergonomics of a growing code base can lead to significant labor and financial costs. Therefore, there is a need to evaluate the ergonomics of the software.
Goal. Provide a standardized approach to evaluating software ergonomics.
Results. The issue of evaluating the ergonomics of the software is considered, the characteristics of the ergonomics of the code are determined. A software ergonomics certificate was compiled, consisting of 4 comprehensive criteria.
Practical significance. An ergonomic certificate, a hierarchical design approach, and ways to evaluate code ergonomics should provide a standardized approach to evaluating software ergonomics. This will simplify the process of software support and refinement.

Keywords: hierarchy, ergonomic certificate, ergonomics assessment, hierarchy, software, ergonomics.

Горячкин Борис Сергеевич
кандидат технических наук, доцент,
Московский государственный технический университет им. Н.Э. Баумана
bsgor@mail.ru
Клюкин Никита Александрович
Московский государственный технический университет им. Н.Э. Баумана
klyukin.n21@yandex.ru

Аннотация. *Постановка проблемы*. В условиях современного программирования, где поддержка и развитие программного обеспечения становятся приоритетными задачами, важно учитывать, что не эргономичность разрастающейся кодовой базы может привести к значительным трудовым и финансовым затратам. В связи с чем возникает необходимость оценочных инструментов эргономичности программного обеспечения.
Цель. Предоставить стандартизированный подход к оценке эргономичности программного обеспечения.
Результаты. Рассмотрен вопрос оценки эргономичности программного обеспечения, определены характеристики эргономичности кода. Был составлен сертификат эргономичности программного обеспечения, состоящий из 4 комплексных критериев.
Практическая значимость. Эргономический сертификат, иерархический подход к проектированию и способы оценки эргономичности кода должны предоставить стандартизированный подход к оценке эргономичности программного обеспечения. Это позволит упростить процесс поддержки и доработки программного обеспечения.

Ключевые слова: иерархичность, эргономический сертификат, оценка эргономичности, иерархичность, программное обеспечение, эргономика.

Введение

Одним из важнейших аспектов качества ПО является его эргономичность. В условиях современного программирования, где поддержка и развитие программного обеспечения становятся приоритетными задачами, важно учитывать, что не эргономичность разрастающейся кодовой базы может привести к значительным трудовым и финансовым затратам. Однако, несмотря на значимость эргономичности, на сегодняшний день отсутствуют универсальные и стандартизированные инструменты для её оценки.

Существующие стандарты, такие как ГОСТ Р ИСО/МЭК 9126-93 «Информационная технология. Оценка программной продукции. Характеристики качества и руководство по их применению», определяют ключевые характеристики качества ПО, включая функциональность, надежность, практичность, эффективность, сопровождаемость и переносимость [1]. Однако в них не предусмотрено конкретных методик или инструментов для оценки этих характеристик, что ограничивает их практическую применимость.

Данное исследование направлено на решение этой проблемы путем разработки эргономичного сертификата программного обеспечения, основанного на принципе иерархичности. Использование иерархического подхода обеспечивает системность оценки, позволяя учитывать как общие правила, так и конкретные показатели, необходимые для соответствия.

Виды иерархичности

В вопросе построения иерархической структуры программной системы ключевыми характеристиками

иерархии являются: количество вершин, количество рёбер, высота — количество эшелонов, ширина — максимальное количество вершин на 1 эшелоне. В рамках этих общих характеристик важно выполнение правила «7±2», означающее, что на одном уровне иерархии необходимо располагать 5–9 вершин, каждая из которых должна обладать не более чем с 5–9 рёбрами, с количеством уровней не более 5–9.

При этом, важно отметить, что в случае, когда использование явных абстракций не хватает для построения эргономичного ПО используют неявные. Примером такого использования является разбиение всего программного кода на домены, за которые он отвечает: в паттерне проектирования MVC это будет Модель — Представление — Контроллер. В рамках данного исследования мы концентрируемся на анализе иерархий явных абстракций.

