Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова»

К защите

Руководитель направления

д.т.н. профессор

Архипов И. О. «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2018 г.

Девятов Алексей Витальевич

ОЦЕНКА СБАЛАНСИРОВАННОСТИ УЧЕБНЫХ КУРСОВ ПО ПРОГРАММИРОВАНИЮ

230100 «Информатика и вычислительная техника»

230100.68-11 «Разработка программно-информационных систем»

Диссертация на соискание академической степени

магистра

Магистрант

Девятов А. В.

Научный руководитель

к.т.н., профессор Тарасов В. Г.

Руководитель программы

д.т.н., профессор

Архипов И. О.

Ижевск 2018

СОДЕРЖАНИЕ

[ТЕРМИНЫ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4](#_Toc265485551)

[РЕФЕРАТ 6](#_Toc265485552)

ВВЕДЕНИЕ…………………………………………………………………....7

[1.АНАЛИЗ 11](#_Toc265485553)1

[1.1. Методы контроля знаний при подготовке программистов 11](#_Toc265485554)

[1.1.1. Уровни подготовки программиста 11](#_Toc265485555)

[1.1.2. Доклады и рефераты 13](#_Toc265485556)

[1.1.3. Тестирование 13](#_Toc265485557)

[1.1.4. Промежуточные программные продукты 14](#_Toc265485558)

[1.1.5. Решение алгоритмических задач на время 15](#_Toc265485559)

[1.1.6. Законченные программные продукты 15](#_Toc265485560)

[1.1.7. Публичная защита программного продукта. 16](#_Toc265485561)

[1.2. Языки программирования, применяемые в обучении 17](#_Toc265485562)

[1.2.1. Pascal 17](#_Toc265485563)

[1.2.2. C 18](#_Toc265485564)

[1.2.3. C++ 19](#_Toc265485565)

[1.2.4. Java 19](#_Toc265485566)

[1.2.5. Python 20](#_Toc265485567)

[1.3. Обзор систем контроля знаний в форме контеста 22](#_Toc265485571)

[1.3.1. Описание системы ejudge 22](#_Toc265485572)

[1.3.2. Описание системы TopCoder 25](#_Toc265485573)

[1.3.3. Описание системы UVA Online Judge 27](#_Toc265485574)

[1.3.4. Описание АС Олимп 28](#_Toc265485575)

[1.3.5. Описание системы BACS win32 29](#_Toc265485576)

[1.4. Постановка цели и задач исследования 30](#_Toc265485577)

[1.5. Выводы 31](#_Toc265485578)

[2. ГЛАВА ТЕОРЕТИЧЕСКАЯ 32](#_Toc265485579)

[2.1. Понятие сложности программы. Комплексная мера 32](#_Toc265485580)

[2.2. Анализ метрик сложности программы 34](#_Toc265485581)

[2.2.1. Классификация метрик 35](#_Toc265485582)

[2.2.2. Метрика Холстеда 36](#_Toc265485583)

[2.2.3. Метрика Мак-Кейба. 38](#_Toc265485584)

[2.2.4. Метрика Джилба 42](#_Toc265485585)

[2.2.5. Дополнительные метрики 42](#_Toc265485586)

[2.3. Расчет весовых коэффициентов 44](#_Toc265485587)

[2.3.1. Прямая расстановка 44](#_Toc265485588)

[2.3.2. Ранжирование факторов 45](#_Toc265485589)

[2.3.3. Присвоение коэффициентов факторам*.* 46](#_Toc265485590)

[2.3.4. Метод анализа иерархий 47](#_Toc265485591)

[2.3.5. Пример расчета весовых коэффициентов по МАИ 50](#_Toc265485592)

[2.4. Пример расчета сложности программы 51](#_Toc265485593)

[2.5. Выводы 54](#_Toc265485594)

[3. ПРИМЕНЕНИЕ МЕТОДИКИ. АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 55](#_Toc265485595)

[3.1. Описание разработанного инструмента 55](#_Toc265485596)

[3.2. Описание системы BACS 57](#_Toc265485597)

[3.2. Анализ полученных результатов 58](#_Toc265485598)

[3.2.2. Оценка авторских решений 67](#_Toc265485599)

[3.3. Выводы 70](#_Toc265485602)

[ЗАКЛЮЧЕНИЕ 71](#_Toc265485603)

[СПИСОК ЛИЕТРАТУРЫ 72](#_Toc265485604)

[ПРИЛОЖЕНИЕ Исходный код блока расчета метрик инструмента оценки сложности задач 74](#_Toc265485605)

# ТЕРМИНЫ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ГА – генетический алгоритм

МНК – метод наименьших квадратов

МАИ – метод анализа иерархий

ЯП – язык программирования

АОС – автоматизированные обучающие системы представляют собой комплексы научно-методической, учебной и организационной поддержки процесса обучения, проводимого на базе компьютерных, или, как их также называют, информационных технологий.

Методика — совокупность методов практического выполнения чего-либо, обучения чему-либо, воспитания.

Метод— систематизированная совокупность шагов, действий, которые необходимо предпринять, чтобы решить определенную задачу или достичь определенной цели.

Метрика программного обеспечения — это [мера](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B0), позволяющая получить численное значение некоторого свойства [программного обеспечения](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) или [его спецификаций](http://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F).

Алгоритм, от имени учёного [аль-Хорезми](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D1%8C-%D0%A5%D0%BE%D1%80%D0%B5%D0%B7%D0%BC%D0%B8) - точный набор [инструкций](http://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%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%D0%B5)), описывающих порядок действий исполнителя для достижения результата [решения задачи](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87) за конечное время.

Статический анализ кода ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Static program analysis) — анализ [программного обеспечения](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), производимый без реального выполнения исследуемых программ. В большинстве случаев анализ производится над какой-либо версией [исходного кода](http://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), хотя иногда анализу подвергается какой-нибудь вид объектного кода, например [P-код](http://ru.wikipedia.org/wiki/P-%D0%BA%D0%BE%D0%B4) или код на [MSIL](http://ru.wikipedia.org/wiki/MSIL).

Динамический анализ кода ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Dynamic program analysis) — анализ [программного обеспечения](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), выполняемый при помощи выполнения программ на реальном или виртуальном [процессоре](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80). Утилиты динамического анализа могут требовать загрузки специальных библиотек, перекомпиляцию программного кода. Некоторые утилиты могут модифицировать исполняемый код в процессе исполнения или перед ним.

Контест – форма проведения соревнования по спортивному программированию, заключается в решении задач по программированию на время.

# РЕФЕРАТ

Магистерская диссертация содержит введение, 3 главы с выводами, заключение, библиографический список использованной литературы и приложение, изложенные на 85 страницах машинописного текста, содержит 8 рисунков и 5 таблиц.

Во введении приведен анализ потенциала применения информационных технологий для обучения программированию, обоснована значимость контроля знаний во время обучения, обоснована актуальность темы исследования, описан объект и предмет исследования, установлена научная новизна работы, освещена практическая значимость работы, а также приведено краткое содержание диссертации.

В первой главе произведен 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 методы контроля знаний в образовательном процессе.

Во второй главе представлена разработка модели сложности программы и комплексной меры, исследование и анализ методов расчета комплексной меры, исследование и анализ метрик программного обеспечения их достоинств и недостатков, исследование методов определения весовых коэффициентов на основе экспертных мнений и обоснование применения выбранного метода для расчета.

В третьей главе представлена структурная схема инструмента оценки сложности и функциональные схемы основных его составляющих блоков. Приведены и проанализированы результаты применения разработанной методики к задачам из архива системы проведения соревнований по программированию BACS, проведено сравнение полученных результатов с экспертными оценками. Сделан общий вывод о работе инструментов.

В заключении приведены основные результаты работы.

# ВВЕДЕНИЕ

Развитие методологии программированного обучения и компьютеризации всех сфер человеческой деятельности привели к широкому применению ЭВМ в учебном процессе. Компьютерное обучение предполагает использование специализированного программного обеспечения — обучающие программы, автоматизированные обучающие системы (АОС).

Одним из важных элементов обучения является контроль усвоения знаний, умений и навыков. Контролирующие программы и модули обучающих систем предназначены для текущего и итогового контроля. При этом АОС позволяют накапливать статистическую информацию по нескольким параметрам и проследить успеваемость каждого обучаемого в динамике, определить эффективность обучения в зависимости от начального уровня знаний обучаемого, сложности и объема материала, времени, затраченного на проработку темы и т. д.

Контроль знаний является неотъемлемой частью на каждом этапе обучения, поэтому АОС в области программирования должна включать улучшенную подсистему тестирования знаний, умений и навыков обучаемых. Эта подсистема призвана обеспечивать оценку исходного уровня подготовки обучаемого, текущий и итоговый контроль усвоения материала. Основным видом заданий для такого контроля являются задачи на составление программ.

Для постоянного контроля качества знаний учащихся необходимо разработать множество задач. Выделяются различные тематики, каждой из которых требуется посвятить отдельный учебный курс. Задачи требуется также относить к определенному этапу обучения, различая их по сложности и по охватываемому материалу.

Для постоянного контроля качества знаний учащихся необходимо разработать множество задач. Выделяются различные тематики, каждой из которых требуется посвятить отдельный учебный курс. Задачи требуется также относить к определенному этапу обучения, различая их по сложности и по охватываемому материалу.

Одной из важных задач в обучении является построение качественного и сбалансированного учебного курса. Для начального обучения необходимо выбрать базовые темы. В дальнейшем нужно переходить к более сложному материалу, при этом он должен логически выходить из уже усвоенного. Возникает необходимость автоматизации выбора задач для построения курса обучения.

**Целью работы является** разработка методики вычисления оценки сбалансированности учебного курса на основе анализа исходного кода практических задач.

Для достижения поставленной цели в работе решаются следующие **задачи:**

* исследовать параметры, определяющие сложность программы;
* разработать метод расчета комплексной меры сложности программы на базе количественных показателей программного кода – метрик;
* разработать программный инструмент для оценки сбалансированности учебного курса по программированию на основе сложности практических задач;
* протестировать полученный инструмент, произвести анализ полученных результатов.

**Объектом исследования** является обучение программированию и оценка сбалансированности учебного курса по программированию.

**Предметом исследования** является методика оценки сбалансированности учебного курса по программированию на основе сложности практических задач.

**Научная новизна** работы состоит в разработке методики оценки сбалансированности учебного курса по программированию на основе сложности практических задач. Сложность задачи вычисляется, путем измерения отдельных свойств программы – метрик. Для получения интегральной оценки сложности разработан метод расчета комплексной меры.

**Практическая ценность работы** состоит в полученной методике, на базе которой был разработан программный инструмент для оценки сбалансированности тем и задач учебного курса по программированию на основе исходного кода решений.

Данный инструмент может быть использован для решения следующих задач:

* построение динамической траектории обучения;
* группировка базы задач по уровню сложности;
* анализ разработанного курса задач, выявление сильного разброса задач по сложности;
* построение оптимального курса с учетом анализа сложности тем.

