МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

Отчет по практической работе

«Система тестирования и анализа производительности алгоритмов»

ПО ДИСЦИПЛИНЕ «Методология разработки программного обеспечения»

Факультет: АВТ
Группа: АВТ-315
Студенты: Катунов Е.Ю.,
Юргина Т.С., Потапейко Д.А.,
Утбасарова Л.А., Ситников А.А,
Оленев Б.А, Анчугов Н. Е.,
Волков С. А., Ахременко А.В.,
Им Э.А. Васильев Д.В.,
Владыкин И.А.

Преподаватель: Романов Е.Л.

Новосибирск 2017

Оглавление

Постановка задачи

Цель: разработать систему тестирования и анализа производительности алгоритмов (программ).

Приложение для рабочего стола позволяет создавать тестовые наборы и тестировать по ним программы (алгоритмы) на предмет производительности, отсутствия ошибок (зависаний, исключений) и соответствия тестовому результату. Вывод и сравнительный анализ результатов в графической форме, подбор зависимости. Генерация и сохранение тестовых наборов и результатов выполнения в файлах. Исполнение алгоритм (программа) оформляется в виде метода класса, в котором она может использовать тестовые наборы данных и датчики событий. Разработка командного языка для создания и настройки объектов и их передачи между модулями программы.

1. Проектирование

1.1. Алгоритм работы приложения

Предполагаемый алгоритм работы приложения:

1. Генерируется случайный набор данных определенного типа;
2. Над созданными данными выполняется алгоритм;
3. Во время выполнения алгоритма детекторы фиксируют значения некоторых метрик;
4. Шаги 1-3 повторяются некоторое количество раз; при этом, размерность набора данных увеличивается с каждым повтором.
5. Результаты снятия метрик изображаются в виде графиков; на каждом графике отображается значение размерности массива данных и значение метрики.
6. Выбирается одна из нескольких доступных аппроксимирующих функций и вычисляется среднее значение аппроксимации. Аппроксимирующая функция также выводится на графике.

1.2. Глоссарий

Курсивом выделены определения, которые также можно найти в глоссарии.

Элемент данных - единица данных, обрабатываемых алгоритмом. Набор таких данных генерируется на шаге 1. Каждый элемент обладает определенным *типом.*

Тип данных (I_DataElement) - тип *элементов данных*.

Тестовый набор (TestCase) - набор *единиц данных*, обрабатываемых алгоритмом. Генерируется на шаге 1, обрабатывается алгоритмом.

Тестовый план (TestPlan) - план тестирования; содержит вектор тестовых наборов. Содержит в себе данные для выполнения тестирований на шаге 4.

Генератор размерностей (I_DimensionGenerator) - генератор размерностей для *тестового плана.* Генерирует размерности тестовых наборов, которые используются в *тестовом плане* (шаг 4).

Генератор последовательностей (I_SequenceGenerator) – генератор случайных *элементов данных.* Генерирует значения по одному.

Единичный результат (OneResult) - результат выполнения алгоритма над одним *тестовым набором* с выбранным типом *детектора.*

Результат (TestPlanResult) - результат выполнения тестового плана. Содержит набор *единичных результатов*.

Детектор *(*Detector*)* - детектор, содержащий набор *измерений*, снятых во время алгоритма (шаг 3).

Измерение (I_DetectorValue) - единичное значение некоторой метрики, снятой во время выполнения одного алгоритма.

Общесистемные интерфейсы

Именуемый объект (I_Name) - объект, обладающий именем. Имя используется при построении визуального интерфейса. Например, объект типа "Детектор" может обладать именем "Время выполнение".

Имя типа объекта (I_TypeName) - название типа объекта. Имя используется при построении визуального интерфейса. Например, "Детектор", "Алгоритм" и т.д.

Имя объекта (I_ObjectName) - агрегирует интерфейсы *именнуемого объекта* (I_Name) и *имени типа объекта* (I_TypeName). Позволяет задать имя объекта.

Сохраняемый тип (I_File) - объект, который можно сохранить через поток данных и восстановить из потока данных.

Фабрики

Фабрики агрегируют все существующие объекты приложения, принадлежащие к определенному интерфейсу, и позволяют предоставить доступ к этим объектам по именам. Используются при построении визуального интерфейса.

Фабрика (I_NameFactory) - общий интерфейс для фабрик, создающих некоторые объекты по заданному имени.

Фабрика типов (TypeFactory) - фабрика для агрегации объектов по *типам*. К типам относятся понятия, такие как “Детектор”, “Алгоритм” и проч.

1.3. Архитектурные требования

Приложение должно поддерживать функциональную расширяемость по следующим элементам**:**

1. Типы генерируемых данных и виды генераторов данных (например, различные виды случайного распределения).
2. Виды алгоритмов - алгоритм содержит в себе информацию о детекторах, который он поддерживает. Разные алгоритмы могут поддерживать одни и те же детекторы.
3. Виды значений, которые измеряет детектор.
4. Виды генераторов размерности. Генератор задает некоторое правило, по которому изменяется размерность массива случайных данных при каждом тестировании (например, арифметическая последовательность, геометрическая последовательность и проч.). Обладает набором параметров – изначальное значение, количество значений,

которое необходимо сгенерировать и некоторый параметр, смысл которого зависит от генератора.

5. Различные виды аппроксимирующих функций (линейная, экспоненциальная, и проч.).

2. Реализация

При реализации системы проект был разбит на пять основных модулей, каждый из которых разрабатывался определенными участниками команды. Список модулей, входящих в них пакетов, интерфейсов, и классов, а также распределение участников команды приведено в таблице 1. Диаграмма классов приводится в приложении 1.

Таблица 1. Распределение участников команды по модулям.

Модуль	Разрабатываемые пакеты и классы	Участники
Генерация данных	Пакет: *algoanalyst.testcase* Классы/Интерфейсы: *I_DataElement* *I_DimensionGenerator* *I_SequenceGenerator* *TestCase* *TestPlan*	Катунов Е.Ю., Юргина Т.С.
Алгоритм	Пакет: *algoanalyst.algorithm* *algoanalyst.result* Классы/Интерфейсы: *Algorithm* *I_DetectorValue* *Detector* *OneResult* *TestPlanResults* *CountDetectorValue* *TimeDetectorValue* *AlgorithmException* *BubbleSort* *MergeSort* *QuickSort*	Утбасарова Л.А., Потапейко Д.А.
Графики	Пакет: *algoanalyst.graphs* Классы/Интерфейсы: *Controller* *Graphic* *Qwer*	Им Э.А., Ахременко А.В.
Визуальный интерфейс, обработка исключений	Пакет: *algoanalyst.view* Классы/Интерфейсы: *Factory_MainAndAll* *Main_View* *Point_Algoritm* *Point_Generator* *Point_TypeData* *Point_Type_Float* *Point_TypeInt* *Point_TypeString*	Владыкин И.А., Васильев Д.В.

Модуль	Разрабатываемые пакеты и классы	Участники
Аппроксимация	Пакет: *algoanalyst.approx* Классы/Интерфейсы: *I_Function* *FunctiobLinear* *FunctionExponential* *FunctionLogLin* *FunctionSquare* *FunctionLogarithmic* *Approximate*	Анчугов Н.Е., Волков С.А.
Командный язык	Пакет: *algoanalyst.language* Классы/Интерфейсы: *DimensionGeneratorConverter* *SequenceGeneratorConverter* *CreateAlgorithmCommand* *CreateDimensionGeneratorCommand* *CreateSequenceGeneratorCommand* *CreateTestPlanCommand* *ExecutePlanCommand* *ShowCommand* *SimpleCommand* *TheTest* *CommandInterpreter* *Commands* *XmlParser*	Ситников А.А., Оленёв Б.А.

