МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМЕНИ Н. Э. БАУМАНА

Факультет информатики и систем управления

Кафедра теоретической информатики и компьютерных технологий

УТВЕРЖДАЮ:

Заведующий кафедрой ИУ-9

_____(И.П. Иванов)

«__»_____201__г.

Расчётно-пояснительная записка

к дипломному проекту

«Функциональный язык программирования с динамической типизацией и ML-подобным синтаксисом»

Исполнитель: Ю.А. Волкова

Группа: ИУ9-121

Руководитель

квалификационной работы

А.В. Дубанов

Содержание

Введение

В настоящее время существует множество языков программирования. У всех есть свои сильные стороны и под конкретные задачи выбирается конкретный язык программирования, наиболее подходящий для этих целей. Так, например, для написания кросс-платформенных приложений используются Java или C++.[1] Для браузерных расширений и сайтов – JavaScript.

Однако, для большинства языков, выигрыш от их выбора не так очевиден. Пользователи языка Python считают код на своём языке простым для понимания, потому что чтение программы на Python напоминает чтение текста на английском языке. Это позволяет сосредоточится на решении задачи, а не на самом языке. Lisp является программируемым языком программирования, что позволяет изменять и дополнять его под конкретные задачи. Язык OCaml благодаря системе вывода типов позволяет писать высокоэффективные и безопасные приложения.

Анализ текущего состояния в разработке языков программирования показал, что существует необходимость в языке программирования, ориентированном на быстрой разработке сценариев (скриптов), первоначальном обучении программированию и исследовательском программировании.

Для этих целей, на наш взгляд, должен существовать функциональный язык с «дружелюбным» синтаксисом, который подразумевает инфиксную нотацию в записи арифметических выражений.

1 Языки-прототипы

1.1 Lisp, Scheme

Lisp (LISt Processing language) – функциональный язык программирования с динамической типизацией. Он был создан для символьной обработки данных.[1] *Scheme* – диалект Lisp'a, использующий хвостовую рекурсию и статические области видимости переменных.[2] Scheme – высокоуровевый язык общего назначения, поддерживающие операции над такими структурами данных как строки, списки и векторы. Scheme используется для написания текстовых редакторов, оптимизирующих компиляторов, операционных систем, графических пакетов и экспертных систем, вычислительных приложений и систем виртуальной реальности.

В Scheme реализована «сборка мусора».

Все объекты, в том числе функции, являются данными первого порядка, что позволяет присваивать функции переменным, возвращать функции и принимать их в качестве аргументов. Все переменные и ключевые слова объедены в области видимости, а программы на языке имеют блочную структуру.

Как и во многих других языках программирования процедуры на языке Scheme могут быть рекурсивными. Все хвостовые рекурсии оптимизируются.

Scheme поддерживает определение произвольных структур с помощью *продолжений*. Продолжения сохраняют текущее состояние программы и позволяют продолжить выполнение с этого момента из любой точки программы. Этот механизм удобен для перебора с возвратом, многопоточности и сопрограмм.

Scheme также позволяет создавать синтаксические расширения для написания процедур трансформации новх синтаксических форм в существующие.[3]

В Scheme используется префиксная нотация. Выражения являются списками с оператором во главе этого списка. Такой способ записи позволяет, например, сложить сразу несколько значений. Но выражения с несколькими операторами трудны для восприятия и нагромождены большим количеством скобок.

Функции на языке Scheme могут иметь произвольное число аргументов. Тем переменным, наличие которых необходимо, присваиваются имена, а остальные можно получить из списка оставшихся.

1.2 Prolog

Prolog (PROgramming in LOGic) – язык, объединяющий в себе логическое и алгоритмическое программирование. Он используется в системах обработки естественных языков, исследованиях искуственного интеллекта и экспертных систем.

Ключевыми особенностями языка Prolog являются унификация и перебор с возвратом. Унификация показывает как две произвольные структуры могут быть равными. Процессоры Prolog'а используют стратегию поиска, которая пытается найти решение проблемы перебором с возвратом по всем возможным путям, пока один из них не приведёт к решению[4]

Язык использует сопоставление с образцом (в том числе повторное использование). Программа на языке Prolog представляет собой набор фактов и правил. В прологе отсутствует логическое отрицание, поэтому в условиях нет ветки «иначе».

