Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт *Новых материалов и технологий*

Кафедра *Информационные технологии и автоматизация проектирования*

ДОПУСТИТЬ К ЗАЩИТЕ ПЕРЕД ГЭК

Зав. кафедрой ИТиАП

(подпись) Д.В. Куреннов (Ф.И.О.)

«____» _____ 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка алгоритмов и программного обеспечения для обработки геометрической информации 2D-объектов специального типа

Руководитель: Петунин А.А., доцент, д.т.н. (Ф.И.О., ученая степень, ученое звание (должность)) _____ (подпись)

Консультант: Минеева Т.А., доцент, к.э.н. (Ф.И.О., ученая степень, ученое звание (должность)) _____ (подпись)

Нормоконтролер: Мухоморов В.Л., ст. преподаватель (Ф.И.О., ученая степень, ученое звание (должность)) _____ (подпись)

Студент группы НМТ-491506 Ф.И.О Молоков Л.К. _____ (подпись)

Екатеринбург
2023

Реферат

Отчёт содержит 91 страницу, 0 рисунков, 0 таблиц, 12 формул, 0 приложен.

Ключевые слова: коневертер, алгоритмы, программное обеспечение, примитивы, разработка.

Объект ВКР — форматы хранения данных и представления графической информации.

Цель работы — разработать программное обеспечение для дешифрования информации из файлов типа DXF, извлечении из них информации о геометрических примитивах и формирования файлов других форматов, содержащих всю необходимую информацию в удобном для последующей работы виде.

Методы работы и исследования: теоретический анализ и последующий синтез информации, моделирование, разработка, тестирование.

Результатом работы стали алгоритмы и программное обеспечение — конвертер «DXF Primiview».

Область применения разработанных алгоритмов и программного обеспечения — удобное использование получаемых форматов данных при технологической разработке управляющих программ для производства изделий на станках с числовым программным управлением.

Значимость работы заключается в использовании разработанных алгоритмов и программного обеспечения при разработке специальных САПР и работе в них, а также, выполнение работ по хозяйственному договору УрФУ с ООО "Униматик".

Содержание

Глоссарий

В настоящей работе применяют следующие термины с соответствующими определениями:

Программный продукт — объект, состоящий из программ, процедур, правил, а также, если предусмотрено, сопутствующих им документации и данных, относящихся к функционированию системы обработки информации. ГОСТ 28806-90 «Качество программных средств. Термины и определения» (утверждён и введён в действие Постановлением Госстандарта СССР от 25 декабря 1990 г. № 3278).

Система автоматизированного проектирования — организационно-техническая система, входящая в структуру проектной организации и осуществляющая проектирования при помощи комплекса средств автоматизированного проектирования (КСАП). ГОСТ 23501.101-87 «Системы автоматизированного проектирования. Основные положения» (утверждён и введён в действие Постановлением Государственного комитета СССР по стандартам от 26.06.87 № 2668).

Язык разметки — набор символов или последовательностей символов, вставляемых в текст для передачи информации о его отображении или строении.

Обозначения и сокращения

В настоящей ВКР применяют следующие сокращения и обозначения:

САПР — Система автоматизированного проектирования.

УП — Управляющая программа.

ЧПУ — Числовое программное управление.

ПО — Программное обеспечение.

ПП — Программный продукт.

ЯП — Язык программирования.

DXF — Drawing eXchange Format. Формат файлов для обмена графической информацией между приложениями САПР, созданный фирмой Autodesk для системы AutoCAD в 1982 г.

TXT — Текстовый формат файлов, представляющий собой последовательность строк электронного текста.

SVG — Scalable Vector Graphics. Язык разметки масштабируемой векторной 2D-графики на основе XML, поддерживающий интерактивность и анимацию. Разрабатывается консорциумом World Wide Web с 1999 года по сегодняшний день.

XML — eXtensible Markup Language. Расширяемый язык разметки подобный HTML.

JSON — JavaScript Object Notation. Текстовый формат обмена данными, основанный на языке программирования JavaScript.

Введение

Актуальность темы исследования. В настоящее время в рамках проектов Уральского Федерального Университета (УрФУ) и Уральского межрегионального научно-образовательного центра (УМНОЦ) происходит разработка специализированных САПР. Создаваемые ПП работают с файлами, содержащими геометрическую 2D-информацию. Используются следующие форматы файлов:

а) DXF,
б) TXT,
в) SVG,
г) JSON.

В связи с этим требуется разработать ПО по конвертации данных форматов файлов.

Степень разработанности темы исследования. В открытом доступе сети Интернет существуют онлайн-конвертеры файлов DXF в другие форматы. Большая их часть лишь визуализирует графическую информацию, представленную в том или ином DXF-файле, и конвертирует в наиболее популярные форматы изображений (JPEG, PNG). Для использования конвертера на предприятии-заказчике необходим особый формат данных, в которые конвертируется DXF. Такие форматы не могут быть обеспечены существующим в открытом рынке ПО — таким, как, например, «DXF Reader GT» от компании «Gray Technical». Несмотря на высокую степень проработки программы, формат выходных данных не соответствует требованиям, предъявляемым к TXT-, SVG- и JSON-файлов, компании «Unimatic». Ещё одним недостатком зарубежного ПО является ежемесячная плата разработчикам. Одной из задач современной Российской Федерации является импортозамещение на производстве, поэтому разработка

собственного ПП увеличит независимость и самостоятельность предприятий и компаний, пользующихся этим ПП.

Цель работы заключается в разработке программного обеспечения для дешифрования информации из файлов типа DXF с геометрической информацией объектов специального типа, извлечении из них информации о геометрических примитивах, вывода полученной графической информации на экран для верификации, формирования текстового файла (TXT) с выводом данных в исходном виде DXF, формирования текстового файла (TXT) с выводом данных в виде координат точек и радиуса примитивов (линий, дуг) между ними, формирования файла-описание двумерной векторной графики в формате SVG с выводом данных в исходном виде DXF, формирования текстового файла (JSON) с выводом данных в виде координат точек и степенями кривизны примитивов между ними.

Данные конвертеры необходимы, в частности, при разработке таких САПР, как Сириус, ТокКТЭ и других.

Для достижения этой цели поставлены следующие **задачи**:

1) проанализировать входные данные DXF, в соответствии с разрабатываемым ПО;

2) выявить структурно-содержательные особенности файлов формата DXF для последующей работы с разрабатываемым ПО;

3) проанализировать возможности применения разных ЯП для разработки ПО;

4) разработать алгоритмы;

5) разработать ПО;

6) провести анализ экономической целесообразности разрабатываемого проекта.

Объект исследования — форматы файлов по обмену графической информацией.

Предмет исследования — проектирование алгоритмов и ПО для конвертации файлов и формата DXF в форматы TXT, SVG, JSON.

Теоретическая и практическая значимость работы

а) разработка универсальна, так как данные файловые конвертеры могут быть использованы любым пользователем в собственных целях (и вне САПР);

б) подход к разработке конвертеров универсален, что значит, что алгоритмы и методы, применённые в данной работе, могут быть использованы при создании аналогичного ПО;

в) разработанное ПО можно внедрять в существующие САПР, работающие с форматом файлов DXF;

г) САПР, использующие разработанный модуль по конвертации файлов с графической информацией, получают независимость от использования аналогичного ПО зарубежного производства;

д) положенные в основу алгоритмы и написанное ПО имеет открытый доступ в сети Интернет и имеет перспективу развития в полноценный модуль импорта/экспорта файлов;

е) коммерческая выгода заказчика.

1 Задача конвертации данных и графических файлов DXF. Анализ текущего состояния проблемы исследования

В данном разделе описаны теоретические основы, требующиеся для разработки алгоритмов и программного обеспечения для обработки геометрической информации 2D-объектов специального типа.

1.1 Стнадартизованные файловые форматы обмена графической информацей

CAD-, CAM- и CAE-системы работают, помимо прочего, с графической информацией с целью визуализации данных. При этом по мере развития различных САПР от всевозможных организаций по всему миру, для каждой системы создавались специальные форматы данных, с которыми удобно работать той или иной системе.

AutoCAD от компании Autodesk, являясь первой в мире CAD-системой, задала норму файлового формата для работы с чертежами — DXF.

Кроме этого, разработчики специализированных САПР до сих пор создают всё новые форматы, в которых работает именно их система.

В рамках работы были изучены 5 форматов файлов, работающих с геометрической информацией. Среди них следующие:

— DXF,
— TXT(DXF-type),
— TXT(x,y,r),
— SVG,
— JSON.

Работа ориентирована именно на эти форматы данных, что подробно объясняется в разделе 1.3.

Рассмотрим детально каждый из стандартизованных форматов данных.

1.1.1 Формат DXF

Краткая характеристика

Название:	AutoCAD DXF
Также известен как:	AutoCAD Drawind Interchange Format, DXF, .DXB, .SLD, .ADI
Тип данных:	Векторный
Сжатие:	нет
Максимальный размер изображения:	неограниченный
Несколько изображений в одном файле:	нет
Разработчик:	Autodesk
Поддерживающ. приложения:	AutoCAD, различные САПР, CorelDraw, др.
Использование:	Хранение и обмен САПР-(проектировочными) и векторными данными
Комментарии:	Сложный формат, в основном, потому что может содержать в себе много различных типов данных. Формат разработан и поддерживается компанией Autodesk с целью применения в CAD-системе AutoCAD. Самая распространённая форма DXF — 7-битный текст, однако существует, также, два схожих двоичных формата, один из которых представляется в виде расширения DXF, а другой — DXB.

Обзор

Форматы AutoCAD DXF (Drawing Interchange Format) и AutoCAD DXB (Drawing Interchange Binary) связаны с CAD-системой AutoCAD, созданной и поддерживаемой Autodesk. DXB — это упрощенная двоичная версия файла DXF. Другими форматами файлов, связанными с AutoCAD, являются форматы слайдов (.SLD) и графиков (.ADI).

Несмотря на то, что DXF был разработан для представления данных в САПР, он используется многими другими программами как формат обмена многими различными типами данных, чаще всего векторно-ориентированной информацией, а также текстом и 3D-полигонами. Как формат САПР, он также может выражать общие концепции черчения, такие как ассоциативные размеры.

Почти любой тип данных может быть каким-либо образом представлен в DXF. Например, программа для рисования CorelDraw! экспортирует контуры чертежа с объектом AutoCAD POLYLINE, в то время как 3D-программа может экспортировать только объекты 3DFACE, представляющие трёх- и четырёхсторонние многоугольники. DXF, также, позволяет создавать множество способов делать почти одно и то же, например, описывать объекты как отдельные редактируемые группы. Одна программа может размещать объекты на разных слоях рисунка, в то время как другая может использовать разные цвета пера, а третья может использовать именованные «блоки» для группировки данных.

Хоть DXF и широко используется для обмена простыми линейными данными, разработчик приложений, желающий поддерживать в них DXF, должен учитывать, что AutoCAD может хранить эти многочисленные типы данных различными способами.

Иногда правильная интерпретация файла DXF может быть очень сложной. Предполагаемый внешний вид линий и областей может зависеть от многих, казалось бы, непонятных настроек в заголовке DXF-файла. По-

скольку файлы DXF очень сложно правильно интерпретировать, многие разработчики приложений решают экспортировать только DXF.

Даже среди программ, заявляющих об импорте DXF, можно обнаружить, что они поддерживают лишь часть всего, что возможно в DXF. Если есть необходимость создать свои собственные файлы DXF для передачи данных в программу, которая утверждает, что импортирует DXF, нужно убедиться, что известно, какие представления она понимает.

С каждой новой версией AutoCAD, DXF изменяется. AutoCAD версии 13 расширил формат DXF во многих отношениях, чтобы представить специализированные данные нового механизма геометрии. Эти дополнения хранят информацию о сложных поверхностях и твёрдых телах для геометрического механизма ACIS компании Spatial Technology, который теперь является частью AutoCAD. Не вся эта информация была задокументирована и должна быть пропущена любым читателем DXF. В версии 13 собственный допуск AutoCAD для числовых файлов DXF также изменился, поскольку он расширил шаг аудита, который проверяет достоверность импортируемых файлов DXF.

Очевидно, что формат файла DXF довольно сложный и тонкий. Далее приведена базовая структура любого файла DXF.

Организация файла

Файл DXF состоит из семи разделов: заголовка, таблиц, блоков, классов, объектов, сущностей и маркера конца файла.

— Раздел HEADER содержит переменные, представляющие состояние внутренних настроек AutoCAD. Например, для переменной версии AutoCAD «$ACADVER» установлено значение «AC1012» в файле DXF, сохраненном AutoCAD версии 13. Другие переменные задают единицы измерения углов, значения по умолчанию для снятия фаски, смещения, масштабирования и т. д.

— Раздел TABLES содержит несколько массивов информации, используемой в остальной части чертежа, например список типов линий, имён слоев, шрифтов и предустановленных видов чертежа.

— Раздел BLOCKS содержит предопределенные элементы чертежа, которые могут присутствовать на чертеже. Например, блок может определять стандартную канавку, которая размещается на каждой секции вала определённого диаметра на чертеже. На определения блоков ссылаются в разделе ENTITIES с помощью команды INSERT.

— Разделы CLASSES и OBJECTS были представлены начиная с AutoCAD версии 13. Раздел CLASSES содержит описание любых определяемых приложением классов объектов, которые могут быть реализованы в разделах BLOCKS или ENTITIES.

— Раздел OBJECTS содержит неграфические части чертежа. Все сущности, которые не являются частью сущностей или таблиц символов, являются «объектами». Например, здесь хранятся словари AutoCAD.

— Раздел ENTITIES содержит фактические данные объекта чертежа. Сюда могут входить необработанные данные, такие как объекты LINE и ARC, а также команды INSERT, которые помещают предопределенное определение блока в определенную позицию на чертеже.

— Конец данных DXF отмечается директивой EOF в последней строке файла.

Детали файла Файл DXF состоит из пар групповых кодов и связанных значений. Каждое из них находится в отдельной строке текстового файла. Целочисленный групповой код указывает тип значения, за которым следует. Групповые коды встречаются в диапазонах. Например, за групповыми кодами от 0 до 9 следуют строки, и каждый отдельный групповой код используется в определенных случаях. Групповой код 0 указывает на начало объекта, таблицы или индикатора конца файла. Код 1 указывает основное текстовое значение объекта. Код группы 2 используется для имён,

таких как имена разделов, блоков, имен таблиц и т. д. Код 9 вводит имя переменной раздела заголовка. Например, в начале каждого файла DXF код группы 0 предшествует команде SECTION, за которой следует код группы 2 со строкой, указывающей тип раздела, например HEADER:

```
0
SECTION
2
HEADER
9
$ACADVER
1
AC1012
```

Диапазоны групповых кодов указывают тип данных, которым следует следовать. Групповые коды от 10 до 59 используются для значений с плавающей точкой, таких как координаты точек. Коды с 60 по 79 хранят целочисленные значения. Например, для сохранения местоположения 2D-точки сначала используется групповой код 10 для значения X, затем код 20 используется для значения Y. Если объект имеет вторичное значение координаты, он также будет использовать групповые коды 11 и 21. Вот минимальный, но полный файл DXF, который описывает линию от точки (1,2) до (3,4) в плоском пространстве:

```
0
SECTION
2
ENTITIES
999
This is just a line
0
LINE
8
0
10
1.0
20
2.0
11
```

```
16 3.0  21
17 4.0  0
18 ENDSEC
19 0
20 EOF
```

Код группы 999 предшествует комментарию. Эта строка будет помещена на слой 0, на что указывает групповой код 8. Этот минимальный файл является примером файла «только объекты», который будет принят практически любой программой, которая утверждает, что импортирует DXF.

