Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
Вычислительная техника

УТВЕРЖДАЮ
Заведующий кафедрой

_____ О. В. Непомнящий
подпись
«____» _____ 2024 г.

БАКАЛАВРСКАЯ РАБОТА

Фреймворк модульного тестирования
для функционально-потоковых параллельных программ

09.03.01 - «Информатика и вычислительная техника»

Руководитель _____ доцент кафедры _____ В. С. Васильев
подпись, дата вычислительной техники

Выпускник _____ _____ Д. М. Дудинков
подпись, дата подпись, дата

Красноярск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одним из подходов повышения надежности программного обеспечения (ПО) является модульное тестирование. Для широко используемых языков программирования существуют инструменты, обеспечивающие поддержку процесса тестирования. Высокая эффективность процесса тестирования достигается с помощью его автоматизации, которая позволяет снизить негативное влияние человеческого фактора на результаты тестирования.

Целью работы является создание фреймворка модульного тестирования для языков функционально потоковой параллельной (ФПП) парадигмы. Функционально-потоковая парадигма имеет ряд особенностей, влияющих на процесс тестирования кода.

Исследований, связанных с реализацией инструментов модульного тестирования ФПП программ не проводилось, поэтому данная работа является **актуальной**.

Практическим результатом работы являются: предложенный подход к разработке тестов, а также созданный набор инструментов, позволяющий автоматизировать запуск тестов и сбор результатов тестирования. Произведена интеграция данных инструментов в разрабатываемую интегрированную среду разработки ФПП языков программирования.

Апробация работы. Результаты работы докладывались и обсуждались на международных конференциях:

- "Проспект Свободный";

- "Кибернетика, информатика, аналитика: модели, инструменты, методы".

Структура работы отражает решаемые **задачи**. Пояснительная записка состоит из введения, 3 глав, заключения, списка литературы из 19 наименований, 2 приложений, содержит 13 рисунков и 2 таблицы.

В первой главе описаны особенности функционально потоковой парадигмы влияющие на организацию системы модульного тестирования, а также инструментальные средства ФПП программирования с которыми интегрируется фреймворк модульного тестирования. Предложен подход к организации тестирования и структура программных средств, учитывающие специфику ФПП программирования. Разработана спецификация требований к инструментам модульного тестирования.

Вторая глава содержит результаты проектирования и реализации инструментальных средств модульного тестирования ФПП программ, описание особенностей использования созданных инструментов.

В рамках третьей главы разработаны примеры программ на которых выполнялось тестирование; созданы модульные тесты к существующим комплектам ФПП программ; выполнена проверка работоспособности инструментов модульного тестирования; описан процесс сборки фреймворка.

1 Постановка задачи

Целью модульного тестирования является проверка корректности работы модулей (функций) на заранее заготовленных наборах данных. Тесты представляют собой код на том же языке программирования, что и основная программа, выполняющий вызов тестируемых функций и сверки результатов с эталонными.

Тестирование — процесс сверки результатов работы программы с эталонными. Исходный код любой программы разделяется на функции, именно для них разрабатываются тесты. Существуют различные способы тестирования. Особенностями модульного тестирования являются:

- объектом тестирования являются функции;

- в ходе тестирования описываются отдельные тестовые случаи, представляющие собой пару - аргумент тестируемой функции и эталонное значение результата. Рекомендуется придерживаться правила "одна проверка на тест" [1];

- тестовые случаи, связанные с одним и тем же модулем, группируются в тестовые наборы.

Фреймворк модульного тестирования традиционно решает задачу идентификацию тестового кода и позволяет выполнять тесты отдельно от основной программы.

1.1 Особенности функционально-потокового параллельного программирования

В контексте модульного тестирования кода функционально-потоковые языки обладают рядом особенностей. На основе исходного кода каждой функции строится реверсивный информационный граф (РИГ) и управляющий граф (УГ) [2, 3]. Этот набор графов используется другими инструментами среды ФПП программирования [4]. Процесс формирования этих графов и их

использования показа на рисунке 1.1. Сплошными линиями обозначены операции генерации данных, пунктирными - информационные зависимости, а штрих-пунктирными зависимости между инструментами.

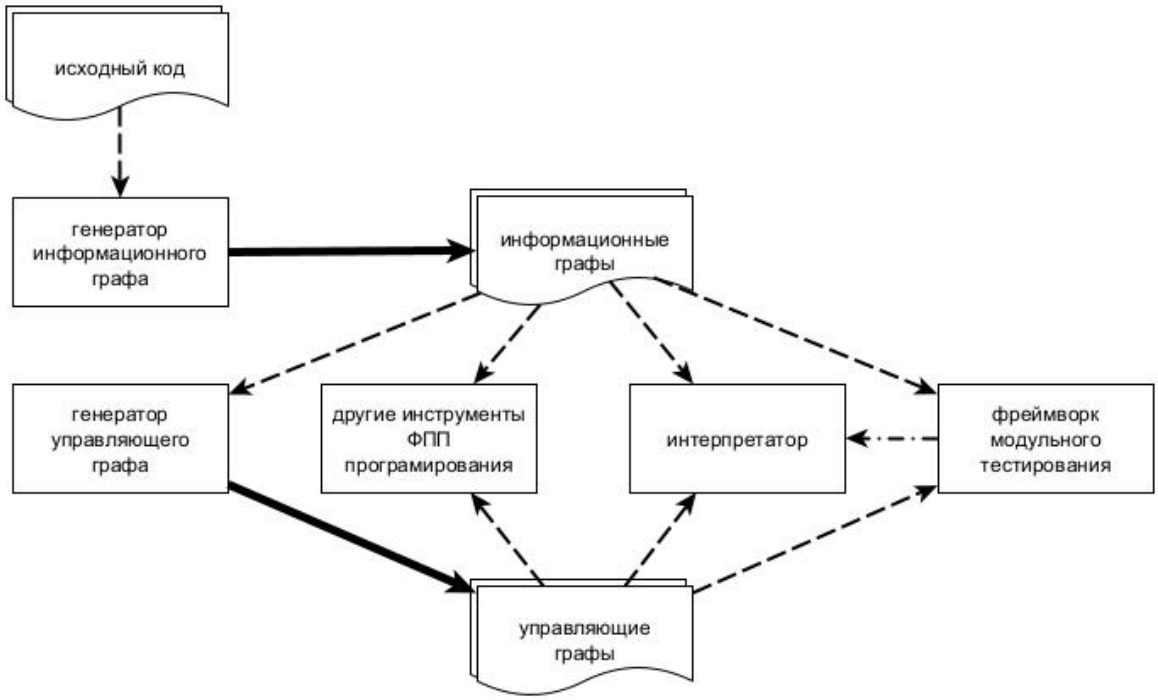

Рисунок 1.1 — Связь фреймворка модульного тестирования с другими инструментами ФПП программ

Исходный код функции и связанные с ней графы помещаются в специальный репозиторий [5]. В настоящий момент наиболее развитым является репозиторий на основе файловой системы, при этом каждая функция размещается в отдельном каталоге. Имя каталога совпадает с именем функции. Для исполнения функции на вход интерпретатора подается имя функции и аргумент, а необходимые для работы программы графы извлекаются из репозитория.

В ФПП языках полиморфизм реализуется за счёт возможности описания различных версий для каждой функции [6]. В соответствующих подкаталогах репозитория размещаются наборы графов для этих версий. На рисунке 1.2

изображен фрагмент репозитория, использующего для хранения данных файловую систему.