**Публикации.** По теме данной диссертации опубликована 1 статья.

**Структура и объем работы.** Магистерская диссертация содержит введение, 3 главы с выводами, заключение, библиографический список использованной литературы и два приложения изложенных на 98 страницах машинописного текста, содержит 10 рисунков и 5 таблиц.

Во введении приведен анализ потенциала применения информационных технологий для обучения программированию, обоснована значимость контроля знаний во время обучения, обоснована актуальность темы исследования, описан объект и предмет исследования, установлена научная новизна работы, освещена практическая значимость работы, а также приведено краткое содержание диссертации.

В первой главе произведен 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. Поставлены цели и задачи исследования.

Во второй главе определены понятия сложности программы и комплексной меры. Выбран метод расчета комплексной меры. Проведено исследование и анализ метрик программного обеспечения их достоинств и недостатков. Исследованы методы определения весовых коэффициентов на основе экспертных мнений. Проведены примеры расчета оценки сложности для типовых программных конструкций.

В третьей главе представлена функциональная схема инструмента оценки сложности, основные его составляющие блоки. Описана АОС, на базе которой была произведена апробация разработанной методики, исследован процесс расчета сложности задач на тестовой выборке, исследованы проблемы, возникающие при оценке большого количества решений, и предложены методы их решения. Для адекватной оценки проведено сравнение полученных результатов с экспертными оценками.

В заключении приведен анализ результатов работы.

# 1.АНАЛИЗ

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 методы контроля знаний в образовательном процессе.

## 1.1. Методы контроля знаний при подготовке программистов

### 1.1.1. 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. 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 [4].

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, 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 (или проектов);
* решение алгоритмических задач за лимитированный промежуток времени;
* публичную защиту программных продуктов.

### ****1.1.2. 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 вычисли тельных 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 [4]. 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 применения полученной информации для практического решения задач.

### ****1.1.3. 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) вы явить связи данного вопроса с другими вопросами данной (или смежной) дисциплины. Преимущества использования тестирования: 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.

### 1.1.4. 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 [4]. 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.

### ****1.1.5. 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. Данный вид контроля осуществляется посредством проведения контестов. В каждом контесте для участников предлагается решения некоторого числа задач различной сложности и тематики за ограниченный промежуток времени, что позволяет выявлять 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.

### ****1.1.6. 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 [4]. Оценка за курсовую роботу выставляется коллективом преподавателей, при этом учитываются глубина проработки темы курсовой работы, содержание и правильность по строения доклада, аргументация в защиту принятых решений, умение отвечать на дополнительные вопросы.

### ****1.1.7. 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 [4].

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.

Учитывая вышесказанное, можно сделать вывод, 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.

## 1.2. Языки программирования, применяемые в обучении

### 1.2.1. 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

### 1.2.2. 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

### 1.2.3. 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

### 1.2.4. 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

### 1.2.5. 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

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 | - | + | + | - | - | + |

## 1.3. Обзор систем контроля знаний в форме контеста

### 1.3.1. 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.

### 1.3.2. 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.

### 1.3.3. 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.

### 1.3.4. Описание АС Олимп

Автоматизированная система Олимп – собственная разработка кафедры ПО ИжГТУ. Активно использовалась в ИжГТУ с 2001 года для тренировок команды участников соревнований ACM ICPC. В 2003 году предпринималась попытка разработки новой олимпиадной системы AcmOlimp на базе АС Олимп, но данный проект не был доведен до конца.

Система представляет собой набор утилит на языке C и shell-скриптов для операционной системы Linux. Администрирование системы производится вручную, путем редактирования файлов настроек. Система предоставляет простой веб-интерфейс для пользователей, в котором можно посмотреть условия задач, таблицу результатов, а также отослать решение на проверку.

**Достоинства:**

* Открытый архив олимпиадных задач. Впоследствии из этого архива была сформирована основная база задач для проекта, описываемого в данной дипломной работе.

**Недостатки:**

* В открытом доступе нет документации по системе. Не документированы настройки и процедуры администрирования системы.
* Возможности системы очень ограничены.
* В системе не поддерживаются современные языки программирования, такие как C# и Java.
* Нестабильная работа системы, постоянные зависания и сбои.

### 1.3.5. Описание системы BACS win32

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 следующие недостатки:

* система BACS осуществляет контроль безопасности с помощью списка запрещенных слов, и при обновлении языков могут появиться новые процедуры, в результате чего система станет уязвимой.

## 1.4. Постановка цели и задач исследования

**Целью** диссертационной работы является разработка методики оценки сбалансированности учебного курса по программированию на основе сложности исходного кода практических задач.

Для достижения цели поставлены следующие **задачи:**

1. Исследовать параметры, определяющие сложность исходного кода программы.
2. Произвести анализ и выбор эффективных метрик программного обеспечения.
3. Разработать метод расчета комплексной меры сложности программы на базе измеримых характеристик программы - метрик.
4. Проанализировать применимость полученной методики для оценки сбалансированности учебного курса по программированию.
5. Разработать программный инструмент для оценки сложности задач по программированию.
6. Протестировать разработанный инструмент, произвести анализ полученных результатов.

## 1.5. Выводы

В данной главе описана специфика автоматизированного обучения программированию. Описаны основные методы контроля знаний при подготовке высококвалифицированного инженера - программиста. Произведен обзор языков программирования, наиболее часто применяющихся в обучении школьников, выделены их плюсы и минусы. Произведен обзор и анализ современных систем контроля знаний в данной области, указаны их достоинства и недостатки. Сформулирована постановка задачи исследования.

# 2. ГЛАВА ТЕОРЕТИЧЕСКАЯ

Целью данной главы является разработка модели сложности программы и комплексной меры, исследование и анализ методов расчета комплексной меры, исследование и анализ метрик программного обеспечения их достоинств и недостатков, исследование методов определения весовых коэффициентов на основе экспертных мнений и обоснование применения выбранного метода для расчета. Рассчитать весовые коэффициенты на основе выбранного метода, провести тестирование методики оценки сложности на контрольном примере.

## 2.1. Понятие сложности программы. Комплексная мера

На протяжении всего обучения необходимо оценивать уровень подготовленности обучающихся. В начале – для того, чтобы узнать начальный уровень, на последующих – чтобы наблюдать за прогрессом обучения. Чтобы определить этот уровень, необходимо знать с каким уровнем сложности задач может справиться студент на текущем этапе обучения. Для того чтобы решить задачу по программированию, необходимо разработать программу, которая должна на тестовый набор входных данных выдать соответствующий набор выходных данных удовлетворяющих условию задачи, поэтому для определения сложности задачи необходимо оценить сложность ее решения.

Зачастую под сложностью программы понимают различные понятия. Одним из них является сложность алгоритма. Трудоёмкость алгоритмов по-разному зависит от входных данных. Для некоторых алгоритмов трудоемкость зависит только от объема данных, для других алгоритмов — от значений данных, в некоторых случаях порядок поступления данных может влиять на трудоемкость. Трудоёмкость многих алгоритмов может в той или иной мере зависеть от всех перечисленных выше факторов .

Одним из упрощенных видов анализа, используемых на практике, является асимптотический анализ алгоритмов. Целью асимптотического анализа является сравнение затрат времени и других ресурсов различными алгоритмами, предназначенными для решения одной и той же задачи, при больших объемах входных данных. Используемая в асимптотическом анализе оценка функции трудоёмкости, называемая сложностью алгоритма, позволяет определить, как быстро растет трудоёмкость алгоритма с увеличением объема данных [15].

Иногда под сложностью понимают, насколько легко разобраться в программе, прочитав ее исходный код. Здесь оценивается стилистика и понятность и читабельность кода, которая завит от количества комментариев, поясняющих код, от разбиения больших участков кода на подпрограммы и.т.д. [13]

В данной работе под понятием сложности программы понимается следующее.

Сложность программы зависит от: размера программы; ее структур данных; структур управления, определяемых логикой программы; правильного ее разбиения на модули; внутренних связей каждого модуля; межмодульных связей в программе [18]. Таким образом сложность программы является латентной величиной, т.е. ее нельзя непосредственно измерить, но можно судить о значении сложности по определенным индикаторам ( свойствам программы ) – метрикам.

Так как о сложности программы приходится судить по ряду показателей - метрик, то для получения интегральной оценки воспользуемся комплексной мерой. Для расчета комплексной оценки используем формулу:

, (1)

где Mi – значения выбранных метрик, Ri ­­ - значения весовых коэффициентов.

Формула (1) строится на основе формулы Кокола [18]. Отличие заключается в том, что в первоначальном виде комплексная оценка строилась на базе одной главной метрики и корректировалась за счет некоторого количества дополнительных. В данной же формуле, отсутствует деление метрик на главные и корректирующие, а их влияние на итоговую оценку определяется соответствующим весовым коэффициентом, рассчитанным на основе экспертных мнений.

В итоге применение разработанной методики вычисления комплексной оценки производится в два этапа. Первый – это выбор метрик из уже существующих удовлетворяющих предметной области, и разработка собственных метрик для получения полноты данных о всех составляющих программы. Второй – расстановка весовых коэффициентов.

## 2.2. Анализ метрик сложности программы

На данном этапе определяются свойства программы, которые будут использованы для получения комплексной меры, и таким образом их можно измерить. Обзор метрик показал, насколько велико и разнообразно их количество. При выборе метрик необходимо учитывать ограничения, которые исходят из формы проведения практических занятий в форме контеста. Решение, подготовленное учащимся, представляет собой небольшую программу менее 500 строк кода. В программах нельзя использовать сторонние библиотеки. Исходный код программы должен быть в одном файле. Ввод данных производится из входного файла или со стандартного ввода.

Поэтому большинство метрик, используемых для оценок сложности, качества и надежности больших проектов, не будут эффективны для оценки программ обучающихся, исходя из этого необходимо исследовать существующие наборы метрик, которые оптимально подходят для предметной области.

### 2.2.1. Классификация метрик

Метрики сложности программ принято разделять на три основные группы [11]:

* объемные метрики;
* метрики сложности потока управления программ;
* метрики сложности потока данных программ.

К первой группе относятся метрики, основанные на количественной характеристике исходного кода программы. Эти метрики очень популярны, ввиду своей простоты. К наиболее известным метрикам данной группы относятся число операторов программы, количество строк исходного текста, набор метрик Холстеда[17]. Метрики этой группы ориентированы на анализ исходного текста программ.

Ко второй группе относятся метрики, основанные на анализе управляющего графа программы. Данный граф строится в виде ориентированного графа, в котором вершины – вычислительные операторы или выражения, а дуги – передача управления между узлами.

К третьей группе относятся метрики, основанные на оценке использования, конфигурации и размещения данных в программе. В первую очередь это касается глобальных переменных. К данной группе относятся метрики Чепина[19], подсчет спена[11].

**Метрические шкалы**

В зависимости от характеристик и особенностей применяемых метрик им ставятся в соответствие различные измерительные шкалы.

Номинальной шкале соответствуют метрики, классифицирующие программы на типы по признаку наличия или отсутствия некоторой характеристики без учета градаций.