Поскольку AutoCAD расширяется с каждой новой версией, добавляются новые групповые коды. При написании программы, которая читает файлы DXF, можно обеспечить совместимость в будущем, игнорируя неопределенные пары кода группы и значения. Одним любопытным аспектом DXF является то, что он не содержит цветовой палитры, однако большинству объектов в файле DXF можно присвоить отдельное значение цвета с групповым кодом 62. Каждому объекту чертежа может быть присвоен номер от 1 до 255, известный как AutoCAD Color Index, или ACI, также описанный в более ранней документации как «номер пера». Это отражает происхождение AutoCAD как пакета САПР, в котором чертежи обычно печатались на перьевом плоттере с несколькими чернильными перьями, но без стандартного соответствия фактическим значениям RGB или даже цветам линий на экране. AutoCAD теперь устанавливает цвет RGB по умолчанию для каждого ACI, когда он появляется на экране, но они не сохраняются в файле DXF [1].

1.1.2 Формат SVG

Краткая характеристика

Название: image/svg+xml

Также известен как: SVG, SVGZ (изображения, сд=жатые с помощью gzip)

Тип данных:	Векторный/растровый
Сжатие:	SVGZ
Максимальный размер изображения:	неограниченный
Несколько изображений в одном файле:	нет
Разработчик:	World Wide Web Consortium (SVG Working Group)
Поддерживающ. приложения:	браузеры, редакторы изображений, инструменты и библиотеки
Использование:	Доступный обзор и обмен векторными изображениями, построение графиков, сложные элементы пользовательского интерфейса, логотипы, простые игры
Комментарии:	Язык для описания двумерной графики в XML, созданный Консорциумом Всемирной паутины (W3C). Поддерживает как неподвижную, так и анимированную интерактивную графику — или, в иных терминах, декларативную и скриптовую. Не поддерживает описания трёхмерных объектов.

Обзор

SVG - это язык для описания двумерной графики в XML [XML10, XML11]. SVG позволяет создавать графические объекты трех типов: векторные графические фигуры (например, контуры, состоящие из прямых линий и кривых), мультимедиа (растровые изображения, видео и аудио) и текст.

Документы SVG могут быть интерактивными и динамическими. Анимации могут быть определены и запущены либо декларативно (то есть

путем встраивания анимационных элементов SVG в содержимое SVG), либо с помощью сценариев.

Достоинством формата SVG можно назвать, среди прочего, его представление в текстовом формате, то есть файлы SVG можно читать и редактировать при помощи обычных текстовых редакторов. При просмотре документов, содержащих SVG-графику, имеется доступ к просмотру кода просматриваемого файла и возможность сохранения всего документа. Кроме того, SVG-файлы обычно получаются меньше по размеру, чем сравнимые по качеству изображения в форматах JPEG (Joint Photographic Experts Group) или GIF (Graphics Interchange Format), а также хорошо поддаются сжатию.

Масштабируемость изображения в формате SVG подразумевает возомжность увеличить любую часть изображения SVG без потери качества.

Широко доступно использование растровой графики в SVG-документах позволяет вставлять элементы с изображениями в форматах PNG, GIF или JPG.

Текст в графике SVG является текстом, а не изображением, поэтому его можно выделять и копировать, он индексируется поисковыми машинами, для этого не требуется создавать дополнительные метафайлы для поисковых роботов.

Анимация реализована в SVG с помощью языка SMIL (Synchronized Multimedia Integration Language), разработанного также консорциумом W3C. Поддерживаются скриптовые языки на основе спецификации ECMAScript. SVG-элементами можно управлять с помощью JavaScript. Применение скриптов и анимации в SVG позволяет создавать динамичную и интерактивную графику. В SVG обеспечивается событийная модель, отслеживаются события (загрузка страницы, изменение её параметров, события мыши, клавиатуры и др.). Анимация может запускаться по определённому событию (например «onmouseover» или «onclick»), что придаёт

графике интерактивность. У каждого элемента есть свои собственные события, к которым можно привязывать отдельные скрипты.

SVG — открытый стандарт. В отличие от некоторых других форматов, SVG не является чьей-либо собственностью.

SVG-документы легко интегрируются с HTML-(HyperText Markup Language — «язык гипертекстовой разметки») и XHTML-документами. Внешние SVG подключаются через тег <object>, значение атрибута data — имя файла с расширением «.svg», содержащего разметку SVG, и имеющего MIME-тип image/svg+xml. Атрибуты width и height определяют размеры области SVG по горизонтали и по вертикали. Элементы SVG совместимы с HTML и DHTML (Dynamic HTML).

SVG предоставляет все преимущества XML:

— Возможность работы в различных средах;

— Интернационализация (поддержка Юникода);

— Широкая доступность для различных приложений;

— Лёгкая модификация через стандартные API — например, DOM. SVG поддерживает стандартизированную W3C объектную модель документа DOM, обеспечивая доступ к любому элементу, что даёт широкие возможности по динамическому изменению элементов, их атрибутов и событий.;

— Лёгкое преобразование таблицами стилей XSLT. Как любой основанный на XML формат, SVG даёт возможность использовать для его обработки таблицы трансформации (XSLT). Преобразуя XML-данные в SVG с помощью простого XSL, можно легко получить графическое представление любых данных, например визуализировать химические молекулы, описанных на языке CML.

Организация файла

Фрагмент документа SVG состоит из любого количества Элементы SVG содержится в "svg"элемент, включающий "svg"элемент.

Фрагмент документа SVG может варьироваться от пустого фрагмента (т. Е. без содержимого внутри "svg"элемент), для очень простого Фрагмент документа SVG содержащая один SVG графический элемент такая как "прямоугольник"для сложной, глубоко вложенной коллекции элементы контейнера и графические элементы.

Фрагмент документа SVG может существовать сама по себе как автономный файл или ресурс, в этом случае Фрагмент документа SVG является документом SVG, или он может быть встроен в качестве фрагмента в родительский XML-документ.

В следующем примере показано простое содержимое SVG, встроенное в виде фрагмента в родительский XML-документ. Обратите внимание на использование пространств имен XML, чтобы указать, что "svg"и "эллипс"элементы принадлежат пространству имен SVG:

```
<?xml version="1.0"?>
<parent xmlns="http://example.org"
  xmlns:svg="http://www.w3.org/2000/svg">
  <!-- parent contents here -->
  <svg:svg width="4cm" height="8cm" version="1.2"
    baseProfile="tiny" viewBox="0 0 100 100">
  <svg:ellipse cx="50" cy="50" rx="40" ry="20" />
  </svg:svg>
  <!-- ... -->
</parent>
```

Фрагмент документа SVG может содержать только один "svg"элемент, это означает, что "svg"элементы не могут отображаться внутри содержимого SVG.

Во всех случаях для соответствия либо Пространству имён в XML 1.0, либо Пространству имён в XML 1.1 согласно Рекомендациям [XML-NS10, XML-NS], объявление пространства имён SVG должно находиться в области видимости для "svg"элемента, так что все Элементы SVG идентифицируются как принадлежащие к пространству имен SVG.

Например, "xmlns"атрибут без префикса может быть указан на "svg"элемент, что означает, что SVG является пространством имён по умолчанию для всех элементов в пределах области действия элемента с атрибут "xmlns":

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny">
  <desc>Demonstrates use of a default namespace
    prefix for elements.</desc>
  <rect width="7" height="3"/>
</svg>
```

Если префикс пространства имён указан на атрибуте "xmlns"(например, xmlns:svg="http://www.w3.org/2000/svg") тогда соответствующее пространство имён не будет являться пространством имён по умолчанию, поэтому элементам должен быть присвоен явный префикс пространства имён [2]:

```
<?xml version="1.0"?>
<s:svg xmlns:s="http://www.w3.org/2000/svg"
  version="1.2" baseProfile="tiny">
  <s:desc>Demonstrates use of a namespace prefix
    for elements.
  Notice that attributes are not
    namespaced</s:desc>
<s:rect width="7" height="3"/>
```

```
6     </s:svg>
```

1.1.3 Формат JSON

Краткая характеристика

Название:	json
Также известен как:	JSON, JavaScript Object Notation
Тип данных:	Текстовый
Разработчик:	Дуглас Крокфорд
Поддерживающ. приложения:	текстовые редакторы/обзорщики, специализированное ПО
Использование:	API, Web-сервисы, обмен большим объёмом данных
Комментарии:	Простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером.

Обзор JSON основан на подмножестве языка программирования JavaScript, определенного в стандарте ECMA-262 3rd Edition - December 1999. JSON - текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными.

JSON основан на двух структурах данных:

1) Коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, запись, структура, словарь, хэш, именованный список или ассоциативный массив,

2) Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Это универсальные структуры данных. Почти все современные языки программирования поддерживают их в какой-либо форме. Логично предположить, что формат данных, независимый от языка программирования, должен быть основан на этих структурах.

Организация файла

Следующий пример показывает JSON-представление данных об объекте, описывающем человека. В данных присутствуют строковые поля имени и фамилии, информация об адресе и массив, содержащий список телефонов. Как видно из примера, значение может представлять собой вложенную структуру.