Входной точкой в программу на языке Prolog является запрос. Ответом на него может быть либо список подошедших шаблонов, либо `false`, означающий, что совпадений не найдено.

1.3 ML, OCaml

ML (Meta Language) – семейство языков с полиморфным выводом типов и обработкой исключений. Языки ML не являются чистыми функциональными языками.

OCaml – самый распространённый в практической работе диалект ML.

Он включает в себя «сборку мусора» для автоматического управления памятью. Как и в Lisp, функции являются объектами первого класса.

OCalm использует статическую проверкую типов, что позволяет увеличить скорость сократить количество ошибок во время исполнения. Но присутствует параметрический полиморфизм, что даёт возможность создавать абстракции и работать с разными типами данных. Механизм автоматического вывода типов позволяет избегать тщательного определения типа каждой переменной, вычисляя его по тому как она используется. Кроме того, в OCaml, как и в Scheme, есть поддержка неизменяемых данных. Также, поддерживаются алгебраические типы данных и сопоставление с образцом, чтобы определять и управлять сложными структурами данных.[6]

1.4 Haskell

Haskell – чистый функциональный язык программирования. Это означает, что результат вычислений, производимых функциями не зависит от состояния программы. Он зависит только от набора входных параметров, а значит уже вычисленные значения не изменятся. А значит при следующем вызове функции с таким же набором входных параметров это значение можно уже не вычислять.

Кроме того, Haskell является ленивым, что позволяет не вычислять значения, пока это не нужно.

Haskell обладает статической сильной полной типизацией с атоматическим выводом типов.

Также, Haskell поддерживает функции высшего порядка и частичное применение.

Haskell – краткий и элегантный.[7] В языке отсутствуют лишние ключевые слова такие как `define` в Scheme или `function` в OCaml для определения функций. Также, как в Python вместо скобок, ограничивающих блоки кода (Scheme, C, JavaScript) и разделительных знаков (C, OCaml) в нём используются отступы. Это делает код легче для восприятия, так как сразу видна вложенность. Однако, в отличие от Python, где отступы обязаны быть одинаковыми, в Haskell важно лишь, чтобы отступ вложенной операции был больше родительской.

Короткие программы легче поддерживать. И, как правило, в них меньше возможности допустить ошибку.

1.5

Разрабатываемый язык должен быть содержать минимум ключевых слов и разделяющих знаков, как в Haskell. При этом, он должен быть с динамической типизацией, как Scheme `WHY??`. В нём должно быть сопоставление с образцом как в OCaml и повторное использование переменных как в Prolog.

2 Синтаксис функционального языка программирования

Для описания синтаксиса языка используются расширенная форма Бэкуса-Наура (РБНФ). Альтернатива обозначается символом '|'. '*' после выражения означает, что оно может быть включено 0 и более раз, '+' - 1 и более, '?' - 0 или 1 раз.[8] Нетерминальные символы начинаются с заглавной буквы. Терминальные либо начинаются малой буквой, либо состоят целиком из заглавных букв. Правила записываются в виде `Идентификатор ::= выражение`, где идентификатор – нетерминал, а выражение – соответствующая правилам РБНФ комбинация терминальных и нетерминальных символов и специальных знаков. Точка в конце — специальный символ, указывающий на завершение правила.

Язык является регистрозависимым.

2.1 Словарь и представление

Существуют следующие виды токенов: идентификатор, число, строка, операторы и ключевые слова. Пробельные символы игнорируются, если они не существенны для разделения двух последовательных токенов.

- Идентификаторы – последовательности символов, исключающие пробельные символы, точки, запятые, скобки, кавычки, вертикальную черту. В отличие от большинства языков идентификаторы могут начинаться с цифр. В этом случае в состав этого идентификатора должен входить хотя бы один символ, не являющийся точкой или буквой `e`.

  Примеры идентификаторов: `day-of-week`, `0->1`, `nil?`, `%2!0?`.

- Число – целочисленная, вещественная константа, дробь или комплексное число. Число (в том числе дробь и вещественное число) может быть записано в шестнадцатеричной системе счисления.