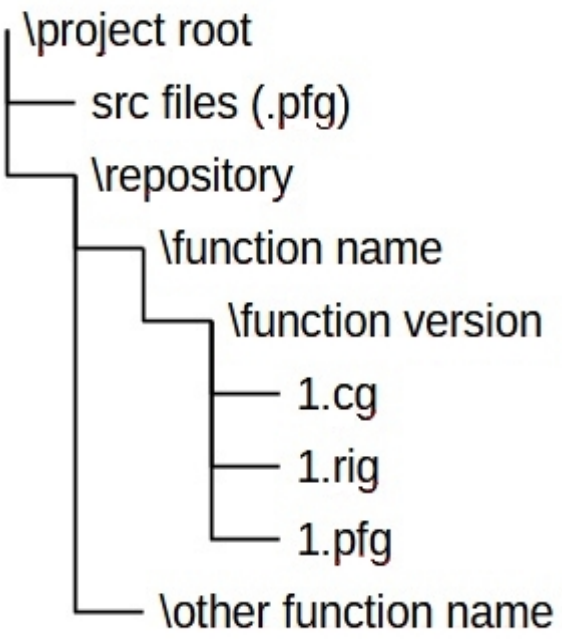

Рисунок 1.2 — Структура репозитория на основе файловой системы

1.2 Подход к разработке модульных тестов

Исходный код тестов практически не отличается от остального кода, однако в ходе тестирования нужно отделить его и выполнить, существует несколько подходов его идентификации. Зачастую в объектно-ориентированных языках код тестов размещается в функциях классов-наследников специально выделенного тестового класса [7]. В фреймворках *Boost.Test* и *GTest* для выделения кода тестов используются макросы [8]. Фреймворки *EUnit* языка *Erlang* [9] и *PyTest* языка *Python* [10] идентифицирует исходный код тестов по имени функции или содержащего этот код файла.

В репозитории не сохраняется информация об имени файла с исходным кодом, в котором описана функция, поэтому для решения проблемы идентификации кода тестов, решено дополнять имена соответствующих ему функций префиксом "*test_*". На рисунке 1.3 приведен пример кода на

функционально-потоковом языке программирования Пифагор, выполняющего умножение матриц и тестов для неё, а на рисунке 1.4 — описание структуры модульного теста.

```
matrix_mul << funcdef X {

  A << X:1;
  B << X:2:#;

  [(X:1:1:|, X:2:|):[!=, =]):?] ^ (
    "bad matrix size",

    {
       block {
         ARows << ((A, M):dup:#:[]);
         BColumns << ((B:|, A:|):dup:[]);
         ResultElements << (ARows, BColumns, row_col_mul):map3;
         ResultElements >> break;
       }
    }
  ):. >> return;
}

row_col_mul << funcdef X {

  Row << X:1;
  Column << X:2;

  return << ((Row, Column, vec_mul):map3:[]:sum_list);
}

test_matrix_mul << funcdef X {
  Cases << [
    ("(2x3)x(3x2)",
      ( (((1,2,3), (4,5,6)), ((7,8),(9,1),(2,3))):matrix_mul,
      ((31,19),(85,55)) ):equals),
    ("(2x2)x(2x1)",
      ( (((1,2),(3,4)), ((5), (6))):matrix_mul,
      ((17), (39)) ):equals),
    ("(2x3)x(2x3)",
      ( (((1,2,3), (4,5,6)), ((1,2,3), (4,5,6))):matrix_mul,
      "bad matrix size" ):equals)
  ];
  return << Cases;
}
```

Рисунок 1.3 — Фрагмент исходного кода на Пифагор

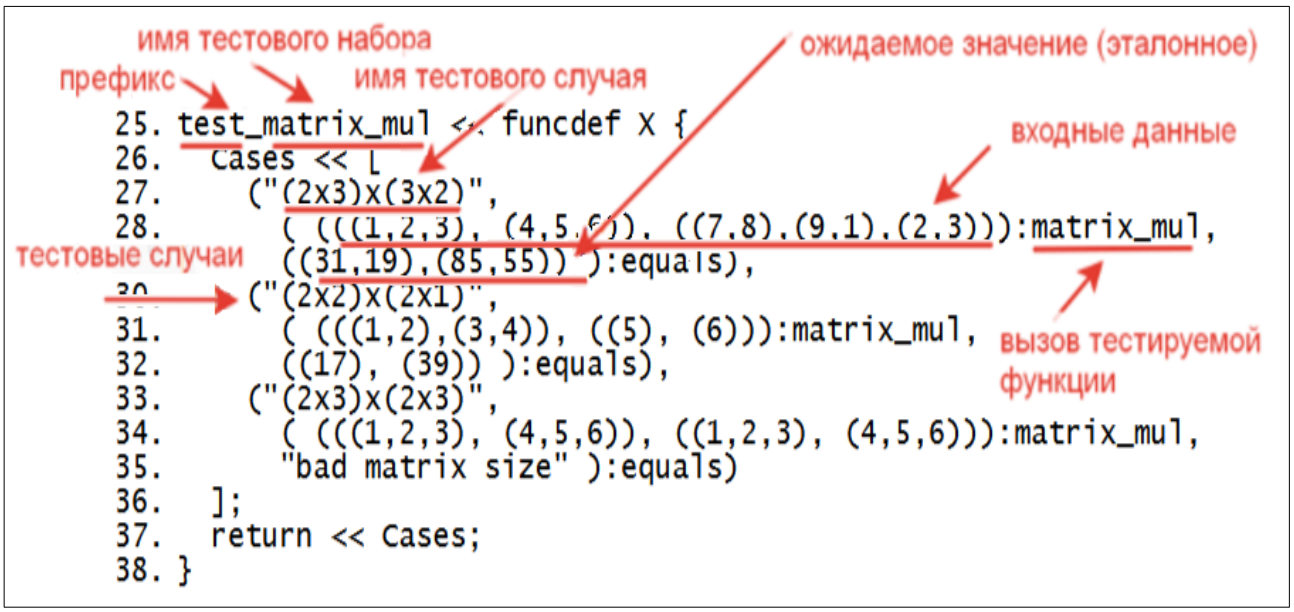

Рисунок 1.4 — Фрагмент записи тестов для функции умножения матриц

На рисунках 1.3 и 1.4 показано что для описания тестового набора создается отдельная функция, возвращающая список кортежей вида:

```
<"имя тестового случая">, <результат работы тестового случая (true/false)>
```

Сравнение полученных результатов с ожидаемыми эталонными значениями удобно реализовывать с использованием функции *equals*, приведённой на рисунке 1.5, выполняющую сравнение двух значений любого типа [11].

```
equals << funcdef X {
  A << X:1;
  B << X:2;
  EqOp << [((X:|,3):[<,=]):?]^(=, X:3);
  TypeA << A:type;
  TypeB << B:type;

  [((TypeA, TypeB):[!=, =]):?]^(
    false,
    {
      block {
        [((TypeA, datalist):[!=, =]):?]^(
          {(X:1, X:2):EqOp},
          {X:vec_equals}
        ) >> break
      }
    }
  ):.:. >> return;
}

vec_equals << funcdef X {
  VectorA << X:1;
  VectorB << X:2;

  SizeA << VectorA:|;
  SizeB << VectorB:|;

  [((SizeA, SizeB):[!=,=]):?]^(
    false,
    {
      block {
        EqOp << [((X:|,3):[<,=]):?]^(=, X:3);
        Ops << (EqOp, SizeA):dup;
        Pairs << (VectorA, VectorB, Ops):#:[];
        CountEquals << ((Pairs:equals):?):|;
        [((CountEquals, SizeA):[!=,=]):?]^(false, true) >> break
      }
    }
  ):. >> return;
}

math_is_close << funcdef X {
  A << X:1;
  B << X:2;
  Eps << X:3;

  return << ((A,B):-:abs,Eps):<;
}

fuzzy_compare << funcdef X {
(X:[], 0.000001):math_is_close >> return;
}

abs << funcdef X
{
  (X:-, X):[(X,0):(<,=>):?] >>return;
}
```

Рисунок 1.5 — Исходного кода функции equals на Пифагор

1.3 Интерпретация тестовых наборов

Тестовый фреймворк решает не только задачу описания и группировки тестовых случаев, но также обеспечивает запуск тестов и сбор результатов их

выполнения. При этом учитываются различные способы завершения функции, поддерживаемые в целевом языке программирования, например – возврат результата или вырабатывание исключения, или по таймауту.