```
{
    "firstName": "Ivan",
    "lastName": "Ivanov",
    "address": {
        "streetAddress": "Moskovskoe sh., 101,
          kv.101",
        "city": "Leningrad",
        "postalCode": 101101
    },
    "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
    ]
}
```

1.2 Способы описания геометрии дуг

Дуги можно описывать различными способами. Разработчики САПР ищут наиболее выгодные из них с целью удобства дальнейшей работы с примитивами.

В AutoCAD для описания дуг используются следующие параметры:

— координата центра по Ox,
— координата центра по Oy,
— радиус,
— начальный угол,
— конечный угол.

При этом, в AutoCAD условно принято, что дуги рисуются по часовой стрелке.

Для описания геометрии дуги в SVG используются следующие параметры:

— координата начала по Ox,
— координата начала по Oy,
— радиус,
— флаг большой/малой дуги,
— флаг направления дуги,
— координата конца по Ox,
— координата конца по Oy.

Как можно заметить, для описания дуг требуется достаточно много параметров, обрабатывать которые не всегда удобно.

Поэтому AutoCAD разработали собственный формат описания геометрии дуг (внутри полилиний) в DXF. Он содержал координаты центра дуги и параметр выпуклости (bulge).

Параметр «bulge»

Особый интерес представляет параметр *bulge* (выпуклость) для каждой из вершин полилинии. Чтобы понять сущность данного параметра, который представляет собой степень кривизны дуги окружности между двумя точками, необходимо сначала разобраться с геометрией дуг.

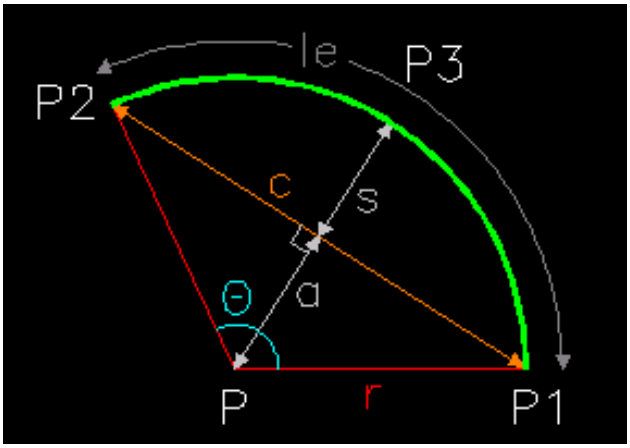

Рисунок 1.1 — Геометрия дуги окружности

Так как дуга окружности описывает часть этой окружности, то она и обладает всеми атрибутами данной окружности (см. рис. 1.1). Среди них:

— Радиус (r) — радиус дуги такой же, как и у окружности;
— Центр (P) — тот же, что и у окружности;
— Центральный угол (Θ) — в окружности равен 360°;
— Длина дуги (le) — является частью периметра (длины) окружности.

Для дальнейшей работы с геометрией дуг примем, также, следующие специфичные атрибуты:

— Начальная и конечная точка ($P1, P2$) — это «вершины» дуги. Хотя иногда и целесообразно говорить о конкретных точках, не лежащих на концах дуги;

— Длина хорды (c) — у дуг и окружностей можно провести бесконечное количество хорд, но для нас интерес представляет только хорда, проходящая через её вершины;

— Середина дуги ($P3$) — точка, делящая дуги с данными вершинами на две, равные по длине, дуги;

— Апофема (a) — это отрезок, вершинами которого являются середина дуги и её центр. Апофема перпендикулярна хорде;

— Высота дуги (s) — это отрезок, проведённый из середины дуги перпендикулярно к хорде.

Кроме самой себя, дуга может, также, и описывать другие геометрические формы: круговой сегмент и сектор. Обе геометрические формы включают в себя все вышеперечисленные атрибуты, однако для выведения формулы параметра *bulge* (выпуклости), потребуется рассмотрение только кругового сектора.

В документации AutoCAD [3] выпуклостью называется тангенс четверти угла дуги между выбранной вершиной и следующей вершиной в списках вершин полилиний. Отрицательность параметра *bulge* указывает на то, что дуга отрисовывается по часовой стрелке от выбранной вершины к следующей. Выпуклость, равная нулю — прямой сегмент, выпуклость, равная единице — половина окружности.

Проблема «расшифровки» атрибутов дуги для дальнейших манипуляций с ней заключается в том, что входными данными являются только координаты вершин и рассматриваемый параметр — *bulge*.

В самом деле, взяв арктангенс от параметра *bulge* и умножив его на 4, легко получить центральный угол, на который опирается рассматриваемая дуга. Результат получен в радианах. Для перевода значения в градусы, необходимо умножить это значение на π и разделить на 180°.

Для вывода данной зависимости, рассмотрим дугу окружности (см. рис. 1.2).

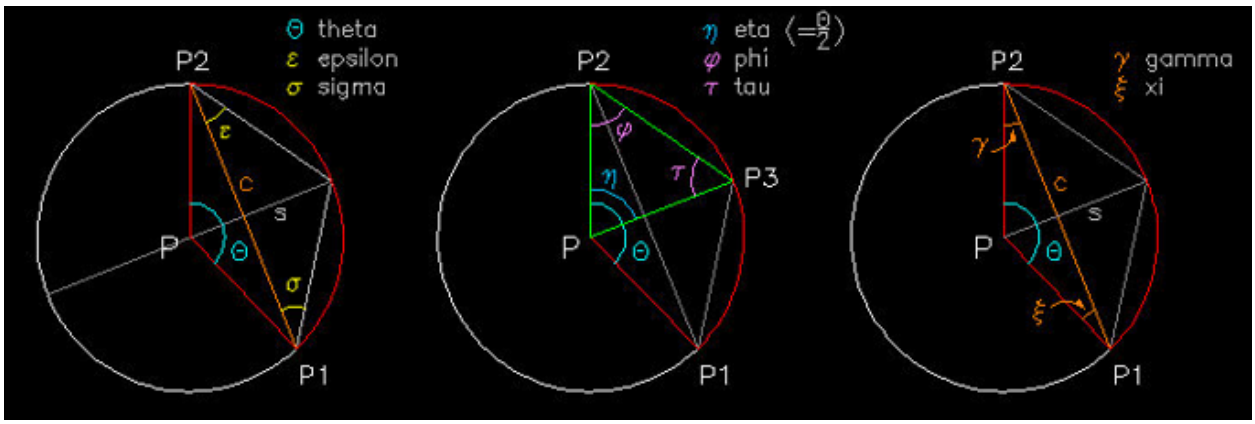

Рисунок 1.2 — Дуга окружности с проведённой хордой и углами при ней

Если провести к углу Θ биссектрису, то получится синий угол η. В итоге, мы получим равнобедренный треугольник (зеленый), в котором углы φ и τ равны. Поскольку сумма углов в треугольнике всегда равна 180° градусам, мы теперь знаем, что углы φ и τ равны следующему (1.1):

$$\varphi = \tau = \frac{(180^\circ - \frac{\Theta}{2})}{2} \Rightarrow \varphi = 90^\circ - \frac{\Theta}{4} \tag{1.1}$$

Теперь посмотрим на хорду c, проведённую от $P1$ до $P2$. Вместе с красными катетами угла Θ она тоже образует равнобедренный треугольник, а значит, $\gamma = \xi$. Угол при вершине треугольника $P - P1 - P2$ — это центральный угол Θ, поэтому γ и ξ вычисляются следующим образом (1.2):

$$\gamma = \xi = \frac{180^\circ - \Theta}{2} \Rightarrow \gamma = 90^\circ - \frac{\Theta}{2} \tag{1.2}$$

Таким образом, желтый угол ε должен быть равняться разнице между фиолетовым углом φ и оранжевым углом γ. Другими словами, ε — это четверть центрального угла Θ (1.4):

$$\varepsilon = (90^\circ - \frac{\Theta}{4}) - (90^\circ - \frac{\Theta}{2}) \Rightarrow \varepsilon = \frac{\Theta}{2} - \frac{\Theta}{4} = \frac{\Theta}{4} \tag{1.3}$$

Параметр *bulge* (выпуклость) описывает, насколько дуга «выпирает» из вершин, то есть насколько велика высота дуги (s) (или расстояние

от $P3$ до $P4$). Высота образует катет прямоугольного треугольника с углом, равным четверти центрального угла (см. желтый треугольник $P-P2-P3$ на рис. 1.3), и поскольку тангенс описывает отношение между катетами в прямоугольном треугольнике, легко описать геометрию с помощью этого одного угла (1.4):

$$\frac{\sin(\varepsilon)}{\cos(\varepsilon)} = \tan(\varepsilon) \tag{1.4}$$

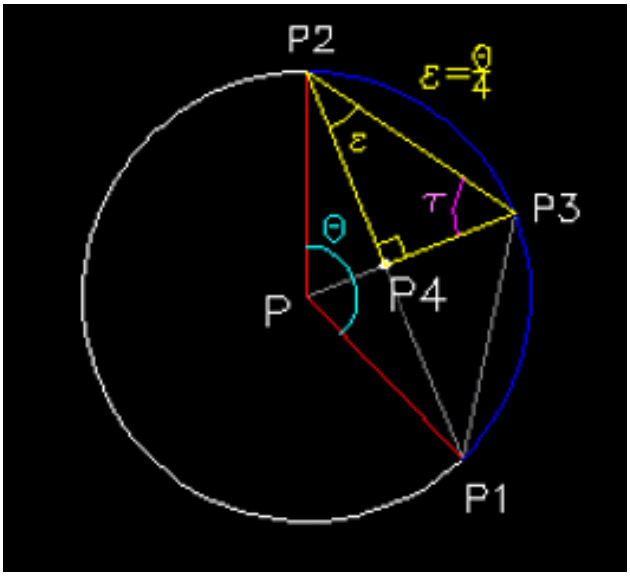

Рисунок 1.3 — Связь угла ε с центральным углом

Мы, также, могли бы найти тангенс угла ε, просто разделив противолежащий катет на смежный катет — что означает высоту дуги s, делённую на половину длины хорды c, — но не зная s и уже имея тангенс ε, полезнее найти s (1.5):

$$s = \frac{c}{2} \cdot \tan(\varepsilon) \tag{1.5}$$

Примем

$$\tan(\varepsilon) = bulge \tag{1.6}$$

Тогда

$$s = \frac{c}{2} \cdot bulge \tag{1.7}$$

Таким образом, радиус дуги может быть найден следующим образом (1.8):

$$r = \frac{(\frac{c}{2})^2 + s^2}{2s} \tag{1.8}$$

Знак той или иной выпуклости важен для определения дуги относительно вершин. Если выпуклость положительна, это означает, что дуга идёт против часовой стрелки от начальной вершины до конечной вершины. Если выпуклость отрицательна, это означает, что дуга идет, наоборот — по часовой стрелке.

Поэтому все приведенные выше формулы должны касаться абсолютного значения выпуклости, а не фактического значения, иначе можно получить отрицательный радиус.

Итак, поняв, что $bulge = tan(\frac{\Theta}{4})$, в согласовании с документацией AutoCAD [?] примем, что *bulge* положителен, когда при передвижении от начальной точки дуги к конечной движение происходит против часовой стрелки.

Ясно, что когда $\Theta = 0$, то и $bulge(\Theta) = 0$. Для углов в 180° условно принимается, что $bulge(\Theta) = \pm 1$. В случае, когда $\Theta = 90°$, получим следующее (1.9):

$$bulge(90°) = \tan(\frac{90°}{4}) = \tan(\frac{\pi}{8}) \tag{1.9}$$

Используя зависимость для тангенса половинного аргумента (1.10):

$$\tan(\frac{\alpha}{2}) = \pm\frac{\sin(\frac{\alpha}{2})}{\cos(\frac{\alpha}{2})} = \pm\frac{2\sin^2(\frac{\alpha}{2})}{2\sin(\frac{\alpha}{2})\cos(\frac{\alpha}{2})} = \pm\frac{1-\cos(x)}{\sin(x)} \tag{1.10}$$

Для $\alpha = \frac{\pi}{8}$ получим (1.11):

$$bulge(90^\circ) = \tan(\frac{\pi}{8}) = \pm\frac{1-\cos(\frac{\pi}{4})}{\sin(\frac{\pi}{4})} = \pm\frac{1-\frac{\sqrt{2}}{2}}{\frac{\sqrt{2}}{2}} = \pm\frac{1-\frac{1}{\sqrt{2}}}{\frac{1}{\sqrt{2}}} = \pm(\sqrt{2}-1) \tag{1.11}$$

В результате, математические данные совпадают с документацией AutoCAD [3] и гласят, что

а) $bulge = 0$ для отрезка прямой,

б) $bulge = \pm 1$ для дуги в 180° (половина окружности),

в) $bulge = \pm(\sqrt{2}-1) \approx 0.41421...$ для четвертей окружностей, когда угол раствора дуги равен 90°.

1.3 Актуальное состояние проблемы использования файловых форматов в области САПР

САПР используют множество разных форматов хранения и передачи данных. Проблема заключается в унификации форматов с целью сокращения их числа и снижении нагрузки с процессов переконвертации и обработки различных форматов данных для получения необходимой информации внутри САПР.

Хотя задача полной унификации используемых форматов файлов вряд ли может быть полностью решена для современных ПП, однако, в последние десятилетия развития технологий разработки программного обеспечения накоплены подходы, позволяющие значительно снизить остроту проблем, прежде всего за счёт продуманного использования открытых форматов хранения и обмена данными. При этом важны как функциональность формата, то есть то, какие именно данные он содержит, так и организация, то есть представление хранимых данных.

Например, для хранения и обмена геометрической информацией в САПР «Сириус» используется унаследованный двоичный формат DBS. Однако сложность чтения двоичного формата, неудобство хранения геомет-

рической информации, а также актуальность акцент формата на экономии памяти препятствуют эффективной работе по обмену информацией между ПО. По этим причинам программистами САПР «Сириус» принято решение заменить данный формат другим — более простым и удобным JSON.

На разных этапах как научных исследований, так и технологической подготовки производства, возникает потребность визуализации разнообразной геометрической информации, такой как геометрия деталей и ограничивающих их контуров, положение допустимых точек врезки и выключения инструмента, маршруты, получаемые в ходе решения различных классов задач резки, а также маршруты движения резака, получаемые после обработки постпроцессором и т.п.

Разработка в этих целях специальных графических утилит является традиционным подходом. Альтернативой, как пишет к.т.н. Уколов [4], является визуализация путём экспорта в удобочитаемый и поддерживаемый широкораспространёнными приложениями формат, например, в SVG. Векторные изображения, хранящиеся в этом формате, можно открывать с помощью любых современных браузеров; для формата доступно большое количество готовых библиотек; а также, формат кросс-платформенный, что означает возможность обозрения файлов данного типа на большинстве платформ и операционных систем.

В целях хранения промежуточных геометрических данных для дальнейшей обработки, а также для контроля содержания необходимых (поддерживаемых) примитивов (объектов) в DXF-файле, в рамках данной работы разработан новый формат хранения данных в текстовом документе — TXT(DXF-type), который описан в разделе 1.4. Также, с помощью данного формата может производится расчёт длины траектории контура детали (обычно, в поперечном её сечении). Это может быть полезно при применении ПО в области лазерной резки с помощью станков с ЧПУ, а, в частности, в САПР «Сириус».

В рамках этой работы был, также, изобретён формат хранения данных в текстовом файле — TXT(x,y,r) в виде координат и радиуса (описание в разделе 1.4). Он применяется для автоматизированного технологического проектирования, для формирования УП. Информация в данном формате о примитивах изображения контура детали используется для непосредственного составления УП, так как каждая последующая точка имеет не только плоские координаты, но и способ достижения этой точки (тип примитива: отрезок, если радиус равен нулю; дуга, если радиус ненулевой).

В целом, разрабатываемый набор конвертеров (модуль экспорта) представляет собой цельный ПП, сочетающий в себе набор необходимых разработчику УП начальных функций для автоматизированного технологического проектирования. Это ПО может быть интегрировано в разные ПП, так как по сути универсально в своём применении (используется в области 2D-резки, токарной обработке).

1.4 Постановка задачи

Необходимо разработать алгоритмы и приложение по конвертации данных из формата **DXF** в форматы **TXT (DXF-type)**, **TXT(x,y,r)**, **SVG**, **JSON**.

Формат TXT (DXF-type) должен содержать названия объектов (сущностей) с аттрибутом в скобках, под которыми они отображаются в DXF-формате (LINE, ARC и т.д.) в отдельных строках, после каждой из которых указываются основные параметры этих объектов. Для линий указываются координаты начала и конца. Для дуг и окружностей — координаты центра и радиус. Для полилиний — координаты точек, соединённых линиями. Например:

```
1 LINE(#38)
2 156.732  67.105
3 124.332  67.105
4 ARC(#75)
5 -3.0     15.0     5.0
```

```
CIRCLE(#2B9)
0.0  0.0  20.0
LWPOLYLINE(#2A9)
0.0  0.0  10.0  0.0  10.0  10.0  0.0  10.0
```

Формат выходного файла TXT(x,y,r) содержит в каждой строке по три параметра: координата абсциссы точки, координата ординаты и радиус перехода от данной точки к следующей (последний *bulge* DXF-файла, при незамкнутом контуре полилинии, не имеет смысла). Пример содержания такого файла:

```
158.33  59.61  0
108.33  59.61  0
108.33  78.61  0
118.33  68.11  0
120.33  66.11  2.00
```

Формат SVG стандартизован (см. раздел 1.1.2). Он должен корректно отображать поддерживаемое содержание входного DXF при открытии любым из доступных способов (с помощью Интернет-браузера, например).

Формат JSON также стандартизован (раздел 1.1.3), однако после конвертации файлы в этом формате должны содержать данные по двум тэгам: *partid* и *paths*, например:

```
[{
    "partid": "LIST",
    "paths": [
    [
    [0, 0, 0],
    [0, 2000, 0],
    [5000, 2000, 0],
    [5000, 0, 0],
    [0, 0, 0]]
    ]},{
    "partid": "00112",
    "paths": [
    [
    [510, 1150, 0],
    [510, 585, 0],
```

