Научная статья
УДК 004.42; DOI: 10.61260/2218-130X-2026-2-104-110

СПОСОБЫ РАСЧЕТА ИНФОРМАЦИОННОГО ОБЪЕМА ФОРМ-СОДЕРЖАТЕЛЬНЫХ ПРЕДСТАВЛЕНИЙ ПРОГРАММЫ

✉Израилов Константин Евгеньевич.
Санкт-Петербургский университет ГПС МЧС России, Санкт-Петербург, Россия
✉konstantin.izrailov@mail.ru

Аннотация. Работа посвящена изучению свойств программы на различных этапах разработки, определяемых авторской форм-содержательной концепцией ее представлений. Суть данной концепции заключается в том, что программа рассматривается через дихотомию ее формы как внешнего описания на заданном языке или нотации и содержания, определяющего внутреннюю логику или функциональность; при этом, по мере разработки содержание остается инвариантным, а форма эволюционирует от человеко-ориентированной к машинно-ориентированной. Выделены и описаны следующие представления: идея, концептуальная модель, архитектура, алгоритмы, исходный код, дерево абстрактного синтаксиса, ассемблерный код, машинный код. В качестве измеряемого свойства программы рассматривается информационной объем представления, соответствующий размеру формы, используемой при описании программы. Для каждого из представлений приведен способ расчета такого объема, а также его формализованная запись. Указаны основные пути уточнения способов расчета объема и дальнейшего развития исследования.

Ключевые слова: программа, характеристики, форма, содержание, информационный объем, способ расчета

Для цитирования: Израилов К.Е. Способы расчета информационного объема форм-содержательных представлений программы // Научно-аналитический журнал «Вестник Санкт-Петербургского университета Государственной противопожарной службы МЧС России». 2026. № 2. С. 104–110. DOI: 10.61260/2218-130X-2026-2-104-110

Scientific article

METHODS FOR CALCULATING THE INFORMATION VOLUME OF FORM-CONTENT PRESENTATIONS OF THE PROGRAM

✉Izrailov Konstantin E.
Saint-Petersburg university of State fire service of EMERCOM of Russia, Saint-Petersburg, Russia
✉konstantin.izrailov@mail.ru

Abstract. This paper examines the properties of a program at various stages of development, as defined by the author's form-content concept of its representations. The concept centers on viewing a program through the dichotomy of its form, as an external description in a given language or notation, and its content, which defines its internal logic or functionality. While the content remains invariant during development, it evolves from human-oriented to machine-oriented in form. The following representations are identified and described: idea, conceptual model, architecture, algorithms, source code, abstract syntax tree, assembler code, and machine code. The measurable property of a program is the information volume of a representation, corresponding to the size of the form used to describe the program. A method for calculating such volume and a formalized notation are provided for each representation. The main ways to refine the methods for calculating the volume are indicated, and avenues for further research development are offered.

Keywords: program, characteristics, form, content, information volume, calculation method

For citation: Izrailov K.E. Methods for calculating the information volume of form-content presentations of the program // Scientific and analytical journal «Vestnik Saint-Petersburg university of State fire service of EMERCOM of Russia». 2026. № 2. P. 104–110. DOI: 10.61260/2218-130X-2026-2-104-110