В ФПП языках реализован специфический механизм обработки ошибок — любая функция может вернуть кортеж, один из элементов которого является специально выделенное значение ошибки. В семантике языка определены правила интерпретации кортежей с кодами ошибок [12]. Обработка ошибок выполняется с использованием тех же синтаксических средств, что и любых других задач, а проверка вырабатывания ошибки функцией сводится к сравнению вычисленного функцией значения с кодом ошибки. При написании модульных тестов для этой цели может использоваться подход, описанный в разделе 1.2.

Для формирования отчета утилита модульного тестирования обрабатывает результаты работы интерпретатора считывая их с выходного потока. Фрагмент вывода интерпретатора для функции test_matrix_mul, содержащей тестовый набор для функции умножения матриц на языке Пифагор, показан на рисунке 1.6.

Запуск интерпретатора осуществляется командой:

inter2.exe <имя функции>.

Перед интерпретацией необходимо заранее транслировать аргумент, для этого выполняется следующая команда:

trans2.exe -c <имя файла с аргументом>

Однако, так как функция, описывающая набор тестовых случаев по описанной схеме, не принимает никаких аргументов, — подготавливать аргументы не требуется.

```
Func is Spec of 0
I'm  ":"  in the state  2  and I got signal  1
Interpreter called!
Function:        "."
Argument:        "true"
Func is Spec of 0
I'm  "return"  in the state  0  and I got signal  1
I'm  "(---)"  in the state  1 ; my len is  2  and I got signal  2
I'm  "[---]"  in the state  0  and I got signal  1
I'm  "return"  in the state  1  and I got signal  1
==========================================
Final result is  "[((\"(2x3)x(3x2)\"),true),((\"(2x2)x(2x1)\"),true),
((\"(2x3)x(2x3)\"),true)]"
==========================================
Final result: [(("(2x3)x(3x2)"),true),(("(2x2)x(2x1)"),true),
(("(2x3)x(2x3)"),true)]
success
```

Рисунок 1.6 — Пример результата работы интерпретатора

Если в результате выполнения тестов получен ненулевой код ошибки или в конце его выходного потока отсутствует подстрока вида:

Final result: <Результат>
success

— набор тестов считается «проваленным», при этом отдельные тестовые случаи не рассматриваются.

При успешном завершении работы интерпретатора анализируется его стандартный поток вывода и формируется отчёт, содержащий результаты проверки отдельных тестовых случаев. Пример отчета для тестового набора test_matrix,_mul:

```
   # test_matrix_mul
- (2x3)x(3x2) >>> true
- (2x2)x(2x1) >>> true
- (2x3)x(2x3) >>> true
```

Утилита модульного тестирования принимает на вход список функций, которые необходимо исполнить или опцию *-a*, приводящую к запуску всех тестовых наборов, содержащихся в репозитории. Задать ограничение времени исполнения одного тестового набора можно с помощью опции *-t <time_sec>*. Опция -h выводит справку по опциям утилиты.

1.4 Утилита модульного тестирования с графическим интерфейсом

На рисунке 1.7 показан макет графического интерфейса утилиты модульного тестирования.

В нижней части окна при выборе режиме «выборочных тестов» расположен список найденных утилитой тестовых наборов, позволяющий выбрать тестовые наборы, которые будут выполняться при запуске. Список наборов скрывается при выборе опции «*Select all tests*» и отображается при выборе «*Deselect all tests*», это сделано потому, что обычно требуется запускать все наборы, а список наборов при этом расходует пространство окна. Выбор опции «*Select all tests*» переводит окно в режим выполнения всех модульных тестов.

В верхней части окна выводятся результаты тестирования. Для каждого набора, с помощью отступа, отображаются вложенные тестовые случаи. Для каждого случая выводится результат (*true/false*), при этом случаи, завершившиеся успешно выделяются зеленым цветом, а проваленные — красным.

Для запуска тестов нужно: выбрать, тестовые наборы; нажать кнопку «*Run*». В поле *TimeOut* можно указать ограничение времени исполнения для одного тестового набора в секундах.

а) режим выполнения всех модульных тестов

б) режим выборочных тестов

Рисунок 1.7 — Макет интерфейса утилиты модульного тестирования

1.5 Выводы по главе

Сформулированы задачи, решаемые фреймворком модульного тестирования, описаны особенности функционально-потоковой парадигмы программирования, влияющие на процесс тестирования. Предложен подход к разработке модульных тестов на ФПП языке программирования Пифагор. Разработана спецификация требований, учитывающая специфику существующих инструментов ФПП программирования.

2 Проектирование. Особенности реализации

Для удобства интеграции создаваемой утилиты с существующими инструментами ФПП программирования, она должна предоставлять схожий с ними интерфейс. Созданные ранее инструменты представляют собой консольные утилиты, взаимодействующие с исходным кодом программ и репозиторием. Для наиболее востребованных сценариев применения утилит, использовались *shell* и *bash* скрипты для *Linux* и *Windows* соответственно.

В настоящее время ведется разработка интегрированной среды разработки (*IDE, integrated development environment*) для функционально-потоковых языков программирования, при этом используется язык *C++* и фреймворк *Qt*.

Для обеспечения лучшей переносимости кода, инструмент модульного тестирования решено реализовать на языке *Python*. В ходе этой работы, скрипты для всех других инструментов ФПП программирования также были переписаны на языке *Python*. Оконное приложение, макет которого показан на рисунке 1.7 разрабатывается на языке *C++* с использованием библиотеки *Qt* для упрощения встраивания в создаваемую *IDE*.

2.1 Алгоритм и реализация утилиты модульного тестирования

При использовании консольной утилиты, пользователь открывает терминал, переходит в каталог с проектом и запускает скрипт модульного тестирования, передавая ему имена тестовых наборов и необязательную опцию таймаута. Например:

```
python tests.py foo bar -t 10
```

Тут: *foo* и *bar* — функции, реализующие тестовые наборы, *10* — ограничение времени выполнения одного тестового набора в секундах.

Текущим каталогом при запуске скрипта должен являться каталог проекта с исходным кодом. Формирование результатов, описанных в первой главе работы, обеспечивается следующим алгоритмом:

а) считываются параметры запуска, обработать:

а.1) сохраняется значение *timeout*, переданное с опцией *-t*. Если опция *-t* не была установлена, то *timeout = 15*;

а.2) в переменную *test_suits* сохраняется список запускаемых тестовых наборов. Если передана опция *-a*, то для этого выполняется поиск в репозитории функций, имена которых соответствуют шаблону имени тестового набора. Иначе, в *test_suits* помещаются имена функций, переданные в качестве аргументов скрипта и соответствующие шаблону;

б) создается пустой список для хранения результатов тестирования (*results*).

в) для каждого элемента списка *test_suits* выполняется:

в.1) в дочернем процессе запускается интерпретация функции, задающей тестовый набор.

в.2) главный процесс ожидает завершения дочернего процесса *timeout* секунд. Если дочерний не успевает выполнить вычисления за это время, он принудительно останавливается, информация об ошибки записывается в переменную *results.* Иначе, считывается стандартный вывод процесса.

в.3) выполняется обработка результатов дочернего процесса в соответствии с правилами, описанными в разделе 1.3 и помещаются в *results.*

В таблице 2.1 приведены форматы ошибок различных видов. Разбор сообщений в этих форматах должны выполнять все утилиты, использующие результаты тестирования, в частности оконное приложение, приведенное на рисунке 1.7.

Таблица 2.1 — Форматы ошибок тестирования

Тип ошибки	Формат ошибки	Пример ошибки
завершение процесса по таймауту	`# <Имя функции>` `- error >> timeout error`	`# test_math_sin` `- error >> timeout error`
некорректный формат вывода тестового набора	`# <Имя функции>` `- error >> error`	`# test_math_sin` `- error >> wrong data format`

2.2 Интеграция модульных тестов в среду ФПП программирования

Расширяемость среды ФПП программирования обеспечивается за счет применения слоистой архитектуры. Распределение некоторых модулей по слоям схематично показано на рисунке 2.1. Цветом выделены модули, созданные в рамках выполнения настоящей работы.