2.1. Описание программы

2.1.1. Пакет *algoanalyst.testcase.*

Таблица 2. Описание классов в пакете *algoanalyst.testcase.*

Класс	Описание
ExpDimentionGenerator	Генератор размерности тестовых наборов с экспоненциальным распределением. **Параметры генератора:** *int start* – начальная степень экспоненты. *int count* – количество тестовых наборов в тестовом плане. *double pow* – по умолчанию равно числу Эйлера (2,71828), но можно задать и другое значение. **Конструктор:** *public ExpDimentionGenerator()*- устанавливает значение параметра *pow* равным числу Эйлера. **Методы:** *void setParams(int start, int count, double pow) throws UNIException* – устанавливает параметры класса из входных значений. Функция также переопределена *void setParams(int*

Класс	Описание
	start, int count). *Vector<Integer> generate()* – возвращает вектор сгенерированных размерностей. *String getTypeName()* - возвращает тип распределения.
FibDimentionGenerator	Генератор размерности тестовых наборов определяемый последовательностью Фиббоначчи. **Параметры генератора:** *int count* - количество наборов. *Vector<Integer> vec* - вектор с размерностями наборов. **Конструктор:** *public FibDimentionGenerator ()* - устанавливает значения параметров в ноль. **Методы:** *void setParams(int start, int count, double step) throws UNIException* – устанавливает параметры класса из входных значений. Функция также переопределена void *setParams(int count).* *Vector<Integer> generate()* – возвращает вектор сгенерированных размерностей. *String getTypeName()* - возвращает тип распределения.
FloatRndFixSequencer	Генератор случайных чисел с плавающей запятой для тестового набора. Параметрами являются минимальное и максимальное значение интервала чисел. **Параметры секвенсора:** *max* - максимум диапазона генерации чисел. *min* - минимум диапазона генерации чисел. **Методы:** *I_DataElement getNext()* - генерирует возвращает следующий элемент последовательности. *void setParams(HashMap<String, String> inList)* - устанавливает параметры, необходимые для генерации последовательности - минимум и максимум диапазона генерации чисел. *String []getParamNames()* - возвращает в массиве имена параметров, необходимы для генерации чисел последовательности. *String getName()* – возвращает название последовательности. *String getTypeName()* - возвращает тип данных последовательности.
FloatTrendSequencer	Генерируют следующее число с плавающей точкой последовательности. **Параметры секвенсора:** *firstValue* - первое возвращаемое значение; *minStep, maxStep* - минимально и максимально возможное приращение от текущего числа для получения следующего, рекомендуется устанавливать числа одного знака; *range* - возможное отклонение от возрастающего/убывающего тренда **Типовые схемы применения:**

Класс	Описание
	minStep=maxStep, range=0 - линейно возрастающая/убывающая последовательность *minStep!=maxStep,* range=0 - случайно возрастающая/убывающая последовательность *minStep=maxStep, range>maxStep* - переменно возрастающая/убывающая последовательность. **Методы:** *I_DataElement getNext()* - возвращает следующий элемент последовательности. *void setParams(HashMap<String, String> inList)* - устанавливает параметры класса, необходимые для генерации последовательности. *String []getParamNames()* - возвращает в массиве имена параметров класса. *String getName()* – возвращает название последовательности. *String getTypeName()* - возвращает тип данных последовательности.
IntRndFixSequencer	Генератор случайных целых чисел в диапазоне, задаваемом параметрами “max” и “min”, для тестового набора. **Параметры секвенсора:** *max* - максимум диапазона генерации чисел. *min* - минимум диапазона генерации чисел. **Методы:** *I_DataElement getNext()* - генерирует возвращает следующий элемент последовательности. *void setParams(HashMap<String, String> inList)* - устанавливает параметры, необходимые для генерации последовательности - минимум и максимум диапазона генерации чисел. *String []getParamNames()* - возвращает в массиве имена параметров, необходимы для генерации чисел последовательности. *String getName()* – возвращает название последовательности. *String getTypeName()* - возвращает тип данных последовательности.
IntTrendSequencer	Генерируют следующее целое число последовательности. **Параметры секвенсора:** *firstValue* - первое возвраащемое значение; *minStep, maxStep* - миниамльно и максимально возможное приращение от текущего числа для получения следующего, рекомендуется устанавливать числа одного знака; *range* - возможное отклонение от возрастающего/убывающего тренда **Типовые схемы применения:** *minStep=maxStep, range=0* - линейно возрастающая/убывающая последовательность *minStep!=maxStep, range=0* - случайно возрастающая/убывающая последовательность *minStep=maxStep, range>maxStep* - переменно

Класс	Описание
	возрастающая/убывающая последовательность **Методы:** *I_DataElement getNext()* – генерирует и возвращает следующее число последовательности. *void setParams(HashMap<String, String> inList)* - устанавливает параметры класса. *String []getParamNames()* - возвращает строковый массив имен параметров, необходимых для создания последовательности. *String getName()* – возвращает название последовательности. *String getTypeName()* - возвращает тип данных последовательности.
LinearDimentionGenerator	Генератор размерности тестовых наборов с линейным распределением. **Параметры:** *int start* –длина первого набора данных. *int count* – количество набор. *double step* – число, на которое изменяется размерность каждого последующего набора. *Vector<Integer> vec* – вектор хранящий размерности тестовых наборов данных. **Конструктор:** *public LinearDimentionGenerator ()* - по умолчанию устанавливает все параметры в ноль. **Методы:** *void setParams(int start, int count, double step) throws UNIException* – устанавливает параметры класса из входных значений. *Vector<Integer> generate()* – возвращает вектор сгенерированных размерностей. *String getTypeName()* - возвращает тип распределения.
StringRndFixSequencer	Генератор случайной строки последовательности с возможностью задания длины строки. **Параметр:** *Len* – длина строки. **Методы:** *void setParams(HashMap<String, String> inList)* – принимает параметром длину генерируемой строки. *I_DataElement getNext()* – получает следующую строку последовательности. *String getName()* – возвращает название последовательности. *String getTypeName()* - возвращает тип данных последовательности.
StringTrendSequencer	Генератор строк тестового набора. **Параметры секвенсора:** Direction – направление возрастание убывание строки. Length – длина строки **Типовые схемы применения:** Direction>0 – возрастающие строки, то есть от «A» до «Z», при

Класс	Описание
	длине строки равной 1. Direction<=0 – убывающие строки, то есть от «Z» до «A», при длине строки равной 1. **Методы:** I_DataElement getNext() – возвращает следующую сгенерированную строку последовательности. void setParams(HashMap<String, String> parList) – задает параметры : длину и направление(возрастание/убывание) строки. *String getName()* – возвращает название последовательности. *String getTypeName()* - возвращает тип данных последовательности. *void setParams(HashMap<String, String> parList)*- устанавливает значения параметров класса. *String[] getParamNames()* – получает строковый массив названий параметров, необходимых для создания последовательности.
TestCase	Тестовый набор. **Параметры:** *Vector <I_DataElement> testData* – вектор, содержащий элементы данных и представляющий собой тестовый набор. **Конструктор:** *public TestCase(int size, I_SequenceGenerator gen)* – в конструкторе с параметрами (вызывается метод генерации значений набора) генерируется набор значений.Принимает размер –size и генератор последовательности – gen, при помощи которого получает новые значения в набор. **Методы:** *List<I_DataElement> getTestData()* - возвращает тестовый набор в виде списка. *void load(DataInputStream in) throws UNIException* – выполняет загрузку набора в файл. *void save(DataOutputStream out) throws UNIException* – загружает данные из потока в тестовый набор *(testData)*
TestPlan	Тестовый план. **Параметры:** *Vector<TestCase> testCases* – хранит вектор наборов. *Vector<Integer> len* – хранит вектор размерностей наборов. **Конструктор:** *public TestPlan(String name, I_SequenceGenerator sequenceGen, I_DimensionGenerator dimenGen)* - в конструкторе параметрами передаются название плана (может и не быть), объект генератора последовательности и генератора размерности наборов. Здесь заполняются параметры плана. **Методы:** *Vector<TestCase> getTestCases()* – возвращает вектор, содержащий наборы данных, то есть тестовый план. *void save(DataOutputStream out) throws UNIException* – сохраняет данные в поток. *void load(DataInputStream in) throws UNIException* – загружает данные из потока в параметры (переопределяя их или заполняя).