```
16      [10, 585, 0],
17      [10, 1150, 0],
18      [510, 1150, 0]],
19      [
20      [322, 1045, 1],
21      [448, 1045, 1],
22      [322, 1045, 0]],
23      [
24      [72, 1045, 1],
25      [198, 1045, 1],
26      [72, 1045, 0]],
27      [
28      [197, 785, 1],
29      [323, 785, 1],
30      [197, 785, 0]]
31      ]}
```

Здесь в тэге *pathid* указывается наименование детали, а в тэге *paths* — примитивы, задающиеся по типу TXT(DXF-type).

Схема работы приложения представлена на рисунке 1.4.

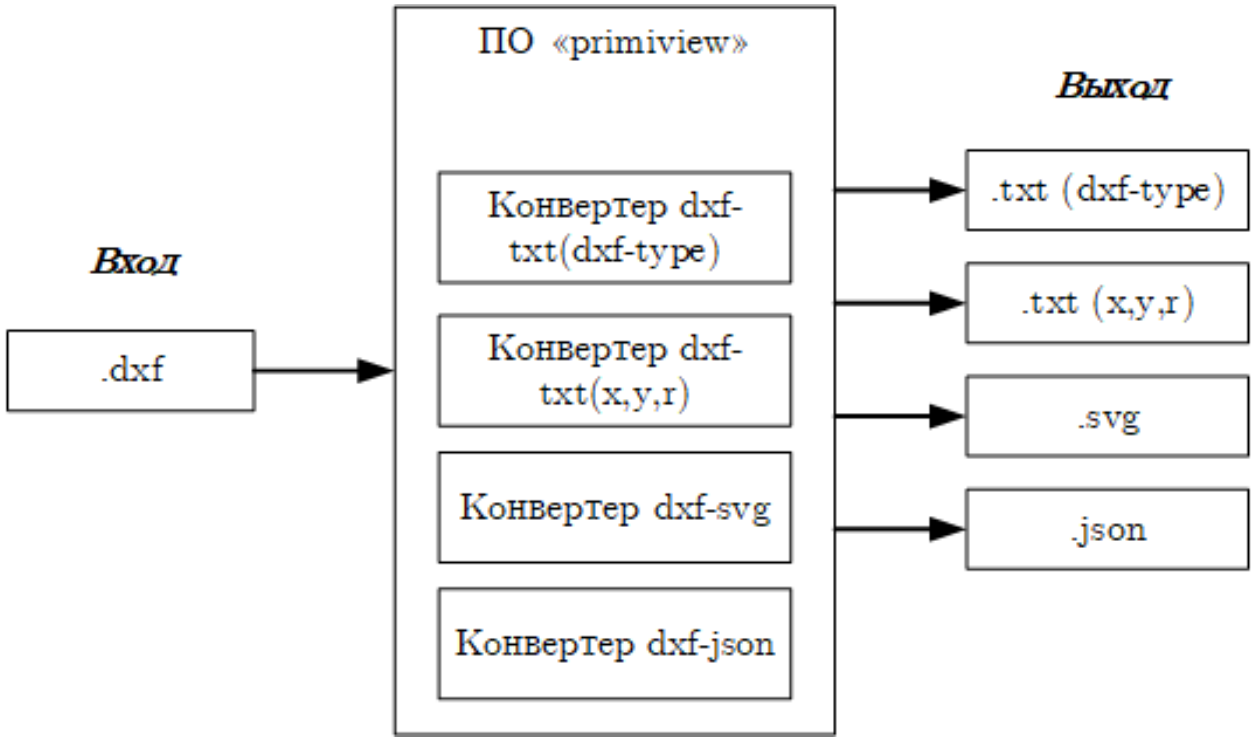

Рисунок 1.4 — Схема работы разрабатываемого ПО

Кроме конвертации данных, необходимо предусмотреть в приложении визуализацию прочитанных из DXF-файла геометрических данных для проверки корректности чтения их программой (т.е. для верификации).

Главные требования ко входному DXF-файлу

Так как данная работа нацелена на создание ПО для обработки геометрической информации 2D-объектов **специального типа**, то конвертироваться из DXF-файлов должна не вся информация, содержащаяся в них. В первую очередь, заказчиком работы было определено, что на входе будет подаваться **2D-контур** деталей типа «Втулка», то есть **тел вращения**. Так как сконвертированная геометрия данных объектов в последующем предполагает разработку УП для токарных станков с ЧПУ, то геометрическая информация должна содержать определённый набор геометрических примитивов, с которым может работать система исполнительных органов станков с ЧПУ. Этот набор ограничивается тем, что исполнительные органы станков с ЧПУ способны перемещаться либо с помощью **линейной**, либо с помощью **круговой интерполяции**. Из этого следует, что для корректной работы САПР, для которых предназначаются разрабатываемые конвертеры, геометрия в DXF-файле на входе конвертеров должна состоять из, как минимум одного из представленных далее примитивов:

а) линия (отрезок),
б) полилиния,
в) дуга,
г) окружность.

Остальные типы геометрии, реализуемой в формате DXF, такие как *эллипс*, *сплайн*, будут игнорироваться ПО.

На вход разрабатываемому конвертеру подаётся файл формата DXF. Формат DXF представляет собой совокупность данных с тегами всей информации, содержащейся в файле чертежа AutoCAD. Тегированные дан-

ные означают, что каждому элементу данных в файле предшествует целое число, называемое групповым кодом. Значение группового кода указывает, какой тип данных имеет следующий элемент. Это значение также указывает смысл элемента данных для данного типа объекта. Практически вся указанная пользователем информация в файле чертежа может быть представлена в формате DXF [3]. В DXF файлах, в зависимости от их содержания, существуют сущности, представляющие для нас интерес. Среди них следующие:

а) LINE (Линия),
б) LWPOLYLINE (Полилиния),
в) ARC (Дуга),
г) CIRCLE (Окружность),
д) INSERT (Вставка).

Как уже и было отмечено, существуют и другие примитивы (ELLIPSE, SPLINE и др.), однако, основываясь на конкретных целях заказчика по возможности применения выходных файлов для генерации УП, ПП проектируется только с указанными примитивами и сущностями DXF.

Рассмотрим каждую из сущностей подробнее.

LINE. Рассмотрим тэги сущности *Линия*, необходимые для её реального отображения (см. табл. 1.4).

Таблица 1.4 — Рассматриваемые групповые коды сущности LINE

Групповой код	Описание
39	Толщина (необязательный; по умолч. = 0)
10	Начальная точка (в с.к. объекта) DXF: значение X
20, 30	DXF: Y и Z значения начальной точки (в с.к. объекта)
11	Конечная точка (в с.к. объекта) DXF: значение X
21, 31	DXF: Y и Z значения конечной точки (в с.к. объекта)

LWPOLYLINE. Рассмотрим тэги сущности *Полилиния*, необходимые для её реального отображения (см. табл. 1.5).

Таблица 1.5 — Рассматриваемые групповые коды сущности POLYLINE

Групповой код	Описание
70	«Флаг» полилинии (бит-закодировано); по умолч. = 0; 1 – закрыта
39	Толщина (необязательный; по умолч. = 0)
10	Координаты вершин (в с.к. объекта), множественные вхождения; по одному вхождению для каждой вершины DXF: значение X
20	DXF: значение Y координат вершин (в с.к. объекта), множественные вхождения; по одному вхождению для каждой вершины
42	*Bulge*. Выпуклость (множественные вхождения - для каждой вершины), (необязательно; по умолч. =0)

ARC. Рассмотрим тэги сущности *Дуга*, необходимые для её реального отображения (см. табл. 1.6).

Таблица 1.6 — Рассматриваемые групповые коды сущности ARC

Групповой код	Описание
39	Толщина (необязательный; по умолч. = 0)
10	Центр дуги (в с.к. объекта) DXF: значение X
20, 30	DXF: Y и Z значения центра дуги (в с.к. объекта)

Продолжение на след. стр.

Продолжение таблицы 1.6

40	Радиус
50	Начальный угол
51	Конечный угол

CIRCLE. Рассмотрим тэги сущности *Окружность*, необходимые для её реального отображения (см. табл. 1.7).

Таблица 1.7 — Рассматриваемые групповые коды сущности CIRCLE

Групповой код	Описание
39	Толщина (необязательный; по умолч. = 0)
10	Центр дуги (в с.к. объекта) DXF: значение X
20, 30	DXF: Y и Z значения центра дуги (в с.к. объекта)
40	Радиус
50	Начальный угол
51	Конечный угол

INSERT. Данная сущность представляет собой вставку блоков с геометрией. Её необходимо рассматривать, так как геометрия может быть вложенной и, таким образом, не видна обзорщиком сущностей, так как вложена. У этой сущности поиск информации по тэгам в программе не потребуются.

1.5 Выводы по главе 1

Анализ состояние вопросов конвертации данных и графических файлов DXF показал, что:

— Задача по конвертации данных из разных форматов с целью обмена графической информацией в САПР сложна и требует, как специальных

знаний в области изучаемых форматов, так и опыт в разработке алгоритмов и ПО по ним для САПР;

— Существующие предложения открытого рынка по ПП, конвертирующим файлы, не удовлетворяют потребностям, предъявляемым к выходным форматам данных;

— Для корректной работы программы, разработчику алгоритмов и ПО необходимы знания геометрии примитивов, с которыми ведётся работа.

2 Разработка файловых конвертеров в виде приложения «primiview»

Программное обеспечение решено назвать **«primiview»**, где **«primi»** — сокращение от англ. *primitive* (*примитив*), **view** — в переводе с англ. *обзор*.

В данном разделе описаны экономические аспекты проекта по созданию ПО «primiview» для обработки геометрической информации 2D-объектов специального типа.

2.1 Выбор языка программирования

Для написания программы выбран язык программирования Python версии 3.11. Выбор языка программирования обоснован несколькими факторами.

Библиотеки. Для создания описанного ПП необходимо привлечение различных библиотек. Кроме стандартной библиотеки, с Python можно использовать множество прикладных библиотек, несколько из которых будут описаны в следующих частях работы. Специфичные библиотеки, позволяющие «читать» DXF-файл и обрабатывать содержимое внутри него, написаны не для каждого ЯП. Библиотека «eazy dxf» для Python позволяет быстро и удобно выполнять данные операции.

Кроссплатформенность. Большинство программ, написанных на Python, выполняются корректно на всех основных платформах. Перенос программы между операционными системами реализуется простым копированием кода. Кроме того, в процессе разработки ПО, для реализации пользовательского интерфейса используется набор расширений Qt, который тоже работает на таких платформах, как Linux и другие UNIX-подобные ОС, macOS и Windows.

Скорость и удобство разработки. Удобочитаемсть, ясность и высокое качество этого языка позволяют повысить производительность разработчика во много раз, сравнивая, например, с компилирующими или строго типизированными языками, такими как C, C++ и Java. Объём программного кода на языке Python обычно составляет треть или даже пятую часть эквивалентного программного кода на языке C++ или Java. Кроме того, при запуске программы, написанной на ЯП Python минуются длинные этапы компиляции и связывания, необходимые в некоторых других ЯП, что, также, увеличивает производительность труда программиста [5].

В ходе сравнительного анализа языков программирования для использования особое внимание уделялось чтению DXF. В тоткрытом доступе были найдены библиотеки для Python (*ezdxf*), C#(*netDxf*), для C++ и других языков были найдены исходные коды, позволяющие обрабатывать DXF-файлы.

В итоге, учитывая вышеприведённые аспекты-преимущества языка Python, а также относительно большой опыт работы по написанию ПО на этом языке, в сравнении с другими, учитывая крайне высокую степень разработки библиотеки *ezdxf* было принято решение использовать язык Python и библиотеку *ezdxf* для разработки ПО.

2.2 Принцип работы приложения «primiview»

Для создания программного обеспечения необходимо сначала разработать концепцию функционирования программы, опираясь на её назначение, на основные её функции. Когда определены модули, блоки и функциональные части ПО, можно приступать к разработке его на выбранном ЯП.

Основываясь на цели, поставленной во введении, разрабатываемая утилита должна принимать на входе DXF-файл, то есть открывать его и обрабатывать его содержимое. Для проверки правильности обработанных

данные, то есть, для верификации содержимого DXF-файла, программа должна визуализировать для пользователя обработанное. После верификации обработанных данных и, соответственно, подтверждения соответствия их исходным, пользователю должна предоставляться возможность конвертировать эти данные в какой-либо из предлагаемых форматов. За это отвечает модуль экспорта, который, в свою очередь, подразделяется на четыре модуля, отвечающие за преобразование данных в различные форматы. Среди них следующие:

а) Модуль экспорта в TXT-файл, где данные будут представлены в такой же форме, как и в оригинальном DXF-файле, за исключением того, что содержаться в нём будут только поддерживаемые сущности (LINE, POLYLINE, ARC, CIRCLE).

б) Модуль экспорта в TXT-файл, в котором поддерживаемые сущности будут представлены сочетанием двух строк, первая из которых — начальная точка примитива, вторая — конечная. Вторая строка содержит в себе радиус скругления примитива, переходящего из первой точки во вторую.

в) Модуль экспорта в формат SVG, который, при открытии, векторно отображает информацию в нём.

г) Модуль экспорта JSON-формат. В данном формате, по подобию формату TXT (x, y, r), содержаться точки, олицетворяющие начало и конец того или иного примитива. В данном случае информация в файле тэгированная, что означает, что в дальнейшем несложно будет получить желаемые куски данных из потенциально объёмного JSON-файла путём обращения по желаемому тэгу.

Схему, отображающую основное содержание разрабатываемого ПО, можно наблюдать на рисунке 2.1.

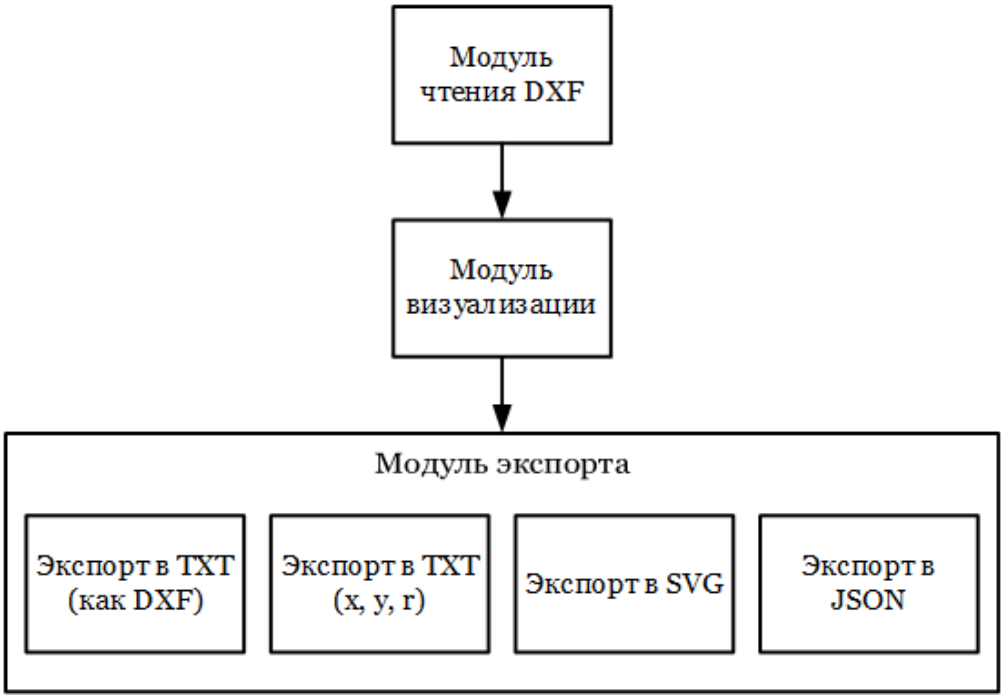

Рисунок 2.1 — Принципиальная структура ПО «Primiview»

2.3 Внутрепрограммная репрезентация информации о геометрических примитивах

Учитывая то, что проблема с работой по чтению входных DXF-файлов решена с помощью библиотеки *ezdxf*, задача разработки ПО фактически свелась к внутренней респезентации необходимой геометрической информации внутри программы для дальнейшей работы с ней.

Есть несколько вариантов хранения геометрических данных в программе на ЯП Python: списки, словари и классы.

В списках обычно хранятся объекты одного типа (например, только координаты в виде чисел). Это не удовлетворяет потребности обмена данными, так как нужная геометрическая информация из DXF содержит в себе и строковые, и булевы значения, а также другие типы данных.

Словари обеспечивают более простой и понятный доступ к полям, чем списки (то есть не по числовым индексам, а по ключам). Однако сло-

вари имеют некоторые ограничения, которые могут оказаться существенными по мере разработки программы.

Во-первых, в словарях не предусмотрено место для централизованного хранения логики обработки записей. Это допустимо в разрабатываемом ПО, так как функции работы с геометрическими примитивами в частях программы имеют одинаковый характер, поэтому не требуется прописывать поведение каждого класса объектов [6].

Однако недостаток использования словарей для представления записей, заключающийся в том, что со временем их становится трудно расширять, является решающим при выборе типа данных для хранения информации о геометрических примитивов. Так как ПО «primiview» требуется постоянно модифицировать для работы с новыми файловыми форматами, то основные изменения исходят из взятия из исходных DXF-файлов дополнительной информации для её последующей обработки. Поэтому сложность дополнения словарей новой информацией недопустима при разработке данного ПО.

Кроме прочих преимуществ хранения данных с помощью классовых свойств, ценна возможность создания списков внутри классов, которые позволяют объединять однородную по типу информацию, что упрощает её дальнейшую обработку.

Таким образом, учитывая вышеприведённые аргументы, а также следуя принципам объектно-ориентированного программирования (ООП) в разработке ПО, принято решение хранить каждый примитив в виде объекта класса этих примитивов.

Предусмотрен отдельный класс данных (*DxfData*), работающий с примитивами, извлечёнными из DXF. Он содержит в себе списки объектов соответствующих классов (линия, дуга, др.), а также методы (функции внутри класса) по работе с ними.

Схема, отображающая структуру хранения данных из DXF-файлов внутри программы приведена на рисунке 2.2.

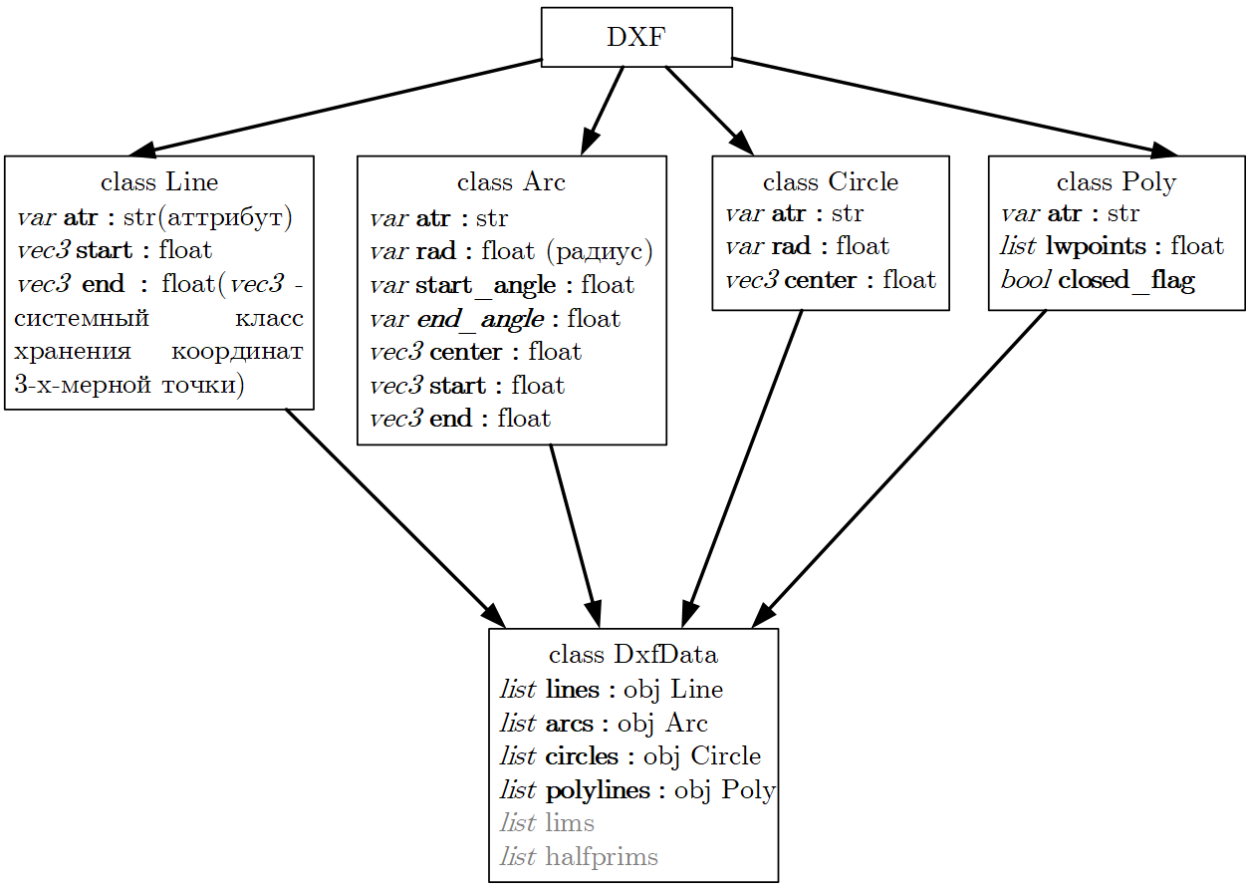

Рисунок 2.2 — Структура хранения данных в ПО «primiview»

2.4 Разработка алгоритмов

Алгоритм 1 показывает схему работы процесса извлечения поддерживаемых ПО Primiview примитивов из выбранного DXF-файла.

На первой итерации осуществляется разбиение блоков, в которых могут быть «спрятаны» остальные сущности. В случае, если пропустить данный этап, то объекты, находящиеся внутри блоков не будут видны библиотекой *ezdxf*, которая используется для чтения DXF-файлов.

После «разрушения» всех блоков примитивы становятся «видимыми». Далее запускается цикл, итерирующий объекты пространства объектов модели (*modelspace*). В случае совпадения объекта с одной из поддерживаемых сущностей, она сохраняется в список соответствующих объектов в оперативной памяти программы. Списком далее будет называться

изменяемый упорядоченный тип данных, представляющих собой последовательность элементов, разделённых между собой запятой и заключённых в квадратные скобки. Данный тип данных используется в ЯП Python, поэтому для соблюдения общности, термин будет применяться и в части **??** этого раздела.

Примем ряд условных обозначений:

msp — пространство объектов модели (modelspace);

$\dashrightarrow$ — запись объекта в множество.

Исходные данные: путь к DXF-файлу

Результат: массивы примитивов и информации о них в оперативной памяти программы