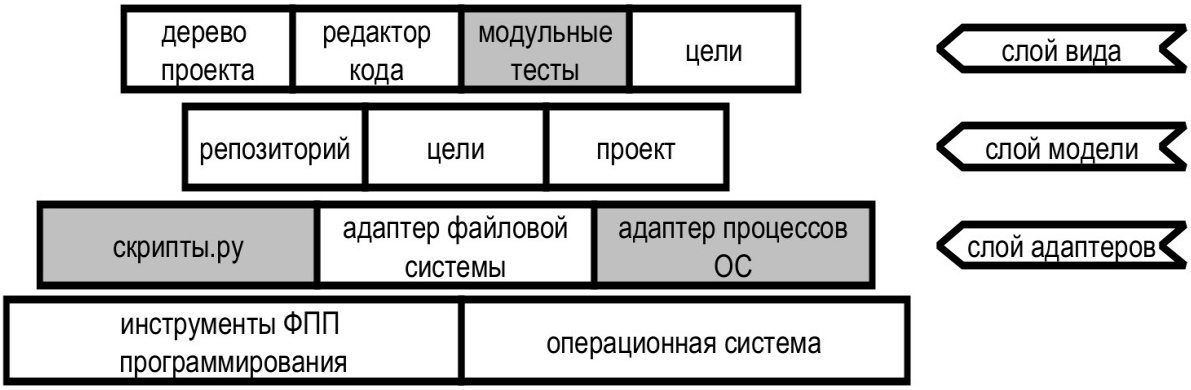

Рисунок 2.1 — Слои и модули среды разработки ФПП программ

На нижнем слое находятся средства операционной системы и созданные ранее инструменты ФПП программирования. К средствам операционной системы тут относятся механизмы работы с файловой системой, потоками и процессами.

Слой адаптеров/фасадов облегчает перенос системы между операционными системами. Помимо скриптов тут созданы адаптеры для удобной работы с файловой системой и запуска процессов, исполняющих скрипты на языке *Python*. За счет инкапсуляции логики работы с отдельными

инструментами ФПП программирования и изоляции платформозависимого кода, модули этого слоя позволяют изменять эти инструменты без модификации кода *IDE*, расположенного на верхних слоях.

Модели создаются для изоляции пользовательского интерфейса (вида) от способа получения данных и форматов их хранения. Например, модель проекта позволяет работать с файлами исходного кода. Модели не обращаются напрямую к инструментам ФПП программирования или системным функциям, вместо этого они используют адаптеры. Для модульных тестов нет отдельной модели, потому что тесты хранятся в репозитории и не нуждаются в дополнительных данных.

Для работы со скриптом был создан класс *ScriptExecutor* позволяющий исполнять скрипты на языке *Python* и считывать их стандартный поток вывода. На рисунке 2.1 он обозначен как «Адаптер процессов ОС». Работа со скриптами была реализована с помощью класса *QProcess* библиотеки *Qt*.

Для того чтобы наглядно продемонстрировать функциональность встраиваемого виджета модульного тестирования и взаимодействие новых и созданных ранее модулей при выполнении тестов из интегрированной среды разработки ФПП программ, на рисунках 2.2 и 2.3 приведены диаграмма прецедентов и диаграмма последовательности соответственно.

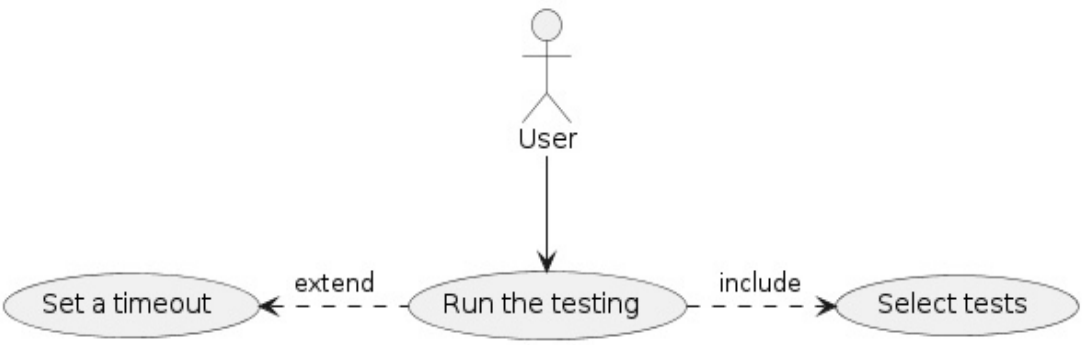

Рисунок 2.2 — Диаграмма показывающая набор действий, доступных пользователю при работе с виджетом модульного тестирования

Скрипт модульного тестирования ожидает, что в текущем каталоге или в каталогах, указанных в переменной окружения *PATH* будут находиться репозиторий и интерпретатор. *IDE* позволяет настроить путь к инструментам разработки, а репозиторий размещается всегда в каталоге открытого проекта. Перед запуском модульных тестов эти пути подставляются в переменную *PATH*.

Набор открытых и закрытых функций созданных и зависимых модулей показан на рисунке 2.4. Используется нотация диаграммы классов, хотя часть модулей реализована в процедурном стиле. Программная реализация фреймворка выполнена в соответствии с этой схемой. По схеме можно проследить распределение ответственности между модулями и ключевые публичные функции этих модулей. Исходный код размещен в репозитории [13].

