Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ ГАГАРИНА Ю.А.
(СГТУ им. Гагарина Ю.А.)

Кафедра “Прикладные информационные технологии”

# ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ ЛАБОРАТОРНАЯ РАБОТА ПО ВЫБОРУ И УСТАНОВКЕ ОПЕРАЦИОННОЙ СИСТЕМЫ

(промежуточный, этап 1)

Программные и аппаратные технологии умного города

Исполнитель НИР,
студент б1-ПИНФ-41 _____ Нефедов Д.В.

Руководитель НИР,
канд. техн. наук, доц. _____ Федукин А.Ю

Саратов 2021

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Компиляторы — это критические важные программные системы. Без компиляторов было бы невозможно представить себе сегодняшний мир, ведь написание любой сколько-нибудь сложной компьютерной программы требует использования языков программирования высокого уровня.

Такие языки избавляют программистов от трудоемкости и сложности написания программного кода на языках ассемблера, которые требуют от программиста детального знания устройства ISA конкретного процессора [1]. Языки программирования высокого уровня позволяют абстрагироваться от деталей конкретной архитектуры и сфокусироваться на решение задачи, которая стоит перед разрабатываемой компьютерной программой.

Кроме того, использование ЯП высокого уровня позволяет допускать гораздо меньше ошибок по сравнению с языками ассемблера. В предотвращении ошибок помогает встроенные в компилятор проверки синтаксиса и семантики. Это особенно актуально для строго типизированных языков программирования.

Зачастую компиляторы генерируют гораздо более эффективный ассемблерный код, чем смог бы написать программист [2]. Это не удивительно, ведь написание эффективного ассемблерного кода требует от программиста глубочайшего понимания архитектуры процессора, под который пишется программа, внимание к мельчайшим деталям, стоимость выполнения каждой инструкций в микрооперациях и многое другое. Хорошо известного, что человеку тяжело работать с большим объемом чисел и просчитывать мельчайшие детали. Для такой работы и были придуманы компьютеры, а значит именно у них получится генерировать более эффективный низкоуровневый код, чем мог бы написать программист.

К сожалению, компиляторы сами по себе являются сложнейшими компьютерными программами, которые состоят из десятков модулей, требуют

знания в самых различных областях информатики [3]. Оптимизирующие компиляторы требуют особых навыков, так как разработка алгоритмов оптимизации находится на рубежах науки и людей, которые могли бы разработать и реализовать определенный алгоритм оптимизации в компиляторе не так уж и много.

Более того, так как компиляторы — это огромные компьютерные программы, состоящие из десятков миллионов строк кода, то столь же мало людей, которые разбираются в кодовой базе, причем в основном каждый разработчик разбирается только в своем модуле, не понимая ничего, либо крайне мало о других модулях компилятора.

Для примера, в компиляторе GCC по состоянию на 5 января 2015 года было чуть более 14,5 миллионов строк кода [4]. Что интересно, код компилятора GCC был настолько сложен, что в нем существовал модуль под названием “reload”, функционал которого до сих пор не до конца понятен самим разработчикам GCC [5].

Чрезвычайная сложность компиляторов представляет большую проблему для студентов: нет простых и понятных примеров, на которые можно было бы взглянуть. Учебники по разработке компиляторов порой слишком обстоятельны и формальны, что усложняет усвоение информации — достаточно вспомнить классическую “Книгу дракона”.

Одним из возможных решений данной проблемы является создание простого и понятного компилятора, в исходном коде и архитектуре которого легко разобраться. Наша АИС направлена на решение этой проблемы.

# 1 Описание предметной области

## 1.1 Цель курсовой работы

Целью курсовой работы являются закрепление теоретических знаний, получение практических навыков и новых знаний в области проектирования информационных систем.

Цель работы достигается в результате участия в решении практических задач и проблем, возникающих при разработке информационной системы компилятора языка C.

## 1.2 Словесное описание предметной области

Информационная система компилятора языка C предназначена для образовательных целей и предназначена для обучения студентов и прочих желающих внутреннему устройству и построению в частности компиляторов, а в общем случае — программ-трансляторов с одного языка программирования в другой.

Система ориентирована в первую очередь на студентов технических направлений. Техническое задание по разработанной системе находится в Приложении А.