  Примеры чисел: `#xA9.F`, `4/7`, `2+7i`, `#x7/ad`.

```
1 Digit   ::= '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'.
2 Hexd    ::= 'A'|'B'|'C'|'D'|'E'|'F'|'a'|'b'|'c'|'d'|'e'|'f'.
3 Hdigit  ::= hexd|digit
4 Dec     ::= ('+'|'-')? digit+.
```

```
5  Hex      ::= '#x' hdigit+.
6  Int      ::= dec|hex.
7  Fracdec  ::= dec '/' digit+.
8  Frachex  ::= hex '/' hdigit+.
9  Frac     ::= fracdec|frachex.
10 Expint   ::= dec 'e' dec.
11 Realdec  ::= dec '.' (digit+ ('e' dec)? )?.
12 Realhex  ::= hex '.' hdigit+
13 Real     ::= realdec|realhex.
14 Number   ::= int|real|frac.
15 Complex  ::= number? ('+'|'-') number 'i'.
16 Num      ::= number|complex.
```

- Строки – последовательности символов, заключённые в двойные(") кавычки.

  Примеры строк: `"valid string"`, `"it's a beautiful day"`.

- Операторы и ключевые слова – специальные символы, пары символов и слова, зарезервированные системой.

  Список таких символов:

  `( ) [ ] { } . : \ < <= > >= ! != = ^`

  `- + ++ / // % * ** || && <- -> #t #f`

  Список таких слов: `scheme`, `mod`, `if`, `zero?`, `eval`, `abs`, `odd?`, `even?`, `div`, `round`, `reverse`, `null?`, `not`, `sin`, `cos`, `tg`, `ctg`, `eq?`, `eqv?`, `equal?`, `gcd`, `lcm`, `expt`, `sqrt`.

3 Способы реализации языка программирования

4 Компоненты среды разработки

5 Технико-экономическое обоснование

Разработка программного обеспечения — достаточно трудоемкий и длительный процесс, требующий выполнения большого числа разнообразных операций. Организация и планирование процесса разработки программного продукта или программного комплекса при традиционном методе планирования предусматривает выполнение следующих работ:

- формирование состава выполняемых работ и группировка их по стадиям разработки;
- расчет трудоемкости выполнения работ;
- установление профессионального состава и расчет количества исполнителей;
- определение продолжительности выполнения отдельных этапов разработки;
- построение календарного графика выполнения разработки;
- контроль выполнения календарного графика.

Трудоемкость разработки программной продукции зависит от ряда факторов, основными из которых являются следующие: степень новизны разрабатываемого программного комплекса, сложность алгоритма его функционирования, объем используемой информации, вид ее представления и способ обработки, а также уровень используемого алгоритмического языка программирования. Чем выше уровень языка, тем трудоемкость меньше.

По степени новизны разрабатываемый проект относится к *группе новизны B* – разработка программной продукции, имеющей аналоги.

По степени сложности алгоритма функционирования проект относится к *3 группе сложности* - программная продукция, реализующая алгоритмы стандартных методов решения задач.

По виду представления исходной информации и способа ее контроля программный продукт относится к *группе 12* - исходная информация представлена в форме документов, имеющих различный формат и структуру и *группе 22* - требуется печать документов одинаковой формы и содержания, вывод массивов данных на машинные носители.

5.1 Трудоемкость разработки программной продукции

Трудоемкость разработки программной продукции (τ_{PP}) может быть определена как сумма величин трудоемкости выполнения отдельных стадий разработки программного продукта из выражения:

$$\tau_{PP} = \tau_{TZ} + \tau_{EP} + \tau_{TP} + \tau_{RP} + \tau_V,$$

где τ_{TZ} — трудоемкость разработки технического задания на создание программного продукта; τ_{EP} — трудоемкость разработки эскизного проекта программного продукта; τ_{TP} — трудоемкость разработки технического проекта программного продукта; τ_{RP} — трудоемкость разработки рабочего проекта программного продукта; τ_V — трудоемкость внедрения разработанного программного продукта.