Порядковой шкале соответствуют метрики, позволяющие ранжировать некоторое характеристики путем сравнения с опорными значениями, т.е. измерение по этой шкале фактически определяет взаимное положение конкретных программ.

Интервальной шкале соответствуют метрики, которые показывают не только относительное положение программ, но и то, как далеко они отстоят друг от друга.

Относительной шкале соответствуют метрики, позволяющие не только расположить программы определенным образом и оценить их положение относительно друг друга, но и определить, как далеко оценки отстоят от границы, начиная с которой характеристика может быть измерена.

### 2.2.2. Метрика Холстеда

Метрика Холстеда относится к метрикам, вычисляемым на основании анализа числа строк и синтаксических элементов исходного кода программы.

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

* NUOprtr (Number of Unique Operators) — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
* NUOprnd (Number of Unique Operands) — число уникальных операндов программы (словарь операндов);
* Noprtr (Number of Operators) - общее число операторов в программе;
* Noprnd (Number of Operands) - общее число операндов в программе.
* На основании этих характеристик рассчитываются оценки:
* Словарь программы (Halstead Program Vocabulary, HPVoc): HPVoc = NUOprtr + NUOprnd;
* Длина программы (Halstead Program Length, HPLen): HPLen = Noprtr + Noprnd;
* Объем программы (Halstead Program Volume, HPVol): HPVol = HPLen log2 HPVoc;
* Сложность программы (Halstead Difficulty, HDiff): HDiff = (NUOprtr/2) × (NOprnd / NUOprnd);
* На основе показателя HDiff предлагается оценивать усилия программиста при разработке при помощи показателя HEff (Halstead Effort): HEff = HDiff × HPVol.

Данная метрика дает наиболее полную оценку о размере и сложности программы, основанную на анализе синтаксических элементов исходного кода программы.

На рисунках а) б) приведены примеры двух простейшие участков кода

void func()

{

cout << 1 << 2 << 3;

}

void func ()

{

cout << 1;

cout << 2;

cout << 3;

}

Если бы в качестве метрики сложности по размеру программы выступала LOC(lines of code ) то мы бы получили соответственно величины 2 и 4, хотя очевидно что эти два участка кода идентичны, за исключением того, что потребуется дополнительное время затратить на написание двух дополнительных раза «cout». Преимущество метрики Холстеда очевидно, так как и в первом и во втором случае получается одинаковое количество уникальных операторов 2, а количество уникальных операндов 4, общее число операндов в первом случае 4 во втором 6. Получаем сложность по Холстеду соответственно (2) и (3):

C\_H1 = 2/2 \* 4/4 = 1. (2)

C\_H2 = 2/2 \* 6/4 = 1.5. (3)

С другой стороны - одной метрики Холстеда не достаточно, так как эта метрика не исследует логику программы это видно на примере:

void func()

{

b = -a;

a -= c \* b;

c \*= a;

}

void func ()

{

if(a < b)

c = a + b;

else

c = a;

}

Участки кода имеют одинаковую сложность 7, так как имеют одинаковое число уникальных операторов 6, число уникальных операндов 3, число всех операндов 7.

C\_H = 6/2 \* 7/3 = 7. (4)

Однако, код с ветвлением должен обладать большей сложностью. Он более сложен в проектировании, более сложен в отладке и нужны дополнительные знания ЯП. Поэтому необходимо в совокупности метрики Холстеда использовать метрики анализирующие логическую структуру программы.

### 2.2.3. Метрика Маккейба

Относится к группе метрик сложности потока управления программ. Как правило, с помощью этих оценок оперируют либо плотностью управляющих переходов внутри программ, либо взаимосвязями этих переходов[3].

И в том, и в другом случае стало традиционным представление программ в виде управляющего ориентированного графа G=(V, E), где V – вершины, соответствующие операторам, а E – дуги, соответствующие переходам.

Показатель цикломатической сложности является одним из наиболее распространенных показателей оценки сложности программных проектов. Данный показатель был разработан ученым Маккейбом в 1976 г., относится к группе показателей оценки сложности потока управления программой и вычисляется на основе графа управляющей логики программы (control flow graph). Данный граф строится в виде ориентированного графа, в котором вычислительные операторы или выражения представляются в виде узлов, а передача управления между узлами – в виде дуг.

Показатель цикломатической сложности позволяет не только произвести оценку трудоемкости реализации отдельных элементов программного проекта и скорректировать общие показатели оценки длительности и стоимость проекта, но и оценить связанные риски и принять необходимые управленческие решения.

Упрощенная формула вычисления цикломатической сложности представляется следующим образом:

C = e – n + 2, (5)

где e – число ребер, а n – число узлов на графе управляющей логики.

Как правило, при вычислении цикломатической сложности логические операторы не учитываются.

В процессе автоматизированного вычисления показателя цикломатической сложности, как правило, применяется упрощенный подход, в соответствии с которым построение графа не осуществляется, а вычисление показателя производится на основании подсчета числа операторов управляющей логики (if, switch и т.д.) и возможного количества путей исполнения программы.

Цикломатическое число Маккейба показывает требуемое количество проходов для покрытия всех контуров сильно связанного графа или количества тестовых прогонов программы, необходимых для исчерпывающего тестирования по принципу «работает каждая ветвь».

Показатель цикломатической сложности может быть рассчитан для модуля, метода и других структурных единиц программы.

Существует значительное количество модификаций показателя цикломатической сложности.

«Модифицированная» цикломатическая сложность – рассматривает не каждое ветвление оператора множественного выбора (switch), а весь оператор как единое целое.

«Строгая» цикломатическая сложность – включает логические операторы.

«Упрощенное» вычисление цикломатической сложности – предусматривает вычисление не на основе графа, а на основе подсчета управляющих операторов.

Данная метрика не может различать циклические конструкции и условные конструкции. Также недостатком метрики является невозможность оценки сложности предикатов, поэтому программы, имеющие одинаковый граф будут иметь равную сложность, хотя будут иметь разные по сложности предикаты.

Рассмотрим следующие два примера с использованием конструкций if else и switch исходные коды и управляющие графы представлены на рисунке 2.

If(a == 1)

A();

else

if(a == 2)

B();

Else

C();

switch(a)

{

case 1: A(); break;

case 2: B(); break;

case 3: C(); break;

}

Рисунок 2. Управляющий граф программы примера 1(слева), примера 2 (справа)

Вычисли цикломатическую сложность для первого и второго примера (6) (7):

C\_M1 = 6 – 5 + 2 = 3. (6)

C\_M2 = 7 – 6 + 2 = 3. (7)

Таким образом, использование модифицированной цикломатической сложности, которая не учитывает каждое ветвление оператора switch, является не лучшим выбором для оценки решений, где каждый оператор имеет важное значение.

Существуют и очевидные минусы данной метрики в примере

while( a != 1)

A();

if(a == 1)

A();

else

B();

if(a == 1)

A();

Управляющие графы представлены на рисунке 3.



Рисунок 3. Управляющие графы программ

Цикломатическая сложность равна для всех 2, несмотря на то, что это три различные конструкции. К тому же при изменении условия в одном из примеров:

(a ==1) на (a == 1 && b!= 0 || b == 13) цикломатическая сложность не изменится. Таким образом, возникает необходимость для анализа составных условий, и четкое разделение конструкций на условные и циклические.

### 2.2.4. Метрика Джилба

Одной из наиболее простых, но, как показывает практика, достаточно эффективных оценок сложности программ является метрика Т. Джилба [7], в которой логическая сложность программы определяется как насыщенность программы выражениями типа IF-ELSE. При этом вводятся две характеристики: 1) CL – абсолютная сложность программы, характеризующаяся количеством операторов условия; 2) cl – относительная сложность программы, характеризующаяся насыщенностью программы операторами условия, т.е. cl определяется как отношение CL к общему числу операторов.

Дополнив метрику Джилба подсчетом циклических конструкций, можно получить метрику, которая позволит в совокупности с метрикой Маккейба эффективно определять цикломатическую сложность программы и распознавать наличие обеих ветвей условного оператора, а также анализировать циклические конструкций.

### 2.2.5. Дополнительные метрики

Для более полного анализа характеристик программ возникла необходимость в разработке дополнительных метрик. Данные метрики были разработаны в результате исследования особенностей спортивного программирования.

Метрика суммарной сложности условий. Целью данной метрики является расчет сложности составных условии в условном операторе if и циклических операторов for и while. Ни одна из выше описанных метрик не уделяет внимания на сложность условий. Однако чем больше составное условие, тем сложнее отлаживать работу программы и зачастую в составных условиях допускаются ошибки.

Расчет метрики производится следующим образом. Для каждого составного условия подсчитывается количество логических операторов cntLO. Итоговое значение метрики получается по формуле :

где Cslo  - итоговое значение метрики, cntLOi – значение количества логических операторов i – го составного условия.

Для примера рассмотрим два условных оператора:

if(a == 1)

f();

if(!(fl && a == 2) )

f();

Для первого примера величина метрики равна Cslo  = 1 (один логический оператор “==”), для второго Cslo  = 3 (“!”, “&&”, “==”).

Вторая дополнительная метрика, основана на вычислении каждой уровня вложенности каждой подпрограммы, и получении суммарного уровня вложенности программы. Расчет метрики осуществляется по формуле

где Csld  - суммарный уровень вложенности программы, maxLDi – максимальный уровень вложенности i-той подпрограммы. Рассмотрим вычисление данной метрики на примере:

int sum = 0;

int f1(int m)

{

if(m < 0)

return sum;

}

void f2(int a)

{

a++;

}

Для функции f1() максимальный уровень вложенности равен 2, для f2() – 1, тогда суммарный уровень данного участка кода равен 3.

## 2.3. Расчет весовых коэффициентов

### 2.3.1. Прямая расстановка

Эксперты расставляют веса факторам, исходя из некоторого требования, например, чтобы сумма всех весов была равна 1 либо 100%, или может быть выбрано и любое другое значение.

Трудности этого подхода заключаются в том, что эксперту приходится одновременно держать в поле видимости все факторы в неявном виде, поскольку, присваивая определенное числовое значение конкретному фактору, эксперт должен параллельно сопоставить его со всеми остальными. С увеличением числа факторов, сложность растет в геометрической прогрессии.

Есть еще и техническое затруднение в работе эксперта, связанное с необходимостью контроля текущей суммы весовых коэффициентов, чтобы не оказаться перед фактом превышения заданной константы или оставить на последние факторы слишком большой вес. Если это происходит, то приходится переопределять уже присвоенные коэффициенты, что может происходить несколько раз, пока этот своеобразный цикличный процесс не завершится. Число повторений увеличивается по мере роста количества факторов.