Введение

Информационные технологии являются неотъемлемой частью современного мира, в основе их лежит программное обеспечение [1]. Однако, хотя практическое создание выполняемых программ и имеет достаточно долгую историю, их теоретические свойства изучены еще недостаточно. Ранее автором была предложена форм-содержательная концепция представлений, через которые проходит любая программа (по крайней мере, в случае применения императивной, процедурной, объектно-ориентированной и других классических парадигм программирования). В рамках концепции любое представление программы (например, исходный код) может быть рассмотрено как его дихотомия на внешний вид – то есть форму, и внутреннюю логику – то есть содержание [2], что было обосновано экспериментально и формализовано [3]. Без внесения уязвимостей в процессе разработки программы ее содержание остается неизменным (изначальная функциональность тождественна конечной выполняемой), а форма преобразуется от человеко-ориентированной к машинно-ориентированной. При этом внешний вид программы постепенно «разрастается», поскольку первые представления ориентированы на восприятие человеком, оперирующим небольшим количеством емких конструкций, а последние предназначены для выполнения на некотором процессоре, поддерживающим разнообразный набор инструкций и их операндов. Как результат, поиск заложенных в высокоуровневых представлениях уязвимостей [4] по низкоуровневым оказывается неэффективным из-за их недостаточной локализации в коде. Тем не менее остается нерассмотренным теоретико-практический вопрос оценки формы программы [5] в каждом из представлений – его информационного объема, что является задачей текущего исследования и в интересах решения которой далее будут предложены соответствующие способы расчета.

Представления программы

Ранее был выделен ряд представлений, графовидная схема перехода между которыми позволяет задавать жизненный цикл создаваемой программы. Далее будет рассмотрена наиболее распространенная ветка такого графа, соответствующая классическому пути большинства программ – от задумки до выполняемого образа. Краткое описание представлений такого пути с указанием их назначений, особенностей форм и используемых базисных элементов приведено далее.

Идея является самым первым представлением программы, поскольку закладывает ее требуемую функциональность – то, что программа должна делать в конечном итоге. Вид представления является максимально неформализованным, поскольку предназначен для работы с ним человека. В качестве базисных конструкций представления могут считаться слова и предложения естественного языка.

Концептуальная модель осуществляет базовую формализацию идеи путем выделения основных элементов функциональности программы и их взаимосвязей [6]. Модель задает основную цель, на достижение которой направлена вся разработка. Базисными конструкциями представления являются текстовые описания блоков основных сущностей предметной области и их связей.

Архитектура программы считается первым представлением, имеющим непосредственное отношение к программному коду, поскольку выделяет иерархию логических модулей (компонентов). Каждый из них может интерпретироваться как одна из частных целей, достигаемых программой в процессе функционирования. Таким образом, с точки зрения базисных элементов, модули в представлении имеют более формализованную запись и строгие взаимосвязи, чем в случае концептуальной модели.

Алгоритмы программы являются составными частями архитектурных модулей и предназначены для выполнения отдельных задач, совокупность которых приводит к достижению частных целей [7]. Классически алгоритмы являются совокупностью названия, входных и выходных параметров, а также внутренней логики. Базисные элементы состоят из блоков алгоритмов, таких как вычисление, условие, цикл, вызов, которые связаны логикой выполнения.

Исходный код предназначен для непосредственного кодирования программы на специализированном языке программирования. Такая запись, хотя все еще ориентирована на понимание разработчиком, однако уже более учитывает необходимость выполнения на процессоре. Базисные элементы зависят от конкретного языка, а точнее, используемых в нем минимальных единиц синтаксиса – токенов.

Дерево абстрактного синтаксиса является служебным представлением, которое строится по программе в процессе ее компиляции соответствующими средствами [8]. Назначением дерева является отвязка программы от специфики языка программирования путем записи операций в унифицированной форме. Базисными элементами представления являются узлы такого графа.

Ассемблерный код может считаться первым представлением, ориентированным именно на выполнение, а не разработку, поскольку он состоит из последовательности текстовых строк с формализованным описанием процессорных инструкций и их операндов. Соответственно, базисные элементы представления определяются токенами нотации такой ассемблерной записи.

Машинный код подходит для непосредственного выполнения на процессоре, поскольку содержит бинарную запись процессорных инструкций (то есть тех, которые имели текстовую запись в ассемблерном коде) [9]. Базисными элементами являются компактно заданные (за счет бинарной записи) инструкции, включающие в себя как саму выполняемую операцию, так и ее операнды.