5.1.1 Трудоемкость разработки технического задания

Расчёт трудоёмкости разработки технического задания (τ_{TZ}) [чел.-дни] производится по формуле:

$$\tau_{TZ} = T^Z_{RZ} + T^Z_{RP},$$

где T^Z_{RZ} — затраты времени разработчика постановки задачи на разработку ТЗ, [чел.-дни]; T^Z_{RP} — затраты времени разработчика программного обеспечения на разработку ТЗ, [чел.-дни]. Их значения рассчитываются по формулам:

$$T^Z_{RZ} = t_Z * K^Z_{RZ},$$

$$T^Z_{RP} = t_Z * K^Z_{RP},$$

где t_Z – норма времени на разработку ТЗ на программный продукт (зависит от функционального назначения и степени новизны разрабатываемого программного продукта), [чел.-дни]. В нашем случае по таблице получаем значение (группа новизны – В, функциональное назначение – технико-экономическое):

$$t_Z = 37.$$

K^Z_{RZ} — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задачи на стадии ТЗ. В нашем случае (совместная разработка с разработчиком ПО):

$$K^Z_{RZ} = 0.65.$$

K^Z_{RP} — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком программного обеспечения на стадии ТЗ. В нашем случае (совместная разработка с разработчиком постановки задач):

$$K^Z_{RP} = 0.35.$$

Тогда:

$$\tau_{TZ} = 37 * \ (0.35 + 0.65) = 37.$$

5.1.2 Трудоемкость разработки эскизного проекта

Расчёт трудоёмкости разработки эскизного проекта (τ_{EP}) [чел.-дни] производится по формуле:

$$\tau_{EP} = T^E_{RZ} + T^E_{RP},$$

где T^E_{RZ} — затраты времени разработчика постановки задачи на разработку эскизного проекта (ЭП), [чел.-дни]; T^E_{RP} — затраты времени разработчика программного обеспечения на разработку ЭП, [чел.-дни]. Их значения рассчитываются по формулам:

$$T^E_{RZ} = t_E * K^E_{RZ},$$

$$T^E_{RP} = t_E * K^E_{RP},$$

где t_E – норма времени на разработку ЭП на программный продукт (зависит от функционального назначения и степени новизны разрабатываемого программного продукта), [чел.-дни]. В нашем случае по таблице получаем значение (группа новизны – В, функциональное назначение – технико-экономическое):

$$t_E = 77.$$

K^E_{RZ} — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком постановки задачи на стадии ЭП. В нашем случае (совместная разработка с разработчиком ПО):

$$K^E_{RZ} = 0.7.$$

K^E_{RP} — коэффициент, учитывающий удельный вес трудоемкости работ, выполняемых разработчиком программного обеспечения на стадии ТЗ. В нашем случае (совместная разработка

с разработчиком постановки задач):

$$K_{RP}^{E} = 0.3.$$

Тогда:

$$\tau_{EP} = 77 * \ (0.3 + 0.7) = 77.$$

5.1.3 Трудоемкость разработки технического проекта

Трудоёмкость разработки технического проекта (τ_{TP}) [чел.-дни] зависит от функционального назначения программного продукта, количества разновидностей форм входной и выходной информации и определяется по формуле:

$$\tau_{TP} = (t_{RZ}^{T} + t_{RP}^{T}) * K_V * K_R,$$

где t_{RZ}^{T} — норма времени, затрачиваемого на разработку технического проекта (ТП) разработчиком постановки задач, [чел.-дни]; t_{RP}^{T} — норма времени, затрачиваемого на разработку ТП разработчиком ПО, [чел.-дни]. По таблице принимаем (функциональное назначение — технико-экономическое планирование, количество разновидностей форм входной информации — 1 (файл с текстом программы на исходном языке), количество разновидностей форм выходной информации — 1 (файл с текстом программы на языке Scheme)):

$$t_{RZ}^{T} = 30,$$

$$t_{RP}^{T} = 8.$$

K_R — коэффициент учета режима обработки информации. По таблице принимаем (группа новизны — B, режим обработки информации — реальный масштаб времени):

$$K_R = 1.26.$$

K_V — коэффициент учета вида используемой информации, определяется по формуле:

$$K_V = \frac{K_P * n_P + K_{NS} * n_{NS} + K_B * n_B}{n_P + n_{NS} + n_B},$$

где K_P — коэффициент учета вида используемой информации для переменной информации; K_{NS} — коэффициент учета вида используемой информации для нормативно-справочной информации; K_B — коэффициент учета вида используемой информации для баз данных; n_P —