Теория анализа сложности иерархий программного кода хорошо изучена научным сообществом. Так, выделяют ряд различных метрик, которые позволяют оценить иерархии модулей, классов и функций.

В рамках оценки иерархичности можно прибегнуть к оценке невязки [2].

Под невязкой подразумевают оценку отличия структуры иерархии от дерева, где 0 соответствует дереву, а 1 — полному графу, выраженную формулой:

$$Nev = \frac{2 \times (e - n + 1)}{(n - 1) \times (n - 2)}$$

Где e — количество вершин, а n — количество рёбер.

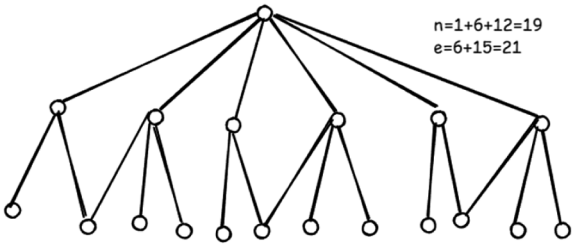
Согласно формуле невязкости имеем, что для соблюдения идеальной структуры дерева необходимо, чтобы общее количество рёбер в иерархии было на 1 меньше, чем количество вершин. Однако, возникает ситуация, когда абстракция нижнего уровня используется несколькими вышестоящими. Для того, чтобы определить оптимальное значение рассчитаем значение для следующих типов иерархий: полное дерево, широкое дерево (широкий второй уровень), узкое дерево (малый второй уровень, но более широкий последующие). При этом возьмём условие, что на уровень абстракции для каждой второй вершины нарушается правило идеального дерева (рис. 1).

Проведя расчёты получили, что максимальное значение невязки имеет широкое дерево, следовательно допустимым значением для программного обеспечения будет значение невязки равное 0,05.

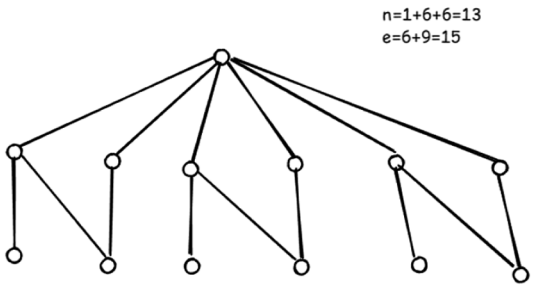
Однако, приведенный выше расчёт не учитывает особенности иерархий, формируемых абстракциями.

Особенностями иерархии функция является возможное наличие на 1 уровне иерархии управляющего слоя. Это вызвано тем, что функция может отвечать за выполнение одной глобальной задачи, требующее выполнения множества действий, и, в целях эргономичного размещения, допустимо выделение логического-управляющего уровня. Поэтому для расчёта коэффициента невязкости для этой иерархии абстракций следует произвести расчёт для структуры, использующей разделяемые вершины.

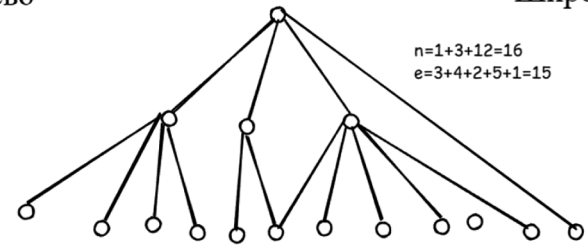
Для иерархии наследования классов, согласно Р. Байнедеру, крайне желательно, чтобы у каждого наследника



Полное дерево



Широкое дерево



Узкое дерево

Рис. 1. Анализируемые типы деревьев

был только один, что связано с тем, что, наследуясь от нескольких родительских классов, мы получаем из обоих свойства и методы, что сильно запутывает архитектуру.

Чаще всего иерархию модулей можно условно разделить на 3 части: модули, отвечающие за отображение информации, модули бизнес-логики, модули взаимодействия с базой данных. В такой структуре явным является факт малого количества модулей, ответственных за доступ к данным из базы данных, поскольку обычно используется единый интерфейс взаимодействия. Тоже самое справедливо и для модулей ответственных за отображение информации, поскольку они также регламентируют единый интерфейс вывода [3]. Таким образом, типовой для этой абстракции является иерархия в форме юлы — верхние и нижние уровни узкие, а средние — широкие.