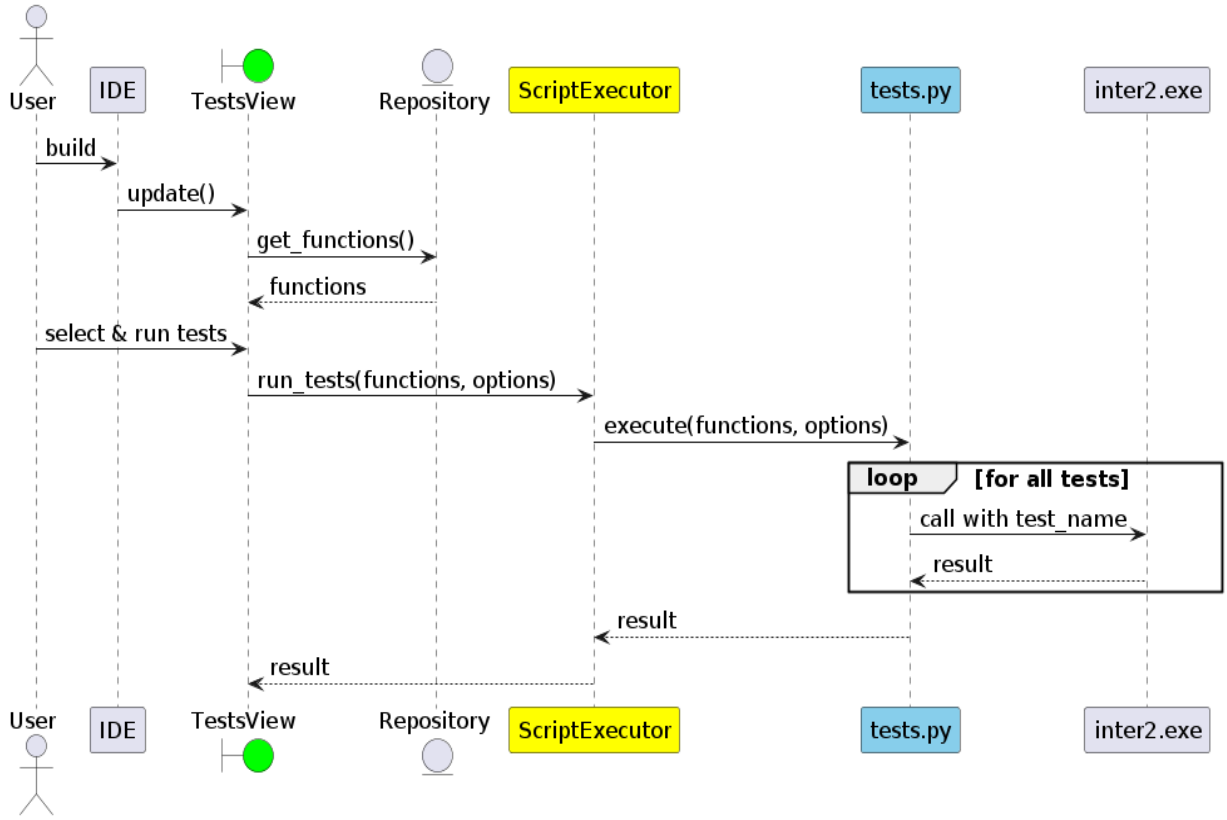

Рисунок 2.3 — Взаимодействие созданных модулей в процессе исполнения тестов

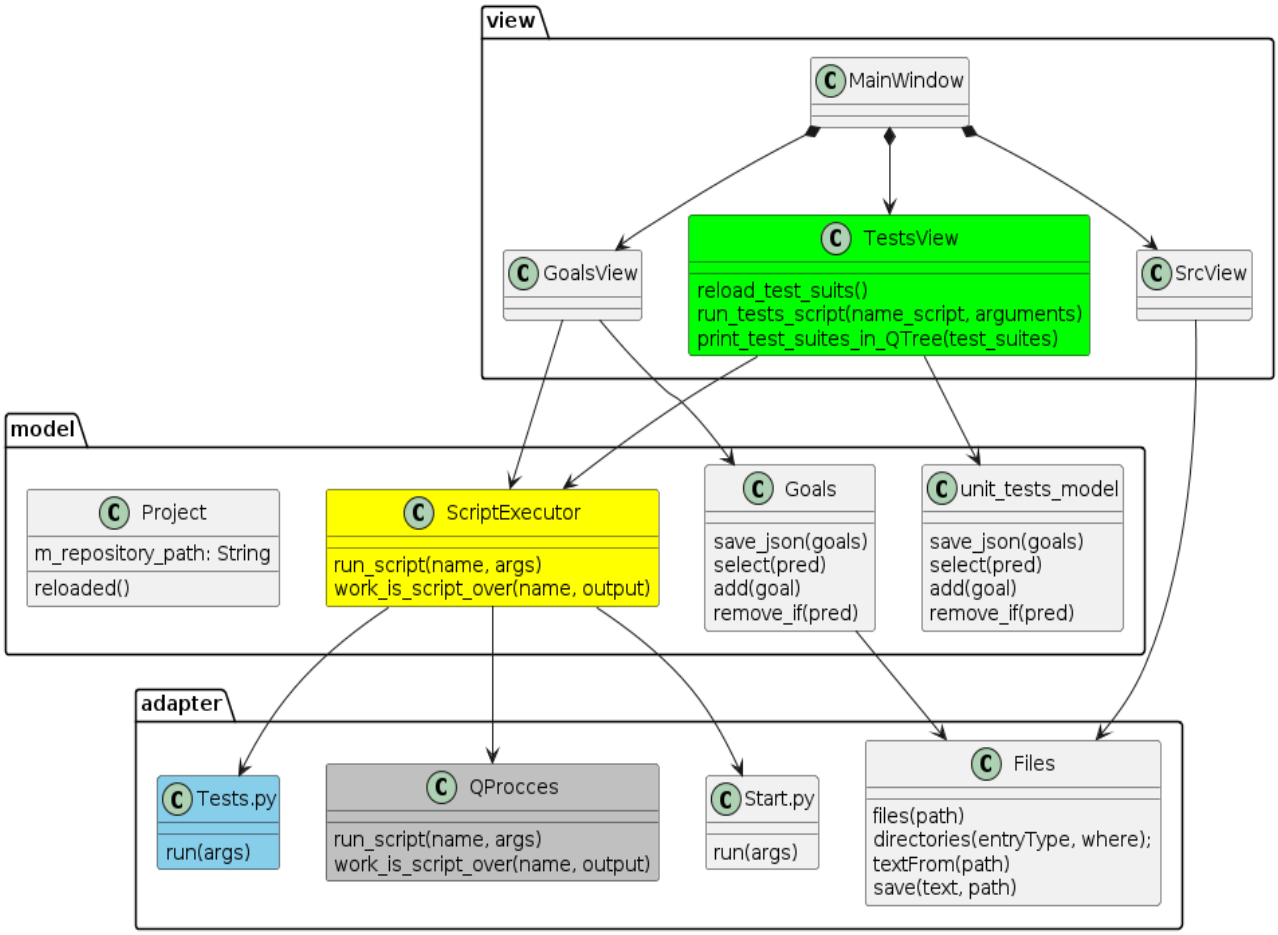

Рисунок 2.4 — Модули *IDE* и интегрированного фреймворка модульного тестирования

2.3 Выводы по главе

На основе требований, приведенных в первой главе, выполнено проектирование. В результате предложен вариант декомпозиции системы на модули, учитывающий необходимость интеграции фреймворка в среду разработки ФПП программ. Выполнена программная реализация системы.

3 Тестирование и особенности использования инструментов модульного тестирования

3.1 Сборка. Требования к инструментам

Утилита, выполняющая запуск тестов оформлена в виде консольного приложения на Python. Для ее успешной работы необходимо разместить в одном каталоге со скриптами собранные интерпретатор, генератор УГ и транслятор (*inter2.exe, cgen2.exe, trans2.exe*).

Оконное приложение, выполняющее тесты интегрировано в IDE, поэтому для его сборки необходимо скомпилировать среду разработки. Разработан файл проекта, автоматизирующий сборку с использованием системы *qmake*. Для сборки проекта из консоли необходимо:

а) установить библиотеку *Qt* версии не ниже 5.14 и компилятор *C++* с поддержкой стандарта *C++17*;

б) добавить в переменную среды *PATH* пути к *qmake* и компилятору *C++*;

в) выполнить в каталоге с файлом проекта (*ide2324.pro*) команды:

```
qmake.exe -spec win32-g++
```

Тут опция *-spec* задает информацию об используемом компиляторе. В результате выполнения команды в текущем каталоге будет создан *Makefile*.

```
mingw32-make.exe qmake_all
```

При успешном выполнении сборки в каталоге *..\..\bin* будет создан файл *ide2324.exe*.

Для сборки *IDE* со всеми указанными выше зависимостями используется файл проекта *make-min.pro*. Процесс сборки при этом аналогичен описанному выше. Результаты сборки также будут размещены в каталоге *..\..\bin*.

3.2 Установка. Требования к системе

Для корректной работы приложения требуется установить и прописать в *PATH* пути к *Python* версии не ниже 3.5.

Для работы *IDE* с интегрированными модульными тестами требуется набор *DLL* библиотек, которые размещаются в одном каталоге с исполняемым файлом. Для их автоматического поиска в системе и копирования в каталог используется утилита *windeployqt*, являющаяся элементом фреймворка *Qt*. Необходимо дописать путь к ней в переменную среды *PATH* и использовать ее следующим образом:

а) перейти в каталог с файлом *ide.exe*.

б) выполнить команду:

```
windeployqt.exe *.exe
```

в) перейти в каталог с библиотекой *Qt*, выполнить команду:

```
cp libstdc++-6.dll libgcc_s_seh-1.dll path_to_ide
```

Скомпилированная и собранная среда разработки программ на языке Пифагор для операционной системы Windows доступна на хостинге [14].

3.3 Проверка корректности работы утилит

На рисунке 3.1 приведен снимок окна среды разработки с интегрированным модулем тестирования. Окно тестирования встроено в виджет, задающий способы запуска программы, в отдельную вкладку.

На рисунке 3.2 приведены тесты, созданные для функции умножения матриц, показанной на рисунке 1.3, а также тесты, написанные с ошибками для проверки реакции на них приложения - *test_error* и *test_timeout_error* проверяющие неверный вывод и работу таймаута.

Для более полной проверки работоспособности созданных инструментов разработаны модульные тесты к математической библиотеке [15] и библиотеке

обработки строк [16] языка Пифагор. Исходный код библиотек и тестов к ним доступен в репозитории [17].