количество наборов данных переменной информации; n_{NS} — количество наборов данных нормативно-справочной информации; n_B — количество баз данных. Коэффициенты находим по таблице (группа новизны - В):

$$K_P = 1.00,$$

$$K_{NS} = 0.72,$$

$$K_B = 2.08.$$

Количество наборов данных, используемых в рамках задачи:

$$n_P = 10,$$

$$n_{NS} = 0,$$

$$n_B = 0.$$

Находим значение K_V:

$$K_V = \frac{1.00 * 10 + 0.72 * 0 + 2.08 * 1}{10 + 0 + 1} = 1.098.$$

Тогда:

$$\tau_{TP} = (30 + 8) * 1.098 * 1.26 = 53.$$

5.1.4 Трудоемкость разработки рабочего проекта

Трудоёмкость разработки рабочего проекта (τ_{RP}) [чел.-дни] зависит от функционального назначения программного продукта, количества разновидностей форм входной и выходной информации, сложности алгоритма функционирования, сложности контроля информации, степени использования готовых программных модулей, уровня алгоритмического языка программирования и определяется по формуле:

$$\tau_{RP} = (t^R_{RZ} + t^R_{RP}) * K_K * K_R * K_Y * K_Z * K_{IA},$$

где t^R_{RZ} — норма времени, затраченного на разработку рабочего проекта на алгоритмическом языке высокого уровня разработчиком постановки задач, [чел.-дни]. t^R_{RP} — норма времени, затраченного на разработку рабочего проекта на алгоритмическом языке высокого уровня разработчиком ПО, [чел.-дни]. По таблице принимаем (функциональное назначение —

технико-экономическое планирование, количество разновидностей форм входной информации — 1 (файл с текстом программы на исходном языке), количество разновидностей форм выходной информации — 1 (файл с текстом программы на языке Scheme)):

$$t_{RZ}^{R} = 8,$$

$$t_{RP}^{R} = 51.$$

K_K — коэффициент учета сложности контроля информации. По таблице принимаем (степень сложности контроля входной информации — 12, степень сложности контроля выходной информации — 22):

$$K_K = 1.00.$$

K_R — коэффициент учета режима обработки информации. По таблице принимаем (группа новизны — B, режим обработки информации — реальный масштаб времени):

$$K_R = 1.26.$$

K_Y — коэффициент учета уровня используемого алгоритмического языка программирования. По таблице принимаем значение (интерпретаторы, языковые описатели):

$$K_Y = 0.8.$$

K_Z — коэффициент учета степени использования готовых программных модулей. По таблице принимаем (использование готовых программных модулей составляет менее 25

$$K_Z = 0.8.$$

K_{IA} — коэффициент учета вида используемой информации и сложности алгоритма программного продукта, его значение определяется по формуле:

$$K_I A = \frac{K'_P * n_P + K'_{NS} * n_{NS} + K'_B * n_B}{n_P + n_{NS} + n_B},$$

где K'_P — коэффициент учета сложности алгоритма ПП и вида используемой информации для переменной информации; K'_{NS} — коэффициент учета сложности алгоритма ПП и вида используемой информации для нормативно-справочной информации; K'_B — коэффициент учета сложности алгоритма ПП и вида используемой информации для баз данных. n_P —

количество наборов данных переменной информации; n_{NS} — количество наборов данных нормативно-справочной информации; n_B — количество баз данных. Коэффициенты находим по таблице (группа новизны - B):

$$K'_P = 1.00,$$

$$K'_{NS} = 0.48,$$

$$K'_B = 0.4.$$

Количество наборов данных, используемых в рамках задачи:

$$n_P = 10,$$

$$n_{NS} = 0,$$

$$n_B = 1.$$

Находим значение K_{IA}:

$$K_{IA} = \frac{1.00 * 10 + 0.48 * 0 + 0.4 * 1}{10 + 0 + 1} = 0.945.$$

Тогда:

$$\tau_{RP} = (8 + 51) * 1.00 * 1.26 * 0.8 * 0.8 * 0.945 = 45.$$

5.1.5 Трудоемкость выполнения стадии «Внедрение»

