Аналитический обзор

Целью данной главы является обзор и сравнение наиболее популярных языков программирования, применяемых для обучения основам программирования, анализ их сильных и слабых сторон, а также исследование особенностей обучения программированию и наиболее распространенных методов контроля знаний в образовательном процессе.

## Методы контроля знаний при подготовке программистов

### Уровни подготовки программиста

На современном этапе развития на смену традиционным формам контроля знаний приходят новые формы, построенные на широком применении средств вычислительной техники.

Подготовка профессионального программиста — это многоуровневый процесс.

На первом, базовом, уровне подготовки студент приобретает навыки структурирования информации и построения простейших вычислительных алгоритмов. Также на этом уровне студент приобретает навыки распространения одного алгоритма решения задачи на круг однотипных задач, унификации однотипных блоков алгоритма и создания процедур.

На втором, общепрофессиональном, уровне подготовки студент овладевает методами и приемами выбора оптимального решения (одного из многих) поставленной задачи. Студент не только осваивает приемы поиска нужной информации, но и овладевает доказательным аппаратом принятия решения. Овладение доказательным аппаратом является одним из основных условий подготовки профессионального программиста, способного к конкурентному продвижению созданного программного продукта.

На третьем, профилирующем, уровне подготовки производится специализация профессионального программиста в узкой области экономики, производства, науки и т. д. На этом уровне детально изучаются особенности выбранной области деятельности.

На различных уровнях подготовки профессионального программиста используются различные виды контроля знаний и умений. Контроль знаний и умений студента решает триединую задачу:

* определить глубину усвоения студентом учебного материала;
* решить вопрос о переходе студента к изучению следующего учебного материала (при положительной оценке) либо предложить ему дополнительные вопросы для изучения текущего учебного материала (при неудовлетворительной оценке);
* подобрать индивидуальное задание для практической и самостоятельной работы студента.

При подготовке профессиональных программистов наряду с хорошо зарекомендовавшими себя традиционными формами контроля знаний — зачетом и экзаменом, — которые используются на этапе итогового контроля, широкое применение находят различные варианты промежуточного контроля знаний, в том числе построенные с использованием информационных технологий. К наиболее популярным видам промежуточного контроля можно отнести:

* доклады и рефераты, выполненные с использованием технологии презентаций и включающие в себя графические, звуковые и анимационные элементы;
* различные виды тестирования;
* создание промежуточных и законченных программных продуктов (проектов);
* решение алгоритмических задач за лимитированный промежуток времени;
* публичную защиту программных продуктов.

### Доклады и рефераты

Как правило, этот вид контроля знаний используется на первом уровне подготовки профессионального программиста по дисциплинам общепрофессионального блока. К таким дисциплинам можно отнести «Технические средства информатизации», «Архитектура ЭВМ и вычислительных систем», «Операционные системы и среды» и некоторые другие. Доклад выполняется в виде презентации с использованием иллюстративного материала, а при необходимости — звуковых эффектов и видеоклипов. Организуется публичное выступление студента с докладом и последующее обсуждение доклада .

К достоинству этого вида контроля следует отнести возможность проверки таких качеств студента, как умение подобрать материал по теме доклада из разных источников информации, многосторонний взгляд на тему доклада, анализ собранной информации, способность делать выводы, овладение приемами публичной аргументации в защиту полученных результатов.

К недостаткам можно отнести то, что данный вид контроля не оценивает способности студента к применению полученной информации для практического решения задач.

### Тестирование

Тестирование — широко распространенный вид контроля знаний. Он используется по многим дисциплинам многих специальностей, как технических, так и гуманитарных. Тестирование позволяет оперативно и достаточно точно определить уровень знаний студента и применяется с целью промежуточной аттестации учащегося и решения вопроса о его допуске к зачету или экзамену. Использование тестирования, с одной стороны, повышает качество подготовки специалиста, с другой стороны, облегчает труд экзаменатора, так как к экзамену допускаются студенты, показавшие по результатам тестирования наличие некоторого минимума знаний. Использование тестирования в качестве итоговой аттестации, по нашему мнению, не целесообразно, так как ответы на вопросы теста жестко фиксированы и не позволяют (или с большими трудностями позволяют) выявить связи данного вопроса с другими вопросами данной (или смежной) дисциплины.

Преимуществами использования тестирования являются оперативность получения сведений о знаниях студентов, объективность полученных результатов, возможность получения статистических данных об успеваемости каждого студента и группы студентов, возможность получения сведений о динамике успехов (или о снижении результатов) студента, возможность определения тем и вопросов, слабо освоенных студентом.

К недостаткам тестирования следует отнести трудности формулировки вопросов теста, составления нескольких однозначных и точных ответов на поставленный вопрос, определения качественного состава и количества вопросов, входящих в один тест, оценки результатов тестирования в силу неодинаковости (по значимости и сложности) вопросов, входящих в один тест, сложности идентификации и аутентификации тестируемого студента.

### Промежуточные программные продукты

При планировании лабораторных работ по дисциплинам «Основы программирования и алгоритмизации», «Базы данных», «Разработка и эксплуатация удаленных баз данных» и т. д. необходимо стремиться, чтобы каждая последующая лабораторная работа дополняла и уточняла предыдущую. Тем самым, достигается цель получения некоторого программного продукта. Так как лабораторные работы имеют своей целью освоение студентами практических навыков использования методов и приемов программирования, то после выполнения лабораторных работ можно говорить о промежуточном программном продукте, в котором отсутствуют, или слабо выражены потребительские качества. Поэтому при публичной защите лабораторных работ упор делается на правильное и эффективное использование методов обработки информации, а также отработку доказательного аппарата правильности выбора данного решения.