# 2 Уточнение описания предметной области

## 2.1 IDEF0

IDEF0 является методологией функционального моделирования. Она используется для создания функциональной модели, которая отображает структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции. Компонентами синтаксиса IDEF0 являются блоки, стрелки, диаграммы и правила [6].

Диаграмма IDEF0 состоит из *блоков* и *стрелок*, направление которых заданы правила. Блоки изображаются в виде прямоугольников и содержат в себе описание той функции, которую они представляют, и номер. Номер блока должен быть расположен в правом нижнем углу.

К каждому блоку должно вести несколько стрелок, такие как:

— *левые стрелки* (направлены в блок) обозначают информацию или продукты, которую функция получает на входе;

— *правые стрелки* (направлены из блока) обозначают информацию или продукты, которую функция дает на выходе;

— *верхние стрелки* (направлены в блок) обозначают документы, которые регламентируют работы системы;

— *нижние левые стрелки* (направлены в блок) обозначают механизмы, которые влияют на работу нашей системы;

— *нижние правые стрелки* (направлены из блока) обозначают обращение к дополнительной информационной системе, которая существует совершенно отдельно от нашей и которая необходима для осуществления процесса или функции. Может как присутствовать на схеме, так и нет.

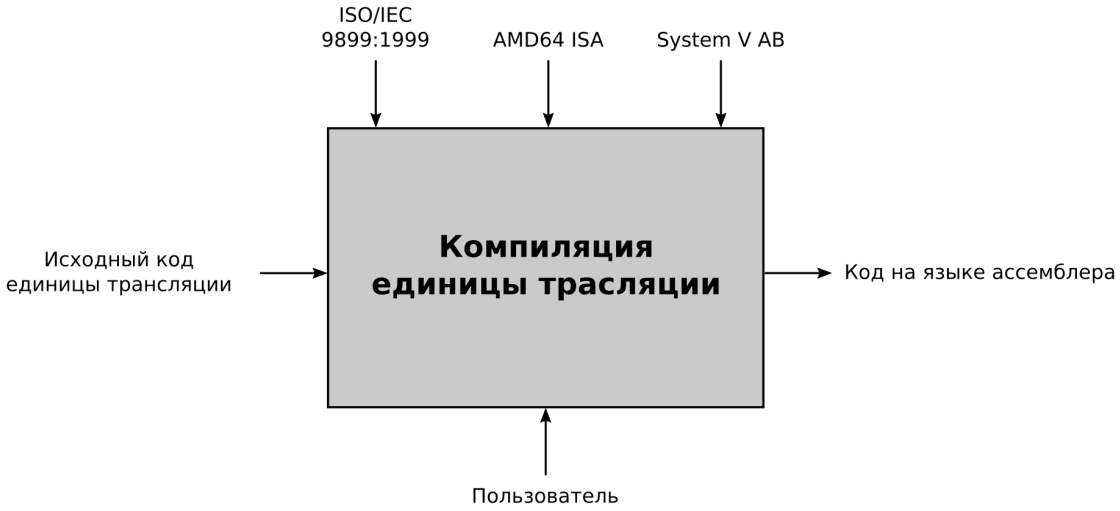


Рисунок 2.1 – Комплексная диаграмма IDEF0

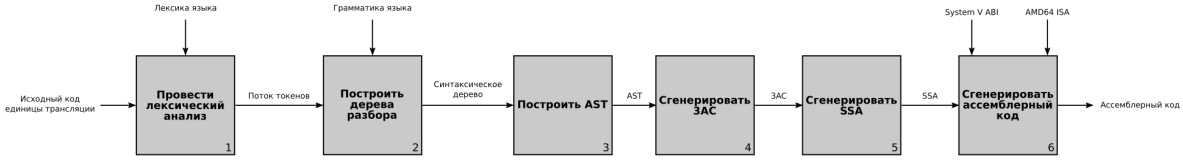


Рисунок 2.2 – Декомпозиция комплексной диаграммы IDEF0

## 2.2 IDEF3

Методология IDEF3 —- это методология описания процессов. В отличие от IDEF0 она рассматривает последовательность выполнения задач, а также причинно-следственные связи между ними [7]. Диаграммы IDEF3 состоят из блоков, описывающих функции, стрелок-связей и перекрёстков, которые показывают, как именно выполняются процессы.