Способы расчета

Исходя из приведенного описания представлений программы, их можно считать достаточно разнородными как с точки зрения назначения и формы, так и используемых базисных элементов. Таким образом, для каждого из представлений можно предположить собственный способ расчета его информационного объема (естественно, с позиции формы), что и приводится далее.

Идея, как представление программы, имеет наименее формализованный вид. Однако, если ее рассматривать как некоторое текстовое описание на естественном языке, то объем представления может быть рассчитан по количеству соответствующих слов и знаков (которые также играют смысловую роль):

$$\begin{cases} Idea \equiv (IdeaElement_i) \\ Volume_{Idea}() \coloneqq |\{IdeaElement_i\}|' \end{cases}$$

где $Idea$ – представление идеи программы; $(\dots)$ – здесь и далее – последовательность элементов, в отличие от $()$, являющегося постфиксом для указания на способ вычисления (как некоторую функцию); $IdeaElement_i$ – элемент описания на естественном языке; $Volume_{Idea}()$ – способ вычисления информационного объема для представления.

Для уточнения способа расчета можно использовать лингвистические метрики текста, семантическую сложность его элементов, а также оценивать читаемость и формализацию.

Объем представления *концептуальной модели* программы может быть вычислен по количеству ее функциональных элементов, а также их связей, несущих определенную содержательную нагрузку:

$$\begin{cases} ConceptModel \equiv \langle \{ConceptElement_i\} | \{ConceptLink_j\} \rangle \\ Volume_{ConceptModel}() \coloneqq |\{ConceptElement_i\}| + |\{ConceptLink_j\}| \end{cases},$$

где $ConceptModel$ – представление концептуальной модели программы; $ConceptElement_i$ и $ConceptLink_j$ – i-ый элемент и их j-ая связь в концептуальной модели; $Volume_{ConceptModel}()$ – способ вычисления информационного объема для представления.

Для уточнения способа расчета можно ввести типизацию связей, уровни иерархий элементов (при наличии) и оценивать структурированность.

Объем представления *архитектуры* программы может быть вычислен по количеству ее модулей – как дочерних, так и иерархически их включающих:

$$\begin{cases} Architecture \equiv \langle \{ArchModule_i\}, \{ArchLink_j\} \rangle \\ Volume_{Architecture}() \coloneqq |\{ArchModule_i\}| \end{cases},$$

где $Architecture$ – представление архитектуры программы; $ArchModule_i$ и $ArchLink_j$ – i-ый модуль и их j-ая связь в архитектуре; $Volume_{Architecture}()$ – способ вычисления информационного объема для представления.

Для уточнения способа расчета можно учитывать типы модулей и связей, их роли, оценивать сложность иерархии, применять метрики связности и целостности, понятность описания внутри блоков.

Типичной формой записи *алгоритмов* является их блок-схема [10], представляющая собой граф потока управления для каждой подпрограммы, в узлах которого расположены операции, часто содержащие человеко-ориентированное описание. Таким образом, объем представления может быть рассчитан по количеству таких узлов:

$$\begin{cases} Algorithms \equiv \{Algorithm_i\} \\ Algorithm_i \equiv \langle AlgBlock_{i,j}, AlgLink_{i,k} \rangle \\ Volume_{Algorithms}() \coloneqq |\{AlgBlock_{i,j}\}| \end{cases},$$

где $Algorithms$ – представление алгоритмов программы; $Algorithm_i$ – алгоритм i-ой подпрограммы; $AlgBlock_{i,j}$ и $AlgLink_{i,k}$ – j-ый модуль и их k-ая связь в алгоритме i-ой подпрограммы; $Volume_{Algorithms}()$ – способ вычисления информационного объема для представления.

Для уточнения способа расчета можно учитывать типы блоков с весовой оценкой их сложности, оценивать цикломатическую сложность и степень вложенности условных ветвлений, а также размеры блоков и сложность их описания.