```
1  инициализация;
2  цикл ∀ сущн. типа Вставка (INSERT) ∈ msp выполнять
3  |  разбиение сущности
4  конец цикла
5  цикл ∀ сущн. ∈ msp выполнять
6  |  если сущ явл. линией тогда
7  |  |  сущн. --→ множ. линий
8  |  конец условия
9  |  иначе если сущн. явл. дугой тогда
10 |  |  сущн. --→ множ. дуг
11 |  конец условия
12 |  иначе если сущн. явл. окружностью тогда
13 |  |  сущн. --→ множ. окружностей
14 |  конец условия
15 |  иначе если сущн. явл. полилинией тогда
16 |  |  сущн. --→ множ. полилиний
17 |  конец условия
18 конец цикла
```

Алгоритм 1 — Сохранение поддерживаемых примитивов из DXF в оперативную память программы

Алгоритм записи примитивов в TXT (как DXF). Принцип данного алгоритма (см. алгоритм 2) основан на открытии созданного TXT-файла, а после — перебора прочитанных из DXF примитивов и записи из каждого из них необходимой информации в открытый для редактирования TXT-файл.

Записи в текстовом файле должны выглядеть следующим образом (см. листинг 2.1):

Листинг 2.1 — Пример содержания TXT-файла (как DXF)

```
1    LINE(#01)
2    0.1  0.1
3    0.1  0.1
```

Исходные данные: путь к имя.txt

Результат: имя.txt (как DXF)

```
1  инициализация;
2  цикл ∀ LINE ∈ множ. линий выполнять
3  |  записать в файл: атрибут сущности, x_0, y_0, x_1, y_1; перевести
   |   курсор на новую строку
4  конец цикла
5  цикл ∀ ARC ∈ множ. дуг выполнять
6  |  записать в файл: атрибут сущности, x_0, y_0, x_1, y_1, r;
   |   перевести курсор на новую строку
7  конец цикла
8  цикл ∀ CIRCLE ∈ множ. окруж. выполнять
9  |  записать в файл: атрибут сущности, x_c, y_c, r; перевести
   |   курсор на новую строку
10 конец цикла
11 цикл ∀ LWPOLYLINE ∈ множ. полилин. выполнять
12 |  записать в файл атрибут сущности;
13 |  цикл ∀ точки LWPOLYLINE выполнять
14 |  |  цикл ∀ координаты x, y выполнять
15 |  |  |  записать в файл значение координаты
16 |  |  конец цикла
17 |  конец цикла
18 |  перевести курсор на новую строку
19 конец цикла
```

Алгоритм 2 — Запись примитивов в TXT (как DXF)

В алгоритме 2 x_0, y_0 — координаты начала примитива; x_1, y_1 — координаты конца примитива; x_c, y_c — координаты центра окружности; r — радиус дуги или окружности.

Алгоритм записи примитивов в TXT (x,y,r). Алгоритм 4 призван, так же как и в прошлом случае, в открытый только что созданный

текстовый файл записать информацию о примитивах, которые были прочитаны из выбранного DXF-файла.

Записи в текстовом файле должны выглядеть следующим образом (см. листинг 2.2):

Листинг 2.2 — Пример содержания TXT-файла (x, y, r)

```
1    1.52  1.86  0
2    1.12  2.08  0
3    1.16  2.04  4.0
4    ...
```

Полилиния может содержать в себе, как отрезки, так и дуги. В объектах LWPOINT сущности LWPOLYLINE степень искривления показывает параметр *bulge*, суть которого подробно описана в разделе **??**. Так как желаемый формат вывода информации о примитивах содержит именно радиус примитива, а не параметр искривления, то необходимо удобно получить радиус из *bulge*.

Для этого воспользуемся уже выведенной зависимостью [4] и применим её в принятых обозначениях (2.1):

$$R = |bulge + \frac{1}{bulge}| \cdot \frac{|A - Z|}{4}, \tag{2.1}$$

где A — начальная точка;

Z — конечная точка.

Примем, что $A(x_0,y_0), B(x_1,y_1)$. Тогда $AZ(x_1 - x_0; y_1 - y_0)$. Таким образом, длина вектора через декартовы координаты (2.2):

$$|A - Z| = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \tag{2.2}$$

Исходные данные: текущая точка, следующая точка

Результат: радиус сегмента полилинии

1 инициализация;

2 **если** *bulge (текущей точки)* $= 0$ **тогда**

3 $\quad r = 0$

4 **конец условия**

5 **иначе**

6 $\quad r = |bulge + \frac{1}{bulge}| \cdot \frac{\sqrt{(x_{nexP} - x_{prevP})^2 + (y_{nexP} - y_{prevP})^2}}{4}$

7 **конец условия**

Алгоритм 3 — Вычисление радиуса сегмента полилинии

Примем условные обозначения:

$\rightsquigarrow$ — запись в файл;

newline — перевод на новую строку.

Исходные данные: путь к имя.txt

Результат: имя.txt (x,y,r)

1 **для каждого** *LINE* ∈ *множ. линий* **выполнять**

2 | $x_0 \quad y_0 \quad 0 \rightsquigarrow$ имя.txt; $\quad x_1 \quad y_1 \quad 0 \rightsquigarrow$ имя.txt

3 **конец цикла**

4 **для каждого** *ARC* ∈ *множ. дуг* **выполнять**

5 | $x_0 \quad y_0 \quad 0 \rightsquigarrow$ имя.txt; $\quad x_1 \quad y_1 \quad r \rightsquigarrow$ имя.txt

6 **конец цикла**

7 **для каждого** *CIRCLE* ∈ *множ. окруж.* **выполнять**

8 | $x_c + r \quad y_c \quad 0 \rightsquigarrow$ имя.txt; `/* первая половина окруж. */`

9 | $x_c - r \quad y_c \quad r \rightsquigarrow$ имя.txt

10 | $x_c - r \quad y_c \quad 0 \rightsquigarrow$ имя.txt; `/* вторая половина окруж. */`

11 | $x_c + r \quad y_c \quad r \rightsquigarrow$ имя.txt

12 **конец цикла**

13 **для каждого** *POLYLINE* ∈ *множ. полилиний*

14 *prevPoint=None ;* `/* предыдущая точка */`

15 **выполнять**

16 | **для каждого** *LWPOINT* ∈ *множ. точек. полилинии*
| **выполнять**

17 | | **если** *prevPoint≠None* **тогда**

18 | | | r = алгоритм 3 (prevPoint, LWPOINT)
| | | $x(prevP) \quad y(prevP) \quad 0 \rightsquigarrow$ имя.txt

19 | | | $x(lwpoint) \quad y(lwpoint) \quad r \rightsquigarrow$ имя.txt

20 | | **конец условия**

21 | | $prevPoint = lwpoint$

22 | **конец цикла**

23 | **если** *контур замкнут* **тогда**

24 | | r = алгоритм 3 (prevPoint, LWPOINT)

25 | | $x(lwpoint_{\text{посл}}) \quad y(lwpoint_{\text{посл}}) \quad 0 \rightsquigarrow$ имя.txt

26 | | $x(lwpoint_{\text{перв}}) \quad y(lwpoint_{\text{перв}}) \quad r \rightsquigarrow$ имя.txt

27 | **конец условия**

28 **конец цикла**

Алгоритм 4 — Запись примитивов в TXT (x, y, r)

Алгоритм записи примитивов в SVG. Формирования файла типа SVG отличается от предыдущих двух, так как этот формат представляет собой язык разметки, а значит, имеет правила синтаксиса, грамматики и т.д. Это расширение языка разметки XML, поэтому в начале, в преамбуле, указывается версия XML, кодировка символов и указание синтаксическому анализатору об игнорировании любых объявлений разметки в определении типа документа.

Листинг 2.3 — Первая строка SVG-файлов

```
1   <?xml version="1.0" encoding="UTF-8"
      standalone="no"?>
```

Следующие две строки должны содержать определение типа документа (заголовок DOCTYPE), однако, данное объявление может оказаться источником ошибок при применении в браузере Mozilla Firefox. Поэтому вместо этого используется атрибут **baseProfile** со значением «full» внутри элемента <svg>.

Начиная с четвёртой строки объявляется корневой элемент <svg>:

Листинг 2.4 — Первая строка SVG-файлов

```
1   <svg version="1.1" width="100%" height="100%"
2   viewBox="102.1188828597992 -211.7921423734452
      50.000000000000014 26.0"
3   baseProfile="full"
4   xmlns="http://www.w3.org/2000/svg"
5   xmlns:xlink="http://www.w3.org/1999/xlink"
6   xmlns:ev="http://www.w3.org/2001/xml-events">
```

В листинге 2.4 присутствует необязательный элемент **viewBox**, который представляет собой параметр с четырьмя значениями, отделяемыми пробелами, определяющими квадратную рамку, в которой будет располагаться графика. Данный атрибут позволяет автоматически масштабиро-

вать изображение до размеров указанного контейнера, причём, без потери качества, так как графическая информация храниться и воспроизводится в векторном формате.

Первые два значение — минимальные координаты x и y рамки, в которой располагается изображение. Третье и четвёртое значения — соответственно, ширина и высота рамки, в которой находится изображение. Значения указываются в пикселях.

Таким образом, чтобы перенести данные из DXF в SVG, сначала определяются эти четыре значения. Алгоритмы для их определения: алгоритм 5, алгоритм 6 и алгоритм 7.

Исходные данные: списки примитивов с параметрами

Результат: список координат x, список координат y

1 инициализация; пустой список координат x, пустой список координат y **цикл** $\forall$ *LINE* $\in$ *множ. линий* **выполнять**

2 $\quad$ x_0,x_1 --→ список x

3 $\quad$ y_0,y_1 --→ список y

4 **конец цикла**

5 **цикл** $\forall$ *ARC* $\in$ *множ. дуг* **выполнять**

6 $\quad$ $(x_c + r)$,$(x_c - r)$ --→ список x

7 $\quad$ $(y_c + r)$,$(y_c + r)$ --→ список y

8 **конец цикла**

9 **цикл** $\forall$ *CIRCLE* $\in$ *множ. окруж.* **выполнять**

10 $\quad$ $(x_c + r)$,$(x_c - r)$ --→ список x

11 $\quad$ $(y_c + r)$,$(y_c + r)$ --→ список y

12 **конец цикла**

13 **цикл** $\forall$ *LWPOLYLINE* $\in$ *множ. полилин.* **выполнять**

14 $\quad$ **цикл** $\forall$*точки в множ. LWPOINTS* **выполнять**

15 $\quad\quad$ x --→ список x

16 $\quad\quad$ y --→ список y

17 $\quad$ **конец цикла**

18 **конец цикла**

19 вернуть список координат x и список координат y

Алгоритм 5 — Вычленение координат изображения из DXF в отдельные списки

Исходные данные: список координат x, список координат y

Результат: x_{MIN}, y_{MIN}

1 инициализация;

2 использовать алгоритм 5

3 вернуть x_{MIN}, y_{MIN} из списков стандартными функциями сортировки ЯП

Алгоритм 6 — Поиск наименьших координат изображения из DXF

Исходные данные: список координат x, список координат y

Результат: ширина и высота рамки изображения

1 инициализация; использовать алгоритм 5

2 определить $x_{MIN}, x_{MAX}, y_{MIN}, y_{MAX}$ из списков стандартными функциями сортировки ЯП

3 ширина $= x_{MAX} - x_{MIN}$

4 высота $= y_{MAX} - y_{MIN}$

5 вернуть значения ширины и высоты

Алгоритм 7 — Поиск длины и высоты изображения из DXF

Алгоритм разбиения полилинии В процессе разработки приложения возникает задача визуализации данных из DXF. Проблемы возникают только с объектом Полилиния, который задаётся в DXF компактно: с помощью параметра *bulge*. Библиотека отрисовки в используемом ЯП, однако, не позволяет задавать полилинии ни в каком виде. По этой причине полилинию необходимо разбивать на линии и дуги по алгоритму 8.

В алгоритме происходит перемена углов дуги местами, если параметр кривизны меньше нуля. Это происходит потому, что в библиотеке *matplotlib* направление отрисовки дуг, как и в AutoCAD, всегда по часовой стрелке. А значит, что регулировка «величины» дуги регулируется последовательностью указания углов её отрисовки.

Исходные данные: список Полилиний с lwpoints у каждой

Результат: линия/дуга