Расчёты произведём посредством искажения идеальной структуры дерева. При построении иерархии функции воспользуемся правилом: для каждой пары вершин нарушать идеальную структуру посредством добавления на уровень ниже разделяемой ими вершины. Для иерархии наследования классов: для каждой 10 вершины имеется 2 родителя. Для иерархии модуля: имеется широкий средний уровень в иерархии равный сумме вершин остальных модулей.

Произведя соответствующие расчёты, получили для иерархии функций значение допустимой невязкости в 4 %, для иерархии наследования классов — в 3 %, а для иерархии модулей — в 8 %.

Таблица 1.

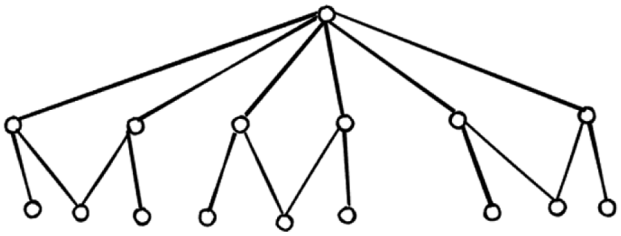
Результаты расчётов невязкости для иерархий абстракций

Функция			Класс			Модуль		
Вершин	Рёбер	Невязка	Вершин	Рёбер	Невязка	Вершин	Рёбер	Невязка
11	12	0,0444444	10	10	0,0277778	10	12	0,0833333
16	18	0,0285714	15	15	0,010989	12	14	0,0545455
21	24	0,0210526	20	21	0,0116959	13	16	0,0606061
23	26	0,017316	30	32	0,0073892	14	16	0,0384615

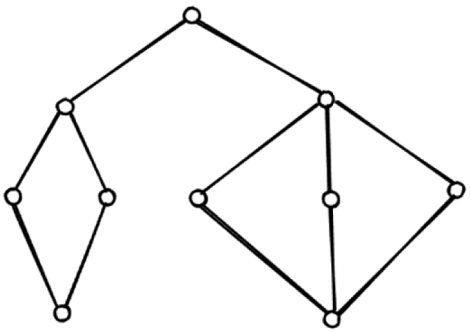
Результаты исследования подтверждаются практикой: так проанализировав архитектуру ПО с открытым исходным кодом видно, что все иерархии абстракции, применяемые в этом ПО приближены к идеальной структуре дерева. К таковым можно отнести иерархию модулей Nginx (5 вершин и 4 ребра), Hadoop Cluster (9 вершин, 8 рёбер), иерархию наследования классов в Git на примере GitObject (5 вершин, 4 ребра), VTK на примере vtkObjectBase (24 вершины, 23 рёбра) [4].

Оценка эргономичности кода

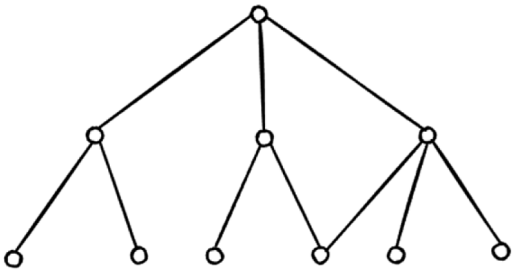
Несмотря на наличие стихийно организованных практик в сообществе, вопрос эргономики программно-



Иерархия функций



Иерархия модулей



Иерархия наследования классов

Рис. 2. Анализируемые типовые иерархии абстракций

го обеспечения требует более глубокого изучения и систематизации. Важно понимать, что такие практики как PEP8 и Java Code Convention являются руководствами к стилю написания кода на конкретном языке. В них отражены функциональные особенности, которые заложены в эти языки: Java — это ООП язык, предназначенный для крупных проектов, где требуется производительность, а Python — скриптовый язык, предназначенный для быстрой разработки. Но, как и в вопросах эргономики интерфейсов, существуют общие правила и закономерности, которые отражают эргономичность кода.