Каждый из алгоритмов имеет текстовую запись на соответствующем языке программирования, что позволяет рассчитать объем представления *исходного кода* по количеству используемых для записи токенов. При этом токены, относящиеся к «синтаксическому сахару» и не влияющие на содержание программы (то есть логику ее работы), могут быть опущены:

$$\begin{cases} SourceCode \equiv (SrcToken_i) \\ SrcToken_i \in SrcTokens_{Content} \\ Volume_{SourceCode}() \coloneqq |\{SrcToken_i\}| \end{cases},$$

где $SourceCode$ – представление исходного кода программы; $SrcToken_i$ – i-ый токен текста; $SrcTokens_{Content}$ – множество всех токенов кода, имеющих смысловую нагрузку; $Volume_{SourceCode}()$ – способ вычисления информационного объема для представления.

Для уточнения способа расчета можно учитывать типизацию токенов, использовать классические метрики программного обеспечения, добавить учет синтаксических конструкций.

Объем представления *дерева абстрактного синтаксиса* может быть рассчитан по количеству его узлов, содержащих, по сути, типы операций, операнды, переменные, константы и пр. При этом все подпрограммы могут быть заданы в рамках единого дерева:

$$\begin{cases} AbstractSyntaxTree \equiv \langle \{AstNode_i\}, \{AstLink_j\} \rangle \\ Volume_{AbstractSyntaxTree}() \coloneqq |\{AstNode_i\}| \end{cases},$$

где $AbstractSyntaxTree$ – представление дерева абстрактного синтаксиса программы; $AstNode_i$ и $AstLink_j$ – i-ый узел и их j-ая связь в дереве; $Volume_{AbstractSyntaxTree}()$ – способ вычисления информационного объема для представления.

Для уточнения способа расчета можно учитывать типы узлов и их семантику, глубину дерева и его поддеревьев, оценивать подобие узлов и расстояние между ними.

Поскольку *ассемблерный код* представляет собой список строк для всех подпрограмм, каждая из которых содержит текстовое описание одной процессорной инструкции (кроме служебной и пр. информации), то его объем можно рассчитать по количеству токенов текста без учета синтаксического сахара (аналогично представлению исходного кода):

$$\begin{cases} AssemblerCode \equiv (AsmLine_i) \\ AsmLine_i \equiv (AsmLineToken_{i,j}) \\ AsmLineToken_{i,j} \in AsmTokens_{Content} \\ Volume_{AssemblerCode}() \coloneqq |\{AsmLineToken_{i,j}\}| \end{cases},$$

где $AssemblerCode$ – представление ассемблерного кода программы; $AsmLine_i$ – i-ая строка текста кода; $AsmLineToken_{i,j}$ – j-ый токен в i-ой ассемблерной строке; $AsmTokens_{Content}$ – множество всех ассемблерных токенов, имеющих смысловую нагрузку; $Volume_{AssemblerCode}()$ – способ вычисления информационного объема для представления.

Для уточнения способа расчета можно различать типы токенов (аналогично исходному коду), наличие излишнего синтаксического сахара, веса сложности используемых конструкций.

Объем *машинного кода* можно вычислить путем суммирования количества машинных инструкций, в которых содержатся все подпрограммы:

$$\begin{cases} MachineCode \equiv (MachineInstr_i) \\ Volume_{MachineCode^{Float}}() \coloneqq |\{MachineInstr_i\}| \end{cases}',$$

где $MachineCode$ – представление машинного кода программы; $MachineInstr_i$ – i-ая инструкция; $Volume_{MachineCode^{Float}}()$ – способ вычисления информационного объема для представления (при процессорных инструкциях переменной длины).

В случае процессорных архитектур с фиксированным размером инструкций достаточно поделить размер машинного кода на длину инструкции (например, в байтах):

$$Volume_{MachineCode^{Fixed}}() \coloneqq \frac{MachineCodeSize}{MachineInstrLength},$$

где $Volume_{MachineCode^{Float}}()$ – способ вычисления информационного объема для представления (при процессорных инструкциях фиксированной длины); $MachineCodeSize$ – размер машинного кода; $MachineInstrLength$ – длина процессорной инструкции.