*Блок* (функциональный элемент) представляет собой прямоугольник, разделенный на три другие прямоугольника: один большой наверху и два маленьких друг рядом с другом внизу. В верхнем прямоугольнике содержится

имя функции, в нижнем левом номер его выполнения, в нижнем правом, при необходимости, находится ссылка на другую функцию. Связи бывают простыми, относительными и связями с условием.

Перекрёстки подразделяются на следующие типы: *И*, *ИЛИ*, *синхронное И*, *синхронное ИЛИ*, а также *исключающее ИЛИ*.

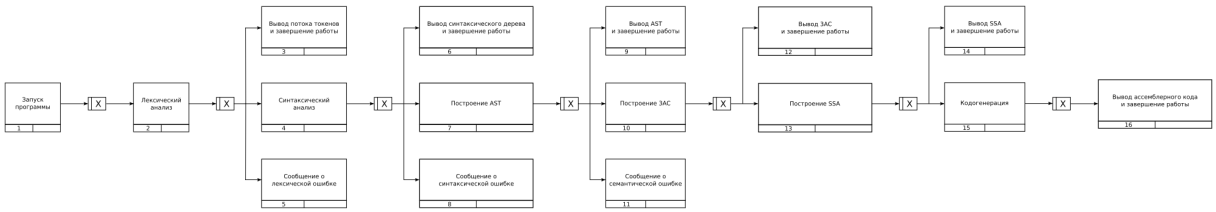


Рисунок 2.3 – IDEF3 диаграмма

## 2.3 DFD

*Диаграммы потоков данных* (англ. *DFD*) —- это способ представления процессов обработки информации. Они показывают, как информация перемещается из одной функции к другой. Подобное представление потока данных отражает движение объектов, их хранение и распространение.

DFD состоит из следующих компонентов: внешняя сущность, процесс, поток данных и хранилище данных [8]. Внешняя сущность представляет собой источник или приёмник информации и изображается прямоугольником с прямыми углами. Процессы в DFD – это функции системы, преобразующие входы и выходы, которые изображаются как прямоугольники со скругленными углами. Потоки данных изображаются стрелками. Если стрелка соединяет какую-либо функцию с хранилищем данных, то на ней должно быть отображено имя, отражающее содержание данного потока. Хранилище данных является прообразом базы данных и изображается как прямоугольник без правой стороны.

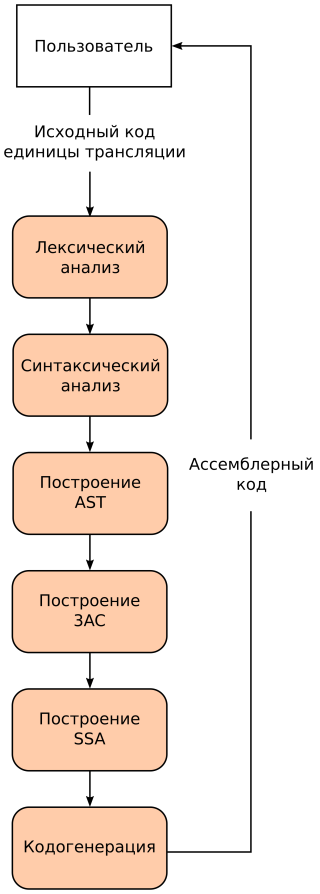


Рисунок 2.4 – DFD диаграмма

## 2.4 UML

*UML* является унифицированным языком моделирования, объекты в котором это упрощенное представление предметов окружающего нас мира [9]. Его используют при разработке ПО для моделирования бизнес-процессов, системного проектирования и отображения организационных структур [10]. UML диаграммы подразделяются на два типа: структурные и поведенческие.

### 2.4.1 Диаграмма пакетов

*Диаграмма пакетов* — это структурная диаграмма UML, которая показывает структуру проектируемой системы на уровне пакетов [11]. Обычно на диаграмме изображаются следующие элементы: пакет, пакетированный элемент, зависимость, импортируемый элемент, импорт пакета, объединение пакета.