Класс	Описание
	String getName() – возвращает имя набора. *void setObjectName(String Name)* – устанавливает имя объекта строкой из параметра функции. *String getTypeName()* – возвращает имя типа – «тестовый план».

2.1.2. Пакеты *algoanalyst.algirithm, algoanalyst.result*

Также в таблице 3 приведено описание общесистемных интерфейсов (методы).

Формальное описание

Пакет алгоритм (algorithm) – пакет содержащий, родительский класс (A_Algorithm) для всех алгоритмов. Также в данном пакете описаны действующие в программе алгоритмы: сортировка пузырьком (BubbleSort), быстрая сортировка(QuickSort) и сортировка слиянием (MergeSort).

Помимо алгоритмов в данном пакете описаны детекторы, общий класс — Detector и два класса имплементирующиеся от интерфейса (I_DetectorValue) — класс, объекты которого хранят грязное время выполнение программы (TimeDetectorValue) и класс, объекты которого хранят числовые замеры, необходимые пользователю для оценки эффективности работы алгоритма (например, количество слияний или обменов) (CountDetecterValue).

При добавлении нового алгоритма в пакет, необходимо отнаследоваться от A_Algorithm (обязательно задать имя алгоритма) и переопределить метод execute, а также метод createDetectors, вызвав в нем сначала родительский метод createDetectors, чтобы создать объект Detector (protos). При этом создастся детектор, присутствующий в программе всегда, — детектор времени грязного выполнения программы. Далее, можно добавлять свои детекторы через protor.add(), передавая в качестве параметра функции новый детектор.

При создании детектора желательно указывать имя детектора — например, protos.add(new CountDetectorValue("Количество рекурсивных слияний")).

Обработка ошибок

Обработка ошибок, произошедших при работе алгоритма, лежит на пользователе и должна обрабатываться в вызывающем классе путем имплементирования от интерфейса AlgorithmException и подписки на события методом - addAlgorithmExceptionListener(AlgorithmException listener).

Алгоритмы

Сортировка пузырьком (BubbleSort). При данной сортировке детектируется грязное время и количество обменов.

Быстрая сортировка(QuickSort) . При данной сортировке детектируется грязное время и количество разделений.

Сортировка слиянием (MergeSort). При данной сортировке детектируется грязное время и количество слияний.

Результаты работы алгоритмов

Пакет результат (result) — содержит классы, необходимые для хранения результатов выполнения тест-плана на заданном алгоритме.

Класс TestPlanResult — хранит результат выполнения всего тест-плана.

Переопределяет методы для сохранения / загрузки результатов.

По умолчанию при сохранении — первое, что заносится это имя текущего объекта класса TestPlanResult, строящееся следующим образом:

Имя_выполненного_тест-плана + Имя_Алгоритма.

Класс OneResult — хранит результат выполнения всего тест-плана для одного выбранного детектора. Получить объект OneResult можно вызвав метод getResult() класса TestPlanResult, передав в него номер детектора (с нуля).

Описание классов и интерфейсов

Таблица 3. Описание классов в пакетах *algoanalyst.algorithm, algoanalyst.result* .

Имя класса/интерфейса	Описание, конструкторы, методы
A_Algorithm implements I_TypeName	Абстрактный класс для создания алгоритмов. **Методы** addAlgorithmExceptionListener(AlgorithmException listener) – добавить слушателя. removeAlgorithmExceptionListener(AlgorithmException listener) – удалить слушателя. TestPlanResults executePlan(TestPlan testPlan) – запустить тестирование алгоритма на входном тест-плане (test-plan). Возвращает объект TestPlanResult. Vector<I_DetectorValue> createDetectors() - создать вектор объектов I_Detector_Value. Возвращает вектор объектов I_Detector_Value. Минимально хранится один детектор грязного времени работы программы.
AlgorithmException	Интерфейс для обработки ошибок. **Методы** void onException(Throwable throwable) – метод для обработки ошибки
BubbleSort extends A_Algorithm	Класс – сортировка пузырьком. **Конструкторы** BubbleSort(String name) – создает объект с заданным именем. BubbleSort() – создает объект с именем по умолчанию – BubbleSort.

Имя класса/интерфейса	Описание, конструкторы, методы
MergeSort extends A_Algorithm	Класс – сортировка слиянием. **Конструкторы** MergeSort(String name) – создать объект с заданным именем. MergeSort() – создать объект с именем по умолчанию – MergeSort.
QuickSort extends A_Algorithm	Класс – быстрая сортировка. **Конструкторы** QuickSort(String name) – создать объект с заданным именем. QuickSort() – создать объект с именем по умолчанию – QuickSort.
I_ObjectName extends I_Name,I_TypeName	Интерфейс расширяющий интерфейсы I_Name, I_TypeName. **Методы** getTypeName() – вернуть тип класса, по умолчанию имя класса setObjectName(String Name) – задает имя объекта String getName() – вернуть имя объекта.
Detector implements I_ObjectName	Класс, содержащий результаты измерений для заданного в protos типа детектора **Конструкторы** Detector(String name, I_DetectorValue proto)- создать объект с заданным именем, и хранящий значения заданного в protos типа. **Методы** add(I_DetectorValue val) – добавить детектор. I_DetectorValue getProto() – получить прототип детектора Vector<I_DetectorValue> getData() – получить вектор значений детектора
I_DetectorValue extends I_ObjectName	Интерфейс для детекторов разных типов. **Методы** long getValue() – получить значение. incValue() – увеличить значение на условную единицу. setValue(long v0) – установить значение переменной value равное v0. I_DetectorValue cloneDetector() – клонировать объект.
CountDetectorValue implements I_DetectorValue	Класс, объекты которого хранят числовые замеры, необходимые пользователю для оценки эффективности работы алгоритма. **Конструкторы** CountDetectorValue(String name) – создать объект с заданным именем.
TimeDetectorValue implements I_DetectorValue	Класс, объекты которого хранят грязное время выполнение программы. **Конструкторы** TimeDetectorValue(String name) – создать объект с заданным именем.
I_File	Интерфейс для единой загрузки/сохранения файлов **Методы**

Имя класса/интерфейса	Описание, конструкторы, методы
	save(DataOutputStream out) – сохранить поток load(DataInputStream in) – загрузить поток
TestPlanResults implements I_ObjectName, I_File	Класс, хранящий результаты выполнения всего тест-плана. **Конструкторы** TestPlanResults(Vector<I_DetectorValue> protos, String name) – создать объект с заданными прототипами детекторов и именем объекта. TestPlanResults(Vector<I_DetectorValue> protos, Vector<Detector> results, Vector<Integer> dimensions, String Name) – создать объект с именем Name, а также с заданными прототипами, результирующей последовательностью и вектором размерностей. **Методы** I_DetectorValue[] createDetectors() – создать детекторы addDetectors(I_DetectorValue[] vals) – добавить детекторы OneResult getResult(int idx) - получить объект OneResult по номеру детектора (с нуля)
OneResult implements I_ObjectName, I_File	Класс, хранящий результат выполнения всего тест-плана для выбранного детектора. **Конструкторы** OneResult(Detector result, Vector<Integer> dimensions) – создать объект с заданным детектором (хранит результаты вычислений для тест-плана для одного типа) и вектором размерностей. **Переменные** Detector result - результирующие данные. Vector<Integer> dimensions - размерности наборов в тест-плане.

2.1.3. Пакет *algoanalyst.language*

Формальное описание

Пакет language содержит классы, описывающие командный язык, предназначенный для запуска приложения, настройки параметров объектов и передачи результатов соответствующим классам. В папку res необходимо добавить файл xml, содержащий команды.