```
1  инициализация; для каждого полилинии выполнять
2      для каждого текущая т., след. т. выполнять
3          если bulge = 0 тогда
4              рисуй линию с соотв. коор.
5          конец условия
6          иначе
7              центр, радиус = алгоритм 9 (текущая т., след. т.)
8              нач.угол. = алгоритм 10 (тек.т., центр)
9              конеч.угол. = алгоритм 10 (след.т., центр) если
                bulge < 0 тогда
10                 поменять местами углы
11             конец условия
12             рисуй дугу с соотв. центром, радиусом и углами.
13         конец условия
14     конец цикла
15 конец цикла
```

Алгоритм 8 — Разбиение полилиний

В работе Уколова С.С. [4] выведена зависимость, связывающая параметр *bulge*, *комплексное* представление координат точек начала (A) и конца (Z) дуги с координатами центра (C) дуги:

$$C = \frac{(1 + ibulge)^2}{4ibulge} \cdot A - \frac{(1 - ibulge)^2}{4ibulge} \cdot Z \tag{2.3}$$

Исходные данные: тек.т., след.т.

Результат: коор.центра дуги, радиус

1 инициализация;

2 $A = x + iy$ (тек.т.)

3 $Z = x + iy$ (след.т.)

4 $C = \frac{(1+ibulge)^2}{4ibulge} \cdot A - \frac{(1-ibulge)^2}{4ibulge} \cdot Z$ ($bulge$ текущей точки, см. раздел 1.4)

5 $rad =$ алгоритм 3

6 верни C(действ.ч., мним.ч.), радиус

Алгоритм 9 — Вычисление координат центра и радиуса дуги

Угол между векторами ищется с помощью известной арктангенса от частного состовляющих y и x координат, соответственно. Результат находится в диапазоне $\pm\pi$, поэтому после этого берется остаток от деления на 360°.

Исходные данные: точка дуги, центр дуги

Результат: угол от полож. направ. Ox

1 инициализация;

2 $vector =$ точка дуги $-$ центр дуги

3 $vecOx$ — вектор положительного направления Ox

4 $\alpha = \frac{\arctan(\frac{vector}{vecOx}) \cdot \frac{180}{\pi}}{360}$

5 верни α

Алгоритм 10 — Угол между векторами

2.5 Разработка программного обеспечения

Разработка программы начинается на создании структуры взаимосвязи её файлов, на их предназначении.

2.5.1 Файловая структура

Проект разбивается по файлам по функциональному признаку на следующие:

— **«__init.py__»**. Файл инициализации, призванный запустить файл главного окна, открыть окно пользовательского интерфейса;

— **«main_window.py»**. Файл главного окна, создающий полотно визуализации содержимого DXF, инициирующий объект класса Buttons. Класс MainWindow наследуется от класса Ui_MainWindow;

— **«buttons.py»**. Файл, описывающий события, инициализирующиеся по нажатию кнопок на форме;

— **«scene.py»**. Файл отрисовки содержимого DXF на полотне пользовательского окна;

— **«filedata.py»**. Файл работы с объектами входного DXF;

— **«math_ops.py»**. Файл математического сопровождения работы программы. Сюда помещены описания крупных математических классов и функций;

— **«iconrsc_rc.py»**. Файл, автоматически создаваемый окружением pyqt для описания внешних графических элементов, использующихся в форме (иконок);

— **«main_dialog.py»**. Файл, содержащий описание взаиморасположения и свойств объектов диалогового окна.

Файловая структура программы представлена схемой на рисунке 2.3.

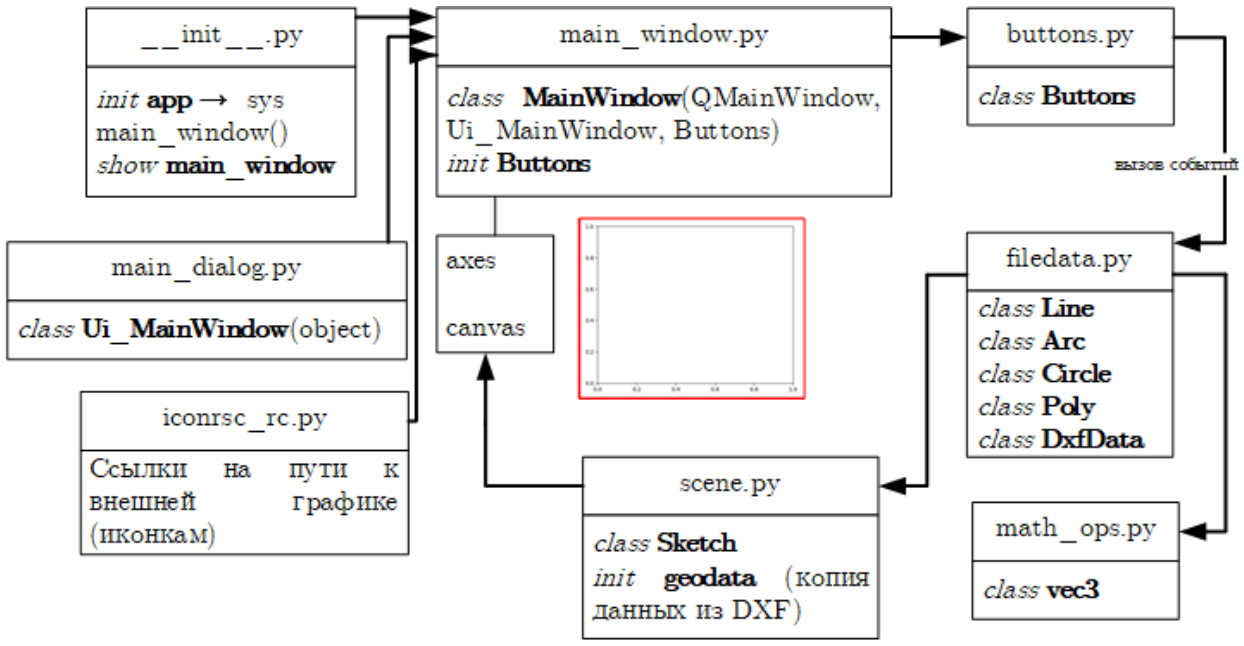

Рисунок 2.3 — Структура файлов и схема их взаимодействия в ПО «primiview»

2.5.2 Модуль визуализации

В целях верификации правильности чтения геометрических данных из входного DXF в программе предусмотрен модуль визуализации, который написан в отдельном файле «scene.py» (см. рис. 2.3.

В качестве инструмента используется библиотека *matplotlib*.

В рассматриваемом файле содержится класс Sketch, в котором описаны методы по работе с окном вывода графической информации приложения.

При инициализации объекта данного класса создаётся копия данных, извлечённых из выбранного DXF-файла.

При открытии каждого нового файла перерисовываются сетка и оси. Сетка перерисовывается, так как необходимо соблюдать масштаб изображения, таким образом, чтобы все объекты изображения попали в область видимости (в рамки окна вывода графики). Перерисовка осуществляется с помощью следующего метода:

```
def draw_grid(self, axes):
    axes.cla()
    axes.set_xlabel('X')
    axes.set_ylabel('Y')
    axes.margins(0.05)
    axes.set_aspect("equal")
    axes.grid()
    axes.set_axisbelow(True)
```

В целом, параметры, задающие геометрию через библиотеку *matplotlib*, совпадают с прямо доступными параметрами извлекаемыми из исходного DXF-файла.

Исключение составляет полилиния.

Проблема с визуализацией полилиний заключается в том, что в используемой библиотеке нет возможности задавать полилинии (только полигоны — замкнутые полилинии). Поэтому найдено решение проблемы — сведение полилиний к линиям и дугам.

Сведениие полилиний к отрезкам (линиям) и дугам заключается в попарном переборе пар точек полилинии. Данная операция осуществляется с помощью встроенной в ЯП функции *zip()*, которая создаёт итератор, объединяющий элементы из предыдущей и далее идущей точками полилинии. Реализация показана далее:

```
    for poly in self.geodata.polylines:
        for current, next in zip(poly.lwpoints[::], poly.lwpoints[1::]):
```

Таким образом, начинается перебор точек полилинии попарно, начиная с нулевого элемента, сравниваемого с первым элементом.

Реализуются алгоритмы 8, 9, 10, приведённые в разделе 2.4:

```
def polyarc_center_rad(self, current, next):
    A = complex(current[0], current[1])
    Z = complex(next[0], next[1])
    C = ((complex(1, current[4])) ** 2) / (complex(0, 4 * current[4])) * A -
        ((complex(1, -current[4])) ** 2) / (complex(0, 4 * current[4])) * Z
```

```
    rad = 0 if current[4] == 0 else (abs(current[4] + 1 / current[4]) *
        math.sqrt((next[0] - current[0]) ** 2 + (next[1] - current[1]) ** 2)
        / 4)
    return [C.real, C.imag], rad

def angle_vectors(self, end, start):
    u1 = np.array(end[:2]) - np.array(start[:2])
    angle = (math.atan2(u1[1], u1[0]) * 180 / math.pi) % 360
    return angle

for poly in self.geodata.polylines:
    for current, next in zip(poly.lwpoints[::], poly.lwpoints[1::]):
        if current[4] == 0:
            axes.add_line(Line2D([current[0], next[0]],
                                 [current[1], next[1]],
                                 color='g',
                                 lw=1.5))

        else:
            center, polyarc_rad = self.polyarc_center_rad(current, next)
            start_angle = self.angle_vectors(current, center)
            end_angle = self.angle_vectors(next, center)
            if current[4] < 0:
                start_angle, end_angle = end_angle, start_angle
            axes.add_patch(Arc((center[0], center[1]),
                                 width=2 * polyarc_rad,
                                 height=2 * polyarc_rad,
                                 theta1=start_angle,
                                 theta2=end_angle,
                                 color='g',
                                 lw=1.5,
                                 fill=False,
                                 alpha=1))
```

Для полилиний, также, выполняется проверка на замкнутость. Для этого в DXF есть специальное булево значение, которое извлекается при чтении программой входного файла. Если полилиния замкнута, то необходимо провести линию от последней точки, к первой:

```
if poly.closed_flag == 1:
    axes.add_line(Line2D([poly.lwpoints[0][0], poly.lwpoints[-1][0]],
```