Говоря об эргономичности конкретного программного кода, выделяют характеристики читаемости, лаконичности, согласованности [5]

Характеристики читаемости — это оценка того, насколько легко другим разработчикам понять код. Эта характеристика стремиться оценить, насколько написанный код следует принципу «7±2», то есть насколько наш код хорошо структурирован, но, в отличие от характеристики «иерархичности» мы смотрим на код не как на набор абстракций, а как на текстовый файл. В связи с этим, в данной характеристики мы стремимся оценить: а как следует представлять и оформлять блоки кода?

При оценке читаемости важным является оценка содержательности имён. Имена для абстракций, используемых в коде, должны работать на восприятие человека-оператора чтобы он понимал её содержание. Так, хорошей практикой является использование в именах функций глаголы или глагольные словосочетания, отражающих их функционал.

Имена переменных используют существительные либо категорию состояния для логических переменных. Также для переменных необходимо детализировать посредством пояснения того, к какой логической части программы они относятся. Так, например использование «numberUser» вместо «num» позволит быстро найти нужную переменную в коде и не создаст путаницы с тем к какой логической части она относиться. Тоже самое применимо для методов и атрибутов функций.

Что же касательно наименования классов, то их принято именовать существительным, при этом, стоит отметить, что в случае использование паттернов проектирования хорошей практикой является явное указание в классе принадлежность к паттерну. Например, для паттерна GOF «Абстрактная фабрика» лучше использовать «InterfaceFactory» вместо «Interface».

Также читаемость кода повышает использование отступов. Знаки присваивания окружены пробелами, обеспечивающими их визуальное выделение. Операторы присваивания состоят из двух основных элементов: левой и правой частей. Пробелы наглядно подчеркивают это разделение. Также более объёмные абстракции, как функция и класс, должны явно выделяться пустыми строками. Согласно устоявшимся практикам, это 1 пустая строка после объявления функции и 2 — после объявления класса. Помимо этого, имеет смысл использовать 1–2 пустые строки в структурных частях кода для отделения неких самостоятельных концепций, поскольку в ходе просмотра листинга взгляд привлекает первая строка, следующая за пустой строкой.

Использование комментариев в программном коде является распространённой практикой, повышающей читаемость программного кода. Например, их использование для статической описательной информации: описание классов и методов, использовать для описания неочевидного блока кода.

Однако использование комментарии нужно осторожно, поскольку они обладают важным свойством — подчёркивают, концентрируют внимание человека-оператора. Примером плохого использования являются избыточные комментарии, недостоверные комментарии, журнальные комментарии.

Характеристики лаконичности — это оценка того, насколько код свободен от избыточности. Лаконичный код достигается за счет использования эффективных алгоритмов и конструкций, которые минимизируют количество строк и символов, не теряя при этом ясности.

Из этого также следует, что в коде должно присутствовать минимальное количество дублирования, поскольку если что-то делается в программе снова и снова, это свидетельствует о том, что какая-то абстракция не нашла представления в коде.

Для определения максимума символов в одной строки обратимся к популярным линтерам. Так, в языке Python популярными линтерами являются Flake8, Black, Pylint. По умолчанию в них установлен размер 79, 88, 100. Для языка Java популярными решениями являются Checkstyle и он по умолчанию устанавливает ограничение в 80 символов. При этом в официальной документации к Flake8 отмечают, что в промышленной разработке разработчики зачастую повышают этот лимит до 100–120, поэтому будем считать максимально допустимым значением символов на строку равным 120 символам.

Касательно вопроса количества строк на функцию/класс прибегнем к исследованию эргономичности программного кода на языке Java, где были получены следующие цифры [6]:

1. Максимальное количество строк в классе — 1065
2. Максимальное количество строк в методе — 25
3. Максимальная длина имена метода — 25

4. Максимальное количество параметров в методе — 5
5. Максимальная длина наименования параметров метода, полей класса — 10
6. Максимальное количество полей в классе — 25
7. Максимальное количество методов в классе — 40
8. Минимальная длина наименования полей в классе, параметров в методе — 3

Характеристика согласованности подразумевает следование единому стилю наименования для параметров, методов, классов и полей, что позволяет разработчикам легче ориентироваться в коде и поддерживать его. Это подразумевает следованию практикам, определенных сообществами языка, а также использования единого стиля в команде.