### 2.3.2. Ранжирование факторов

Этот подход существенно легче предыдущего, потому что не требует контроля общей суммы коэффициентов. Здесь от экспертов требуется провести упорядочивание обследуемых факторов, формирующих объект, по степени проявления их свойств в возрастающем или убывающем порядке:

R11, R21, … , Rn1;

R12, R22, … , Rn2;

………………; (10)

R1m, R2m, … , Rnm

где *Rij* - ранг (место), присвоенный фактору *Oij* - м экспертом в ряду из *n* рассмотренных объектов, ранжированных этим экспертом по степени проявления анализируемого свойства. В случае присвоения двум или более факторам одинакового ранга, он получается дробным. Сводные оценки весовых коэффициентов можно получить путем усреднения частных рангов по столбцам.

По сравнению с другими методами, метод ранжирования является более простым – это его преимущество. Но усреднение рангов ведет к увеличению погрешности весовых коэффициентов и, следовательно, к менее точным результатам. Также, как и при прямой расстановке, сохраняется необходимость держать в поле зрения все факторы.

### 2.3.3. Присвоение коэффициентов факторам

В соответствии с этим методом, факторы оцениваются по некоторой балльной шкале, например от 1 до 10. Тогда получаем выражения (24)

Y11, Y21, … , Yn1;

Y12, Y22, … , Yn2;

………………; (11)

Y1m, Y2m, … , Ynm

где *Yij* — балльная оценка фактора, полученная от *j*-го эксперта, *n* — количество факторов, *m* — число экспертов.

Сводные оценки весовых коэффициентов обычно находят, подбирая соответствующую регрессионную модель. Используя следующие формулы, можно получить среднюю оценку *wi* весовых коэффициентов факторов

где *wij -* вес *i-*го объекта, рассчитанный по оценкам всех экспертов;

где *xij -* оценка фактора *i*, данная экспертом *j*; *n -* число факторов, *m* - число

экспертов.

Преимуществом данного метода является уменьшение зависимости оценки конкретного фактора от остальных. Однако, сопоставление факторов все же требуется – иначе коэффициенты значимости корректно невозможно расставить.

### 2.3.4. Метод анализа иерархий

Этот метод, разработанный Т. Саати [16], позволяет частично избавиться от сложностей, перечисленных выше. Суть метода заключается в сравнении факторов между собой по их влиянию на конечную цель попарно относительно друг друга, не учитывая при этом влияние других факторов. Для того, чтобы осуществить попарное сравнение факторов, автором метода Саати предложена специальная оценочная шкала, состоящая из пяти основных и четырех промежуточных суждений. В ней суждения экспертов представляются следующим образом (табл. 2):

Таблица 2. Иерархия экспертных сравнений соотношения факторов

|  |  |
| --- | --- |
| Суждение | Пояснение |
| 1. Равная важность | Равный вклад факторов в цель |
| 1. … | Промежуточное суждение |
| 3. Умеренное превосходство | Опыт и суждение дают легкое превосходство одного |
| 4. … | Промежуточное суждение |
| 5. Существенное превосходство | Сильное превосходство одного фактора над другим |
| 6. … | Промежуточное суждение |
| 7. Значительное превосходство | Имеется практически значительное превосходство одного фактора над другим |
| 8. … | Промежуточное суждение |
| 9.Очень сильное превосходство | Имеется значительное превосходство одного фактора над другим |

В результате сравнения получается квадратная матрица A = (*aij*) с единичной диагональю (сравнение фактора самого с собой равно единице). Здесь *aij* означает отношение оценок соответствующих элементов; индексы *i* и *j* изменяются от единицы до величины, равной количеству факторов. При последовательном переборе всех возможных пар факторы сравниваются между собой дважды (сначала - фактор *ai* с фактором *aj*, затем - в обратном порядке), поэтому должно выполняться условие «обратной симметричности»:

. (14)

Из условия (14) можно сделать вывод, что необязательно вычислять всю матрицу, а достаточно найти только одну ее часть, лежащую выше или ниже диагонали, что не имеет принципиального значения вследствие элементарного пересчета взаимно обратных значений. Если рассматривается *n* факторов, то всего возможно наличие значащих сочетаний.

Для кодирования, в МАИ используется номер соответствующей строки табл. 1. Каждое из приведенных суждений кодируется числом от 1/9 до 9. Например, если придано существенное превосходство элемента *Ai* над элементом *Аj*, то полагают, что в матрице парных сравнений *aij* = 5 и, соответственно, *aji* = 1/5, поскольку для кодирования используется пятая строка.

Суть обработки матрицы заключается в разложении: *A* ≈ *Z·U*, где . Цель - определение компонент вектора весов Z = (z1, …, z*n*),

что позволяет ранжировать факторы *Ai*.

Вычисление весов можно осуществить несколькими способами. Одним из возможных подходов к аппроксимации вектора весов может служить путь вычисления собственного вектора матрицы парных сравнений, который равен соответствующему максимальному собственному числу.

Положительные стороны метода. Отсутствие необходимости постоянно держать в поле зрения все факторы или, по крайней мере, группу однородных факторов, позволяет эксперту сконцентрировать внимание на конкретной проблеме: на сколько фактор *Аi* превосходит фактор *Bj* или уступает ему. Вследствие этого следует ожидать более точных результатов.

В практике нередко возникают ситуации, когда число влияющих факторов изменяется. В МАИ это приводит только к необходимости сравнения вновь возникших пар или же к вычеркиванию строк и столбцов матрицы парных сравнений, соответствующих изъятых из рассмотрения факторов, т. е. к образованию минора матрицы. Полученные результаты предыдущих опросов сохраняются, и полного обновления анкеты, как это происходит в других случаях, не требуется. С учетом того, что процедура МАИ, в сущности, сводится к поиску собственного вектора соответствующей матрицы, принадлежащего максимальному собственному значению, с «технической» точки зрения включение дополнительных факторов есть увеличение размерности соответствующего линейного пространства за счет добавления прямых слагаемых.

Обычные числовые шкалы не всегда удобны для сопоставления факторов, выражаемых в различных размерностях и понятиях. Особенно сложно сравнивать факторы, показателями которых, с одной стороны, являются количественные величины, а с другой – качественные. Вербально-числовые шкалы, одним из вариантов которых является шкала Саати, как раз и призваны оценивать такие несоответствия показателей влияющих факторов.

**Недостатки метода.** Возникает ряд вопросов при интерпретации результатов, и связаны они, прежде всего, с критерием качества работы эксперта — с отношением согласованности***.*** Отношение транзитивности хорошо работает, когда все характеристики исследуемой системы можно представить числовыми величинами. Но как только это становится невозможным, требование наличия транзитивности зачастую вступает в противоречие с логикой исследователя.

### 2.3.5. Пример расчета весовых коэффициентов по МАИ

Рассмотрим процесс расчета весовых коэффициентов на основе метода анализа иерархий. Эксперт попарно сравнивает влияние метрик на итоговую оценку. На основе опроса эксперта строим матрицу парных сравнений. Результаты представлены в таблице 3.

Таблица 3. Матрица парных сравнений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Метрика Холстеда | Метрика Мак-Кейба | Метрика Джилба | Суммарная сложн. условий | Суммарная вложенность |
|  | 1 | 2 | 3 | 4 | 6 |
| 1 | 1 | 1/5 | 1/3 | 1/2 | 1/2 |
| 2 | 5 | 1 | 2 | 3 | 3 |
| 3 | 3 | 1/2 | 1 | 2 | 2 |
| 4 | 2 | 1/3 | 1/2 | 1 | 1 |
| 5 | 2 | 1/3 | 1/2 | 1 | 1 |
| 6 | 2 | 1/3 | 1/2 | 1 | 1 |

Необходимо найти максимальное собственное число матрицы и соответствующий ему собственный вектор.

Решение несимметричной задачи собственных значений осуществляется в несколько этапов. На первом этапе матрица приводится ортогональным преобразованием к верхней форме Хессенберга. На втором этапе, занимающем больше всего времени, матрица приводится ортогональным преобразованием к верхней форме Шура. Если требуются только собственные значения, то этого достаточно, т.к. собственные числа матрицы располагаются в диагональных блоках квазитреугольной матрицы из канонической формы Шура. Если же требуются собственные векторы, то они могут быть получены по векторам Шура и квазитреугольной матрице путем обратной подстановки (фактически - решения системы линейных уравнений; сам процесс обратной подстановки занимает незначительную часть времени работы алгоритма, но необходимость накапливать проводимые над матрицей преобразования для применения их к собственным векторам замедляет алгоритм более чем в два раза).

Разложение Шура, на которое тратится больше всего времени, осуществляется с использованием QR-алгоритма. Более подробно нахождение в монографии [6].

Для нахождения собственных значений можно воспользоваться готовыми решениями для их расчета, например: библиотека lapack, пакет MathCad.

В итоге получаем (15):

x = (15)

## 2.4. Пример расчета сложности программы

Для примера произведена оценка сложности следующей задачи.

**Условие.**

Дан массив. Нужно найти количество элементов, которые меньше среднего значения элементов массива.

<Input>

Со стандартного устройства ввода вводится целое число N – количество элементов массива, в следующей строке – N целых чисел – элементы массива.

<Output>

Нужно выдать на стандартное устройство вывода целое число – количество искомых элементов.

<Sample Input>

5

2 4 6 10 8

<Sample Output>

2

**Авторское решение.**

#include <stdio.h>

int main()

{

int n[1000],i,s,r;

scanf("%d",&n[0]);

s=0;

for(i=1;i<=n[0];i++)

{

scanf("%d",&n[i]);

s=s+n[i];

}

if(s%n[0]==0)

s=s/n[0];

else

s=s/n[0]+1;

r=0;

i=1;

while(i-1!=n[0])

{

if(n[i]<s)

r++;

i++;

}

printf("%d",r);

return 0;

}

Для начала рассчитаем метрики Холстеда:

* Количество уникальных операндов NUOprnd = 11
* Количество уникальных операторов NUOprtr = 20
* Общее число операндов NOprnd = 64
* Общее число операторов NOprtr = 57

Размерная сложность по Холстеду равна (16):

HDiff = (NUOprtr/2) × (NOprnd / NUOprnd) = 50. (16)

Далее построим управляющий граф программы, который представлен на рисунке 4.



Рисунок 4. Управляющий граф программы

По управляющему графу программы рассчитываем метрику Мак – Кейба. Количество вершин графа v = 11, количество ребер e = 14. Цикломатическая сложность по Мак – Кейбу (17):

C = e – v + 2 = 14 -11+2 = 5. (17)

Далее рассчитаем оставшиеся метрики, значение всех метрик представлено в таблице 4

Таблица 4. Значения метрик для тестового примера

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метрика Холстеда | Метрика Мак-Кейба | Метрика Джилба | Суммарная сложн.условий | Сумм. уровень вложенности |
| 50 | 5 | 5 | 3 | 3 |

Используя формулу (1) и рассчитанные коэффициенты (15) получаем:

H\_M = 50\*1.1 + 5\*6.1 + 5\*3.7 + 3\*2 + 0\*2 + 3\*2 = 116 (17)

## 2.5. Выводы

Определены понятия сложности программы и комплексной меры. Проведен анализ методов расчета комплексной меры. Произведен анализ и исследование метрик программного обеспечения их достоинства и недостатки, представлены примеры расчета метрик. Разработаны дополнительные метрики программы и методы их расчета для более полного анализа сложности программы. Исследованы методы определения весовых коэффициентов на основе экспертных мнений и обосновано применение метода МАИ для расчета весовых коэффициентов, приведен пример расчета весовых коэффициентов для 6 метрик на основе метода МАИ. Приведены результаты тестирования методики оценки сложности на контрольном примере.