```
3                               [poly.lwpoints[0][1], poly.lwpoints[-1][1]],
4                               color='g',
5                               lw=1.5))
```

2.6 Пользовательский интерфейс

2.7 Выводы по главе 2

fsfgsdfs

3 Экономическое обоснование эффективности проекта

В данном разделе описаны экономические аспекты проекта по созданию ПО «Primiview» для обработки геометрической информации 2D-объектов специального типа.

Целью экономического обоснования проекта является представление разработанного ПП в качестве проекта для реализации на предприятии, что позволит провести планирование и корректировку последовательности работ (при необходимости).

3.1 Разработка проекта

Цель проекта — создание модуля конвертации форматов, как отдельного ПП, для внедрения в ПО для автоматизации технологического проектирования обработки деталей типа «Втулка» на станках с ЧПУ к 05.06.2023.

Обоснование цели. Клиент — это производитель станков с ЧПУ. Для создания УП на продаваемые станках клиенты заказчика привлекают САПР, разработанные за рубежом, что негативно влияет на независимость от сторонних организаций, ограниченность в дополнении ПО функционалом, необходимым только на отдельном предприятии, а также, на сохранность информации, используемой при разработке УП. Компания работает над проектом разработки своей CAM-системы для предложения своим клиентам специализированного ПО вместе с оборудованием с целью решить вышеуказанные проблемы.

Численные критерии сравнения состояний системы клиента:

— сложность проектирования,
— сроки (время) проектирования,
— себестоимость проектирования,

— качество результатов проектирования.

Пояснения к критериям. Под сложностью проектирования понимается минимально-необходимая квалификация проектировщика для выполнения задач технологического проектирования.

Трудоёмкость проектирования, в данном случае, выражается в среднем времени создания (написания) одной УП.

Себестоимость проектирования оценивается не в расчёте на одну УП, а в рамках одного рабочего года. В себестоимость проектирования входят такие элементы, как зарплата сотрудника, производящего проектирование, а также, цена годовой лицензии/контракта обслуживания CAM-системы для данного количества оборудования на предприятии.

В качестве численного критерия для оценки качества результатов проектирования принято среднее количество типов ошибок, корректировки по которым оператор станка с ЧПУ вносит после того, как автоматизированное проектирование выполнено.

Текущее состояние системы клиента: основное ПО по автоматизации технологического проектирования введено в эксплуатацию, базовые потребности системы по конвертации DXF-файлов работают.

По численным критериям:

— сложность проектирования: инженер-технолог II категории по квалификационному справочнику [7],
— сроки (время) проектирования: 3 часа,
— себестоимость проектирования: 1420000 руб. (см. раздел 3.4.2),
— качество результатов проектирования: 10 ошибок.

Клиенты заказчика обращались с просьбами и пожеланиями, поставить им продукт, повышающий их эффективность по вышеуказанным критериям.

Клиенты заказчика испытывают трудности в удержании и поиске инженеров-технологов с высокой квалификацией (II категории), поэтому при возможности приобрести ПО, снижающее требования к квалификации инженеров-технологов, изъявили намерение купить такой ПП.

Несмотря на высокую квалификацию инженеров-технологов, занимающихся созданием УП, время проектирования сильно ограничивает мощностные способности клиентов заказчика. Заказчик ПО преследует интересы своих клиентов в сокращении времени проектирования ценой платы за специализированное ПО.

Клиенты заказчика стремятся повысить качество производимой продукции, поэтому нуждаются в дополнительных вложениях в соответствующие мероприятия. В качестве решения заказчик предложил снизить стоимость проектирования путём использования отечественного более дешёвого специального ПО.

Ещё одна причина заинтересованности клиентов заказчика в CAM-системе — качество результатов проектирования. Оно напрямую влияет как на качество самой производимой продукции, так и на количество бракованных изделий. Снизить частоту и характер ошибок призвана разработанная САПР.

Целевое состояние системы клиента: усовершенствованная, более гибкая и универсальная версия этого ПО.

По численным критериям:

— сложность проектирования: инженер-технолог (без категории) по квалификационному справочнику [7],

— сроки (время) проектирования: максимум 1,5 часа,

— себестоимость проектирования: максимум 1 млн.руб.,

— качество результатов проектирования: максимум 5 типов ошибок.

Результатом проекта является факт конвертации файлов из формата DXF в форматы TXT(тип DXF), TXT(x,y,r), SVG, JSON, созданные в

разработанном ПП. Файлы последних форматов содержат в себе геометрическое описание объектов по правилам, описанным в предыдущих разделах ВКР, из входного DXF, удобное для дальнейшей работы в CAM-системе "ТокКТЭ"и других САПР.

Команда проекта сформирована из 3 человек, среди которых:

а) 1 владелец,
б) 2 программиста, среди которых:
   б.1) 1 разработчик,
   б.2) 1 тестировщик.

Владелец проекта организует работу остальной команды, проводит планирование проекта, оценку его экономической эффективности, контроль за выполнением подчинёнными задач проекта.

Программисты занимаются непосредственно созданием продукта проекта, то есть написанием ПО. Разработчики отвечают за написание программного кода по техническому заданию проекта. Тестировщики выполняют проверку работоспособности ПП, ищут и сообщают отделу разработчиков о найденных и необходимых к устранению ошибок и недочётов программы.

3.2 Дерево задач проекта

Целью данного этапа является построение иерархического дерева, включающего в себя последовательное разбиение общей цели проекта на подцели и задачи.

3.2.1 Первый уровень иерархии.

Главной целью проекта, как уже было сказано, является разработка программного обеспечения «Primiview» по конвертации файлов в формате

DXF в форматы TXT, SVG, JSON. Данная цель в проекте единственная и находится на высшем уровне иерархии.

3.2.2 Второй уровень иерархии.

В целях определения и формализации цели, структуры и методов проекта, чтобы исключить неоднозначное их понимание и толкование исполнителями, первый этап, стоящий в иерархии на втором уровне, — это *формирование технического задания (ТЗ).*

При параллельном методе разработке, когда этапы проекта могут начинаться тогда, пока предыдущие ещё не закончились, следующим шагом данного уровня будет *разработка алгоритмов* программного обеспечения. Данный этап необходим, так как именно на нём ещё можно решить несостыковки в логической части программы, исправление которых на следующих этапах редко бывает возможным.

Параллельно с разработкой алгоритмов начинается этап непосредственно *разработки ПО.* Эти этапе происходят одновременно, так как они тесно взаимосвязаны, и, например, не зная на каком ЯП будет разрабатываться ПО, трудно будет рационально подобрать алгоритмы, отвечающие возможностям ЯП.

Завершающим этапом второго уровня является *сдача проекта* заказчику. Эта стадия может быть выполнена только при полном выполнении предыдущих стадий, если иное не было предварительно оговорено с заказчиком.

3.2.3 Третий уровень иерархии.

Этап формирования ТЗ подразделяется на следующие задачи:

а) Определение назначения ПО. Здесь формализуются цели и функции ПП. Впоследующем, они вносятся в ТЗ. Это смысл выполнения проекта, то, к чему стремится вся его команда, что хочет получить в итоге заказчик;

б) Исследование степени разработанности. На стадии предпроектного исследования выполняется проверка, существуют ли аналоги данного продукта в открытом доступе рынка. Если есть, то чем заказчика не устраивает их использование;

в) Требования к продукту. Определяются численные критерии, которым должен соответствовать результат проекта на этапе его сдачи;

г) Определение сроков. Всем участникам проекта необходимо знать, к какому сроку они должны выполнить определённый ранее объём работ. Сотрудникам, при согласовании ТЗ, необходимо оценить эти сроки, и в случае невозможности из соблюдения, просить корректировки и согласования с заказчиком более позднего выполнения задач;

д) Написание ТЗ. Завершающая стадия этапа формирования ТЗ — здесь собирается вся информация с предыдущих стадий и документируется согласно стандартам организации. ТЗ должно быть согласовано со всему участниками проекта.

Этап разработки алгоритмов подразделяется на следующие стадии:

а) Определение принципа работы ПО. Составляется принципиальная схема работы ПО, связь модулей, определяется предназначение каждого из модулей;

б) Разработка алгоритмов для модулей ПП. Происходит решение поставленных задач на уровне логики и математики, строится набор взаимосвязей алгоритмов. По возможности, рассматривается применение уже существующих универсальных (реже, специальный) алгоритмов.

Этап разработки ПО содержит такие стадии:

а) Выбор ЯП. Данная стадия подразумевает проведение сравнительного анализа существующих инструментов различных ЯП применительно к разрабатываемому ПП;

б) Определение структуры ПО. Эта стадия необходима для понимания разработчика функционала каждого из модулей программы. От этого зависит, на какие части (из каких файлов) будет состоять ПО;

в) Написание и отладка программного кода. Главная часть создания продукта. На этой стадии разработчики непосредственно пишут программный код и отлаживают его работу.

г) Тестирование ПО. Команда тестировщиков, также, пишет тестовый код, который проверяет разрабатываемое ПО на корректность работы его функционала.

Завершающий этап на втором уровне иерархии — сдача ПО, содержит следующие стадии:

а) Презентация. Команда проекта презентует результаты своей работы заказчику, демонстрирует работу программы. Отчитывается по выполнению всех этапов, указанных в ТЗ;

б) Внесение правок. По итогам презентации команда проекта заслушивает обратную связь от заказчика и на этой основе вносит в проект правки;

в) Передача ПО заказчику. Организатор проекта передаёт заказчику исходный код ПО, документацию и инструкции, требования, прочие исходные файлы, информацию, обеспечивающую доступ к ПП;

г) Обучение Сотрудников. При сдаче нового ПП, команде разработчиков необходимо обучить персонал заказчика работе с новым ПО;

д) Устранение недостатков. После обучения исполнитель проекта (организация) предоставляет заказчику 7 рабочих дней на проведение внутренних проверок ПО, по результатам которых заказчик предоставляет исполнителю список исправлений, которые команда проекта обязана скорректировать в разработанном ПО;

е) Закрытие задания на проект. Расчёт. Конечным мероприятием сдачи ПО являет закрытие сторонами задания по договору и получение расчёта за выполненную работую.

3.2.4 Четвёртый уровень иерархии.

Описание данного уровня иерархии приведено кратко для примера. В самом деле, каждая из стадий третьего уровня подразделяется на задачи четвёртого уровня.

Стадия исследования степени разработанности проблемы в этапе формирования ТЗ подразделяется на следующие задачи:

а) Исследование отечественного рынка аналогичных ПП. Задача состоит в поиске решений по аналогичным проектам в открытом доступе в рамках отчественного рынка;

б) Исследование зарубежного рынка аналогичных ПП. Данная задача отличается от предыдущей сложностью поиска аналогов (на зарубежном рынке(пространстве)), так как для проведения данного анализа необходим высокий уровень владения иностранным (английским) языком, а также, требуется знание достоверных ресурсов (источников) информации.

Описанные элементы иерархии сведены в иерархическое дерево (см. рисунок 3.1). Уровни иерархической структуры оформлены таким образом, что, чем конкретнее описаны элементы структуры, тем насыщеннее цвет заливки.

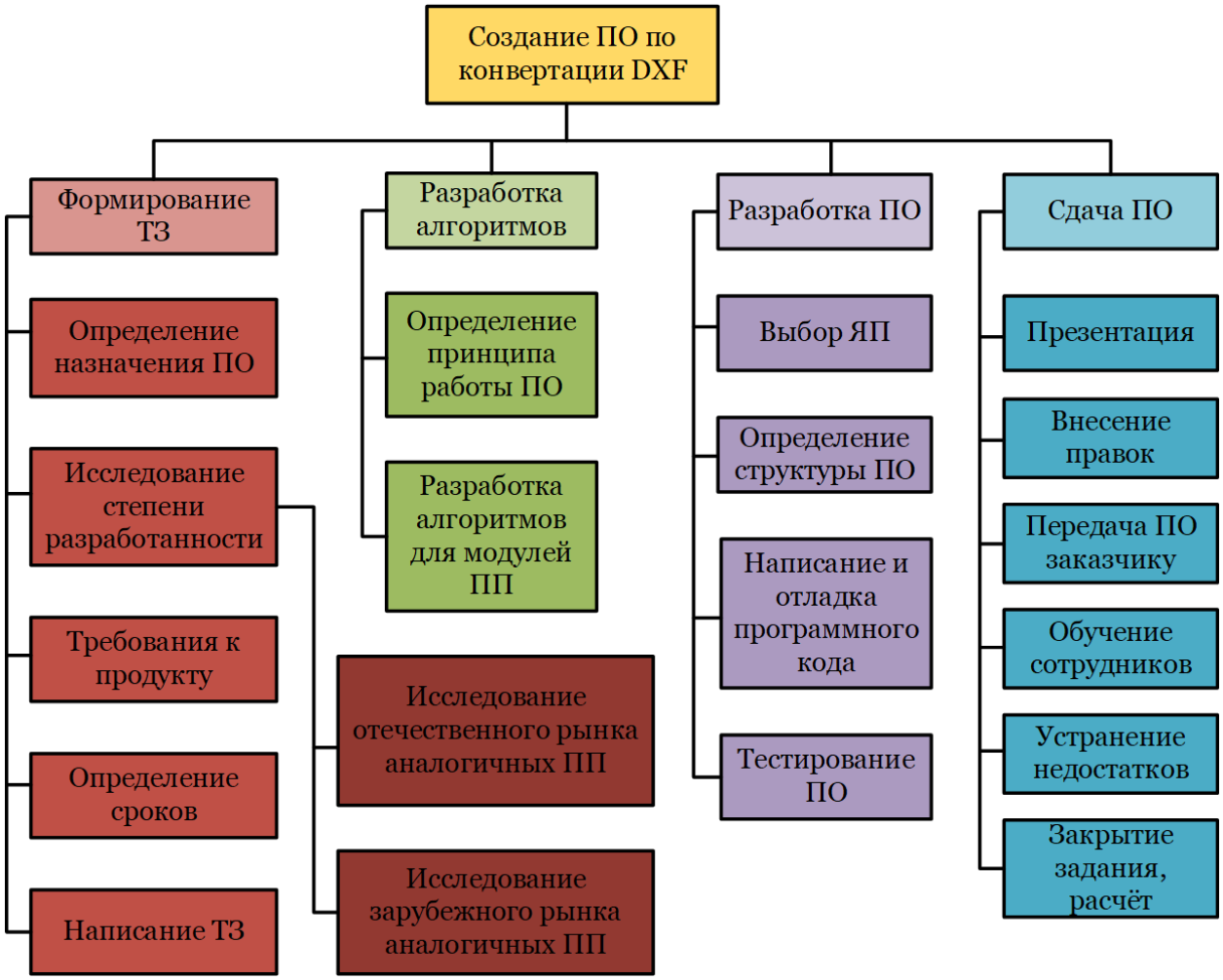

Рисунок 3.1 — Дерево задач проекта

3.3 Построение диаграмм проекта

В целях визуализации информации в области планирования проекта, а также для удобства, наглядности и гибкости контроля за выполнением задач проекта используются методы построения диаграмм и графов проекта.

3.3.1 Диаграмма Ганта

На данном этапе строится диаграмма Ганта (англ. Gantt Chart, также, ленточная диаграмма, график Ганта) — это тип столбчатых диаграмм, использующийся для иллюстрации плана, графика работ проекта. Изоб-

рёл такой тип диаграмм американский «прародитель» менеджмента Генри Лоренс Гант [8]. Также, является методом планирования проекта. Эта диаграмма представляет собой отрезки (графические плашки), размещающиеся на горизонтальной шкале времени. Каждый отрезок соответствует отдельно задаче (подзадаче). Начало, конец и длина отрезка на шкале времени соответствуют началу, концу и длительности задачи. Диаграмма может использоваться для представления текущего состояния выполнения работ: часть прямоугольника, отвечающего задаче, заштриховывается, отмечая процент выполнения задачи; показывается вертикальная линия, отвечающая моменту «сегодня».

Рядом с самой диаграммой располагается таблица со списком работ, строки которой соответствуют отдельным задачам, отображённым на диаграмме, в то время как столбцы содержат дополнительную информацию о задаче.

	Vorgangsname	Длите	Начало	Окончание	Пре
0	**Разработка ПО <<Primiview>>**	**230 дн**	**Вт 19.07.22**	**Пн 05.06.23**	
1	**Формирование ТЗ**	**14 дн**	**Вт 19.07.22**	**Пт 05.08.22**	
2	Определение назначения ПО	2 дн	Вт 19.07.22	Ср 20.07.22	
3	**Исследов. Степени разраб.**	**4 дн**	**Чт 21.07.22**	**Вт 26.07.22**	**2**
6	Определение требов. К продукту	2 дн	Ср 27.07.22	Чт 28.07.22	5
7	Определение сроков	2 дн	Пн 01.08.22	Вт 02.08.22	6
8	Написание ТЗ	3 дн	Ср 03.08.22	Пт 05.08.22	7
9	**Разработка алгоритмов**	**80 дн**	**Пн 08.08.22**	**Пт 25.11.22**	
10	Определение принципа работы ПО	5 дн	Пн 08.08.22	Пт 12.08.22	8
11	Разработка алгоритмов для модулей ПП	75 дн	Пн 15.08.22	Пт 25.11.22	10
12	**Разработка ПО**	**195 дн**	**Пн 08.08.22**	**Пт 05.05.23**	
13	Выбор ЯП	5 дн	Пн 08.08.22	Пт 12.08.22	8
14	Определение структуры ПО	5 дн	Пн 15.08.22	Пт 19.08.22	13
15	Написание и отладка программного кода	185 дн	Пн 22.08.22	Пт 05.05.23	14;10
16	Тестирование ПО	185 дн	Пн 22.08.22	Пт 05.05.23	14;10
17	**Сдача ПО**	**21 дн**	**Пн 08.05.23**	**Пн 05.06.23**	
18	Составление презентации и отчёта	4 дн	Пн 08.05.23	Чт 11.05.23	16;15
19	Внесение правок	5 дн	Пт 12.05.23	Чт 18.05.23	18
20	Передача ПО заказчику	1 дн	Пт 19.05.23	Пт 19.05.23	19
21	Обучение сотрудников	3 дн	Пн 22.05.23	Ср 24.05.23	20
22	Устранение недостатков	7 дн	Чт 25.05.23	Пт 02.06.23	21
23	Закрытие задания, расчёт	1 дн	Пн 05.06.23	Пн 05.06.23	22

Рисунок 3.2 — Диаграмма Ганта для проекта

3.3.2 Сетевой график

С целью точного определения плановых сроков завершения проекта и выявления потенциальных вариантов их сокращения, а также для выявления и планирования критического пути проекта строится сетевой график.