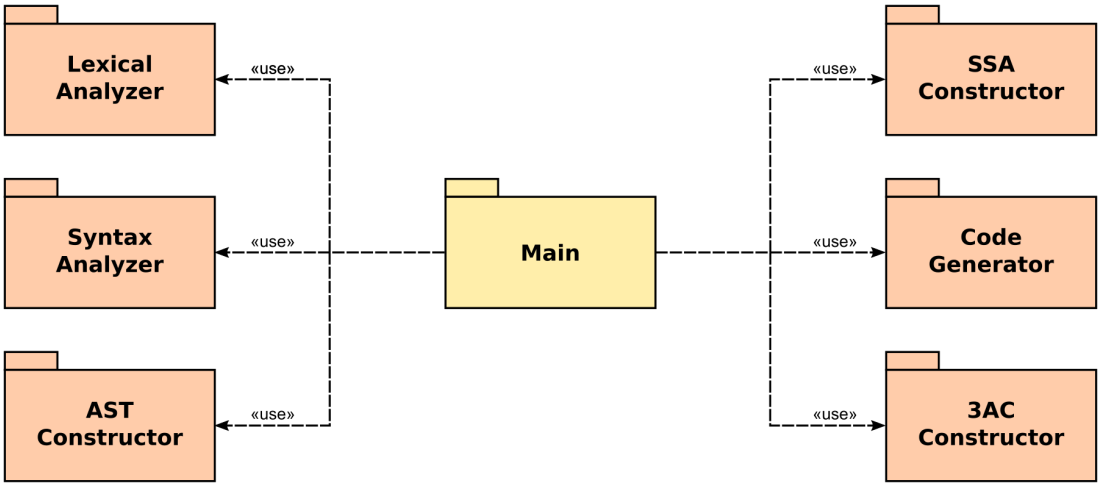


Рисунок 2.5 – Диаграмма пакетов UML

### 2.4.2 Диаграмма компонентов

*Диаграмма компонентов* показывает компоненты, предоставляемые и требуемые интерфейсы, порты и отношения между ними [12]. Этот тип диаграмм используется в компонентно-ориентированной разработке для описания систем с сервисно-ориентированной архитектурой [13].

Компонентно-ориентированная разработка основана на предположения, что ранее сконструированные компоненты могут быть переиспользованы или, при необходимости, заменены на некие “эквивалентные” или “совместимые” компоненты.