Теги xml преобразуются в классы. У каждого такого класса тега есть аннотация @XStreamAlias("name") с параметром, указывающим имя тега. Класс обязан реализовать интерфейс Runnable, если результат его работы не возвращает значение, или интерфейс Callable<T>, если возвращает значение.

Самым внешним тегом является тэг <*commands*>. Этот тег может содержать любое положительное количество команд. Тег преобразуется в класс Commands.

Тег <show> показывает результаты TestPlanResults в окне для отображения графиков. Для этого ему нужны теги, возвращающие TestPlanResults. Тег преобразуется в класс ShowCommand.

Один из тегов, возвращающих TestPlanResult - <execute>. Для этого ему нужен экземпляр алгоритма и тестовый план. Тег преобразуется в класс ExecutePlanCommand. Также у тега можно указать атрибут print=”true”, для вывода результата в стандартный поток вывода.

Для получения алгоритма используется тег <algorithm>. Его единственный атрибут type указывает, какой тип сортировки создавать – bubble, merge или qucksort. Вложенный тег <name> определяет рабочее название алгоритма. Тег преобразуется в класс CreateAlgorithmCommand.

Для создания тестового плана используется тег <testPlan>. Тег преобразуется в класс CreateTestPlanCommand. Вложенный тег имя определит название тестового плана. Для создания плана нужны генераторы последовательностей и размерностей.

Тег <dimen> создает генератор размерностей. Атрибут type определяет тип генератора – linear, fib или exp. Для настройки генератора необходимо вложить теги с параметрами. Параметры для каждого генератора можно узнать в таблице 1. Тег преобразуется в класс CreateDimensionGeneratorCommand.

Тег <seq> создает генератор последовательностей. Необходимые атрибуты – type и data. Type определяет тип генератора – trend или rndFix. Data определяет тип данных – int, float или string. Параметры генератора определяются вложенными тегами.

Класс CommandInterpreter ожидает файл xml с внешним тегом <commands>, после чего преобразует xml в экземпляр класса Commands и выполняет все команды.

Таблица 4. Описание классов в пакете *algoanalyst.language*.

Имя класса/интерфейса	Описание, конструкторы, методы
XmlParser	Класс для чтения команд из файла xml. **Методы** parse(String srt) – конвертирует файл xml в объект.
CommandInterpreter	Класс, который запускает приложение. **Методы** void main(String[] args) – входная точка приложения. Первый параметр метода – путь к файлу
TheTest	Класс для интеграционного тестирования программы. Во время тестирования отработают все возможные комбинации классов. **Методы** void main(String[] args) – метод для запуска теста. Первый параметр метода – путь к файлу

2.1.4. Пакет algoanalyst.approx

Формальное описание

Пакет содержит классы позволяющие определить тип зависимости результатов тестирования. Для этого необходимо вызвать метод approximate2, передав в качестве параметра объект OneResult.

Таблица 5. Описание классов в пакете *algoanalyst.approx.*

Имя класса/интерфейса	Описание, конструкторы, методы
Approximate	Класс, аппроксимирующий результаты тестирования. **Методы** String approximate2(OneResult result) – метод возвращает название функции, которой соответствует зависимость в result
I_Function	Интерфейс объекта функции. **Методы** double calc(double a, double b, double x) – возвращает значение функции с параметрами a и b в точке x double calc(double x) – возвращает значение функции с параметрами a и b в точке x, предполагается, что параметры a и b являются полями объекта.
FunctionLinear	Класс линейной функции. Расширяет интерфейс I_Function.
FunctionLogarithmic	Класс функции логарифма по основанию 2. Расширяет интерфейс I_Function.
FunctionExponential	Класс экспоненциальной функции. Расширяет интерфейс I_Function.
FunctionLogLin	Класс логарифмически-линейной функции. Расширяет интерфейс I_Function.
FunctionSquare	Класс квадратичной функции. Расширяет интерфейс I_Function.

2.1.5. Пакет algoanalyst.graphs

Формальное описание

Пакет algoanalyst.graphs содержит классы для построения графиков по входным данным – тест-планам. Для создания форм с графиками используется платформа JavaFX. Для построения графиков необходимо подать список TestPlanResults-ов в статический метод run класса Graphic.

Таблица 6. Описание классов в пакете *algoanalyst.approx.*

Имя класса/интерфейса	Описание, конструкторы, методы
Qwer	Класс для учета отображенных графиков. **Конструкторы** Qwer(int id,int idx,TestPlanResults testPlanResults,int numberAtDatas) – создание объекта класса Qwer с полями:

Имя класса/интерфейса	Описание, конструкторы, методы
	id – индекс выбранного TestPlanResult-а в комбобоксе idx – индекс выбранного графика в комбобоксе testPlanResults - выбранный TestPlanResult numberAtDatas – индекс отображенного на панели графика.
Graphic	Класс-модель, который инициализирует форму для отображения графиков. **Методы** List<TestPlanResults> getVecResults() – возвращает список тест-планов vecResults. void run(List<TestPlanResults> resultsVector) – запуск формы отображения графиков, находящихся в resultsVector. Detector getResY(int idx, int i) – возвращает idx-й детектор из i-ого элемента списка vecResults. void main(String[] args) – запуск приложения построения графиков с инициализированными тестовыми данными. void start(Stage primaryStage) – загрузка fxml-файла и установка контроллера для формы отображения графиков, а также настройка параметров и запуск формы отображения графиков. void initializeTest() – инициализация тестовых данных.
Controller	Класс-контроллер для формы отображения графиков. В нем описаны методы, срабатывающие по событиям на форме отображения графиков. **Методы** void addDelete() – отображение/скрытие выбранного графика. void sortChange() – изменение набора строк в комбобоксе с видами функции. void setDataTest(int idx) – вспомогательный метод для метода sortChange(), где idx – индекс выбранного тест-плана. public void setResult() – заполнение комбобокса с тест-планами. void drawResult() – отрисовка на графике выбранной функции. void delete(int id, int idx) – удаление выбранной функции на графике. void enter() – отображение координат мыши на графике.

2.1.6. Пакет algoanalyst.wiev

Формальное описание

Пакет algoanalyst.wiev содержит классы для отображения пользовательского внешнего вида. Пользователи могут создавать различные тестовые планы, из выбираемых параметров, - алгоритмы, тип данных, генераторы. Для того, чтобы получить графики и результаты.

Таблица 7. Описание классов в пакете *algoanalyst.wiev.*

Имя класса/интерфейса	Описание, конструкторы, методы
Factory_MainAndAll	Класс для рефлексивного получения выборов параметров, на основе имеющихся классов. **Конструкторы** Factory_MainAndAll(int id,int idx,TestPlanResults testPlanResults,int numberAtDatas) – создание объекта класса Qwer с полями: factory – генератор фабрики box – JComboBox для отображения back – CallBack для обратной связи
Main_View	Класс, выводящий GUI на экран **Методы** initComponents – Отрисовка элементов WhatSSSSSequensor – отображение необходимый полей ввода для определенного секвенсора PaUmalchaniu – вызов настроек при первом запуске void main(String[] args) – запуск приложения. TestActionListener – принимать все параметры из выпадающих полей, и создавать необходимы тестовый план
Point_Algoritm	Класс – деталь для фабрики с рефлексией. Указывает на пакет где хранятся алгоритмы
Point_Generator	Класс – деталь для фабрики с рефлексией. Указывает на пакет где хранятся генераторы последовательностей
Point_TypeData	Класс – деталь для фабрики с рефлексией. Указывает на пакет где хранятся главные классы для определенных типов данных – числа, строки, дробные.
Point_TypeFloat	Класс – деталь для фабрики с рефлексией. Указывает на пакет где хранятся классы секвенсоры для дробных чисел
Point_TypeInt	Класс – деталь для фабрики с рефлексией. Указывает на пакет где хранятся классы секвенсоры для целых чисел
Point_TypeString	Класс – деталь для фабрики с рефлексией. Указывает на пакет где хранятся классы секвенсоры для строк