Для этого зачастую удобно использовать линтеры. Их основная функция заключается в статическом анализе кода, что позволяет находить ошибки и несоответствия до момента выполнения программы. Это особенно полезно для командной работы, где важно придерживаться единого стиля написания кода. Линтеры могут проверять такие аспекты, как использование пробелов, наименование переменных, длина строк и другие соглашения по стилю кодирования. Если код не соответствует установленным стандартам, линтер выдает предупреждения, что позволяет разработчикам быстро исправлять проблемы.

Сертификат эргономичности программного обеспечения

В данном исследовании предлагается оценка эргономичности программного обеспечения на помощи эргономического сертификата. Он представляет собой документ, определяющий эргономическое состояние АСОИУ и оценивающий его эргономическое обеспечение [7]. Это оценка, состоящая из 4 комплексных критериев, отражающих один из аспектов эргономичности программной системы:

1) Комплексный критерий эргономичности кода. Этот комплексный критерий отражает качество реализации конкретного кода. Оценивается его читаемость, лаконичность, согласованность.
2) Комплексный критерий эргономичности функций. Этот комплексный критерий даёт оценку качеству проектирования функций. Оценивается их сложность, иерархичность.
3) Комплексный критерий эргономичности классов. Этот комплексный критерий даёт оценку качеству проектирования классов. Оценивается иерархичность, а также насколько классы ПО следуют принципам ООП: инкапсуляция, наследование, полиморфизм.
4) Комплексный критерий эргономичности модулей. Этот комплексный критерий даёт оценку качеству проектирования модулей. Оценивается их иерархичность, сцепление, связность

Таблица 2.

Сертификат эргономичности программного обеспечения

I. Комплексный критерий эргономичности кода	
1.1. Читаемость (насколько легко другим разработчикам понять код)	Содержательность имён Использование отступов Использование комментариев
1.2. Лаконичность (насколько код свободен от избыточности)	Количество строк в классе; Количество строк в методе/функции; Количество полей в классе; Количество методов в классе; Длина тела метода/функции;
1.3. Согласованность (следование единому стилю наименования для параметров, методов, классов и полей)	Следование стилистическим практикам, определенных в сообществе используемого языка;

Частные эргономические характеристики кода	
Суть характеристики	Значение нормы
1.1. В функциях используется глаголы или отглагольные сочетания	Да
1.2. Имена переменных являются существительными или категорией состояния, если переменная является логической	Да
1.3. Имена переменных детализированы	Да
1.4. Знаки присваивания окружены пробелами	Да
1.5. Минимальное количество пробелов после объявления функции	1
1.6. Максимальное количество пробелов после объявления функции	2
1.7. Количество пробелов после объявления класса	2
1.8. Максимальное количество пробелов между элементами кода	2
1.9. Для описания классов/функций/модулей присутствуют комментарии	Да
1.10. Отсутствуют журнальные комментарии	Да
1.11. Отсутствие явного дублирования в коде	Да
1.12. Максимальные количество символов в строке	120
1.13. Максимальное количество строк в классе	1065
1.14. Максимальное количество строк в методе/функции	25
1.15. Максимальная длина имена метода/функции	25