Расчёт трудоёмкости разработки технического проекта (τ_V) [чел.-дни] производится по формуле:

$$\tau_V = (t^V_{RZ} + t^V_{RP}) * K_K * K_R * K_Z,$$

где t^V_{RZ} — норма времени, затрачиваемого разработчиком постановки задач на выполнение процедур внедрения программного продукта, [чел.-дни]; t^V_{RP} — норма времени, затрачиваемого разработчиком программного обеспечения на выполнение процедур внедрения программного продукта, [чел.-дни]. По таблице принимаем (функциональное назначение — технико-экономическое планирование, количество разновидностей форм входной информации — 1 (файл с текстом программы на исходном языке), количество разновидностей форм выходной информации — 1 (файл с текстом программы на языке Scheme)):

$$t^V_{RZ} = 9,$$

$$t_{RP}^{V} = 11.$$

Коэффициент K_K и K_Z были найдены выше:

$$K_K = 1.00,$$

$$K_Z = 0.8.$$

K_R — коэффициент учета режима обработки информации. По таблице принимаем (группа новизны — B, режим обработки информации — реальный масштаб времени):

$$K_R = 1.26.$$

Тогда:

$$\tau_V = (9 + 11) * 1.00 * 1.26 * 0.8 = 21.$$

Общая трудоёмкость разработки ПП:

$$\tau_{PP} = 37 + 77 + 53 + 45 + 21 = 233.$$

5.2 Расчет количества исполнителей

Средняя численность исполнителей при реализации проекта разработки и внедрения ПО определяется соотношением:

$$N = \frac{t}{F},$$

где t — затраты труда на выполнение проекта (разработка и внедрение ПО); F — фонд рабочего времени. Разработка велась 5 месяцев с 1 января 2016 по 31 мая 2016. Количество рабочих дней по месяцам приведено в таблице 1. Из таблицы получаем, что фонд рабочего времени

$$F = 96.$$

Получаем число исполнителей проекта:

$$N = \frac{233}{96} = 3$$

Для реализации проекта потребуются 1 старший инженер и 2 простых инженера.

Таблица 1: Количество рабочих дней по месяцам

Номер месяца	**Интервал дней**	**Количество рабочих дней**
1	01.01.2016 - 31.01.2016	15
3	01.02.2016 - 29.02.2016	20
4	01.03.2016 - 31.03.2016	21
5	01.04.2016 - 30.04.2016	21
6	01.05.2016 - 31.05.2016	19
Итого		96

5.3 Ленточный график выполнения работ

На основе рассчитанных в главах 5.1, 5.2 трудоёмкости и фонда рабочего времени найдём количество рабочих дней, требуемых для выполнения каждого этапа разработка. Результаты приведены в таблице 2.

Таблица 2: Трудоёмкость выполнения работы над проектом

Номер стадии	**Название стадии**	**Трудоёмкость [чел.-дни]**	**Удельный вес [%]**	**Количество рабочих дней**
1	Техническое задание	37	11	10
2	Эскизный проект	77	24	23
3	Технический проект	53	35	34
4	Рабочий проект	45	25	24
5	Внедрение	21	5	5
Итого		233	100	96

Планирование и контроль хода выполнения разработки проводится по ленточному графику выполнения работ. По данным в таблице 2 в ленточный график (таблица 3), в ячейки столбца “продолжительности рабочих дней” заносятся времена, которые требуются на выполнение соответствующего этапа. Все исполнители работают одновременно.