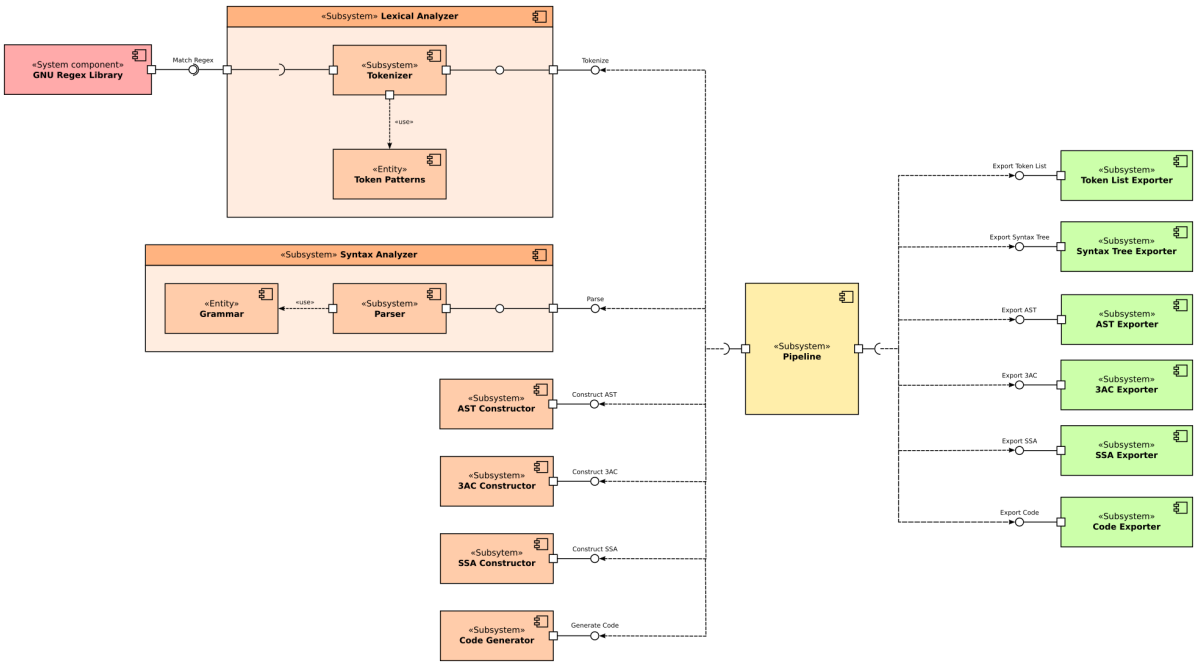


Рисунок 2.6 – Диаграмма компонентов UML

### 2.4.3 Диаграмма потока информации

*Диаграмма потока информации* — это поведенческая диаграмма UML, которая показывает обмен информацией между сущностями системы на высоком уровне абстракции. Потоки информации могут быть полезны для описания циркуляции информации через систему через представление аспектов модели, которые еще не полностью специфицированы или недостаточно детализированы.

Потоки информации не показывают природу информации, механизмы передачи, порядок обмена или какие-либо контрольные условия [14]. Единицы информации могут быть использованы для представления информации, которая протекает через систему вместе с информационными потоками еще прежде, чем их реализация будет проработана.

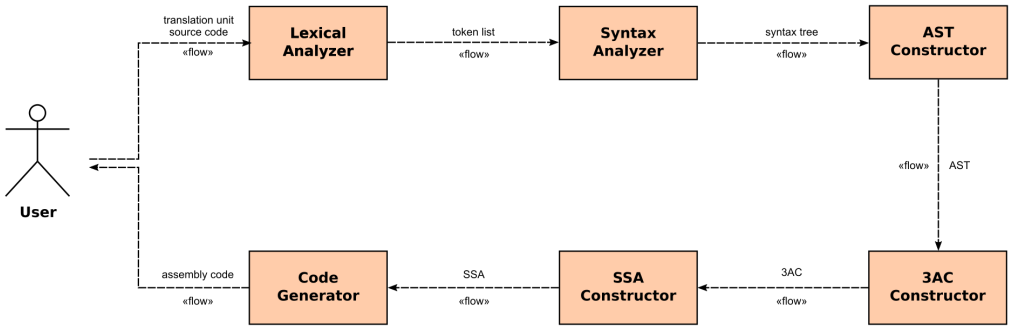


Рисунок 2.7 – Диаграмма потока информации UML

# 3 Выбор средств разработки

Так как для компилятора требуется возможность скомпилировать самого себя, то выбор реализации языка становится очевиден — это язык C стандарта ISO/IEC 9899:1999.

Кроме того, из соображений минимизации зависимостей для более полного понимания всего процесса компиляции единственной зависимостью компилятора будет использование функций GNU C Library, реализующих POSIX-совместимые регулярные выражения, которые не входят в используемый стандарт языка.

В рамках реализации компилятора не ставится задача реализации собственного движка регулярных выражений, так как это довольно простая математическая модель (регулярные языки), разработка которых лишь усложнит код компилятора, который мы стараемся сделать максимально простым.

Для облегчения процесса сборки приложения в качестве средств систем сборки будет использован такой инструмент как GNU Make.

В качестве компилятора может быть использован любой компилятор, который поддерживает стандарт ISO/IEC 9899:1999. Более того, использование разных компиляторов позволит убедится, что исходный код компилятора соответствует используемому стандарту.

# 4 Оценка стоимости работ

Варианты использования описывают основные процедуры работы с системой, поэтому путём их анализа можно определить сложность системы, а, значит, и затраты, необходимые для её разработки. Во многих случаях исследование вариантов использования проще, чем поиск функциональных точек, необходимых для работы других методов оценивания затрат, таких как COCOMO II и Functional Points.

Сложность системы при этом определяется следующими факторами:

— количество и сложность действующих лиц, участвующих в варианте использования;

— дополнительные требования, такие как многозадачность, безопасность и производительность;

— внешние факторы, такие как опыт членов команды разработчиков.

В качестве единицы сложности системы используется прецедентная точка (Use-Case Point, UCP), количество таких точек для каждого варианта использования определяется по формуле:

$$\mathrm{UCP} = \mathrm{TCP} \times \mathrm{ECF} \times \mathrm{UUCP} \times \mathrm{PF} \tag{1}$$

— UCP – количество прецедентных точек;

— TCP – коэффициент технической сложности;

— ECF – коэффициент сложности взаимодействия с окружающей средой;

— PF – эффективность работы разработчиков;

— UUCP – количество нескорректированных прецедентных точек.