Такой вид промежуточного контроля дает точные и достоверные сведения об уровне усвоения знаний и умений, приобретенных студентом, прививает навыки делового общения и аргументации в защиту принятого решения.

### Решение алгоритмических задач на время

Данный вид контроля знаний применяется в основном для тренировок и проведения соревнований по спортивному программированию для студенческих и школьных команд и осуществляется посредством проведения контестов. В каждом контесте участникам предлагается решить некоторое число задач различной сложности и тематики за ограниченный промежуток времени.

Это позволяет выявить способность программиста концентрироваться на поставленной задаче в жестких временных рамках, слаженно работать в команде (при проведении командного соревнования), разрабатывать эффективные алгоритмы решения, анализировать задачи и применять уже известные алгоритмы. Данный вид контроля знаний, может быть использован не только при обучении олимпиадному программированию, но и при обучении базовым навыкам программирования как студентов, так и школьников.

### Законченные программные продукты

Этот вид промежуточного контроля, как правило, осуществляется при выполнении курсовой работы (проекта), когда может быть получен законченный программный продукт, обладающий потребительскими качествами. Здесь выполняется комплексная проверка знаний и умений, полученных студентом. Также организуется публичная защита курсовой работы с приглашением студентов младших курсов. Приглашение этих студентов преследует две цели: образовательную — повышение мотивации к овладению выбранной специальностью и воспитательную — приучение к использованию технических терминов, правилам ведения дискуссии, расширению технического кругозора. Оценка за курсовую работу выставляется коллективом преподавателей, при этом учитываются глубина проработки темы курсовой работы, содержание и правильность построения доклада, аргументация в защиту принятых решений, умение отвечать на дополнительные вопросы.

### Публичная защита программного продукта

Это наиболее эффективный вид контроля знаний, который позволяет оценить работу студента со всех сторон: комплексность решения поставленной задачи, глубину проработки задачи, степень самостоятельности выполненной работы, удобство работы с интерфейсом программы, наличие в ней подсказок и справочников, удобство чтения полученных результатов. Проведение публичной защиты программного продукта (промежуточного или законченного) требует от преподавателя дополнительных усилий по ее организации и проведению. При организации публичной защиты (особенно первой) необходимо обратить серьезное внимание на подготовку студентом доклада, оказать помощь в составлении доклада, как по его структуре, так и по содержательной части. По мере проведения последующих публичных защит помощь преподавателя будет уменьшаться. Несколько первых публичных защит проводится внутри обучаемой группы студентов, без приглашения студентов младших курсов. На первой публичной защите очень важно помочь студенту преодолеть психологический барьер публичного выступления.

Во время проведения публичной защиты следует обратить внимание на организационный момент: поддержание дисциплины, организацию комфортной деловой атмосферы, порядок выступления докладчиков. Как правило, рассматриваются 4–5 работ студентов, коллективом преподавателей проводится сравнительный анализ работ, и выставляются оценки. Перед оглашением полученных студентами оценок обязательно следует обратить внимание присутствующих на положительные моменты работы студента и при необходимости указать на замеченные недостатки. Такой подход обеспечивает при выполнении следующего цикла лабораторных работ или следующей курсовой работы устранение отмеченных недостатков, что в конечном итоге приводит к повышению качества подготовки профессиональных программистов. Публичная защита работ студентами самым благотворным способом сказывается на процессе написания дипломной работы и ее защите перед Государственной комиссией.

Сочетание традиционных видов контроля знаний и умений студентов с новыми видами контроля, основанными на применении информационных технологий, способствует оперативному и эффективному управлению процессом обучения, выявлению тенденции на повышение или понижение уровня усвоения знаний, принятию оперативных мер по повышению качества подготовки специалистов, повышению мотивации студентов к овладению специальностью.

С учетом вышесказанного, можно сделать вывод о том, что правильная организация системы контроля знаний студентов на каждом уровне в отдельности и за все время обучения в целом приводит к повышению качества подготовки профессиональных программистов.

## Модели и методы технологии интеллектуальных вычислений

### Регрессионный анализ

Регрессионный анализ — метод моделирования измеряемых данных и исследования их свойств. Данные состоят из пар значений зависимой переменной (переменной отклика) и независимой переменной (объясняющей переменной). Регрессионная модель есть функция независимой переменной и параметров с добавленной случайной переменной. Параметры модели настраиваются таким образом, что модель наилучшим образом приближает данные. Критерием качества приближения (целевой функцией) обычно является среднеквадратичная ошибка: сумма квадратов разности значений модели и зависимой переменной для всех значений независимой переменной в качестве аргумента.

Регрессионный анализ — раздел математической статистики и машинного обучения. Предполагается, что зависимая переменная есть сумма значений некоторой модели и случайной величины. Относительно характера распределения этой величины делаются предположения, называемые гипотезой порождения данных. Для подтверждения или опровержения этой гипотезы выполняются статистические тесты, называемые анализом остатков. При этом предполагается, что независимая переменная не содержит ошибок. Регрессионный анализ используется для прогноза, анализа временных рядов, тестирования гипотез и выявления скрытых взаимосвязей в данных.

Основными целями регрессионного анализа являются следующие:

* определение степени детерминированности вариации критериальной (зависимой) переменной предикторами (независимыми переменными);
* предсказание значения зависимой переменной с помощью независимой (-ых);
* определение вклада отдельных независимых переменных в вариацию зависимой.