Для уточнения способа расчета можно учитывать длину инструкций (в случае их переменной длины), а также применять подобные ассемблерному коду метрики.

Приведенные способы расчета носят приблизительный характер и предназначены для определения общего направления по созданию соответствующего измерительного инструментария. Так, например, явно не учитываются константные значения в представлениях исходного, ассемблерного и машинного кода. Тем не менее, для базовой оценки информационного объема все приведенные способы и их формулы являются достаточно адекватными.

Заключение

В работе решается задача вычисления информационного объема программы на всех этапах разработки в соответствии с авторской форм-содержательной концепцией ее представлений. Основной результат проведенного исследования состоит в оригинальных способах расчета объема, что обладает новизной. Теоретическая значимость состоит в формализованной записи способов расчета на основе показателей представлений, а практическая – в возможности оценить степень изменения информационного объема реальных программ в процессе их инжиниринга, а также внедренных в них уязвимостей. Продолжением исследования должно стать уточнение ряда способов расчета и их применения для реальных программ.

Список источников

1. Цифровые технологии и проблемы информационной безопасности: монография / Т.И. Абдуллин [и др.]. СПб.: СПГЭУ 2021. 163 с.

2. Израилов К.Е. Моделирование программы с уязвимостями с позиции эволюции ее представлений. Часть 1. Схема жизненного цикла // Труды учебных заведений связи. 2023. Т. 9. № 1. С. 75–93. DOI: 10.31854/1813-324X-2023-9-1-75-93

3. Израилов К.Е. Моделирование программы с уязвимостями с позиции эволюции ее представлений. Часть 2. Аналитическая модель и эксперимент // Труды учебных заведений связи. 2023. Т. 9. № 2. С. 95–111. DOI: 10.31854/1813-324X-2023-9-2-95-111

4. Архитектурные уязвимости моделей телекоммуникационных сетей / М.В. Буйневич [и др.] // Научно-аналитический журнал «Вестник Санкт-Петербургского университета Государственной противопожарной службы МЧС России». 2015. № 4. С. 86–93.

5. Израилов К.Е. Рассмотрение представлений кода программ с позиций метаданных // Фундаментальные исследования и инновации в национальных исследовательских университетах: материалы Всерос. науч.-метод. конф. Санкт-Петербург, 2012. С. 176–180.

6. Вялых А.В., Гуляев Д.А. Понятие концептуальной модели программой системы // Концепции и модели устойчивого инновационного развития общества: сб. статей Междунар. науч.-практ. конф. Оренбург, 2023. С. 17–19.

7. Малясова С.В. Алгоритмизация, программирование и технология программирования // Информатика и образование. 2010. № 2. С. 22–36.

8. Горчаков А.В., Демидова Л.А., Советов П.Н. Автоматизированный анализ текстов программ при помощи представлений на основе цепей Маркова и машин экстремального обучения // Вестник Рязанского государственного радиотехнического университета. 2022. № 82. С. 89–103. DOI: 10.21667/1995-4565-2022-82-89-103

9. Белеванцев А., Журихин Д., Мельник Д. Компиляция программ для современных архитектур // Труды Института системного программирования РАН. 2009. Т. 16. С. 31–50.

10. Баскакова Е.В., Шевко Н.Р. Способы описания алгоритмов: блок-схемы, псевдокод и их роль в разработке // Теоретические и практические аспекты развития современной науки: теория, методология, практика: сб. науч. статей по материалам XVIII Междунар. науч.-практ. конф. Уфа, 2025. С. 190–195.