1.16. Максимальное количество параметров в методе/функции	5
1.17. Максимальная длина наименования параметров метода, полей класса	10
1.18. Максимальное количество полей в классе	25
1.19. Максимальное количество методов в классе	40
1.20. Минимальная длина наименования полей в классе, параметров в методе	3
1.21. Наименования классов, методов, параметров, полей указано в одном из стилей, определенного в сообществе используемого языка	Да
1.22. Максимальное количество действий в одной строке	1
II. Комплексный критерий эргономичности функции	
Характеристика сложности	Разрядность функции Функциональная сложность
Характеристика иерархичности	Высота, ширина иерархии
Частные эргономические характеристики функции	
Суть характеристики	Значение нормы
2.1. Максимальная разрядность функции	3
2.2. Максимальная ширина уровня иерархии	9
2.3. Максимальное количество рёбер для одной вершины	9
2.4. Максимальное количество уровней в иерархии	9
2.5. Максимальное значение коэффициента невязки	0,04
III. Комплексный критерий эргономичности класса	
Характеристика использования ООП	Оценка инкапсуляции, Оценка наследования, Оценка полиморфизма
Характеристика иерархичности	Высота, ширина иерархии
Частные эргономические характеристики класса	
Суть характеристики	Значение нормы
3.1. Поля класса защищены и для них определён интерфейс взаимодействия	Да
3.2. Используется механизм абстрактных классов для реализации общего поведения между классами	Да
3.3. Максимальная ширина уровня иерархии	9
3.4. Максимальное количество рёбер для одной вершины	9
3.5. Максимальное количество уровней в иерархии	9
3.6. Максимальное значение коэффициента невязки для структуры наследования	0,03
IV. Комплексный критерий эргономичности модуля	
Характеристика иерархичности	Высота, ширина иерархии; Оценка Сцепления; Оценка Связности; Оценка Невязки.
Частные эргономические характеристики модуля	
Суть характеристики	Значение нормы
4.1. Максимальное значение коэффициента сцепления	4
4.2. Минимальное значение коэффициента связности	9
4.3. Максимальная ширина уровня иерархии	9
4.4. Максимальное количество рёбер для одной вершины	9
4.5. Максимальное количество уровней в иерархии	9
4.6. Максимальное значение коэффициента невязки	0,08

Заключение

Таким образом, в данной работе был изучен вопрос оценки эргономичности ПО. Важным в вопросе эргономичности ПО и выполнения правила «7±2», в частности, является оценка того, какие иерархии формируют эти абстракции. Для её определения в работе использовался коэффициент невязки, был проведён эксперимент, результатом которого стали числовые пороговые значения для каждого вида иерархий абстракций.

Также было проведено исследование того, какими характеристиками должен обладать эргономичный код, где был выявлен ряд характеристик: читаемость, лаконичность, согласованность.

В результате, был составлен сертификат эргономичности, состоящий из 4 комплексных критериев: критерий эргономичности кода, критерий эргономичности функций, критерий эргономичности классов, критерий эргономичности модулей. Структурно, в первом критерии оценивается эргономичность фактической реализации ПО, а в последних трёх критериях — насколько эргономично она спроектирована.

ЛИТЕРАТУРА

1. ГОСТ Р ИСО/МЭК 9126-93 «Информационная технология. Оценка программной продукции. Характеристики качества и руководство по их применению».
2. С. Орлов. Технологии разработки программного обеспечения: Учебник — СПб.: Питер, 2002. — 464 с.
3. Мартин Фаулер. Архитектура корпоративных программных приложений. — М.: Вильямс, 2006. — 544 с.
4. Чандлер Р., Брайант Р., Брайант Р. Архитектура приложений с открытым исходным кодом. Lulu.com, 2011. — 415 с.
5. Горячкин Б.С. Бакланов Н.В. Понятный код // E-Scio, 2023 г. — № 2 (77), Стр. 19–30.
6. Горячкин Б.С., Черненький С.В., Саросек М.С. Сертификат эргономичности программного кода на языке Java//Международный научный журнал «Динамика сложных систем — XXI век»: Издательство «Радиотехника» — Москва, 2022 — № 1, Стр. 13–21.
7. Горячкин Б.С. Эргономический сертификат автоматизированной системы обработки и отображения информации и управления — Стр. 25–29. Выпуск: № 9 (51) Часть 2. 2016 г. DOI: https://doi.org/10.18454/IRJ.2016.51.101

Журнал «Современная наука: актуальные проблемы теории и практики»