Таблица 3: Ленточный график выполнения работ

Номер стадии	Продолжительность [раб.-дни]	Календарные дни																						
		01.01.2016 - 03.01.2016	04.01.2016 - 10.01.2016	11.01.2016 - 17.01.2016	18.01.2016 - 24.01.2016	25.01.2016 - 31.01.2016	01.02.2016 - 07.02.2016	08.02.2016 - 14.02.2016	15.02.2016 - 21.02.2016	22.02.2016 - 28.02.2016	29.02.2016 - 06.03.2016	07.03.2016 - 13.03.2016	14.03.2016 - 20.03.2016	21.03.2016 - 27.03.2016	28.03.2016 - 03.04.2016	04.04.2016 - 10.04.2016	11.04.2016 - 17.04.2016	18.04.2016 - 24.04.2016	25.04.2016 - 01.05.2016	02.05.2016 - 08.05.2016	08.05.2016 - 15.05.2016	16.05.2016 - 22.05.2016	23.05.2016 - 29.05.2016	30.05.2016 - 31.05.2016
		Количество рабочих дней																						
		0	0	5	5	5	5	5	6	3	5	3	5	5	5	5	5	5	5	3	4	5	5	2
1	10			5	5																			
2	23					5	5	5	6	2														
3	34									1	5	3	5	5	5	5	5							
4	24																	5	5	3	4	5	2	
5	5																						3	2

5.4 Определение себестоимости программной продукции

Затраты, образующие себестоимость продукции (работ, услуг), состоят из затрат на заработную плату исполнителям, затрат на закупку или аренду оборудования, затрат на организацию рабочих мест, и затрат на накладные расходы.

В таблице 4 приведены затраты на заработную плату и отчисления на социальное страхование в пенсионный фонд, фонд занятости и фонд обязательного медицинского страхования (30.5 %). Для старшего инженера предполагается оклад в размере 120000 рублей в месяц, для инженера предполагается оклад в размере 100000 рублей в месяц.

Таблица 4: Затраты на зарплату и отчисления на социальное страхование

Должность	**Зарплата в месяц**	**Рабочих месяцев**	**Суммарная зарплата**	**Затраты на социальные нужды**
Старший инженер	120000	5	600000	183000
Инженер	100000	5	500000	152500
Инженер	100000	5	500000	152500
Суммарные затраты			2088000	

Расходы на материалы, необходимые для разработки программной продукции, указаны в таблице 5.

Таблица 5: Затраты на материалы

Наименование материала	**Единица измерения**	**Кол-во**	**Цена за единицу, руб.**	**Сумма, руб.**
Бумага А4	Пачка 400 л.	2	200	400
Картридж для принтера HP P10025	Шт.	3	450	1350
Суммарные затраты				1750

В работе над проектом используется специальное оборудование — персональные электронно-вычислительные машины (ПЭВМ) в количестве 9 шт. Стоимость одной ПЭВМ составляет 90000 рублей. Месячная норма амортизации К = 2,7%. Тогда за 5 месяцев работы расходы на амортизацию составят $P = 90000 * 3 * 0.027 * 5 = 36450$ рублей.

Накладные расходы рассчитываются по следующе формуле:

$$C_n = A_n * C_z$$

$$N = 2.1 * 1600000 = 3360000$$

Общие затраты на разработку программного продукта (ПП) составят

$$2088000 + 1750 + 36450 + 3360000 = 5486200 \text{ рублей.}$$

5.5 Определение стоимости программной продукции

Для определения стоимости работ необходимо на основании плановых сроков выполнения работ и численности исполнителей рассчитать общую сумму затрат на разработку программного продукта. Если ПП рассматривается и создается как продукция производственно-технического назначения, допускающая многократное тиражирование и отчуждение от непосредственных разработчиков, то ее цена P определяется по формуле:

$$P = K * C + Pr,$$

где C — затраты на разработку ПП (сметная себестоимость); K — коэффициент учёта затрат на изготовление опытного образца ПП как продукции производственно-технического назначения ($K = 1.1$); Pr — нормативная прибыль, рассчитываемая по формуле:

$$Pr = \frac{C * \rho_N}{100},$$

где ρ_N — норматив рентабельности, $\rho_N = 30\%$;

Получаем стоимость программного продукта:

$$P = 1.1 * 5486200 + 5486200 * 0.3 = 7680680 \text{ рублей.}$$

5.6 Расчет экономической эффективности

Основными показателями экономической эффективности является чистый дисконтированный доход (NPV) и срок окупаемости вложенных средств. Чистый дисконтированный доход определяется по формуле:

$$NPV = \sum_{t=0}^{T} (R_t - Z_t) * \frac{1}{(1+E)^t},$$

где T — горизонт расчета по месяцам; t — период расчета; R_t — результат, достигнутый на t шаге (стоимость); Z_t — текущие затраты (на шаге t); E — приемлемая для инвестора норма прибыли на вложенный капитал.