Значение UUCP определяется числом шагов, необходимых для выполнения варианта использования, действующими лицами каждого варианта и общим количеством, и сложностью вариантов использования рассматриваемом варианте использования 3 шага. Сложность варианта использования определяется в соответствие со следующими правилами:

— *простой* (5 баллов) – предполагается простой пользовательский интерфейс и работа не более, чем с одной сущностью базы данных; количество шагов не превышает 3, а в реализации задействовано не более 5 классов;

— *средней сложности* (10 баллов) – более сложный пользовательский интерфейс, работа с двумя или более сущностями базы данных; количество шагов составляет от 4 до 7, в реализации задействовано 5-10 классов;

— *сложный* (15) – сложная обработка запросов пользователя, работа с тремя и более сущностями базы данных; количество шагов превышает 7, количество классов в реализации более 10.

Варианты использования:

— *простой*: выполнение лексического анализа единицы трансляции; выполнение синтаксического анализа, вывод диагностических сообщений о лексических ошибках в программе;

— *средней*: построение AST по дереву разбора, вывод диагностических сообщений о синтаксических и лексических ошибках;

— *сложный*: построение трехадресного кода, построение SSA, кодогенерация кода под целевую платформу.

$$\mathrm{UUCW} = 3 \times 5 + 2 \times 10 + 3 \times 15 = 80 \tag{2}$$

Действующие лица также разделяются на три класса по следующим правилам:

— *простое действующее лицо* (1 балл) – сторонняя система, обращающаяся к нашей системе через программный интерфейс;

— *действующее лицо средней сложности* (2 балла) – сторонняя система, обращающаяся к нашей по сети;

— *сложное действующее лицо* (3 балла) – пользователь системы.

После этого подсчитывается количество действующих лиц в каждом классе, количество умножается на соответствующие баллы. В результате получается нескорректированный вес действующих лиц (UAW). В нашем варианте одно сложное действующее лицо — пользователь.

$$\mathrm{UAW} = 1 \times 3 = 3. \tag{3}$$

Значение UUCP вычисляется по формуле:

$$\mathrm{UUCP} = \mathrm{UUCW} + \mathrm{UAW} = 80 + 3 = 83. \tag{4}$$

Для вычисления TCF сначала каждому фактору присваивается вес в зависимости от того, насколько сильно его влияние на систему (от 0 до 5). После этого оценивается сложность, связанная с данным фактором (также от 0 до 5). Полученные значения перемножаются, результаты складываются и получают значение PreTCF (см. таблицу 1).

Таблица 1 – Получение значения PreTCF

| Описание | Вес | Сложность | Вес × сложность |
|---|---|---|---|
| Распределённая система | 0 | 5 | 0 |
| Производительность | 0 | 4 | 0 |
| Эффективность взаимодействия с пользователем | 5 | 3 | 15 |
| Сложные алгоритмы обработки данных | 5 | 5 | 25 |
| Возможность повторного использования | 5 | 5 | 25 |
| Простота установки | 5 | 2 | 10 |
| Простота использования | 5 | 3 | 15 |
| Переносимость | 0 | 5 | 0 |
| Простота модификации | 5 | 5 | 25 |
| Многозадачность | 0 | 5 | 0 |
| Безопасность | 0 | 4 | 0 |
| Открытость для сторонних приложений | 5 | 4 | 20 |
| Необходимость специальных навыков пользователей | 5 | 1 | 5 |
| PreTCF | | | 144 |

Значение TCF вычисляется по формуле:

$$\mathrm{TCF} = 0.6 + 0.01 \times \mathrm{PreTCF} = 0.6 + 1.44 = 2.04. \tag{5}$$

Аналогичным образом вычисляется значение PreECF (см. таблицу 2).