3. Руководство пользователя

3.1. Командный язык

Для запуска программы с помощью командного языка используется файл xml. Файл должен иметь следующую структуру.

```
<commands>
    <show>
        <execute print="true | false">
            <testPlan>
                <name>Another Test</name>
                <dimen type='linear | exp | fib'>
                    <!-- Пример параметров -->
                    <start>1000</start>
                    <count>3</count>
                    <step>1.5</step>
                </dimen>
                <seq type="rndFix | trend" data="int | float | string">
                    <!-- Пример параметров -->
                    <min>97</min>
                    <max>120</max>
                    <len>10</len>
                </seq>
            </testPlan>
            <algorithm type="bubble | merge | quicksort">
                <name>Algorithm name</name>
            </algorithm>
        </execute>
    </show>
</commands>
```

Значение тегов можно узнать в описании пакета algoanalyst.language, пункт 2.1.3. Также можно воспользоваться расширенным синтаксисом с возможностью объявлять переменные. Внутри корневого тега описываются все переменные следующим образом:

```
<xsl:variable name="variableName">
    <anyContent>Plan Name</anyContent>
</xsl:variable>
```

После, для того, чтобы воспользоваться переменной, нужно написать тег `<xsl:copy-of select=”$name”/>`.

Структура расширенного синтаксиса:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:variable name="seq">
        <seq type="rndFix" data="string">
            <min>97</min>
            <max>120</max>
            <length>15</length>
        </seq>
    </xsl:variable>
    <!-- Переменных может быть любое количество -->
    <xsl:template match="/">
        <!-- Здесь описываются команды -->
    </xsl:template>
</xsl:stylesheet>
```

3.2. Результат работы.

Графики

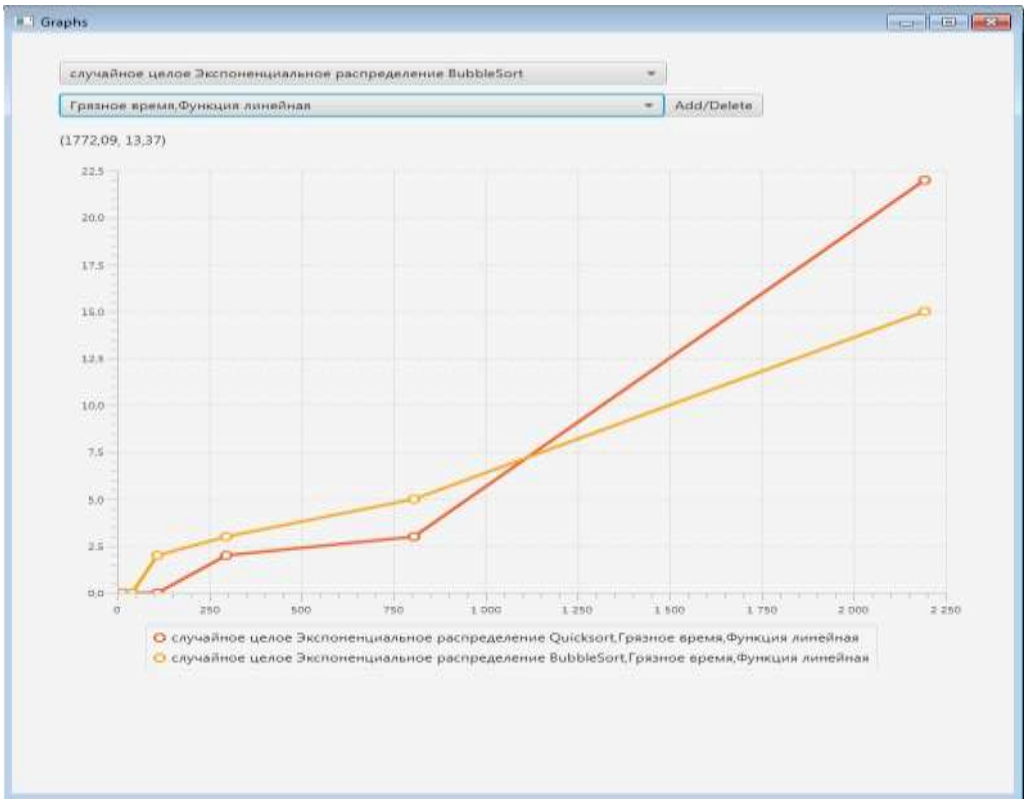


Рисунок 1. Построение графиков

Пользовательские интерфейс

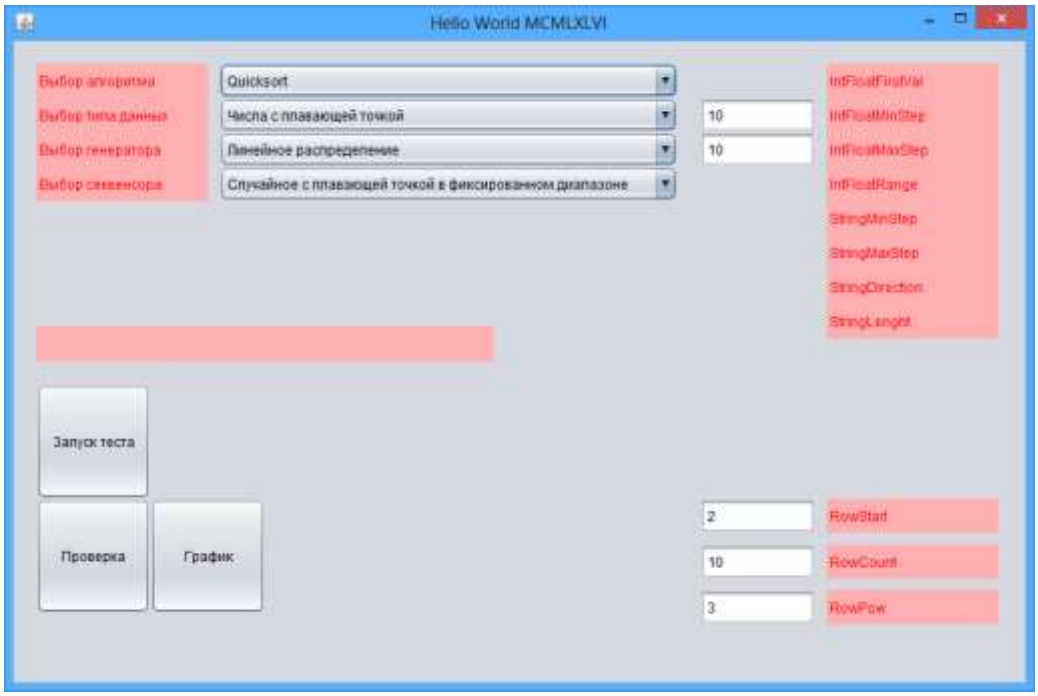


Рисунок 2. Интерфейс пользователя

3.3. Анализ результатов, планируемые изменения в следующей итерации разработки .

3.3.1 Анализ результатов, планируемые изменения в пакетах Algoanalyst.algoritm и Algoanalyst.result.

Недостатки:

- Если происходит ошибка, то об этом сообщается вызывающей стороне, но при этом в TestPlanResult не добавляется значение для ошибочного (упавшего) тест-кейса.