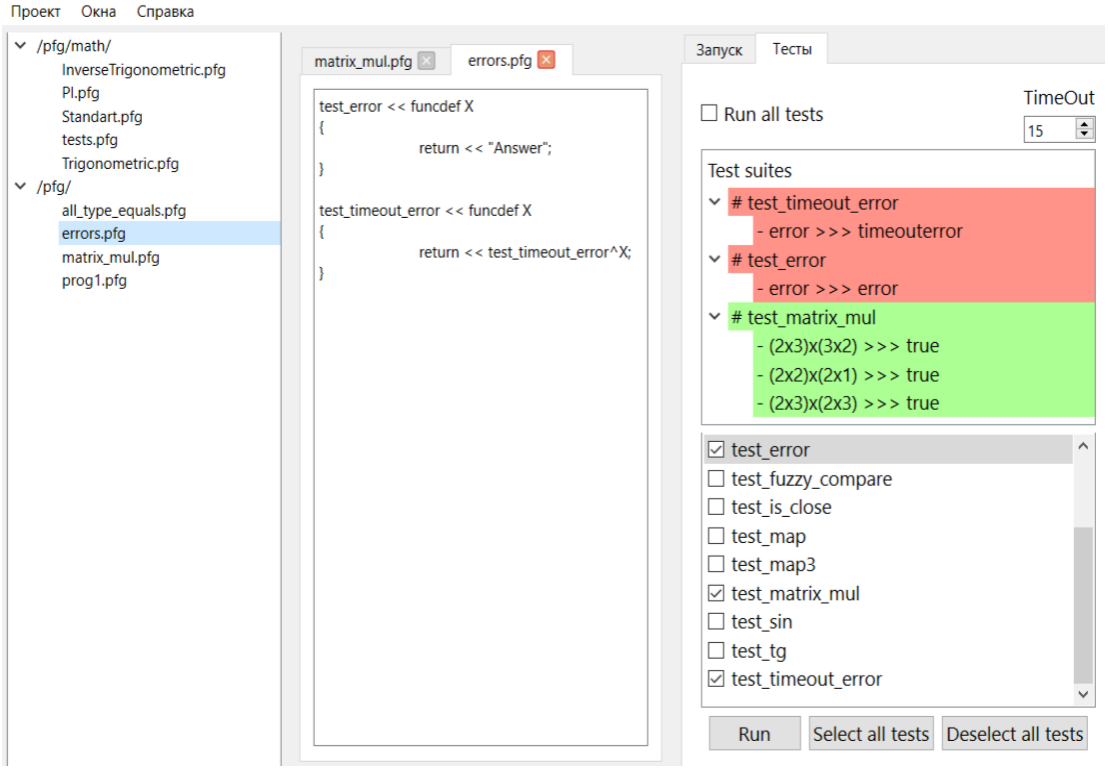

Рисунок 3.1 — *IDE* со встроенными модульными тестами

```
1 test_error << funcdef X {
2   return << "Answer";
3 }
4
5 test_timeout_error << funcdef X {
6   return << test_timeout_error^X;
7 }
8
9 test_fuzzy_compare << funcdef {
10   Cases << [
11     ("int", ( (1, 1):fuzzy_compare, true):equals),
12     ("int equal float", ( (1, 1.00000001)
13           :fuzzy_compare, true):equals),
14     ("float equal float", ( (1.1, 1.09999999)
15           :fuzzy_compare, true):equals),
16     ("int not equal float", ( (1, 1.1)
17           :fuzzy_compare, false):equals),
18     ("float not equal float", ( (1.2, 1.1)
19           :fuzzy_compare, false):equals)
20   ];
21   return << Cases;
22 }
```

Рисунок 3.2 — Фрагмент кода с тестами на ФПП языке Пифагор

Разработанные тестовые случаи описаны в таблице 3.1, исходный код тестов размещен в репозитории приведен в приложении А. В ходе тестирования выявлен ряд ошибок: функции *tg* и *ctg* из библиотеки *math* не учитывают точки неопределённости, функции *strcat, strlistcat, indchr, indchrend, indschr* и *amountchr* из библиотеки *string* некорректно работают с пустыми строками.

Таблица 3.1 — Тестирование библиотек

Функция	**Тестовый случай**	**Ожидаемый результат**	**Результат тестирования**
sin	1	0.8414709848078965	Верно
	-1	-0.8414709848078965	Верно
	0.5	0.479425538604203	Верно
	0	0	Верно
	7	0.6569865987187891	Верно
cos	1	0.5403023058681398	Верно
	-1	0.5403023058681398	Верно
	0.5	0.8775825618903728	Верно
	0	1	Верно
	7	0.7539022543433046	Верно
tg	1	1.5574077246549023	Верно
	-1	-1.5574077246549023	Верно
	0.5	0.5463024898437905	Верно
	0	0	Верно
	π/2	Нельзя вычислить	Неверно, возвращает -100799841.626418903470, вследствие погрешности *
ctg	1	0.6420926159343306	Верно
	-1	-0.6420926159343306	Верно
	0.5	1.830487721712452	Верно
	0	Нельзя вычислить	Возвращает ZERODIVIDE - ошибку деления на 0 *
abs	1	1	Верно
	-1	1	Верно
	0	0	Верно
	-0.5	0.5	Верно

Продолжение таблицы 3.1

Функция	Тестовый случай	Ожидаемый результат	Результат тестирования
powint	(3,4)	81	Верно
	(0.5,3)	0.125	Верно
	(10,-3)	0.001	Верно
	(0.5,-3)	8	Верно
strcat	("abc","def")	"abcdef"	Верно
	("", "def")	"def"	Интерпретатор завершает работу с ошибкой и не возвращает результат **
	("abc","")	"abc"	Неверно, возвращает ('a','b','c',BASEFUNCERROR), происходит ошибка при вычислении *
strlistcat	("abc")	"abc"	Верно
	("abc","def")	"abcdef"	Верно
	("abc","def","g")	"abcdefg"	Верно
	("","def","g")	"defg"	Интерпретатор завершает работу с ошибкой и не возвращает результат **
	("abc","","g")	"abcg"	Интерпретатор завершает работу с ошибкой и не возвращает результат **
	("abc","def","")	"abcdef"	Неверно ('a','b','c','d','e','f',BASEFUNCERROR), происходит ошибка при вычислении *
	("abc","de","f","g")	"abcdefg"	Верно
indchr	("abcde",'a')	1	Верно
	("abcde",'c')	3	Верно
	("abcde",'e')	5	Верно
	("abcde",'f')	0	Верно
	("abcabc",'a')	1	Верно
	("",'a')	0	Интерпретатор завершает работу с ошибкой и не возвращает результат **

Окончание таблицы 3.1

Функция	Тестовый случай	Ожидаемый результат	Результат тестирования
indchrend	("abcde",'a')	1	Верно
	("abcabc",'a')	4	Верно
	("abcabc",'d')	0	Верно
	("",'a')	0	Интерпретатор завершает работу с ошибкой и не возвращает результат **
indschr	("abcde",'a')	(1)	Верно
	("abcabca",'a')	(1,4,7)	Верно
	("abcabca",'d')	(.)	Верно
	("",'a')	(.)	Интерпретатор завершает работу с ошибкой и не возвращает результат **
amountchr	("R",'T')	0	Верно
	("T",'T')	1	Верно
	("TTT",'T')	3	Верно
	("TaTabcTaTaataaT",'T')	5	Верно
	("",'T')	0	Интерпретатор завершает работу с ошибкой и не возвращает результат **
delchr	("abcd",'a')	"bcd"	Верно
	("abcada",'a')	"bcd"	Верно
	("bcd",'a')	"bcd"	Верно
	("",'a')	""	Интерпретатор завершает работу с ошибкой и не возвращает результат **

Тесты помеченные звездочками не добавлены в приложение А.

* нельзя задать ожидаемое значение. Например, функция вычисления тангенса не возвращает значение, которое можно интерпретировать как ошибку. Для исправления тестовых случаев необходимо переписать код функций на языке Пифагор.

** при выполнении тестов аварийно завешается работа интерпретатора, фреймворк модульного тестирования при этом сохраняет работоспособность, однако "ошибочными" будут считаться все тестовые случаи тестового набора.

3.4 Выводы по главе

Описаны тестирование и особенности использования инструментов модульного тестирования, включая инструкции для программиста по сборке системы. Инструкции пользователя не создавались так как в качестве них может использоваться спецификация требований, приведенная в первой главе работы.