Таблица 2 – Получение значения PreECF

| Описание | Вес | Сложность | Вес × сложность |
|---|---|---|---|
| Знакомство с UML | 1 | 1 | 1 |
| Знакомство с предметной областью | 4 | 5 | 20 |
| Знакомство с объектно-ориентированными технологиями | 3 | 2 | 6 |
| Опыт ведущего аналитика | 2 | 3 | 6 |
| Мотивация | 2 | 3 | 6 |
| Постоянство требований | 1 | 1 | 1 |
| Наличие совместителей | 0 | 3 | 0 |
| Трудный язык программирования | 2 | 3 | 6 |
| PreECF | | | 46 |

Значение ECF вычисляется по формуле:

$$\mathrm{ECF} = 1.4 + (-0.03 \times \mathrm{PreECF}) = 1.4 - 1.38 = 0.02. \tag{6}$$

Значение PF определяется как количество человеко-часов, необходимых для реализации одного варианта использования. Данное значение определяется из опыта выполнения предыдущих проектов. Если такой опыт отсутствует, то берётся число из интервала от 15 до 30, обычно 20. Тогда:

$$\mathrm{UCP} = 83 \times 2.04 \times 0.02 \times 20 \approx 67.60. \tag{7}$$

Получили, что для реализации системы потребуется 67.60 человеко-часов.

# ЗАКЛЮЧЕНИЕ

Таким образом, по итогу проделанной работы сделано следующее:

— Изучена предметная область, проведен её анализ, определены цели и задачи системы;

— Сформулировано словесное описание разрабатываемой информационной системы;

— Изучены графические нотации методологий IDEF0, IDEF3, DFD и UML, их принципы построения, а также построены соответствующие модели, описывающие работу данной системы;

— Произведено оценивание стоимости создания информационной системы и выявлено, сколько потребуется человеко-часов для создания данной системы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. *London K.* Introduction to Computers. — 24 Russell Square London WC1 : Faber, Faber Limited, 1968. — C. 186. — ISBN 0571085938.
2. *Hyde R.* The art of assembly language. — 2-е изд. — San Francisco : No Starch Press, 2010. — ISBN 9781593273019.
3. *Aho A. V.*, *Sethi R.*, *Ullman J. D.* Compilers: Principles, Techniques, and Tools. — Reading, Massachusetts : Addison-Wesley, 1986. — C. 585. — ISBN 0-201-10088-6.
4. *Larabel M.* GCC Soars Past 14.5 Million Lines Of Code & I'm Real Excited For GCC 5 [Электронный ресурс]. — 01.2015. — URL: `https://www.phoronix.com/scan.php?page=news_item%5C&px=MTg3OTQ` (дата обр. 14.12.2021).
5. reload [Электронный ресурс] / J. Wakely [и др.]. — 08.2017. — URL: `https://gcc.gnu.org/wiki/reload` (дата обр. 14.12.2021).
6. *U.S. Department of Defense*. Systems Engineering Fundamentals. — Fort Belvoir, Virginia : Defense Acquisition University Press, 01.2001.
7. *Harzallah M.* Incorporating IDEF3 into the Unified Enterprise Modelling Language // Proceedings of the 2007 Eleventh International IEEE EDOC Conference Workshop. — 2007. — C. 133—140.
8. *Gane C.*, *Sarson T.* Structured Systems Analysis: Tools and Techniques. — New York : Improved Systems Technologies, 1977. — C. 373. — ISBN 978-0930196004.
9. *Hunt J.* The Unified Process for Practitioners: Object-oriented Design, UML and Java. — Springer, 2000. — C. 25. — ISBN 1-85233-275-1.
10. *Fowler M.* UML Distilled: A Brief Guide to the Standard Object Modeling Language. — 3-е изд. — Addison-Wesley, 2004. — ISBN 0-321-19368-7.

11. *Martin R. C.* UML for Java Programmers. — Prentice Hall, 2003. — ISBN 0-13-142848-9.

12. *Буч Г.*, *Рамбо Д.*, *Якобсон И.* Краткая история UML. — 2-е изд. — Москва : ДМК Пресс, 2006. — С. 14. 496 с. — ISBN 5-94074-334-X.

13. *Douglass B.* Real-Time UML. — 3-е изд. — Newnes, 2004. — ISBN 9780321160768.

14. *Penker M.*, *Eriksson H.-E.* Business Modeling with UML. — John Wiley & Sons, 2000. — ISBN 0-471-29551-5.

# ПРИЛОЖЕНИЕ А