Планируемые изменения:

- Продумать обработку ошибок, происходящих при выполнении работы алгоритмов на тестовом плане.
- Создать формат "некорректные данные" и заполнять ими TestPlanResult при падении тест-кейса для исключения дальнейших ошибок типа "выход за пределы коллекции".
- Обработка ошибок должна происходить в вызывающих блоках. Можно увеличить набор тестируемых алгоритмов.

3.3.2 Анализ результатов, планируемые изменения в пакете Algoanalyst.graphs.

Недостатки:

- Вызвать форму для построения графиков можно лишь один раз за время работы приложения, при повторной попытке вызова формы приложение падает.
- Отсутствует возможность увидеть аппроксимированную функцию для сравнения с оригинальной функцией.
- Отсутствует возможность построения графиков функций с логарифмической шкалой.

Планируемые изменения:

- Реализация возможности построения логарифмической шкалы, для этого возможно придется создавать собственный класс для отображения графика (на данный момент используется класс JavaFX Linechart). Также будет необходимо добавить на форму 2 чек-бокса – LogX и LogY для управления видом шкалы.
- Добавление кнопки, для вывода выбранной аппроксимированной функции.
- Решение проблемы с падением программы, при повторном вызове формы построения графиков.
- Добавление кнопки «Сохранить» - для возможности сохранения построенных графиков в формате JPEG.
- Добавление элемента класса ColorPick – для настройки цвета выбранного графика.
- Добавление обработки исключений при проверке входных данных.

3.3.3 Анализ результатов, планируемые изменения в пакетах Algoanalyst.testcase и Algoanalyst.testcasetypes.

Планируемые изменения:

- Сменить механизм взаимодействия с пользовательским интерфейсом.
- Ограничение по длине строки на настоящий момент, то есть при размере строки = N, можно получить всего 26^N чисел.
- Добавить секвенсоры к типу данных String - случайно убывающий, случайно-возрастающий секвенсор.

3.3.4 Анализ результатов, планируемые изменения в пакете Algoanalyst.wiew.

Полученный графический интерфейс на данный момент позволяет выбрать необходимые данные, задать настройки и вывести полученные графики. Но этих функций пока недостаточно, чтобы дать пользователю реализовать всего возможности.

В текущей версии графического интерфейса присутствует достаточное количество недостатков, которые должны быть исправлены при последующей разработке.