Регрессия — зависимость математического ожидания (например, среднего значения) случайной величины от одной или нескольких других случайных величин (свободных переменных), вычисляется по формуле (1)

*E*(*y*| *x*) = *f*(*x*), (1)

где *E* — среднее значение случайной величины, *y* — зависимая случайная величина, *x* — независимая случайная величина

Регрессионным анализом называется поиск такой функции *f*, которая описывает эту зависимость. Регрессия может быть представлена в виде суммы неслучайной и случайной составляющих по формуле (2).

*y* = *f*(*x*) + *ν*, (2)

где f — функция регрессионной зависимости, а *ν* — аддитивная случайная величина с нулевым математическим ожиданием. Предположение о характере распределения этой величины называется гипотезой порождения данных. Обычно предполагается, что величина *ν* имеет гауссово распределение с нулевым средним и дисперсией .

Задача нахождения регрессионной модели нескольких свободных переменных ставится следующим образом. Задана выборка — множество {*x*1, …, *xN*| *x*∈ ℝ*M*} значений свободных переменных и множество {*y*1, …, *yN*| *y*∈ ℝ} соответствующих им значений зависимой переменной. Эти множества обозначаются как *D*, множество исходных данных {(*x*, *y*)*i*}. Задана регрессионная модель — параметрическое семейство функций f(\mathbf{w},\mathbf{x}) зависящих от параметров \mathbf{w}\in\mathbb{R} и свободных переменных *x*. Требуется найти наиболее вероятные параметры \bar{\mathbf{w}}:

\bar{\mathbf{w}}=\arg\max\limits_{\mathbf{w}\in\mathbb{R}^W}p(y|x,\mathbf{w},f)=p(D|\mathbf{w},f). (3)

Функция вероятности *p* зависит от гипотезы порождения данных и задается Байесовским выводом или методом наибольшего правдоподобия.

**Линейная регрессия.**

Линейная регрессия предполагает, что функция *f* зависит от параметров \mathbf{w} линейно. При этом линейная зависимость от свободной переменной *x* необязательна,

 y=f(\mathbf{w},\mathbf{x})+\nu=\sum_{j=1}^N w_jg_j(\mathbf{x})+\nu.  (4)

В случае, когда функция g\equiv\text{id}, линейная регрессия имеет вид

 y=\sum_{j=1}^N w_jx_j+\nu=\langle\mathbf{w},\mathbf{x}\rangle +\nu,  (5)

где *xj* — компоненты вектора *x*.

Значения параметров в случае линейной регрессии находят с помощью метода наименьших квадратов. Использование этого метода обосновано предположением о гауссовом распределении случайной переменной.

Разности *yi – f(xi)* между фактическими значениями зависимой переменной и восстановленными называются регрессионными остатками (residuals). В литературе используются также синонимы: невязки и ошибки. Одной из важных оценок критерия качества полученной зависимости является сумма квадратов остатков, вычисляемая по формуле

SSE=\|f(\mathbf{x}_i)-y_i\|_2=\sum_{i=1}^N(y_i-f(\mathbf{w},\mathbf{x}_i))^2. (6)

где *SSE* — Sum of Squared Errors, сумма квадратичных ошибок.

Дисперсия остатков вычисляется по формуле

\bar{\sigma}^2_\nu=\frac{SSE}{N-2}=MSE. (7)

где *MSE* — Mean Squared Error, среднеквадратичная ошибка.

**Метод наименьших квадратов (расчёт коэффициентов).**

На практике линия регрессии чаще всего ищется в виде линейной функции *Y* = *b*0 + *b*1*X*1 + *b*2 *X*2 + ... + *bn* *Xn* (линейная регрессия), наилучшим образом приближающей искомую кривую. Делается это с помощью метода наименьших квадратов, когда минимизируется сумма квадратов отклонений реально наблюдаемых *Y* от их оценок \hat{Y} (имеются в виду оценки с помощью прямой линии, претендующей на то, чтобы представлять искомую регрессионную зависимость):

\sum_{k=1}^{M} (Y_k-\hat{Y_k})^2 \to min, (8)

где *M* — объём выборки. Этот подход основан на том известном факте, что фигурирующая в приведённом выражении сумма принимает минимальное значение именно для того случая, когда *Y* = *y*(*x*1, *x*2, ... *xn*).

Для решения задачи регрессионного анализа методом наименьших квадратов вводится понятие функции невязки, формула:

\sigma(\bar{b})=\frac{1}{2}\sum_{k=1}^{M}{(Y_k-\hat{Y}_k)^2} (9)

Условие минимума функции невязки:

\left\{ \begin{matrix}
\frac{d\sigma(\bar{b})}{db_i}=0 \\
i=0...N
\end{matrix} \right.
\Leftrightarrow
\left\{ \begin{matrix}
\sum_{i=1}^{M}{My_i}=\sum_{i=1}^{M}{\sum_{j=1}^{N}{b_jx_{i,j}}}+b_0M \\
\sum_{i=1}^{M}{y_ix_{i,k}}=\sum_{i=1}^{M}{\sum_{j=1}^{N}{b_jx_{i,j}x_{i,k}}}+Mb_0\sum_{i=1}^{M}{x_{i,k}} \\
k=1...N
\end{matrix} \right. (10)

Полученная система является системой *N* + 1 линейных уравнений с *N* + 1 неизвестными *b*0 ... *bn*.