# 3. ПРИМЕНЕНИЕ МЕТОДИКИ. АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Целью данной главы представить функциональную схему инструмента оценки сложности, основные его составляющие блоки. Описать АОС, на базе которой была произведена апробация разработанной методики, исследовать процесс расчета сложности тестовой выборки задач , описать проблемы возникающие при оценке большого количества решений по одной задаче, и методы их решения. Для адекватной оценки полученные результаты сравнить с экспертными оценками.

## 3.1. Описание разработанного инструмента

Для проверки методики был разработан инструмент оценки сложности программ на основе исходного кода, написанного на языке C++ [14]. Язык программирования C++ был выбран не случайно. Во первых – это на данный момент самый популярный ЯП для решения задач по программированию. Во вторых – большинство решений задач, хранящихся в базе системы BACS, написаны на C++.

Функциональная схема инструмента оценки сложности задачи на основе исходного кода программы представлена на рисунке 5.



Рисунок 5. Функциональная схема инструмента оценки сложности

Блок «Лексический анализатор» - в нем происходит анализ исходного текста программы и преобразование его в список лексем.

Блок «Таблица символов» - справочник лексем конкретного языка программирования.

Блок «Синтаксический анализатор» - в нем анализируются логические конструкции. Также в этом блоке производится семантический анализ, а именно вычисляются те значения метрик, которые требуют изменяемые параметры, или требуют сложного синтаксического анализа.

В расчетном блоке, производится вычисление всех метрик, а затем расчет итоговой оценки сложности программы.

Положительные стороны схемы можно выделить следующие:

* есть возможность анализировать особенности языка;
* извлечение метрик и их расчет разделены, что дает гибкость при разработке и использовании.

К недостаткам данной схемы можно отнести:

* написание анализатора для каждого языка, приводит к тому, что нельзя просто добавить дополнительный язык программирования. С другой стороны, сейчас активно используемых языков не так уж и много. Меняются они не так уж и часто. Таким образом эту работу надо будет сделать только один раз;

Инструмент оценки реализован на языке C++ с использованием библиотек VivaCore [12] и Boost. Часть исходного кода представлены в приложении A.

VivaCore - это открытая библиотека для работы с Си и Си++ кодом. Библиотека предназначена для реализации на ее основе систем рефакторинга кода, систем статического и динамического анализа, систем трансформации или оптимизации кода, расширений языка, подсистем подсветки синтаксиса, систем построения документации по коду и других аналогичных инструментов.

Boost — собрание библиотек, расширяющих [C++](http://ru.wikipedia.org/wiki/C%2B%2B). Cвободно распространяются по лицензии [Boost Software License](http://www.boost.org/more/license_info.html) вместе с исходным кодом. Проект был создан после принятия стандарта [C++](http://ru.wikipedia.org/wiki/C%2B%2B), когда многие были недовольны невключенностью некоторых библиотек в стандарт. Проект является своего рода «испытательным полигоном» для различных расширений языка и [часть библиотек](http://www.open-std.org/jtc1/sc22/wg21/docs/library_technical_report.html) являются кандидатами на включение в следующий стандарт [C++](http://ru.wikipedia.org/wiki/C%2B%2B).

Исходные коды основных блоков инструмента оценки представлены в приложении 1.

## 3.2. Описание системы BACS

В качестве рабочего материала выбрана автоматизированная система для проведения олимпиад по программирования BACS, разработанная и используемая на кафедре ПО ИжГТУ с 2006 г.

Схема системы представлена системы BACS представлена на рисунке 6.



Рисунок 6. Функциональная схема системы BACS

Рассмотрим подсистему проверки. Главный блок - это сервер проверки, в котором происходит компиляция посланных решений и проверка на тестовых наборах. Второй блок – архив задач. Архив задач – это база данных, в которой хранятся условия задачи, тестовые наборы, чекеры, и самое главное авторские решение, которые будут использоваться при оценке сложности задачи. В третьем блоке «база данных», хранятся данные о пользователях, о контестах, а также все посылки, т.е. посланные участниками решения задач, которые также будут проанализированы.

## 3.2. Анализ полученных результатов

В качестве тестового примера работы разработанного инструмента оценки сбалансированности учебных курсов по программированию был выбран курс School Foundation 17/18, проходящий в системе BACS. Курс содержит 18 контестов, каждый контест охватывает определенную тему в программировании.

Для получения оценки сбалансированности была измерена сложность первых задач каждого из 18 контестов.

**3.2.1. Тестовый набор задач**

**Задача 1. Трамвай или троллейбус**

Вася собирается поехать в школу, и у него есть две опции это сделать: использовать трамвай или использовать троллейбус. На троллейбус у Васи есть проездной, а на трамвай — нет. Исходя из этого, он решил действовать по следующему алгоритму:

• Если Вася успевает в школу и на трамвае, и на троллейбусе, то он выбирает троллейбус.

• Если Вася успевает в школу только на одном виде транспорта из двух, то он едет на нём.

• Если Вася не успевает в школу ни на одном виде транспорта, то он тоже выбирает троллейбус — в школу он уже опоздал, а тут хотя бы не надо деньги платить.

Чтобы дойти до трамвайной остановки около своего дома, Васе нужно потратить a1 минут. На трамвае ему нужно ехать a2 минут, а потом a3 минут нужно потратить на путь с остановки до школы.

Аналогично, Вася тратит b1 минут на путь до своей троллейбусной остановки, b2 минут на путь на троллейбусе и b3 минут на путь от троллейбусной остановки около школы до школы.

У Васи осталось T минут до начала уроков. Вася успеет в школу, если его путь суммарно займет не более T минут. Определите, успеет ли Вася в школу, а также какой вид транспорта выберет Вася. Если Вася успевает в школу, определите, сколько у него еще есть времени в запасе, то есть через какое максимальное время он может выйти из дома, чтобы все еще успеть в школу на выбранном им виде транспорта.

**Input**

В первой строке заданы три целых числа a1, a2, a3 (1 ≤ a1, a2, a3 ≤ 100). В второй строке заданы три целых числа b1, b2, b3 (1 ≤ b1, b2, b3 ≤ 100). В третьей строке задано целое число T (1 ≤ T ≤ 300).

**Output**

В первой строке выведите «YES» (без кавычек), если Вася успеет в школу, и «NO» в противном случае.

Во второй строке выведите «Tram» (без кавычек), если Вася поедет на трамвае, или «Trolleybus», если Вася поедет на троллейбусе.

Если в первой строке выведено «YES», то в третьей строке выведите максимальное количество минут, по прошествии которых Вася может выйти из дома и все еще успеть в школу на выбранном виде транспорта (указанном во второй строке).

**Задача 2. Характеристика генома**

Группа британских ученых занимается изучением геномов различных существ. Будем считать, что геном существа является строкой длины n из символов A, C, G, T. Ученые пришли к выводу, что по количеству вхождений каждого символа в геном можно определять некоторые его свойства. Дан геном s. Для каждого символа, встречающегося в нем, выведите количество его вхождений в s. Символы нужно выводить в алфавитном порядке.

**Input**

В первой строке дано целое число n (4 ≤ n ≤ 105 ) — длина генома. Во второй строке дана строка s длины n, состоящая из символов A, C, G, T.

**Output**

Для каждого символа c, встречающегося в s, выведите строку c k, где k — количество этих вхождений. Символы c должны идти в алфавитном порядке. Обратите внимание, что выводить нужно только те символы, для которых k > 0.

**Задача 3. Двоичная запись**

Дано число N. Выведите его представление в двоичном виде в обратном порядке. Например для 6 ответ 011, так как 6 = 0 × 20 + 1 × 21 + 1 × 22 .

**Input**

Со стандартного устройства ввода вводится одно число N (1 ≤ N ≤ 109 ).

**Output**

Нужно выдать на стандартное устройство вывода требуемое представление числа N.

**Задача 4. Вставка после первого вхождения**

Задана строка слов D; слово – это последовательность символов, не содержащая пробела. Слова разделяются ровно одним пробелом, последним словом является ″\*″. Найти первое совпадение слова W в строке слов D, вставить после него слово T и вывести новую строку. Гарантируется, что в строке D есть хотя бы одно вхождение слова W.

**Input**

Со стандартного устройства ввода в первой строке вводится набор слов D, разделенных ровно одним пробелом, последним словом является ″\*″. Во второй строке вводится слово W, в третьей строке вводится слово T. Количество символов в каждой строке не превышает 1000. Все символы имеют коды от 32 до 254.

**Output**

Нужно выдать на стандартное устройство вывода полученную строку.

**Задача 5. Лишние пробелы**

Нужно каждую группу пробелов в строке заменить на один пробел.

**Input**

Со стандартного устройства ввода вводится строка, содержащая местами более одного пробела подряд. Длина строки не превышает 1000 символов.

**Output**

Нужно выдать на стандартное устройство вывода ту же строку, но без лишних пробелов.

**Задача 6. Длинное слово - первое**

Нужно определить самое длинное слово в строке и поменять его местами с первым. Если слов с максимальной длиной несколько, менять следует первое из них.

**Input**

Со стандартного устройства ввода вводится строка, соседние слова в которой разделены одним пробелом. Длина текста не превышает 1000 символов.

**Output**

Нужно выдать на стандартное устройство измененную строку.

**Задача 7. Air Trip**

У Джона и Брюса есть самолет. Они собираются совершить несколько последовательных перелетов. Для i-го перелета необходимо f[i] литров топлива. Сначала самолет заправлен T литрами топлива. Для совершения очередного (i + 1)-го перелета в баке должно быть как минимум f[i + 1] литров топлива. Необходимо посчитать, сколько перелетов можно выполнить.

**Input**

В первой строке находится число N – количество перелетов (1 ≤ N ≤ 50). В следующей строке находится N целых чисел в пределах от 1 до 1000 – количество топлива, необходимое для перелета. В последней строке находится число T – начальное количество топлива в баке самолета.

**Output**

Единственная строка выходного файла должна содержать ответ на задачу.

**Задача 8. Количество положительных и отрицательных чисел**

Дан массив *A* из *N* целых чисел. Найти количество положительных, отрицательных и равных нулю чисел в этом массиве. Вывести исходный массив.

**Input**

Со стандартного устройства ввода в первой строке вводится число *N* (1 *<= N* <= *1000*)—количество элементов массива. Во второй строке через пробел вводятся *N* чисел—элементы массива *A* (-1000 <= *Ai* <= 1000).

**Output**

В первой строке выведите количество положительных элементов массива *A*,во второй строке количество отрицательных элементов массива *A*, в третьей количество элементов массива *A,* равных 0. В четвертой строке выведите исходный массив *A.*

**Задача 9. Часть строки**

Дана строка S и 2 слова из нее. Нужно выдать часть строки, находящуюся между этими словами.