- Так как в программе присутствует командный язык, необходимо специальное поле ввода для задаваемых команд, а также кнопки загрузки и сохранения командных скриптов.
- Чтобы проще тестировать правильность работы алгоритмов, также в графический интерфейс необходимо добавить кнопки для загрузки и сохранения сгенерированных тестовых данных.
- Подписи. Для полного понимания пользователю необходимы понятные подписи для ввода нужных параметров.
- К сожалению, в текущей версии график можно вывести только один раз. Это тоже необходимо исправить.

В будущем, графический интерфейс программы должен стать более понятным для обычного пользователя. Чтобы он, не особо разбираясь, сразу мог получить необходимый ему результат – правильные данные, графики и прочее.

3.3.5 Анализ результатов, планируемые изменения в пакете Algoanalyst.language.

Планируемые изменения:

- Добавить в главное окно программы кнопку для перехода к окну работы с командным языком. Окно состоит из текстовой области и нескольких кнопок: сохранение/загрузка файла с командами; запуск скрипта; добавление в скрипт самых часто употребляемых команд.

3.3.6 Анализ результатов, планируемые изменения в пакете Algoanalyst.approx.

Недостатки:

- Аппроксимация работает не всегда точно. Для более точного аппроксимирования можно попробовать определять коэффициенты аппроксимируемой функции решением СЛАУ.

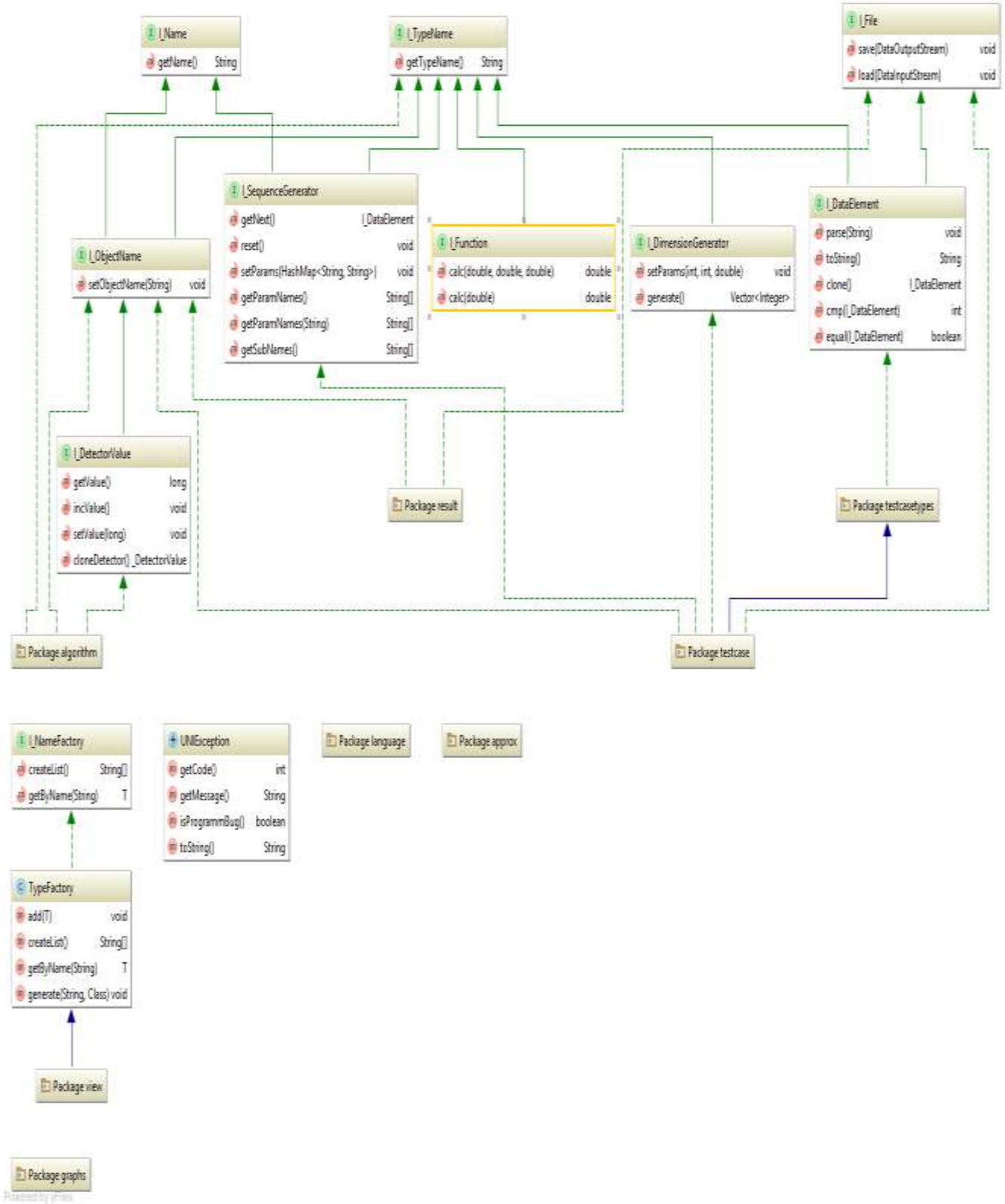
Планируемые изменения:

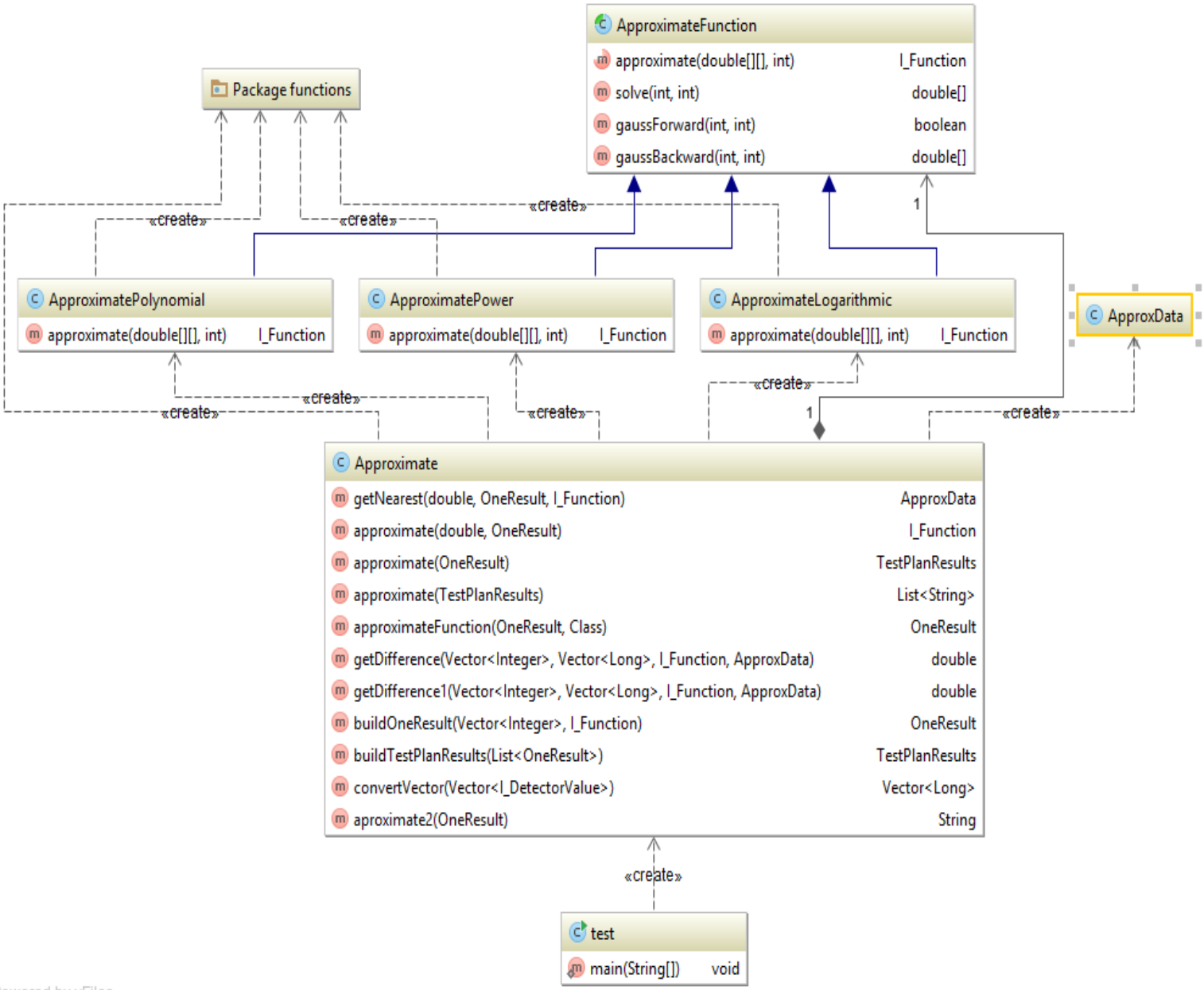
- Следует предусмотреть возможность возвращения построенной функции для построения аппроксимированной функции на графике. Для этого нужно будет изменить графический интерфейс.

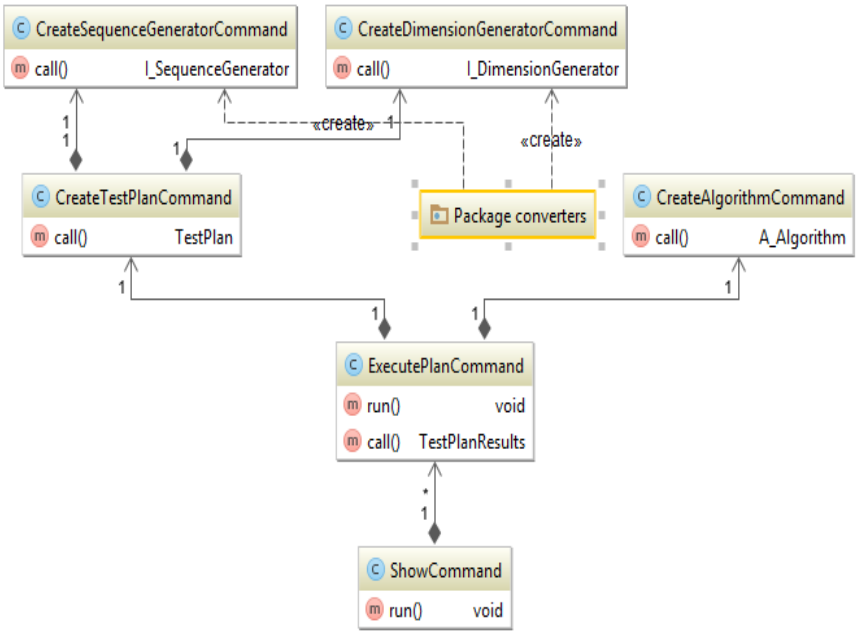
Приложение 1. Метрика проекта.

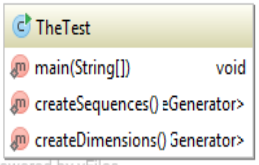
				пакеты	6	8	10	12		
1	Данные,генератор, тестовый набор, план,генератор планов	Катунов, Юргина		testcase,testcasetypes	244	704	839	874		26%
2	Алгоритм,эксперимент, результат	Потапейко, Утбасарова		algoritm	324	402	435	435		13%
3	Графики, обработка	Археменко, Им		result,graphs	102	182	258	353		10%
4	FrontEnd, граф. дизайн, профиль пользователя	Васильев,Владыкин		view	287	326	486	698		20%
5	Подбор зависимостей, генератор функций	Анчугов, Волков		approx	149	221	216	540		16%
6	Командный язык	Оленев, Ситников		language	66	195	262	312		9%