Если представить свободные члены левой части уравнений матрицей (11)

B=\left\{ \begin{matrix}
\sum_{i=1}^{M}{y_i} \\
\sum_{i=1}^{M}{y_ix_{i,1}} \\
... \\
\sum_{i=1}^{M}{y_ix_{i,N}}
\end{matrix} \right\} (11)

а коэффициенты при неизвестных в правой части матрицей (12)

A=\left\{ \begin{matrix}
M & \sum_{i=1}^{M}{x_{i,1}} & \sum_{i=1}^{M}{x_{i,2}} & ... & \sum_{i=1}^{M}{x_{i,N}} \\
\sum_{i=1}^{M}{x_{i,1}} & \sum_{i=1}^{M}{x_{i,1}x_{i,1}} & \sum_{i=1}^{M}{x_{i,2}x_{i,1}} & ... & \sum_{i=1}^{M}{x_{i,N}x_{i,1}} \\
\sum_{i=1}^{M}{x_{i,2}} & \sum_{i=1}^{M}{x_{i,1}x_{i,2}} & \sum_{i=1}^{M}{x_{i,2}x_{i,2}} & ... &  \sum_{i=1}^{M}{x_{i,N}x_{i,2}} \\
... & ... & ... & ... & ... \\
\sum_{i=1}^{M}{x_{i,N}} & \sum_{i=1}^{M}{x_{i,1}x_{i,N}} & \sum_{i=1}^{M}{x_{i,2}x_{i,N}} & ... &  \sum_{i=1}^{M}{x_{i,N}x_{i,N}}
\end{matrix} \right\} (12)

то получаем матричное уравнение A \times X = B, которое легко решается методом Гаусса. Полученная матрица будет матрицей (13), содержащей коэффициенты уравнения линии регрессии:

X=\left\{ \begin{matrix}
b_0 \\
b_1 \\
... \\
b_N
\end{matrix} \right\} (13)

### Генетический алгоритм

Генетический алгоритм — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Является разновидностью эволюционных вычислений. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе [4].

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде вектора («генотипа») генов, где каждый ген может быть битом, числом или неким другим объектом. В классических реализациях ГА предполагается, что генотип имеет фиксированную длину, однако существуют вариации ГА, свободные от этого ограничения.

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет, насколько хорошо описываемый им фенотип решает поставленную задачу.

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения (обычно лучшие особи имеют большую вероятность быть выбранными), к которым применяются «генетические операторы» (в большинстве случаев «скрещивание» — crossover и «мутация» — mutation), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий остановки алгоритма. Таким критерием может быть:

* нахождение глобального, либо субоптимального решения;
* исчерпание числа поколений, отпущенных на эволюцию;
* исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в многомерных пространствах поиска.

Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Задание целевой функции (приспособленности) для особей популяции

2. Создание начальной популяции

(Начало цикла)

1. Размножение (скрещивание)

2. Мутирование

3. Вычисление значения целевой функции для каждой особи

4. Формирование нового поколения (селекция)

5. Если выполняются условия останова, то

(конец цикла),

иначе

(начало цикла)

Перед первым шагом нужно случайным образом создать начальную популяцию; даже если она окажется совершенно неконкурентоспособной, генетический алгоритм все равно достаточно быстро переведет ее в жизнеспособную популяцию. Таким образом, на первом шаге можно не стараться создать приспособленных особей, достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию приспособленности. Итогом первого шага является популяция *H*, состоящая из *N* особей.

Размножение в генетических алгоритмах обычно половое — чтобы произвести потомка, нужны несколько родителей, обычно два.

Размножение в разных алгоритмах определяется по-разному — оно, конечно, зависит от представления данных. Главное требование к размножению — потомок или потомки должны иметь возможность унаследовать черты обоих родителей, «смешав» их каким-либо способом.

Почему особи для размножения обычно выбираются из всей популяции *H*, а не из выживших на первом шаге элементов *H*0 (хотя последний вариант тоже имеет право на существование)? Дело в том, что главный бич многих генетических алгоритмов — недостаток разнообразия в особях. Достаточно быстро выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция «забивается» копиями этой особи. Есть разные способы борьбы с таким нежелательным эффектом; один из них — выбор для размножения не самых приспособленных, но вообще всех особей.

К мутациям относится все то же самое, что и к размножению: есть некоторая доля мутантов *m*, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать *mN* особей, а затем изменить их в соответствии с заранее определенными операциями мутации.

На этапе отбора нужно из всей популяции выбрать определенную ее долю, которая останется «в живых» на этом этапе эволюции. Есть разные способы проводить отбор. Вероятность выживания особи *h* должна зависеть от значения функции приспособленности Fitness(*h*). Сама доля выживших *s* обычно является параметром генетического алгоритма, и ее просто задают заранее. По итогам отбора из *N* особей популяции *H* должны остаться *sN* особей, которые войдут в итоговую популяцию *H'*. Остальные особи погибают.

### Искусственные нейронные сети

Искусственные нейронные сети — математические модели, а также их программные или аппаратные реализации, построенные по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма. Искусственные нейронные сети представляют собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов) [4, 5].

Сеть обучается, чтобы для некоторого множества входов давать желаемое (или, по крайней мере, сообразное с ним) множество выходов. Каждое такое входное (или выходное) множество рассматривается как вектор. Обучение осуществляется путем последовательного предъявления входных векторов с одновременной подстройкой весов в соответствии с определенной процедурой. В процессе обучения веса сети постепенно становятся такими, чтобы каждый входной вектор вырабатывал выходной вектор.

Для решения поставленных задач использована нейронная сеть с обучающим алгоритмом обратного распространения ошибки.

На рисунке 2 показан нейрон, используемый в качестве основного строительного блока в сетях обратного распространения. Подается множество входов, идущих либо извне, либо от предшествующего слоя. Каждый из них умножается на вес, и произведения суммируются. Эта сумма, обозначаемая NET, должна быть вычислена для каждого нейрона сети. После того, как величина NET вычислена, она модифицируется с помощью активационной функции и получается сигнал OUT.