**Input**

Со стандартного устройства ввода вводится строка, соседние слова в которой разделены одним пробелом. В конце строки стоит перевод строки. Во второй и третьей строках идут слова. Гарантируется, что первое слово встретится в строке раньше второго. Длина текста не превышает 1000 символов.

**Output**

Нужно выдать на стандартное устройство вывода требуемую часть строки.

**Задача 10. Последовательность**

Дана последовательность из N целых чисел (2 ≤ N ≤ 15000). Найти количество различных чисел среди элементов этой последовательности. Числа по модулю не превосходят 2 × 109 .

**Input**

В первой строке входных данных записано N — количество чисел. В последующих строках записаны числа последовательности, разделенные пробелом или символом перевода строки.

**Output**

Выведите одно искомое число. Пробелы до и после числа выводить не нужно.

**Задача 11. Двоичный поиск**

Вам даны два массиа чисел. Требуется для каждого элемента второго массива понять, содержится ли он в первом массиве.

**Input**

В первой строке входного файла содержатся числа N и K (0 < N, K < 10001). Во второй строке находится N чисел первого массива, а в третей строке K чисел второго массива, каждое из которых по модулю не превосходящие 2 × 109 .

**Output**

В выходной файл для каждого из K чисел вывести в отдельную строку «YES», если это число встречается в первом массиве и «NO» в противном случае.

**Задача 12. Очень легкая задача**

Сегодня утром жюри решило добавить в вариант олимпиады еще одну, Очень Легкую Задачу. Ответственный секретарь Оргкомитета напечатал ее условие в одном экземпляре, и теперь ему нужно до начала олимпиады успеть сделать еще N копий. В его распоряжении имеются два ксерокса, один из которых копирует лист за x секунд, а другой – за y. (Разрешается использовать как один ксерокс, так и оба одновременно. Можно копировать не только с оригинала, но и с копии.) Помогите ему выяснить, какое минимальное время для этого потребуется.

**Input**

На вход, через пробел, подаются три натуральных числа N, x и y (1 ≤ N ≤ 2 · 108 , 1 ≤ x, y ≤ 10).

**Output**

Выведите одно число - минимальное время в секундах, необходимое для получения N копий.

**Задача 13. Максимум из минимумов**

Задана матрица целых чисел размера M\*N (1≤ M ,N ≤ 100). Все числа в матрице различны и по модулю не превосходят 216-1. В каждой строке находится минимальный элемент, затем среди этих чисел выбирается максимальное. Составьте программу, находящую номер строки матрицы X, в которой расположено выбранное число.

**Input**

В первой строке содержится, через пробел, два числа M и N. В следующих M строках через пробел по N чисел записан исходный массив.

**Output**

Выведите одно число - номер строки в которой содержится максимум из минимумов.

**Задача 14. Максимум из минимумов**

На листке в клетку размером 10 × 10 выбрали какую-то клетку и, начиная от нее, стали закрашивать подряд клетки, двигаясь только вниз и вправо. Затем первую и последнюю закрашенную клетку стерли. Ваша задача — восстановить их. Если вариантов ответа несколько, выведите любой. Ниже слева показан пример предоставленного изображения, а справа — один из возможных ответов для этого примера, который Вы должны найти.

**Input**

Входные данные состоят из 10 строк, содержащих по 10 чисел 0 и 1. Число 0 означает незакрашенную клетку, а 1 — закрашенную.

**Output**

Выведите изображение листка в том же формате, что и во входных данных, только с добавлением первой и последней покрашенных клеток.

**Задача 15. Чётки**

Возьмем веревочку и насадим на нее N костяшек, причем на каждой костяшке нарисуем английскую букву (в любом регистре). Далее свяжем концы веревочки. Получившееся в результате таких махинаций приспособление назовем четками. Даны четки. Они всегда расположены на столе в виде окружности так, что ровно одна костяшка располагается вверху. Назовем ее верхней. Даны три человека: Сережа, Миша и Витя. Сережа, как ярый приверженец деструктивных методов, занимается только тем, что извлекает костяшку, следующую по часовой стрелке сразу за верхней. Аналогично, Миша, как прирожденный конструктор, умеет лишь добавлять костяшку между верхней и следующей за ней по часовой стрелке. Витя выполняет более хитрое действие: он вращает четки против часовой стрелки, в результате чего верхней становится костяшка, следующая за текущей верхней костяшкой по часовой стрелке. Вам необходимо определить, как будут выглядеть четки после некоторой совокупности действий Сережи, Миши и Вити.

**Input**

В первой строке содержится целое число N (1 ≤ N ≤ 200). Во второй – строка состоящая из N символов, обозначающих начальные буквы на костяшках в порядке обхода по часовой стрелке начиная с изначальной верхней костяшки. Далее приведен список действий Сережи, Миши и Вити. Каждое действие записываться в отдельной строке. Если это действие Сережи, записывается буква «D», если Миши – буква «I», затем проблем, затем буква, проставленная на вновь добавленной костяшке. Если за дело берется Витя, записывается буква «M». Буква «E» означает окончание всех действий. Общее количество команд в списке не превышает 50000. Гарантируется, что Витя не попытается удалить костяшку из четок, в которой осталась всего одна костяшка.

**Output**

Выведите строку – конечное состояние четок (буквы на всех костяшках, начиная с текущей верхней и далее по часовой стрелке).

**Задача 16. Скобки**

Коля написал длинное арифметическое выражение и хочет проверить, не ошибся ли он. Но он не знает, как это сделать. Через некоторое время он решил упростить себе задачу: для начала проверить, является скобочная последовательность правильной. Помогите Коле.

Имеется последовательность открывающих и закрывающих скобок. Требуется выяснить, является ли она правильной.

Правильными называются последовательности скобок, удовлетворяющие следующим условиям:

• Пустая последовательность — правильная.

• Если A и B – правильные скобочные последовательности, то (A) и AB — правильные скобочные последовательности.

• Последовательности, которые нельзя получить применением конечного числа указанных выше правил, не являются правильными.

**Input**

Первая строка содержит последовательность скобок. Длина строки не превышает 105 символов.

**Output**

Если последовательность является правильной, выведите «CORRECT». Иначе, выведите «INCORRECT».

**Задача 17. Очередь**

Реализуйте работу очереди. Для каждой операции изъятия элемента выведите её результат. На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда – это либо "+ N", либо "-". Команда "+ N"означает добавление в очередь числа N. Команда "-" означает изъятие элемента из очереди.

**Input**

В первой строке содержится количество команд – M(1 ≤ M ≤ 106 ). В следующих M строках содержатся команды, по одной в каждой строке.

**Output**

Для каждой команды изъятия выведите число, которое удаляется из очереди, по одному в каждой строке. Гарантируется, что изъятий из пустой очереди не производится.

**Задача 18. Движение с ускорением**

В начальный момент времени автомобиль двигался со скоростью V0. После этого он t1 секунд двигался с ускорением a1, а затем – t2 секунд с ускорением a2. Необходимо найти максимальную скорость, с которой двигался автомобиль. Если автомобиль перед началом движения с ускорением a имел скорость V , то через t секунд он будет иметь скорость V +a·t. Положительное ускорение означает, что скорость увеличивается, отрицательное – уменьшается.

**Input**

На вход дается пять целых чисел V0, t1, a1, t2, a2 (0 ≤ V0 ≤ 1000, 0 ≤ t1, t2 ≤ 10, |a1| ≤ 10, |a2| ≤ 10). Гарантируется, что ни в какой момент времени скорость автомобиля не становится отрицательной.

**Output**

Для каждой команды изъятия выведите число, которое удаляется из очереди, по одному в каждой строке. Гарантируется, что изъятий из пустой очереди не производится.

### 3.2.2. Оценка авторских решений

Для каждой задачи в базе данных хранится авторское решение. Оно служит прежде всего для проверки набора тестов, а так же как правило представляет наиболее лучший алгоритм решения поставленной задачи. Поэтому на первом этапе рассчитываются авторские решения по каждой задаче. Результаты расчета сложности авторских решений представлены на диаграмме, показанной на рисунке 7.

Рисунок 7. Результаты оценки сложности по авторским решениям

Значения рассчитанных метрик по каждой задаче приведены в таблице 5.

Таблица 5. Значения метрик на авторских решениях

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Метрика Холстеда | Метрика Мак-Кейба | Метрика Джилба | Сум. ур. влож. | Суммарная сложность усл. опер. |
| Задача 1 | 21,3 | 6 | 4 | 2 | 4 |
| Задача 2 | 19,9 | 10 | 6 | 3 | 7 |
| Задача 3 | 9,3 | 2 | 1 | 2 | 1 |
| Задача 4 | 10,9 | 4 | 1 | 4 | 1 |
| Задача 5 | 19,3 | 4 | 4 | 3 | 2 |
| Задача 6 | 32 | 7 | 6 | 3 | 0 |
| Задача 7 | 30 | 4 | 3 | 3 | 1 |
| Задача 8 | 21,9 | 6 | 6 | 3 | 2 |
| Задача 9 | 8,8 | 1 | 0 | 1 | 0 |
| Задача 10 | 23,4 | 4 | 2 | 3 | 0 |
| Задача 11 | 29,3 | 5 | 4 | 3 | 1 |
| Задача 12 | 35,2 | 5 | 4 | 3 | 3 |
| Задача 13 | 33,2 | 3 | 2 | 3 | 0 |
| Задача 14 | 64,1 | 16 | 12 | 4 | 6 |
| Задача 15 | 15 | 8 | 8 | 4 | 4 |
| Задача 16 | 12,5 | 9 | 8 | 4 | 6 |
| Задача 17 | 27,2 | 5 | 4 | 3 | 2 |
| Задача 18 | 44 | 8 | 5 | 4 | 2 |

График сбалансированности курса можно показать на диаграмме, изображенной на рисунке 8.

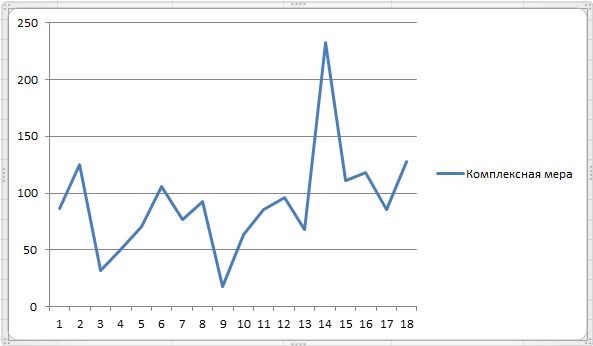


Рисунок 8. График сбалансированности учебного курса School Foundation 17/18

Как видно из графика, сложность задач учебного курса School Foundation 17/18 хоть и возрастает, но очень прерывисто. Пиковые значения графика показывают, что сложность практических задач 3 и 9 занятий следует увеличить, а 14 – уменьшить.