	TeamLeader			root	182	189	193	196		6%
Лабы					1354	2219	2689	3408		
				Приращение		1,64	1,21	1,27		
					1	2	3	4		
2	Обсуждение каркаса	2	2	чел*мес	0,44	0,87	1,31	1,75		
4		6	3	НЕСОВПАДЕНИЕ	**7,6**	**6,4**	**5,2**	**5,0**		
6		6	4							
8		6	5							
10		6	6							
12		6	7							
			8							
			9	чел*мес вычисл	3,3	5,5	6,8	8,7		
			10	срок вычисл	2,8	3,4	3,6	4,0		
			11		**a**	b	c	d		
			12		2,4	1,05	2,5	0,38		

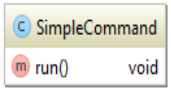
Приложение 2. Диаграммы классов.

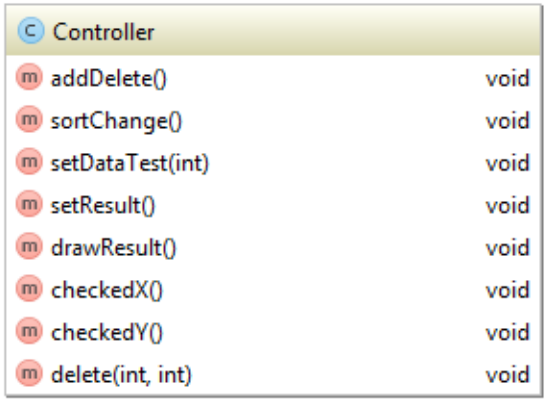


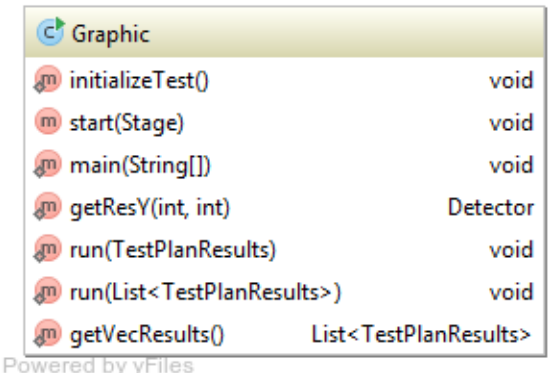



CreateSequenceGeneratorCommand
call() I_SequenceGenerator
CreateDimensionGeneratorCommand
call() I_DimensionGenerator
«create»
«create»
CreateTestPlanCommand
call() TestPlan
Package converters
CreateAlgorithmCommand
call() A_Algorithm
ExecutePlanCommand
run() void
call() TestPlanResults
ShowCommand
run() void

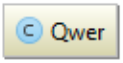

TheTest
main(String[]) void
createSequences() eGenerator>
createDimensions() Generator>
Powered by yFiles

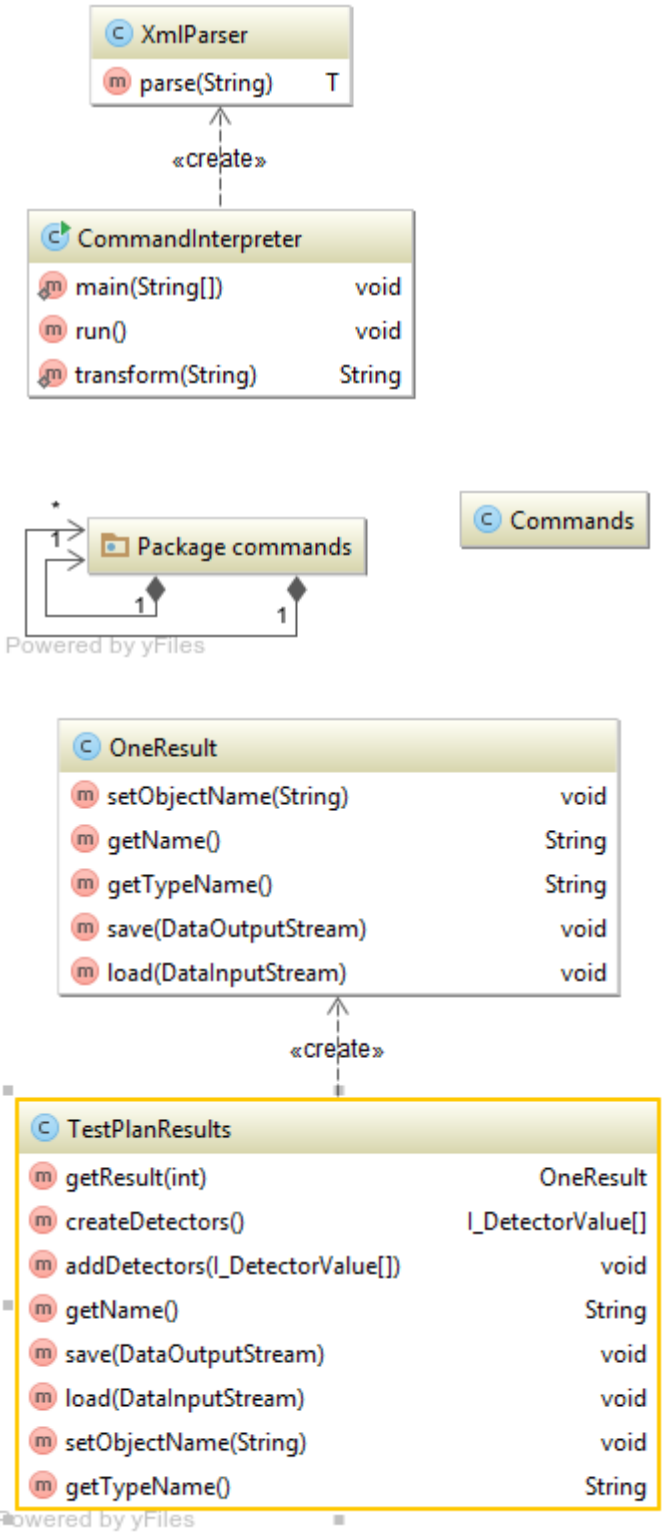

SimpleCommand
run() void

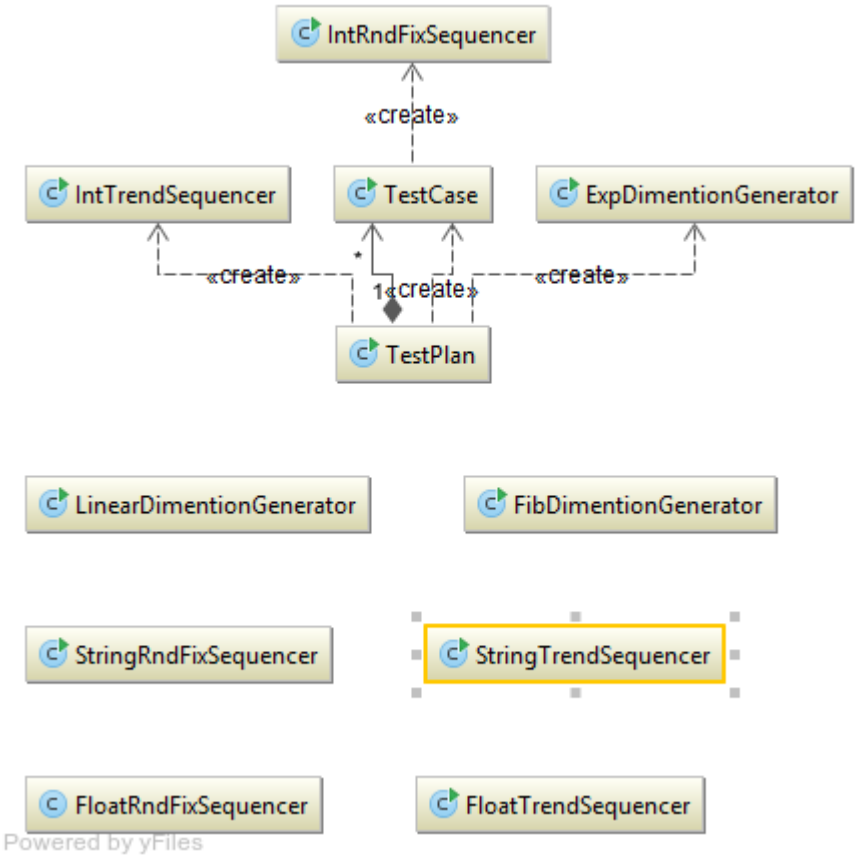

Controller
addDelete() void
sortChange() void
setDataTest(int) void
setResult() void
drawResult() void
checkedX() void
checkedY() void
delete(int, int) void


Graphic
initializeTest() void
start(Stage) void
main(String[]) void
getResY(int, int) Detector
run(TestPlanResults) void
run(List<TestPlanResults>) void
getVecResults() List<TestPlanResults>
Powered by yFiles


graph


Qwer


XmlParser
parse(String) T
«create»
CommandInterpreter
main(String[]) void
run() void
transform(String) String
Package commands
Commands
1
1
1
Powered by yFiles
OneResult
setObjectName(String) void
getName() String
getTypeName() String
save(DataOutputStream) void
load(DataInputStream) void
«create»
TestPlanResults
getResult(int) OneResult
createDetectors() I_DetectorValue[]
addDetectors(I_DetectorValue[]) void
getName() String
save(DataOutputStream) void
load(DataInputStream) void
setObjectName(String) void
getTypeName() String
Powered by yFiles


IntRndFixSequencer
«create»
IntTrendSequencer
TestCase
ExpDimentionGenerator
«create»
«create»
1
«create»
TestPlan
LinearDimentionGenerator
FibDimentionGenerator
StringRndFixSequencer
StringTrendSequencer
FloatRndFixSequencer
FloatTrendSequencer
Powered by yFiles