На рисунке 3 показана активационная функция (14), обычно используемая для обратного распространения.

. (14)

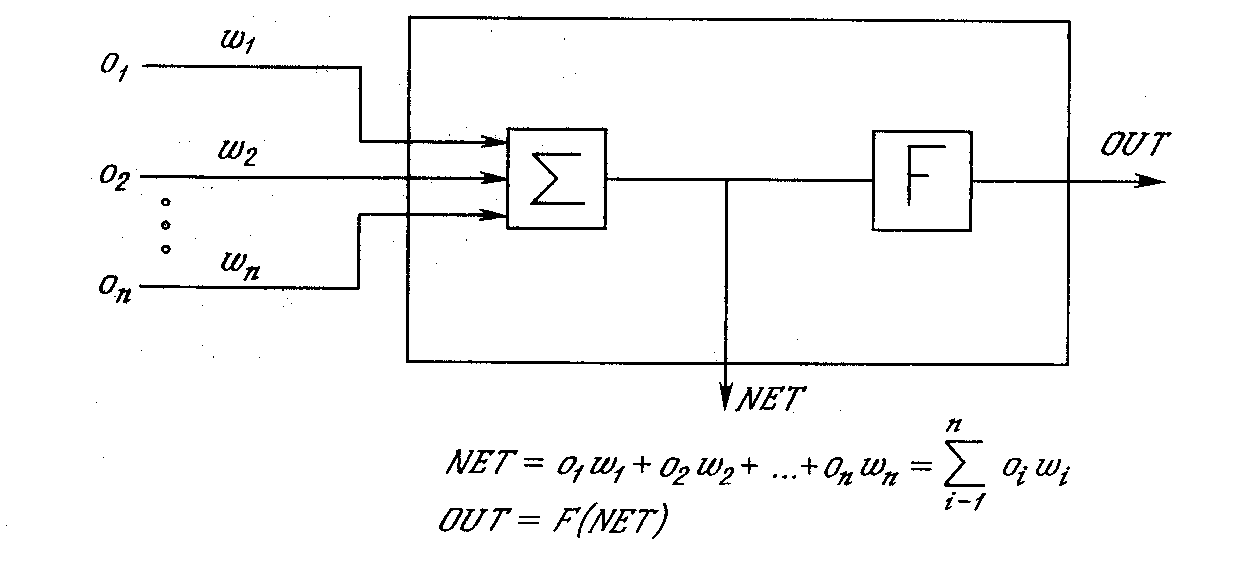


Рисунок 2. Искусственный нейрон с активационнной функцией

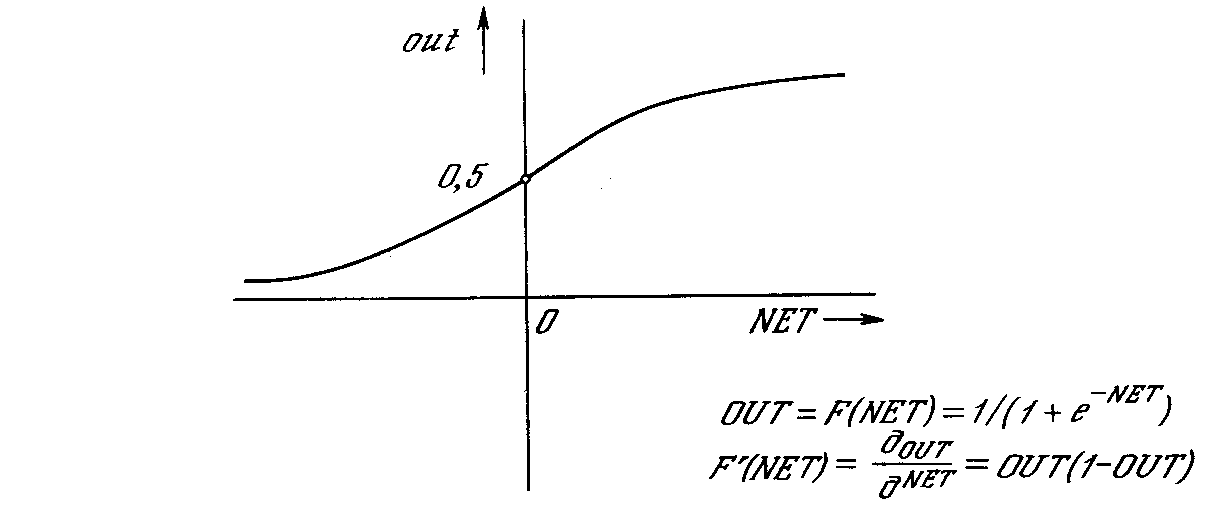


Рисунок 3. Сигмоидальная активационная функция

Как показывает уравнение (15), эта функция, называемая сигмоидом*,* весьма удобна, так как имеет простую производную, что используется при реализации алгоритма обратного распространения.

. (15)

Сигмоид, который иногда называется также логистической, или сжимающей функцией, сужает диапазон изменения NET так, что значение OUT лежит между нулем и единицей. Многослойные нейронные сети обладают большей представляющей мощностью, чем однослойные, только в случае присутствия нелинейности. Сжимающая функция обеспечивает требуемую нелинейность [5].

* 1. Языки программирования, применяемые в обучении
     1. Pascal

Язык Паскаль, названный в честь французского математика и философа Блеза Паскаля (1623-1662), был создан как учебный язык программирования в 1968-71 годах швейцарским ученым Никлаусом Виртом на кафедре информатики Стэнфордского университета. В настоящее время это язык имеет более широкую сферу применения, чем предусматривалось при его создании. Свое признание Паскаль получил с появлением пакета Турбо Паскаль (Turbo Pascal). Этот язык отличается простотой понимания, стройностью и структурностью алгоритмов, быстротой компилятора и удобными средствами создания и отладки программ.