References

1. Cifrovye tekhnologii i problemy informacionnoj bezopasnosti: monografiya / T.I. Abdullin [i dr.]. SPb.: SPGEU 2021. 163 s.
2. Izrailov K.E. Modelirovanie programmy s uyazvimostyami s pozicii evolyucii ee predstavlenij. Chast' 1. Skhema zhiznennogo cikla // Trudy uchebnyh zavedenij svyazi. 2023. T. 9. № 1. S. 75–93. DOI: 10.31854/1813-324X-2023-9-1-75-93
3. Izrailov K.E. Modelirovanie programmy s uyazvimostyami s pozicii evolyucii ee predstavlenij. Chast' 2. Analiticheskaya model' i eksperiment // Trudy uchebnyh zavedenij svyazi. 2023. T. 9. № 2. S. 95–111. DOI: 10.31854/1813-324X-2023-9-2-95-111
4. Arhitekturnye uyazvimosti modelej telekommunikacionnyh setej / M.V. Bujnevich [i dr.] // Nauchno-analiticheskij zhurnal «Vestnik Sankt-Peterburgskogo universiteta Gosudarstvennoj protivopozharnoj sluzhby MCHS Rossii». 2015. № 4. S. 86–93.
5. Izrailov K.E. Rassmotrenie predstavlenij koda programm s pozicij metadannyh // Fundamental'nye issledovaniya i innovacii v nacional'nyh issledovatel'skih universitetah: materialy Vseros. nauch.-metod. konf. Sankt-Peterburg, 2012. S. 176–180.
6. Vyalyh A.V., Gulyaev D.A. Ponyatie konceptual'noj modeli programmoj sistemy // Koncepcii i modeli ustojchivogo innovacionnogo razvitiya obshchestva: sb. statej Mezhdunar. nauch.-prakt. konf. Orenburg, 2023. S. 17–19.
7. Malyasova S.V. Algoritmizaciya, programmirovanie i tekhnologiya programmirovaniya // Informatika i obrazovanie. 2010. № 2. S. 22–36.
8. Gorchakov A.V., Demidova L.A., Sovetov P.N. Avtomatizirovannyj analiz tekstov programm pri pomoshchi predstavlenij na osnove cepej Markova i mashin ekstremal'nogo obucheniya // Vestnik Ryazanskogo gosudarstvennogo radiotekhnicheskogo universiteta. 2022. № 82. S. 89–103. DOI: 10.21667/1995-4565-2022-82-89-103
9. Belevancev A., Zhurihin D., Mel'nik D. Kompilyaciya programm dlya sovremennyh arhitektur // Trudy Instituta sistemnogo programmirovaniya RAN. 2009. T. 16. S. 31–50.
10. Baskakova E.V., Shevko N.R. Sposoby opisaniya algoritmov: blok-skhemy, psevdokod i ih rol' v razrabotke // Teoreticheskie i prakticheskie aspekty razvitiya sovremennoj nauki: teoriya, metodologiya, praktika: sb. nauch. statej po materialam XVIII Mezhdunar. nauch.-prakt. konf. Ufa, 2025. S. 190–195.

Информация о статье:
Статья поступила в редакцию: 20.03.2026; одобрена после рецензирования: 20.04.2026;
принята к публикации: 21.04.2026

Information about the article:
The article was submitted to the editorial office: 20.03.2026; approved after review: 20.04.2026;
accepted for publication: 21.04.2026

Информация об авторах:
Израилов Константин Евгеньевич, профессор кафедры прикладной математики и безопасности информационных технологий Санкт-Петербургского университета ГПС МЧС России (196105, Санкт-Петербург, Московский пр., д. 149), кандидат технических наук, доцент, e-mail: konstantin.izrailov@mail.ru, https://orcid.org/0000-0002-9412-5693

Information about authors:
Izrailov Konstantin E., professor of the department of applied mathematics and information technology security of Saint-Petersburg university of State fire service of EMERCOM of Russia (196105, Saint-Petersburg, Moskovsky ave., 149), candidate of technical sciences, associate professor, e-mail: konstantin.izrailov@mail.ru, https://orcid.org/0000-0002-9412-5693