## 3.3. Выводы

Представлена функциональная схема работы, разработанного инструмента оценки сложности задач по программированию. Описаны достоинства и недостатки данной схемы. Для наглядности представлена схема системы BACS,

Предложена общая схема оценки задач по программированию. Описаны возможные проблемы в процессе оценки и методы из решения.

Проанализирован на сбалансированность учебный курс School Foundation 17/18. Сделаны выводы по его улучшению.

# ЗАКЛЮЧЕНИЕ

1. Исследованы параметры, определяющие сложности программы.
2. Произведен анализ и выбор эффективных метрик программного обеспечения, среди них метрика Холстеда, Маккейба, Джилба. Разработаны дополнительные метрики для полноты оценки программы такие как: суммарный уровень вложенности, суммарная сложность составных условий.
3. Разработан метод расчета комплексной меры сложности программы на базе измеримых характеристик программы - метрик. В основе метода лежит формула комплексной меры Кокола. Исследованы методы расчета весовых коэффициентов на основе экспертных мнений.
4. Разработан программный инструмент для оценки решения задач по программированию, написанных на языке C++. Для разработки программного инструмента использованы библиотеки VivaCore и Boost.
5. Проведено тестирование разработанного инструмента на контрольной выборке задач. Произведен анализ полученных результатов.

# СПИСОК ЛИТЕРАТУРЫ

1. Dirk Giesen Philosophy and practical implementation of static analyzer tools . -Electronic data. -Dirk Giesen, cop. 1998.
2. Joel Jones Abstract syntax tree implementation idioms . -Proceedings of the 10th Conference on Pattern Languages of Programs 2003, cop 2003.
3. T.J. McCabe, "A complexity measure," IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, December, 1976.
4. Агальцов, В. П. Контроль знаний - доминирующая составляющая образовательного процесса // Информатика и образование вып. 2 : Образование и Информатика, 1986. - С.94-96 .
5. Войт Н.Н. Разработка методов и средств адаптивного процессом обучения в автоматизированном проектировании: Ульяновск, 2009.
6. Голуб Дж., Ван Лоун Ч. Матричные вычисления. М.: Мир, 1999. 548 с.
7. Изосимов А.В., Рыжко А.Л. Метрическая оценка качества программ. - М.: МАИ, 1989.
8. Интернет-источники: http://topcoder.com, http://acm.uva.es, http://ejudge.ru, http://cs.istu.ru
9. Майерс Г. Искусство тестирования программ. Пер. с англ. - М.: Финансы и статистика, 1982. – 176 с.
10. Майерс Г. Надежность программного обеспечения. Пер. с англ. - М.: Мир, 1980.
11. [Новичков А. Метрики кода и их практическая реализация. http://www.ibm.com/developerworks/ru/edu/0108novich/index.html](http://www.ibm.com/developerworks/ru/edu/0108novich/index.html)
12. [Рыжков Е., Карпов А. Сущность библиотеки анализа кода VivaCore: http://www.viva64.com/content/articles/code-analyzers/?f=VivaCore\_Librar y\_rus.html&lang=ru&content=code-analyzers](file:///C:\Users\ampirator\AppData\Roaming\Microsoft\Word\Рыжков%20Е.,%20Карпов%20А.%20Сущность%20библиотеки%20анализа%20кода%20VivaCore:%20http:\www.viva64.com\content\articles\code-analyzers\%3ff=VivaCore_Librar%20y_rus.html&lang=ru&content=code-analyzers)
13. Статья «Оценка программ». http://www.structur.h1.ru/ocenka.htm
14. Страуструп Б. Язык программирования С++. Специальное издание. – Бином, Невский Диалект, 2006. – 1104 с.
15. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн Алгоритмы построение и анализ. М. Вильямс, 2005.
16. Тутыгин А.Г.,Коробов В.Б.Преимущества и недостатки метода анализа иерархий // Известия Российского государственного педагогического университета имени А.И.Герцена . - СПб., 2010. - N 122. - С.108-115.
17. Холстед М. Начала науки о программах. - М.: Финансы и статистика, 1981.
18. Черноножкин С.К. Меры сложности программ (Обзор) //Системная информатика. Вып. 5: Архитектурные, формальные и программные модели. — Новосибирск: Наука, 1997. — C. 188—227.
19. Шаров Е. Сайт посвященный метрикам. http://met-rix.narod.ru/index.htm

# ПРИЛОЖЕНИЕ

Исходный код блока расчета метрик инструмента оценки сложности задач.

В данном приложении представлена часть исходного кода расчета метрик инструмента оценки сложности задач

Листинг 1 - файл metrixinfo.cpp

#include "metrixinfo.h"

#include <set>

#include "boost/regex.hpp";

metrixinfo::metrixinfo(void)

{

cyclomatic = 1;

}

metrixinfo::~metrixinfo(void)

{

funclist.erase(funclist.begin(), funclist.end());

}

int metrixinfo::is\_keyword(string str)

{

for(int i = 0; i < 6;i++)

{

if(str == keywords[i])

return i;

}

return -1;

}

int metrixinfo::is\_splitword(string str)

{

for(int i = 0; i < 5;i++)

{

if(str == splitwords[i])

return i;

}

return -1;

}

int metrixinfo::is\_otherkeyword(string str)

{

for(int i = 0; i < 6;i++)

{

if(str == otherkeywords[i])

return i;

}

return -1;

}

int metrixinfo::is\_operword(string str)

{

for(int i = 0; i < 31;i++)

{

if(str == operwords[i])

return i;

}

return -1;

}

int metrixinfo::is\_boolword(string str)

{

for(int i = 0; i < 6;i++)

{

if(str == boolwords[i])

return i;

}

return -1;

}

int metrixinfo::is\_typeword(string str)

{

for(int i = 0; i < 10;i++)

{

if(str == typewords[i])

return i;

}

return -1;

}

int metrixinfo::is\_predicates(string str)

{

for(int i = 0; i < 3;i++)

{

if(str == predicates[i])

return i;

}

return -1;

}

int metrixinfo::is\_function(string str)

{

for(int i = 0; i < funclist.size(); i++)

{

if(str == funclist[i].name)

return i;

}

return -1;

}

void metrixinfo::HM()

{

//string [1000] unikope;

//string [] unik = "";

vector <string> unik;

vector <string> operu;

for(int i = 0; i < tokens.size();i++)

{

bool fl = false;

string teklex(tokens[i].ptr,tokens[i].len);

for(int j = 0; j<unik.size(); j++)

{

if(teklex == unik[j])

{

fl = true;

break;

}

}

if(!fl)

{

unik.push\_back(teklex);

}

}

//holstead\_metrix hm;

hm.NUOprtr = unik.size();

hm.Noprtr = tokens.size();

hm.Noprnd = 1;

hm.NUOprnd = 1;

for(int i=0; i < funclist.size(); i++)

{

hm.Noprnd = hm.Noprnd + funclist[i].vars.size();

bool fl = false;

for(int k =0; k<funclist[i].vars.size();k++)

{

for(int j = 0; j<operu.size(); j++)

{

if(funclist[i].vars[k].name == operu[j])

{

fl = true;

break;

}

}

if(!fl)

{

operu.push\_back(funclist[i].vars[k].name);

}

}

}

hm.NUOprnd = operu.size()+1;

hm.NUOprtr =hm.NUOprtr - hm.NUOprnd;

hm.Noprtr =hm.Noprtr - hm.Noprnd;

hm.dictpr = hm.NUOprnd + hm.NUOprtr;

hm.lenpr = hm.Noprnd + hm.Noprtr;

hm.vpr = (double)(hm.lenpr\*log((long double)hm.dictpr));///log(2.0));

hm.value = (hm.NUOprtr\*2)\*(hm.Noprnd/hm.NUOprnd);

/\*cout<<"hm.NUOprnd:"<<hm.NUOprnd<<endl;

cout<<"hm.NUOprtr:"<<hm.NUOprtr<<endl;

cout<<"hm.Noprtr:"<<hm.Noprtr<<endl;

cout<<"Dic:"<<hm.dictpr<<endl;

cout<<"Len:"<<hm.lenpr<<endl;

cout<<"V:"<<hm.vpr<<endl;

cout<<"value:"<<hm.value<<endl;

\*/

}

void metrixinfo::M1( )

{

bool fltype;

for(int i = 0; i < funclist.size(); i++)

{

fltype = false;

ma\_variable tv;

int ofsj;

ofsj = funclist[i].first;

string tmps(tokens[ofsj].ptr,tokens[ofsj].len);

while(tmps!="{")

{

ofsj++;

tmps.clear();

tmps.assign(tokens[ofsj].ptr,tokens[ofsj].len);

}

for(int j = ofsj; j<funclist[i].last; j++)

{

string teklex(tokens[j].ptr,tokens[j].len);

int index;

//fltype = false;

//if((index = is\_type(teklex)) != -1)

// {

while(is\_typeword(teklex)!=-1 && j<funclist[i].last)

{

fltype = true;

tv.type += teklex;

j++;

teklex.clear();

teklex.assign(tokens[j].ptr,tokens[j].len);

}

if(fltype)

{

if(teklex == "\*" || teklex == "&")

{

j++;

teklex.clear();

teklex.assign(tokens[j].ptr,tokens[j].len);

}

tv.name = teklex;

while(teklex != "," && teklex != ";" && j<funclist[i].last)

{

j++;

teklex.clear();

teklex.assign(tokens[j].ptr,tokens[j].len);

if(teklex == "(" || teklex == "[" )

{

char stack[512];

int stacklen = 1;

//stack[0]=teklex[1];

string back;

if(teklex== "[")

{

back = "]";

stack[0] = '[';

}

if(teklex== "(")

{

back = ")";

stack[0] = '(';

}

int ll;

ll = j;

string tmp;

while(tmp != back ||stacklen != 0 )//

{

ll++;

tmp.clear();

tmp.assign(tokens[ll].ptr,tokens[ll].len);

if(tmp == teklex && stack[stacklen-1]==back[0] || tmp == back && stack[stacklen-1]==teklex [0])

{

stacklen--;

}

else

if(tmp == teklex || tmp == back)

{

stack[stacklen]=tmp[0];

stacklen++;

}

}

if(ll<funclist[i].last)

{

j = ll;

}

}

}

if(teklex == ";")

{

fltype = false;

funclist[i].vars.push\_back(tv);

tv.name.clear();

tv.type.clear();

}

if(teklex == ",")

{

funclist[i].vars.push\_back(tv);

tv.name.clear();

}

}

}

}

return;

}

int metrixinfo::FunctionsAnalyze()

{

int retval = 0;

vector<char> stackdepht;

int depth = 1,maxdepth = 1;

bool flinfunc;

//bool fl1 = true;

for(int i = 0; i < funclist.size(); i++)

{

depth = 1;

maxdepth = 1;

funclist[i].cntoper.num\_if = 0;

funclist[i].cntoper.num\_switch = 0;

funclist[i].cntoper.num\_for = 0;

funclist[i].cntoper.num\_while = 0;

flinfunc = false;

funclist[i].flrekursion = false;