Достоинства:

* Простой синтаксис языка. Небольшое число базовых понятий. Программы на Паскале достаточно легко читаемы.
* Достаточно низкие аппаратные и системные требования, как самого компилятора, так и программ, написанных на Паскале.
* Универсальность языка. Язык Паскаль применим для решения практически всех задач программирования.
* Поддержка структурного программирования, программирования "сверху-вниз", а также объектно-ориентированного программирования.

Недостатки:

* Распространен только в странах бывшего СССР
* Очень мало разработанного ПО
* Морально устарел
  + 1. C

С – компилируемый язык, который в 1969-1973 годах разработал Деннис Ричи. Изначально он разрабатывался для того чтобы реализовать операционную систему Unix, но позже его перенесли и на другие платформы. Успеху языка С способствовало в значительной мере то, что его конструкции очень близки к типичным машинным инструкциям, а это делает возможным его применение во многих проектах – начиная от операционных систем и заканчивая прикладным программным обеспечением для множества устройств и встраиваемых систем.

Достоинства:

* процедурный стиль программирования
* простой и логичный синтаксис
* статическая строгая типизация
* актуален
* прост в изучении
* дает понимание работы ЭВМ

Недостатки:

* достаточно бедная стандартная библиотека
  + 1. C++

C++ — [компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на [Java](https://ru.wikipedia.org/wiki/Java) и [C#](https://ru.wikipedia.org/wiki/C_Sharp).

Синтаксис C++ унаследован от языка [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как [компиляторами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

Достоинства:

* расширяет язык C
* имеет богатую стандартную библиотеку

Недостатки:

* сложен в изучении
  + 1. Java

История создания языка Java начинается в июне 1991 года, когда Джеймс Гослинг создал проект для использования в одном из своих многочисленных сет-топ проектов. Язык, который рос вне офиса Гослинга, как дуб, **Oak** - первоначальное название Java до 1995 года, после в дальнейшем история Java продолжалась под именем **Green**, а позже был переименован как Java.

Но официальной датой создания языка Java считается 23 мая 1995 года, после выпуска компанией Sun первой реализации Java 1.0. Она гарантировала «**Напиши один раз, запускай везде**», обеспечивая недорогой стоимостью на популярных платформах.

13 ноября 2006 года, Sun выпустила большую часть как свободное и открытое программное обеспечение в соответствии с условиями GNU General Public License (GPL).

После 8 мая 2007 года судьба Java сложилась иначе. Компания завершила процесс, делая все чтобы исходный код был бесплатным и открытым, кроме небольшой части кода, на который компания не имела авторских прав.

Достоинства:

* платформо-независимый
* безопасный
* актуальный

Недостатки:

* менее производительный, по сравнению с C/C++
* полностью объектно-ориентированный
  + 1. Python

Python начал разрабатываться в конце восьмидесятых годов сотрудником Голландского Национального Исследовательского Института Математики и Информатики Гвидо ван Россумом.

Python вобрал в себя черты многих популярных в то время языков программирования: Algol-68, C, C++, Modula-3 ABC, SmallTalk.

Версия 1.0 появилась в 1994 году, 2.0 в 2000-м, а 3.0 в 2008-м году. На данный момент активно развиваются вторая и третья версии этого языка. Поддержка Python'a осуществляется командой разработчиков все того же института, при этом за ван Россумом осталось право решающего голоса в вопросах развития языка.

Достоинства:

* краток и лаконичен
* платформо-независимый

Недостатки:

* динамическая типизация
* низкая производительность
  + 1. Язык ассемблера

Язык ассемблера — машинно-ориентированный язык низкого уровня с командами, не всегда соответствующими командам машины, который может обеспечить дополнительные возможности вроде макрокоманд; автокод, расширенный конструкциями языков программирования высокого уровня, такими как выражения, макрокоманды, средства обеспечения модульности программ.

Автокод — язык программирования, предложения которого по своей структуре в основном подобны командам и обрабатываемым данным конкретного машинного языка.

Язык ассемблера — система обозначений, используемая для представления в удобочитаемой форме программ, записанных в машинном коде. Язык ассемблера позволяет программисту пользоваться алфавитными мнемоническими кодами операций, по своему усмотрению присваивать символические имена регистрам ЭВМ и памяти, а также задавать удобные для себя схемы адресации (например, индексную или косвенную). Кроме того, он позволяет использовать различные системы счисления (например, десятичную или шестнадцатеричную) для представления числовых констант и даёт возможность помечать строки программы метками с символическими именами с тем, чтобы к ним можно было обращаться (по именам, а не по адресам) из других частей программы (например, для передачи управления).

Перевод программы на языке ассемблера в исполнимый машинный код (вычисление выражений, раскрытие макрокоманд, замена мнемоник собственно машинными кодами и символьных адресов на абсолютные или относительные адреса) производится ассемблером — программой-транслятором, которая и дала языку ассемблера его название.