В рамках тестирования созданного программного обеспечения разработаны тестовые наборы для существующих модулей на функционально-потоковом языке программирования Пифагор. Правильная работа фреймворка на достаточно большом наборе тестов подтверждает корректность созданного программного обеспечения.

ЗАКЛЮЧЕНИЕ

В рамках работы решена задача инструментальной поддержки процесса модульного тестирования функционально-потоковых программ:

а) предложен подход к разработке тестов;

б) разработан фреймворк модульного тестирования в виде скрипта на языке *Python*;

в) создано оконный интерфейс к фреймворку в виде *Qt*-виджета на языке *C++,* выполнена его встраивание в интегрированную среду разработки функционально-потоковых языков программирования.

В результате доклада работы на конференциях заняты 1 и 3 места. Дипломы приведены в приложение.

В рамках дальнейших работ возможно:

а) создание системы оценки качества тестового покрытия [18]. Для этого необходимо выполнить доработку интерпретатора функционально-потоковых параллельных программ – ввести в него режим исполнения программ, помечающий вершины реверсивного информационного графа после их использования;

б) адаптация системы тестирования для языка Smile использующего, в отличии от Пифагор, строгую типизацию [19]. Этот язык ориентирован на компиляцию программ, поэтому фреймворк модульного тестирования для этого языка должен собирать тестовые наборы и формировать из них точку входа.

СПИСОК СОКРАЩЕНИЙ

IDE – (integrated development environment) интегрированная среда разработки;

ОС – операционная система;

ПО – программное обеспечение;

РИГ – реверсивный информационный граф;

УГ – управляющий граф;

ФПП – функционально потоковой параллельной.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мартин, Р. Чистый код. Создание, анализ и рефакторинг. Библиотека программиста. – Санкт-Петербург, 2014. – 464 с.

2. Легалов, А. И. Функциональный язык для создания архитектурнонезависимых параллельных программ / А. И. Легалов // Вычислительные технологии. – 2005. – Т. 10. – № 1. – С. 71-89.

3. Деннис, Д. Б. Схемы потока данных / Д. Б. Деннис, Д. Б. Фоссин, Д. П. Линдерман // Теория программирования. – 1972. – Т. 2. – С. 7-43.

4. Легалов, А. И. Инструментальная поддержка создания и трансформации функционально-потоковых параллельных программ / А. И. Легалов, В. С. Васильев, И. В. Матковский, М. С. Ушакова // Труды Института системного программирования РАН. – 2017. – Т. 29, № 5. – С. 165–184.

5. Легалов, А. И. Особенности хранения функционально-потоковых параллельных программ / А. И. Легалов, И. В. Матковский, А. В. Анкудинов // Сибирский журнал науки и технологий. – 2013. – № 4. – С. 53-57.

6. Легалов, А. И. Инструментальная поддержка эволюционного расширения программ средствами процедурно-параметрического программирования / А. И. Легалов, П. В. Косов // Решетневские чтения. 2015. №19. URL: https://cyberleninka.ru/article/n/instrumentalnaya-podderzhka-evolyutsionnogo-rasshireniya-programm-sredstvami-protsedurno-parametricheskogo-programmirovaniya (дата обращения 16.05.2024).

7. Макгрегор, Д. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие: пер. с англ./Д. Макгрегор, Д. Сайкс: ООО «ТИД "ДС"». – 2002. – 432 с.

8. Качалов, С. К. Методика создания средства автоматического тестирования кроссплатформенного программного обеспечения / С. К. Качалов // Наукосфера. – 2021. – № 4-2. – С. 129-134. – EDN QEZZJU.

9. Loder, W. Erlang and Elixir for Imperative Programmers / W. Loder. – Apress, 2016.

10. pytest Documentation, Release 0.1. – 2023. – URL: https://buildmedia.readthedocs.org/media/pdf/pytest/latest/pytest.pdf (дата обращения 16.11.2023)

11. Васильев, В. С. Сравнение двух значений (любого типа) на Пифагор. Блог программиста. – URL: https://pro-prof.com/forums/topic/equals_pifagor (дата обращения 16.11.2023)

12. Функциональный язык параллельного программирования «Пифагор». – 2004. – URL: http://www.softcraft.ru/parallel/fpp/fpp02/part02.pdf (дата обращения 16.11.2023)

13. Репозиторий с исходным кодом программ. – URL: https://gitflic.ru/project/fpp/pifagor_tools_fork?branch=unit_test_framework (дата обращения 16.05.2024)

14. Сайт с скомпилированной и собранной средой разработки программ на языке Пифагор для операционной системы Windows. – URL: https://gitflic.ru/project/fpp/pifagor_tools_fork/release/53825713-7493-46ed-9025-8bfd8014959a (дата обращения 16.11.2023)

15. Удалова, Ю. В. Библиотека математических функций для языка функционально-потокового параллельного программирования Пифагор / Ю. В. Удалова // Вестник Бурятского государственного университета. Математика, информатика. – 2019. – № 4. – С. 57-64. – DOI 10.18101/2304-5728-2019-4-57-64. – EDN NTIXAY.

16. Удалова, Ю. В. Библиотека обработки строк для языка функционально-потокового параллельного программирования Пифагор /

Ю. В. Удалова // Международный научно-исследовательский журнал. – 2020. – № 12-1(102). – С. 83-87. – DOI 10.23670/IRJ.2020.102.12.014. – EDN JZVEOH.

17. Репозиторий с примерами программ на языке Пифагор. – URL: https://gitflic.ru/project/fpp/fpp_sources (дата обращения 16.05.2024)

18. Сарычева, Ю. Ю. Обзор инстументов для измерения тестового покрытия кода / Ю. Ю. Сарычева, Ю. С. Белов // E-Scio. – 2023. – № 4(79). – С. 72-81. – EDN TTNVLU.

19. Легалов, А. И. Поддержка статической типизации в функционально-потоковой модели параллельных вычислений / А. И. Легалов, Н. К. Чуйкин // Параллельные вычислительные технологии (ПаВТ'2023): Короткие статьи и описания плакатов. Материалы XVII всероссийской научной конференции с международным участием, Санкт-Петербург, 28–30 марта 2023 года. – Челябинск: Издательский центр ЮУрГУ, 2023. – С. 173-185. – EDN JTYATB.