// cout<<funclist[i].name<<" "<<funclist[i].first<<" "<<funclist[i].last<<endl;

for(int j = funclist[i].first; j<funclist[i].last; j++)

{

string teklex(tokens[j].ptr,tokens[j].len);

int index;

if((index = is\_keyword(teklex)) != -1)

{

stackdepht.push\_back('d');

depth++;

if(maxdepth < depth)

maxdepth = depth;

switch(index)

{

case 0: funclist[i].cntoper.num\_if ++; cyclomatic++;break;

case 1: cyclomatic++;break;

case 2: funclist[i].cntoper.num\_switch++; cyclomatic++;break;

case 3: funclist[i].cntoper.num\_for++; break;

case 4: funclist[i].cntoper.num\_while++; break;

}

string ll(tokens[j].ptr,tokens[j].len);

vector < char > s;

int cntpred = 1;

int sl = 0;

if(index != 1 && index != - 1)

{

while(ll != ")" || sl != 1)

{

if(ll == "(")

{

s.push\_back('(');

sl++;

}

if(ll == ")")

{

s.pop\_back();

sl--;

}

if(ll == "&&" || ll == "||")

{

cntpred ++;

}

ll.clear();

j++;

ll.assign(tokens[j].ptr,tokens[j].len);

}

funclist[i].predval.push\_back(cntpred);

}

}

/\*if(stackdepht.size() > 1)

if(teklex == "{" && stackdepht.back() =='}' || teklex == "}" && stackdepht.back() =='{')

{

fl1 = false;

stackdepht.pop\_back();

}

\*/

if(teklex == ";" && stackdepht.size()>1)

{

if(stackdepht[stackdepht.size()-1] == 'd' && stackdepht[stackdepht.size() - 2]=='d')

//while(stackdepht.back() == 'd')

{

stackdepht.pop\_back();

depth --;

}

}

if(teklex == "{")

{

if(!flinfunc)

flinfunc = true;

/\*if(stackdepht.size() > 1)

{

if( stackdepht.back() == 'd')

{

}

}

\*/

stackdepht.push\_back(teklex[0]);

}

if(teklex == "}")

{

if(stackdepht.size() > 1)

{

if(stackdepht.back() == '{')

{

stackdepht.pop\_back();

}

if(stackdepht.back() == 'd')

{

while(stackdepht.back() == 'd')

{

stackdepht.pop\_back();

depth --;

}

}

}

}

if((retval = is\_function(teklex)) != -1 && flinfunc)

{

funclist[i].call.push\_back(retval);

if(retval==i)

{

funclist[i].flrekursion = true;

funclist[i].cntrecurs++;

}

}

/\*if(fl1 && (teklex == "}" || teklex =="{"))

{

if(stackdepht.size() > 1)

if( stackdepht.back() == 'd')

{

depth++;

if(maxdepth < depth)

maxdepth = depth;

}

stackdepht.push\_back(teklex[0]);

}

if(stackdepht.size() > 1)

if(stackdepht.back() == 'd')

{

stackdepht.pop\_back();

depth--;

}

\*/

}

funclist[i].depth = maxdepth;

}

//for(int i = 0; i < funclist.size(); i++)

//{

//cout<<funclist[i].name<<endl;

//cout<<"depth: "<<funclist[i].depth<<" "<<endl<<"call: ";

// for(int j = 0; j < funclist[i].call.size(); j++)

//cout<<funclist[funclist[i].call[j]].name <<" ";

/\*cout<<endl;

cout<<"cycles:"<<funclist[i].cntoper.num\_for + funclist[i].cntoper.num\_while<<" "<< "if: "<<funclist[i].cntoper.num\_if<<endl;

cout<<"predictes: ";

\*/

// for(int j = 0; j < funclist[i].predval.size();j++)

{

/// cout<<funclist[i].predval[j]<<", ";

}

// cout<<endl;

// if(funclist[i].flrekursion)

// cout<<"rekursion: true"<<endl<<endl;

//else

// cout<<"rekursion: false"<<endl<<endl;

//}

M1();

HM();

// cout<<endl<<endl;

return 0;

}

int metrixinfo::MyAnalyze(Lex::TokenContainer tok)//(VivaCore::Program \*program)

{

//VivaLex Lexer(program);

//VivaParser Parser(Lexer,VivaErrorLog::Instance(),program);

// tokens(Lexer.GetTokenContainer());

tokens = tok;

bool flisfuncion = false;

char figskobki[255];

int figskobkilen = 0;

/\*for(int kk = 0; kk < tokens.size();kk++)

{

string str(tokens[kk].ptr,tokens[kk].len);

cout<<str;

}

\*/

for(int i = 0; i < tokens.size();i++)

{

string teklex(tokens[i].ptr,tokens[i].len);

if(teklex == "{")

{

string tmp(tokens[i-1].ptr,tokens[i-1].len);

if(tmp == ")")

{

int j = i-2;

char stack[255];

int stacklen = 1;

stack[0]=')';

while((tmp != "(" || stacklen != 0) && j != 0)

{

tmp.clear();

tmp.assign(tokens[j].ptr,tokens[j].len);

j--;

if(tmp == "(" && stack[stacklen-1]==')' || tmp == ")" && stack[stacklen-1]=='(')

{

stacklen--;

}

else

if(tmp == "(" || tmp == ")")

{

stack[stacklen]=tmp[0];

stacklen++;

}

}

functinfo finf;

if(j-1>=0)

{

//string keywords = "else if while switch for";

finf.name.assign(tokens[j].ptr,tokens[j].len);

//string::size\_type idx = keywords.find(finf.name);

if(is\_keyword(finf.name)== -1)//idx==keywords.npos)

{

flisfuncion = true;

finf.first=j - 1;

funclist.push\_back(finf);

}

}

}

}

if(teklex == "}" && figskobki[figskobkilen - 1] == '{' || teklex == "{" && figskobki[figskobkilen - 1] == '}')

{

figskobkilen--;

if(figskobkilen == 0 && flisfuncion)

{

flisfuncion = false;

funclist.back().last = i;

}

}

else

{

if((teklex == "}" || teklex == "{") && flisfuncion)

{

figskobki[figskobkilen] = teklex[0];

figskobkilen++;

}

}

}

/\*for(int jj = 0; jj < funclist.size();jj++)

{

cout<<funclist[jj].name<<" "<<funclist[jj].first<<" "<<funclist[jj].last<<endl;

}

\*/

FunctionsAnalyze();

int jilb = 0;

int depthsum =0;

int predvalsum =0;

int recsum = 0;

for(int finx = 0; finx < funclist.size(); finx++)

{

jilb += funclist[finx].cntoper.num\_if;

jilb += funclist[finx].cntoper.num\_switch;

jilb += funclist[finx].cntoper.num\_while;

jilb += funclist[finx].cntoper.num\_for;

depthsum += funclist[finx].depth;

recsum += funclist[finx].cntrecurs;

for(int g = 0; g < funclist[finx].predval.size();g++)

{

predvalsum += funclist[finx].predval[g];

}

}

double tothmval = hm.value;

//cout<<"\_\_\_\_\_\_\_\_Total\_\_\_\_\_\_\_\_"<<endl;

cout<<tothmval<<"\t"<<cyclomatic<<"\t"<<jilb<<"\t"<<depthsum<<"\t"<<predvalsum<<"\t"<<recsum<<"\t"<<2;

//cout<<""<<endl;

cout<<endl;

return 0;

}

int metrixinfo::is\_operand(string str)

{

for ( int i = 0; i < operands.size(); i++)

{

if(str == operands[i])

return i;

}

return -1;

}

void metrixinfo::SuperNewM(Lex::TokenContainer tok)

{

//boost regexp используем для нахождения более сложных конструкцийй

boost::regex exp("(\\d{4}[- ]){3}\\d{4}");

vector <char> stackdepth;

int tekdepth = 0;

int maxdepth = -1;

set <int> opers;

set <string> opernds;

holstead\_metrix hm;

int cyclomaticm = 0;

int sumdepth = 0;

int jilb = 0;

int predvalsum = 0;

int recsum = 0;

tokens = tok;

int inx;

hm.init();

for(int i = 0; i < tokens.size();i++)

{

string teklex(tokens[i].ptr,tokens[i].len);

if(!teklex.empty())

{

if(teklex == "{")

{

if(!stackdepth.empty())

if(stackdepth.back() == 'f')

{

stackdepth.pop\_back();

tekdepth--;

}

stackdepth.push\_back('{');

tekdepth++;

maxdepth = max(maxdepth,tekdepth);

}

else

{

if( (inx = is\_keyword(teklex)) != -1)

{

if(inx < 5)

{

stackdepth.push\_back('f');

tekdepth++;

maxdepth = max(maxdepth,tekdepth);

if(inx == 3)

{

stackdepth.push\_back('d');

stackdepth.push\_back('d');

}

}

}

else

{

if(teklex == "}" )

{

if(!stackdepth.empty())

if(stackdepth.back() == '{')

{

tekdepth--;

stackdepth.pop\_back();

if(stackdepth.empty())

{

sumdepth += maxdepth;

maxdepth = -1;

tekdepth = 0;

}

}

}

else

{

if(teklex == ";")

{

if(!stackdepth.empty())

if(stackdepth.back() == 'd')

{

stackdepth.pop\_back();

}

else

{

while( !stackdepth.empty() && stackdepth.back() == 'f')

{

stackdepth.pop\_back();

tekdepth--;

}

}

}

}

}

}

if( (inx = is\_keyword(teklex))!=-1)

{

//упраляющие структуры

switch(inx)

{

case 0: jilb++; cyclomaticm++; break;

case 1: jilb++;cyclomaticm++; break;

case 2: jilb++;break;

case 3: cyclomaticm++;break;

case 4: cyclomaticm++;break;

case 5: cyclomaticm++;break;

}

opers.insert(inx);

hm.Noprtr++;

}

else

{

if((inx = is\_typeword(teklex))!=-1)

{

//разбор связаный с типом данных

}

else

{

if((inx = is\_operword(teklex))!=-1)

{

opers.insert(inx+100);

hm.Noprtr++;

}

else

{

if((inx = is\_splitword(teklex))!=-1)

{

//opers.insert(inx+200);

//hm.Noprtr++;

}

else

{

if((inx = is\_boolword(teklex))!=-1)

{

opers.insert(inx+400);

hm.Noprtr++;

predvalsum++;

}

else

{

if((inx = is\_predicates(teklex))!=-1)

{

predvalsum++;

opers.insert(inx+500);

hm.Noprtr++;

}

else

{

if((inx = is\_otherkeyword(teklex))!=-1)

{

opers.insert(inx+600);

hm.Noprtr++;

}

else

{

opernds.insert(teklex);

hm.Noprnd++;

}

}

}

}

}

}

}

}

}

hm.NUOprnd = opernds.size() - 1;

hm.NUOprtr = opers.size();

hm.calc();

cout<<hm.value<<"\t"<<cyclomaticm<<"\t"<<jilb<<"\t"<<sumdepth<<"\t"<<predvalsum<<"\t"<<recsum<<"\t"<<2<<endl;

}