Достоинства:

* высокое быстродействие
* дает понимание работы ЭВМ

Недостатки:

* очень высокий порог вхождения
* системные ограничения
* не применяется в чистом виде

Таблица 1. Сравнение популярных языков программирования

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Язык  Признак | Pascal | C | C++ | Java | Python | Assembly |
| Лаконичность синтаксиса | - | + | +/- | - | + | - |
| Возможность писать в процедурном стиле | + | + | +/- | - | + | + |
| Производительность | + | + | + | +/- | - | + |
| Актуальность | - | + | + | + | + | - |
| Богатая стандартная библиотека | - | - | + | + | + | - |
| Строгая типизация | + | + | + | + | - | + |
| Низкий порог вхождения | + | + | - | - | + | - |
| Понимание работы ЭВМ | - | + | + | - | - | + |

* 1. Обзор систем контроля знаний в форме контеста
     1. Описание системы ejudge

Основной сайт проекта размещается по адресу <http://www.ejudge.ru> [8]. На сайте доступна документация по системе, последние версии исходных кодов, работает Wiki-раздел и форум. Предоставляется анонимный и пользовательский доступ к системе контроля версий SVN для исходных кодов и документации системы.

Сама система ejudge способна работать в любом более-менее современном дистрибутиве Linux. Для доступа к WEB-интерфейсу требуется настроенный WEB-сервер apache. Система написана на языке C и состоит из совокупности демонов, отвечающих за основные сервисы, CGI-программ, реализующих WEB-интерфейс со стороны сервера, и утилит командной строки и утилит с текстовым интерфейсом. В настоящее время примерный объем исходного кода системы составляет порядка 150000 строк кода на языке C.

Основной интерфейс системы (как пользовательский, так и администраторский) реализован через WEB. Дополнительные административные возможности предоставляются через утилиты командной строки и программы с текстовым интерфейсом. Внешний вид генерируемых WEB-страниц может модифицироваться под потребности сайта. Утилиты командной строки позволяют получить информацию в формате, удобном для дальнейшей обработки (XML- и CSV-форматы).

Многоязыковой интерфейс. Локализация выполняется с помощью соответствующего .po-файла. Поддержка произвольных 8-битных кодировок. В настоящее время в системе поддерживается английский и русский язык сообщений.

Одновременная поддержка многих турниров.

Ведение базы пользователей с привязкой информации о пользователе к турниру, то есть каждый турнир может требовать свой набор полей информации о пользователе, при этом значение одного и того же поля может быть разным для разных турниров.

Проведение турниров с автоматической проверкой задач по нескольким системам выставления оценки. Турниры могут быть как ограниченные, так и неограниченные по времени. Дополнительные ограничения времени могут задаваться для каждой задачи индивидуально. Время участника может быть как абсолютным, так и относительным (т.н. виртуальные турниры).

Тонкое управление правами доступа пользователей. Некоторый пользователь может быть администратором одного турнира и не иметь никаких привилегий в другом турнире.

Административный WEB-интерфейс позволяет выполнить все операции по созданию и настройке нового турнира не прибегая к редактированию конфигурационных файлов.

Поддержка практически всех популярных языков программирования: C, C++, Java, Pascal, Perl, Python и т.д. При компиляции системы автоматически выявляются установленные среды программирования и выполняется начальное конфигурирование. Добавление новых языков возможно и в уже работающую систему.

"Безопасный" запуск тестируемых программ на выполнение (требует специальной поддержки со стороны ядра ОС). Тестируемая программа не может выполнить потенциально опасные операции, такие как создание новых процессов, открытие произвольного файла, создание сокета и т.п.

**Достоинства:**

* Ejudge распространяется в исходных кодах под лицензией GPL. Это означает, что система распространяется бесплатно и открыта для изменения и добавления новых возможностей.
* Настраиваемый интерфейс системы. Можно изменять внешний вид страниц, генерируемых системой.
* Возможность добавления новых языков программирования в систему.

**Недостатки:**

* Система ejudge работает под управлением ОС Linux, что не позволяет использовать языки программирования, которые реализованы только для Windows, такие как C#, Visual Basic, и др.
* Доступная на сайте разработчика документация по системе устарела и не полностью описывает систему.
* Система не предоставляет архив олимпиадных задач. Предполагается, что пользователь должен сам составить задачи и тесты к ним.
  + 1. Описание системы TopCoder

TopCoder — один из сайтов спортивного программирования. В отличие от ACM International Collegiate Programming Contest является индивидуальным [8].

Создан в апреле 2001 года. По состоянию на ноябрь 2007 года насчитывал около 130 000 пользователей, из которых около 22 000 хоть раз участвовали в Algorithm Competition.

Наиболее популярный вид турниров, проходящих в системе TopCoder — соревнование по быстрому решению алгоритмических задач (аналогично школьным и студенческим олимпиадами по программированию). Он заключается в том что каждому участнику даётся 3 задачи, разные по сложности, классифицируемые на 3 уровня. Каждая задача имеет свою максимальную стоимость в баллах. Обычно 250, 500 и 1000.

Такие матчи, называемые SRM (Single Round Match), проходят примерно раз в неделю. Кроме этого проводятся ежегодные турниры. Матч состоит из трёх основных фаз — Coding, Challenging и System Testing.

В первой фазе участники за отведённое время пытаются решить предложенные им три задачи, как правило оцениваемые в 250, 500 и 1000 баллов. Решением является создание указанного в условии класса и реализация указанного в условии метода, проходящая все заранее подготовленные тесты. Участникам разрешается писать решения на одном из следующих языков: C++, C#, Java и VB.NET. Количество очков за решённую задачу нелинейно зависит от времени отправки окончательного решения: чем позже — тем меньше очков. За каждую повторную отправку снимается 10 % стоимости задачи. Количество очков не может быть меньше 30 % стоимости задачи. Решение считается принятым (на этом этапе), если оно прошло все предварительно заложенные в систему тестирования входные данные («тесты»).

Продолжительность тура в регулярных матчах (англ. Single Round Match, сокращенно SRM), а также отборочных соревнованиях турниров (англ. Online Elimination Rounds составляет 75 минут. В очном финале (англ. Onsite Events) продолжительность первой фазы составляет 85 минут.