ПРИЛОЖЕНИЕ А

Примеры описания тестов

```
test_sin << funcdef x {
  Cases << [
    ("sin(1)", (1:sin, 0.8414709848078965, fuzzy_compare):equals),
    ("sin(-1)", (-1:sin, -0.8414709848078965, fuzzy_compare):equals),
    ("sin(0.5)", (0.5:sin, 0.479425538604203, fuzzy_compare):equals),
    ("sin(0)", (0:sin, 0.0, fuzzy_compare):equals),
    ("sin(7)", (7:sin, 0.6569865987187891, fuzzy_compare):equals)
  ];
  return << Cases;
}

test_cos << funcdef x {
  Cases << [
    ("cos(1)", (1:cos, 0.5403023058681398, fuzzy_compare):equals),
    ("cos(-1)", (-1:cos, 0.5403023058681398, fuzzy_compare):equals),
    ("cos(0.5)", (0.5:cos, 0.8775825618903728, fuzzy_compare):equals),
    ("cos(0)", (0:cos, 1.0, fuzzy_compare):equals),
    ("cos(7)", (7:cos, 0.7539022543433046, fuzzy_compare):equals)
  ];
  return << Cases;
}

test_tg << funcdef x {
  Cases << [
    ("tg(1)", (1:tg, 1.5574077246549023, fuzzy_compare):equals),
    ("tg(-1)", (-1:tg, -1.5574077246549023, fuzzy_compare):equals),
    ("tg(0.5)", (0.5:tg, 0.5463024898437905, fuzzy_compare):equals),
    ("tg(0)", (0:tg, 0.0, fuzzy_compare):equals)
  ];
  return << Cases;
}

test_ctg << funcdef x {
  Cases << [
    ("ctg(1)", (1:ctg, 0.6420926159343306, fuzzy_compare):equals),
    ("ctg(-1)", (-1:ctg, -0.6420926159343306, fuzzy_compare):equals),
    ("ctg(0.5)", (0.5:ctg, 1.830487721712452, fuzzy_compare):equals)
  ];
  return << Cases;
}

test_abs << funcdef x {
  Cases << [
    ("abs(1)", (1:abs, 1):equals),
    ("abs(-1)", (-1:abs, 1):equals),
    ("abs(0)", (0:abs, 0):equals),
    ("abs(-0.5)", (-0.5:abs, 0.5):equals)
  ];
  return << Cases;
}

test_powint << funcdef x {
  Cases << [
    ("int and positive power",
       ((3,4):powint, 81.0, fuzzy_compare):equals),
    ("float and positive power",
       ((0.5,3):powint, 0.125, fuzzy_compare):equals),
    ("int and negative power",
       ((10,-3):powint, 0.001, fuzzy_compare):equals),
    ("float and negative power",
       ((0.5,-3):powint, 8.0, fuzzy_compare):equals)
  ];
  return << Cases;
```

```
64 }
65 test_strcat << funcdef x {
66   Cases << [
67     ("2 not empty strings",
68       (("abc","def"):string.strcat, "abcdef"):equals)
69   ];
70   return << Cases;
71 }
72
73 test_strlistcat << funcdef x {
74   Cases << [
75     ("1 not empty strings",
76        (("abc"):string.strlistcat, "abc"):equals),
77     ("2 not empty strings",
78       (("abc","def"):string.strlistcat, "abcdef"):equals),
79     ("3 not empty strings",
80       (("abc","def","g"):string.strlistcat, "abcdefg"):equals),
81     ("4 not empty strings",
82       (("abc","de","f","g"):string.strlistcat, "abcdefg"):equals)
83   ];
84   return << Cases;
85 }
86
87 test_indchr << funcdef x {
88   Cases << [
89     ("element in begin",
90                 (("abcde",'a'):string.indchr, 1):equals),
91     ("element in string",
92                 (("abcde",'c'):string.indchr, 3):equals),
93     ("element in end",
94                 (("abcde",'e'):string.indchr, 5):equals),
95     ("element is not",
96                 (("abcde",'f'):string.indchr, 0):equals),
97     ("multiple elements",
98                 (("abcabc",'a'):string.indchr, 1):equals)
99   ];
100   return << Cases;
101 }
102
103 test_indchrend << funcdef x {
104   Cases << [
105     ("1 element", (("abcde",'a'):string.indchrend, 1):equals),
106     ("multiple elements",
107       (("abcabc",'a'):string.indchrend, 4):equals),
108     ("element is not",
109       (("abcabc",'d'):string.indchrend, 0):equals)
110   ];
111   return << Cases;
112 }
113
114 test_indschr << funcdef x {
115   Cases << [
116     ("1 element",
117        (("abcde",'a'):string.indschr, (1)):equals),
118     ("multiple elements",
119        (("abcabca",'a'):string.indschr, (1,4,7)):equals)
120   ];
121   return << Cases;
122 }
123
124 test_amountchr << funcdef x {
125   Cases << [
126     ("no element",
127        (("R",'T'):string.amountchr, 0):equals),
128     ("one element",
129        (("T",'T'):string.amountchr, 1):equals),
130     ("several elements",
131        (("TTT",'T'):string.amountchr, 3):equals),
132     ("string with several elements",
```

```
133         (("TaTabcTaTaataaT",'T'):string.amountchr, 5):equals)
134   ];
135   return << Cases;
136 }
137
138 test_delchr << funcdef x {
139   Cases << [
140     ("one element",
141        (("abcd",'a'):string.delchr, "bcd"):equals),
142     ("several elements",
143        (("abcada",'a'):string.delchr, "bcd"):equals),
144     ("no element",
145        (("bcd",'a'):string.delchr, "bcd"):equals)
146   ];
147   return << Cases;
148 }
```

ПРИЛОЖЕНИЕ Б

Результаты апробации работы

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ДОНЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ДИПЛОМ

III СТЕПЕНИ

НАГРАЖДАЕТСЯ

Дудинков

Данил Михайлович

студент федерального государственного автономного образовательного учреждения высшего образования «Сибирский федеральный университет»

победитель II Международной научно-практической конференции студентов, аспирантов и молодых ученых «Кибернетика, информатика, аналитика: модели, инструменты, методы»

Секция «Системный анализ и принятие решений в условиях новых вызовов»

Председатель оргкомитета
Конференции, проректор А.В. Белый

Приказ ФГБОУ ВО «ДонГУ» №213-08 от 16.05.2024г.

г. Донецк, 2024 г.

СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ | SIBERIAN FEDERAL UNIVERSITY

ДИПЛОМ

I степени

награждается

Дудинков Данил Михайлович

за доклад на тему:

Среда разработки функционально-потоковых параллельных программ,

успешно представленный на юбилейной XX Международной научной конференции студентов, аспирантов и молодых ученых «Проспект Свободный - 2024»

Научный руководитель: канд. техн. наук, доцент
В. С. Васильев

Проректор по научной работе
Р. А. Барышев

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«17» 06 2024 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Фреймворк модульного тестирования для функционально-потоковых параллельных программ

Руководитель	подпись	17.06.24 дата	доцент, канд. техн. наук должность, ученая степень	В.С. Васильев
Выпускник	подпись	10.06.24 дата		Д.М. Дудинков
Нормоконтролёр	подпись	17.06.24 дата	доцент, канд. техн. наук должность, ученая степень	В.С. Васильев

Красноярск 2024

РЕФЕРАТ

Выпускная квалификационная работа по теме «Фреймворк модульного тестирования для функционально-потоковых параллельных программ» содержит 37 страниц текстового документа, 13 рисунков, 2 таблицы, 19 использованных источников.

ТЕСТИРОВАНИЕ, МОДУЛЬНОЕ ТЕСТИРОВАНИЕ, ИНСТРУМЕНТЫ ТЕСТИРОВАНИЯ, ФУНКЦИОНАЛЬНО-ПОТОКОВОЕ ПАРАЛЛЕЛЬННОЕ ПРОГРАММИРОВАНИЕ.

Цель работы – создание фреймворка модульного тестирования функционально-потоковых параллельных программ. Исследований, связанных с реализацией инструментов модульного тестирования функционально-потоковых программ не проводилось, поэтому данная работа является актуальной.

В первой главе описаны особенности функционально потоковой парадигмы и существующих инструментов. Предложен подход к организации тестирования. Разработана спецификация требований к инструментам модульного тестирования.

Вторая глава содержит результаты проектирования и реализации инструментальных средств модульного тестирования ФПП программ.

В рамках третьей главы разработаны примеры программ на которых выполнялось тестирование; созданы модульные тесты к существующим комплектам ФПП программ; выполнена проверка работоспособности инструментов модульного тестирования; описаны особенности использования созданных инструментов.

В результате работы над ВКР был предложен подход к разработке тестов; создан набор инструментов модульного тестирования; произведена интеграция данных инструментов в интегрированную среду разработки функционально-потоковых языков программирования.

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
«___»_____2023 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Студенту Дудинкову Данилу Михайловичу
фамилия, имя, отчество

Группа КИ20-06Б Направление (специальность) 090301
номер Код

Информатика и вычислительная техника
Наименование

Тема выпускной квалификационной работы: Фреймворк модульного тестирования для функционально-потоковых параллельных программ

Утверждена приказом по университету № _____ от _____

Руководитель ВКР: В.С. Васильев, канд. техн. наук, доцент кафедры ВТ
инициалы, фамилия, учёная степень, должность, место работы

ИКИТ СФУ

Исходные данные для ВКР:

1) примеры программ на языке Пифагор

2)

3)

4)

5)

Перечень разделов ВКР:

1) Постановка задачи

2) Проектирование. Особенности реализации

3) Тестирование и особенности использования инструментов модульного тестирования

Перечень графического материала: презентация

Руководитель ВКР _____ В.С. Васильев
подпись инициалы, фамилия

Задание принял к исполнению _____ _____
подпись инициалы, фамилия

«20» 12 2023 г.
Дата