На момент начала 2016 года, ставка рефинансирования 11% годовых (ЦБ РФ), что

эквивалентно (0.916% в месяц). В виду особенности разрабатываемого продукта он может быть продан лишь однократно. Отсюда получаем

$$E = 0.00916.$$

В таблице 6 находится расчёт чистого дисконтированного дохода. График его изменения приведён на рисунке 1.

Таблица 6: Расчёт чистого дисконтированного дохода

Месяц	**Текущие затраты, руб.**	**Затраты с начала года, руб.**	**Текущий доход, руб.**	**ЧДД, руб.**
Январь	1096890	1096890	0	-1096890
Февраль	1096890	2193780	0	-2183823.7
Март	1096890	3290670	0	-3260891.4
Апрель	1096890	4387560	0	-4328182.7
Мая	1098640	5486200	7680680	2019802

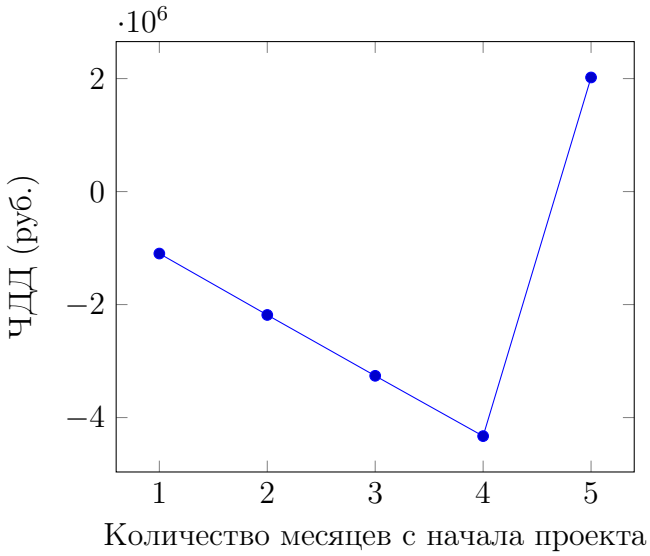

Рисунок 1 — График изменения чистого дисконтированного дохода

Согласно проведенным расчетам, проект является рентабельным. Разрабатываемый проект позволит превысить показатели качества существующих систем и сможет их заменить. Итоговый ЧДД составил: 2019802 рубля.

5.7 Результаты

В рамках организационно-экономической части был спланирован календарный график проведения работ по созданию подсистемы поддержки проведения диагностики промышленных, а также были проведены расчеты по трудозатратам. Были исследованы и рассчитаны следующие статьи затрат: материальные затраты; заработная плата исполнителей; отчисления на социальное страхование; накладные расходы.

В результате расчетов было получено общее время выполнения проекта, которое составило 96 рабочих дней, получены данные по суммарным затратам на создание и разработку функционального языка, которые составили 5486200 рублей. Согласно проведенным расчетам, проект является рентабельным. Цена данного программного проекта составила 7680680 рублей, итоговый ЧДД составил 2019802 рублей.

6 Примеры кода

Список литературы

[1] Peter Seibel. Practical Common Lisp. Publication Date: April 6, 2005

[2] http://www.schemers.org/Documents/Standards/R5RS/HTML/

[3] R. Kent Dybvig. The Scheme Programming Language, Fourth Edition. The MIT Press. 2009

[4] Information technology — Programming languages — Prolog. 1995

[5] A Byte of Python, http://python.swaroopch.com/

[6] Yaron Minsky, Anil Madhavapeddy, Jason Hickey. Real world OCaml. 2013

[7] Miran Lipovača. LEARN YOU A HASKELL FOR GREAT GOOD!. 2011

[8] Скоробогатов С.Ю. Лекции по курсу 'Компиляторы', 2014.

[9] Арсеньев В.В., Сажин Ю.Б. Методические указания к выполнению организационно-экономической части дипломных проектов по созданию программной продукции. М.: изд. МГТУ им. Баумана, 1994. 52 с. 2.

[10] Под ред. Смирнова С.В. Организационно-экономическая часть дипломных проектов исследовательского профиля. М.: изд. МГТУ им. Баумана, 1995. 100 с.