Во второй фазе участники пытаются подобрать тест (вариант входных данных), на котором решения его конкурентов будут работать неверным образом. При этом разрешается смотреть исходный код, но невозможно запускать программы конкурентов. Каждый удачный подход даёт 50 очков, а неудачный отнимает 25 очков. Если подход был удачным, тест добавляется в набор тестов, используемый на следующей фазе. Продолжительность этой фазы составляет 15 минут во всех матчах кроме очных финалов (10 минут).

В третьей фазе происходит тестирование всех решений всех участников, которые не были признаны неверными по итогам второй фазы. Формируются окончательные результаты матчей.

Классификация участников и их итоговая расстановка по местам определяется конечным количеством очков у участников. Участники, имеющие большее количество очков, занимают более высокие места. В случае равенства очков, все участники с данным количеством очков занимают (делят) одно и то же место.

В случае, если в ходе соревнования не происходило никаких технических сбоев, для всех участников пересчитывается рейтинг.

**Достоинства:**

* Огромная база пользователей системы, стабильность и надежность работы.
* Уникальная схема взаимодействия участников соревнований, возможность «оспорить» решения других участников.
* Удобный клиент для работы с системой, который представляет собой Java-апплет и запускается из браузера.
* Система рейтинга пользователей.

**Недостатки:**

* Система TopCoder является закрытой, в нее нельзя вносить изменения, для работы с ней нужен доступ в Интернет.
* Формат проведения соревнований TopCoder отличается от ACM ICPC, поэтому ее возможностей недостаточно для эффективной подготовки к соревнованиям ACM.
* Соревнования в TopCoder проводятся на английском языке, что создает трудности для русскоязычных пользователей.
  + 1. Описание системы UVA Online Judge

Эта олимпиадная система – разработка университета Валладолид (Испания). Сайт Online Judge расположен по адресу <http://acm.uva.es/> [8]. На сайте круглосуточно ведется борьба за первое место в рейтинге среди более 10000 участников со всего мира. Рейтинг участников определяется по количеству решенных задач. В архиве системы более двух тысяч задач, и более пяти миллионов проверенных решений. На базе Online Judge проводятся соревнования, в которых участвуют одновременно сотни программистов по всему миру. Данная система использует программу NetJudge для проверки решений – собственную разработку университета с закрытым исходным кодом.

Участник (или команда) регистрируется на сайте соревнования, заполняет данные о себе и получает идентификатор. После этого в любое время участник решает одну из задач на Pascal или C/C++ и отправляет ее по e-mail judge@uva.es. Текст программы-решения вставляется прямо в текст письма. В начало программы вставляется комментарий, в котором указывается ID участника, номер задачи и язык программирования. Это выглядит так: /\* @JUDGE\_ID: 7834КЕ 314 C \*/

**Достоинства:**

* Большой архив олимпиадных задач, который включает задачи с прошедших полуфинальных (региональных) и финальных соревнований ACM ICPC.
* Регистрация в системе бесплатна, после регистрации можно решать любые задачи и участвовать в соревнованиях.
* Обширная статистика по задачам, возможность обсуждения задач с другими пользователями на форуме проекта.

**Недостатки:**

* Система Online Judge является закрытой, в нее нельзя вносить изменения, для работы с ней нужен доступ в Интернет.
* В системе не поддерживаются современные языки программирования, такие как C# и Java.
* Низкая производительность системы, перегруженный интерфейс.
  + 1. Описание системы BACS

Информация о проекте находится на сайте [8]. Автоматизированная система BACS – собственная разработка кафедры ПО ИжГТУ. Активно используется в ИжГТУ с 2006 года и по настоящее время для тренировок участников соревнований ACM ICPC. При этом идет постоянная доработка возможностей системы.

Система работает под операционной системой Windows 2000 или старше. Проверка безопасности осуществляется с помощью поиска запрещенных слов в тексте программ.

**Достоинства**

Достоинства системы заключаются в следующем:

* система BACS поддерживает многие современные языки, такие как C# и Java.
* возможность добавления новых языков программирования в систему.
* можно запустить параллельно несколько серверов проверки.

**Недостатки**

Можно отметить следующий недостаток:

* система BACS осуществляет контроль безопасности с помощью списка запрещенных слов, и при обновлении языков могут появиться новые процедуры, в результате чего система станет уязвимой.
* неудобный пользовательский интерфейс
  1. Постановка цели и задач исследования

**Целью** диссертационной работы является разработка методики вычисления оценки качества усвоения студентами учебного материала на основе статистики решенных практических задач.

Для достижения цели поставлены следующие **задачи**:

* 1. Исследовать параметры, определяющие качество усвоения студентами учебного материала.
  2. Разработать метод расчета комплексной меры качества усвоения на базе статистических данных.
  3. Разработать программный инструмент для оценки качества усвоения студентами учебного материала.
  4. Протестировать полученный инструмент, произвести анализ полученных результатов. Для тестирования системы использовать практические задачи из обучающего курса по программированию системы BACS.
  5. Выводы

В данной главе описана специфика автоматизированного обучения программированию. Описаны основные методы контроля знаний при подготовке высококвалифицированного инженера- программиста. Произведены описания методов, использованных при разработке методики оценки сложности задач по программированию на основе исходного кода, а именно описана сущность статического анализа кода программы, основные этапы процесса статического анализа. Произведен обзор и анализ современных систем контроля знаний в данной области, указаны их достоинства и недостатки. Также представлены методы решения проблемы подбора весовых коэффициентов. Сформулирована постановка задачи исследования.