Сетевой график представляет собой ориентированный граф, изображающий все необходимые для достижения цели проекта операции в технологической взаимосвязи. Другими словами, это изображение комплекса работ с учётом их длительности, взаимосвязи и перечнем неделимых операций в порядке их выполнения.

Основной элемент сетевого графика — события, характеризующие завершение или начало работы. Оно обычно обозначается окружностью внутри которой располагается номер события (для сверки со справочной таблицей, прилагающейся к сетевому графику).

Стрелками на сетевом графике изображаются работы — реальные процессы или действия, требующие затрат труда, материалов и времени, и приводящие к событию, на которое обращена стрелка. Работы на сетевом графике обычно означают одно из следующего:

— Действительная работа — производственный или творческий процесс, требующий усилий, затрат труда, времени и материальных ресурсов;

— Ожидание — процесс, требующий только затрат времени без привлечения каких-либо ресурсов;

— Зависимость (фиктивная работы) — процесс, не требующий затрат труда, времени и ресурсов.

Продолжительность работы для проектов на глобальном уровне обычно измеряется, как минимум, днями. Однако на разной масштабности может также и быть оценена в часах, неделях, месяцах, годах. Работы

могут иметь количественные показатели, характеризующие трудоёмкость, стоимость, материальные ресурсы.

Временные показатели изображаются над стрелками, обозначение работ — под стрелками.

К сетевым графикам предъявляется требование, чтобы события были определены полно и чётко, их формулировка должна включать результат предшествующих ему работ. Если то или иное событие наступило, то это значит, что сразу могут быть начаты следующие работы.

Путь на сетевом графике — это непрерывная последовательность работ от исходного до завершающего события.

Критическим путём называется путь сетевого графика с наибольшей суммарной продолжительностью работ.

При построении сетевых графиков необходимо придерживаться следующих правил:

а) Последовательное изображение работ от начала к окончанию,

б) Изображение стрелок от предшествующего события к последующему,

в) Отсутствие пересечения стрелок,

г) Обозначение работ: между двумя событиями только одна стрелка,

д) Запрет на замкнутые контуры, тупики, хвостовые работы,

е) Изображение дифференциально-зависимых работ с помощью пунктирных стрелок,

ж) Изображение поставки-результата внешней работы, используемого в данной работе,

з) Кодирование событий.

Согласно описанным принципам построения сетевых графиков был разработан сетевой график для проекта по разработке ПО «Primiview» для конвертирования файлов. На рисунке 3.3 изображена сама сеть. Для расшифровки событий и работ см. таблицы 3.1 и 3.2.

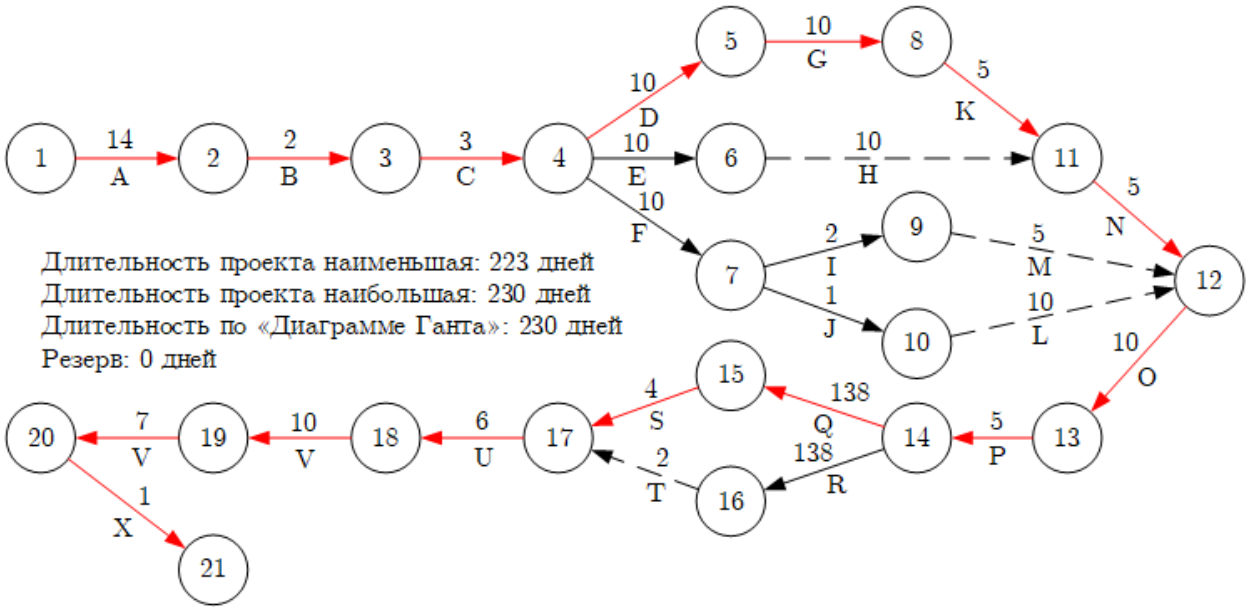

Рисунок 3.3 — Сетевой график проекта

Таблица 3.1 — Работы в сетевом графике проекта

Обозн.	Наименование этапа	Продолж. (раб. дней)	Предш. этап
A	Принятие командой проекта решений по назначению ПО, исследование аналогов, определение требований и сроков	14	-
B	Определение файловых потоков	2	A
C	Определение взаимосвязи файлов ПО	3	B
D	Сравнение концепций хранения данных внутри ПО, схематизация и моделирование вариантов реализации алгоритма, проверка алгоритма на ошибки	10	C
E	Вывод зависимостей между доступными параметрами дуг и необходимой геометрией дуг, разработка алгоритма	10	C

Продолжение на след. стр.

Продолжение таблицы 3.1

F	Разработка алгоритма по перебору и внутрепрограммному сохранению координат изображения	10	C
G	Разработка алгоритма на основе созданной структуры хранения примитивов	10	D
H	Разработка алгоритма с использованием созданных алгоритмов вычленения необходимой информации из DXF-файлов	10	E
I	Применение встроенных в ЯП методов сортировки, применение предыдущего алгоритма	2	F
J	Выполнение операций над краевыми значениями массивов координат	1	F
K	Разработка алгоритма конвертации с использованием радиуса сегмента полилиний	5	G,H
L	Разработка алгоритма конвертации с использованием альтернативного метода описания дуг	10	J
M	Разработка алгоритма с использованием встроенных в ЯП алгоритмов сортировки	2	I
N	Разработка алгоритма с использованием извлечённых из DXF Геометрических данным для визуализации изображения	5	H,K
O	Разработка алгоритма конвертации DXF B JSON с объединением примитивов в контуры	10	L,M,N
P	Сравнительный анализ ЯП в применении к поставленным требованиям к ПО	5	O

Продолжение на след. стр.

Продолжение таблицы 3.1

Q	Написание кода программы и его отладка	138	P
R	Тестирование функционала разработанного ПО	138	Q
S	Подготовка документа-отчёта, а также составление презентации с подготовкой команды проекта к демонстрации разработанного ПП	4	R
T	Подготовка презентации и отчёта с незавершёнными тестами ПО	2	Q
U	Заслушивание обратной связи заказчика после презентации. Составление списка замечаний к исправлению	6	S,T
V	Проведение обучения сотрудников заказчика к пользованию ПП. Передача ПО заказчику. Предоставление времени на внутреннюю проверку нового ПО	10	U
W	Приём списка замечаний сотрудников заказчика после периода внутренней апробации ПП	7	V
X	Обновление переданного ПО заказчику. Завершающая встреча заказчика и владельца проекта для подписания оформленных к завершению проекта документов	1	W

Таблица 3.2 — События в сетевом графике проекта

Обозн.	Наименование события	Предш. событие
1	Начало проекта	-
2	Техническое задание	1
3	Схема работы ПО	2
4	Структура ПО	3
5	Алгоритм внутрепрограммного сохранения примитивов	4
6	Алгоритм вычисления радиуса сегмента полилинии	4
7	Алгоритм вычленения координат из объектов DXF	4
8	Алгоритм конвертации DXF→TXT(DXF-type)	5
9	Алгоритм поиска наименьших координат изображения	7
10	Алгоритм поиска ширины и высоты рамки объектов DXF	7
11	Алгоритм конвертации DXF→TXT(x,y,r)	6,8
12	Алгоритм конвертации DXF→SVG	9,10,11
13	Алгоритм конвертации DXF→JSON	12
14	Утверждённый руководителем проекта ЯП	13
15	Файлы проекта с исходным кодом	14
16	Отчёт о прохождении тестов	15
17	Презентация и отчёт проекта в эл.виде	16
18	Список правок	17
19	Результаты тестирования сотрудников	18
20	Список устранённых недостатков	19
21	Подписанное о закрытии задание проекта. Счёт на оплату. Чек об оплате	20

3.4 Сравнительная экономическая эффективность

Расчеты сравнительной экономической эффективности капитальных вложений (инвестиций) применяются при сопоставлении нескольких возможных для осуществления вариантов инженерных решений: при решении задач по выбору взаимозаменяемых материалов, внедрению новых видов техники, модернизации оборудования, способов организации производственных процессов и т. п. То есть для оценки решений, которые являются альтернативными для обеспечения одинаковых конечных результатов деятельности. При этом конечные результаты (производство конкретной продукции с определенными характеристиками в заданном объеме) уже известны, есть необходимость определить, какой способ ее изготовления на том или ином этапе деятельности предприятия является более выгодным.

3.4.1 Исходные данные

Станкостроительное предприятие рассматривает заказ на создание программного обеспечения для своего оборудования (токарных станков с ЧПУ). Это ПО автоматизирует процесс создания управляющих программ для станков с ЧПУ, взамен работе инженера-технолога-программиста, который, обычно, берёт чертёж детали и либо вручную пишет УП, либо использует иностранные CAM-системы, предварительно создавая 3d-модель по выданному чертежу. В рамках данной (третьей) части ВКР будет рассматриваться инвестиционный проект (ИП) с точки зрения покупателя оборудования у предприятия, которое привлекло силы университета для создания описанного ПО. Сравниваются два варианта – покупка станков без ПО и, соответственно, с ним.

Сравнительная характеристика вариантов. Рассмотрим ситуацию с точки зрения покупателя оборудования рассматриваемого станкостроительного предприятия. Соберём основные данные в таблицу 3.3.

Таблица 3.3 — Сравнительная характеристика вариантов ИП

Вариант 1	Вариант 2
Покупка станка без ПО	Покупка станка вместе с ПО (CAM-системой) за большую цену
Наём во время этапа подготовки производства инженеров-технологов-программистов для написания УП	Привлекаются технологи из имеющегося штата сотрудников для выполнения дополнительных обязанностей по контролю создания УП с помощью купленного ПО
Время написания УП в три раза дольше, чем во втором варианте	Время написания УП в течение одного часа (в среднем)

Варианты рассматриваются с точки зрения потребителя оборудования. За сопоставляемые характеристики принимаются следующие:

— Объём производства (серийное),
— Частота создания УП в год (200 новых УП в среднем).

Выбор единичного периода времени. В качестве единичного периода времени для расчётов примем один год, так как на рассматриваемом предприятии-клиенте ситуация с производством каждый месяц практически не меняется. Также, большинство справочных величин ссылаются именно на годовой период, что тоже является подтверждением равномерной распределённости экономических характеристик внутри отдельно взятых месяцев.

Состав и описание капитальных вложений по вариантам. В капитальные вложения входят следующие величины:

— Цена станка с без ПО – 2 750 000 руб.,
— Цена встроенной CAM-системы на единицу оборудования– 70 000 руб.,

— Наладка полной группы станков – 20 000 руб.

Принятие решения по нормативному сроку окупаемости и его обоснование. Соответствует требованиям к сроку окупаемости дополнительных капитальных вложений, в данном случае – в токарный станок с ЧПУ.

Срок полезного использования оборудования – 10 лет.

Срок контракта на выпуск продукции с использованием данного оборудования – в рассматриваемой ситуации нет ограничений, токарная обработка постоянно проводится на предприятии.

Требования собственника, инвестора – предприятие установило желаемый срок окупаемости – 5 лет.

Следовательно, задаём Тн (нормативный срок окупаемости) равным 5 лет, так как временные рамки требований инвестора меньше срока полезного использования оборудования.

Определение состава затрат по вариантам (результат – перечень затрат). Корректировка затрат в соответствии с возможностями Методики сравнительной эффективности (включаем в расчет только различающиеся по альтернативам затраты). Деление затрат на переменные и постоянные. Формирование списка исходных данных для выполнения расчетов (см. таблицу 3.4).

Таблица 3.4 — Исходные данные для расчётов текущих затрат

	Вариант 1	Вариант 2
Переменные затраты (на единицу объема деятельности (одну УП))		
Зарплата технолога, руб	1500	1500
Время на написание одной УП, час	3	1

Продолжение на след. стр.

Продолжение таблицы 3.4

Постоянные затраты (на единицу оборудования)		
Цена годовой лицензии/контракта обслуживания CAM-системы, руб	50000	55000

3.4.2 Расчёты и анализ

Так как выбран нормативный срок окупаемости, равный одному году, то к нему будут приведены расчёты по приведённым затратам.

Исходные данные.

Среднее годовое количество УП на предприятии-покупателе станков

$$N = 200 \text{ шт};$$

Срок полезного использования оборудования:

$$T_{machinery} = 10 \text{ лет};$$

Требования инвестора по окупаемости ИП:

$$T_{inv} = 5 \text{ лет};$$

Принятая норма окупаемости:

$$T_n = T_{inv} = 5 \text{ лет};$$

Наладка полной группы станков:

$$CAM_{Term} = 20000 \text{ руб};$$

Цена встроенной CAM-системы на единицу оборудования:

$$CAM_2 = 70000 \text{ руб};$$

Цена станка без встроенной CAM-системы:

$$M = 2750000 \text{ руб};$$

Цена годовой лицензии/контракта обслуживания CAM-системы на единицу оборудования для вариантов 1 и 2, соответственно:

$$CAM_{1Perm} = 50000 \text{ руб};$$

$$CAM_{2Perm} = 55000 \text{ руб};$$

Среднее время написания одной УП:

$$t_1 = 3 \text{ часа};$$

$$t_2 = 1 \text{ час};$$

Почасовая оплата технолога:

$$Sal = 1500 \text{ руб};$$

Страховые сбора от заработной платы:

$$fees = 30\%$$

Количество покупаемых станков:

$$N_M = 5 \text{ шт};$$

Расчёт.

Себестоимость использования оборудования и ПО:

$$C_1 = Sal \cdot t_1 \cdot N \cdot (100\% + fees) + N_M \cdot CAM_{1Perm} = \\ = 1500 \cdot 3 \cdot 200 \cdot (100\% + 30\%) + 5 \cdot 50000 = 1420000 \text{ руб}; \quad (3.1)$$

$$C_2 = Sal \cdot t_2 \cdot N \cdot (100\% + fees) + N_M \cdot CAM_{2Perm} = \\ = 1500 \cdot 1 \cdot 200 \cdot (100\% + 30\%) + 5 \cdot 55000 = 665000 \text{ руб}; \quad (3.2)$$

Условно-годовая экономия (на себестоимости):

$$E = |C_1 - C_2| = |1420000 - 665000| = 755000 \text{ руб;} \tag{3.3}$$

Капитальные вложения предприятия-покупателя станков:

$$K_1 = M \cdot N_M + CAM_{Term} = 2750000 \cdot 5 + 20000 = 13770000 \text{ руб;} \tag{3.4}$$

$$\begin{gathered} K_2 = N_M \cdot (M + CAM_2) + CAM_{Term} = \\ = 5 \cdot (2750000 + 70000) + 20000 = 14120000 \text{ руб;} \end{gathered} \tag{3.5}$$

Дополнительные капитальные вложения:

$$K_{extr} = |K_1 - K_2| = |13770000 - 14120000| = 350000 \text{ руб;} \tag{3.6}$$

$$K_{extr} = N_M \cdot CAM_2 = 5 \cdot 70000 = 350000 \text{ руб. (проверка);} \tag{3.7}$$

Срок окупаемости дополнительных капитальных вложений:

$$T_{payback} = \frac{K_{extr}}{E} = \frac{350000}{755000} = 0{,}464 \text{ лет;} \tag{3.8}$$

Приведённые затраты по вариантам:

$$Z_1 = C_1 + \frac{1}{T_n} \cdot K_1 = 1420000 + \frac{1}{5} \cdot 13770000 = 4174000 \text{ руб;} \tag{3.9}$$

$$Z_2 = C_2 + \frac{1}{T_n} \cdot K_2 = 665000 + \frac{1}{5} \cdot 14120000 = 3489000 \text{ руб;} \tag{3.10}$$

Годовой экономический эффект:

$$E_{annual} = |Z_1 - Z_2| = |4174000 - 34890000| = 685000 \text{ руб;} \tag{3.11}$$

Минимальный годовой объём деятельности, при котором обеспечивается приведённый годовой экономический эффект:

$$\begin{gathered} N_{cr} = \frac{N_M \cdot CAM_{1Perm} - N_M \cdot CAM_{2Perm} - \frac{K_{extr}}{T_n}}{S \cdot t_2 \cdot (100\% + fees) - Sal \cdot t_1 \cdot (100\% + fees)} = \\ = \frac{5 \cdot 50000 - 5 \cdot 55000 - \frac{350000}{5}}{1500 \cdot 1 \cdot 100\% + 30\% - 1500 \cdot 3 \cdot 100\% + 30\%} = 24{,}359 \text{ шт;} \end{gathered} \tag{3.12}$$

По вычисленным в формулах 3.9, 3.10 затратам изобразим на графике (см. рисунок 3.4) границы целесообразности рассматриваемых вариантов.

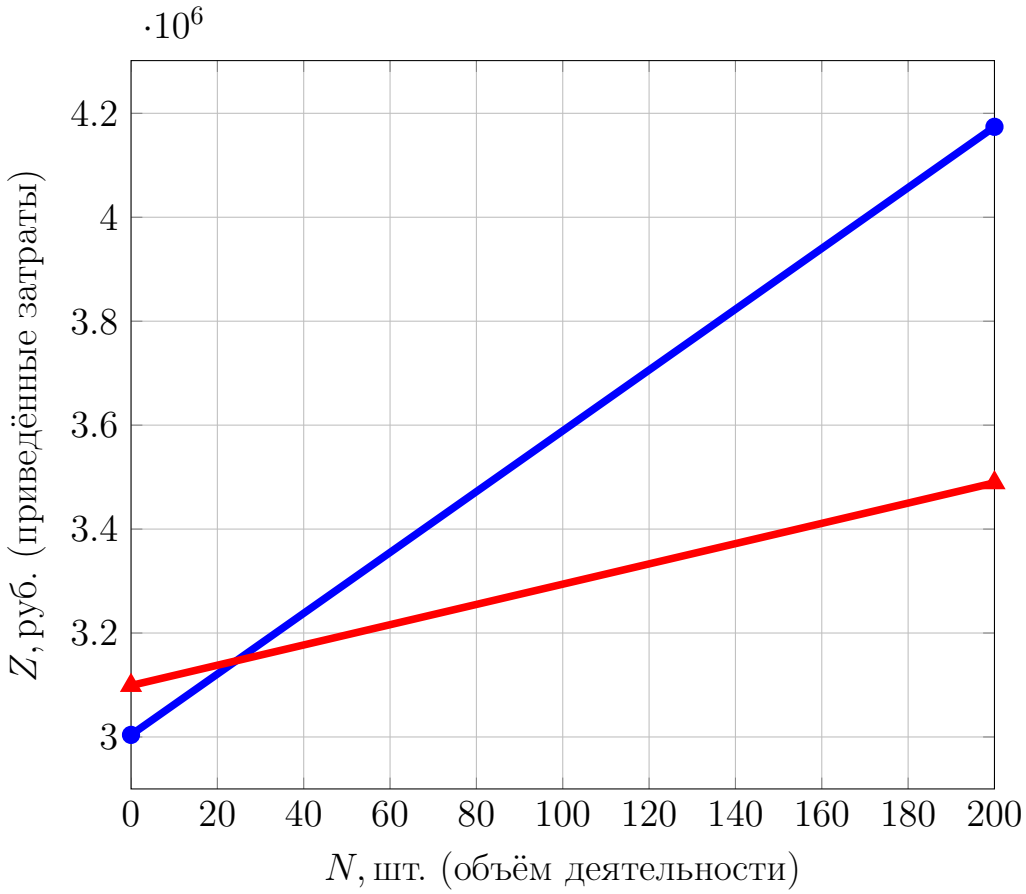

Рисунок 3.4 — Границы целесообразности рассматриваемых вариантов

Получив необходимые значения по критериям сравнения, сведём результаты в таблицу 3.5).

Таблица 3.5 — Сравнительная характеристика рассматриваемых вариантов по показателям эффективности

Наименование показателя	Ед. изм.	По вариантам:		Отклонения показателей
		Вариант без ПО	Вариант с ПО	
Годовой объем деятельности	шт.	200	200	-
Капитальные вложения, всего	руб.	13770000	14120000	350000
в том числе:				
Наладка станков	руб.	20000	20000	-
Цена станка (Вар. 2 + ПО)	руб.	13750000	665000	755000
Срок окупаемости дополнительных кап. вложений		0,464		
Приведённые затраты по вариантам	руб.	4174000	3489000	658000
Годовой экономический эффект			685000	

3.4.3 Выводы по результатам расчётов.

Так как на первых этапах расчёта по методу сравнительной эффективности ИП нельзя было сделать конкретный вывод по поводу целесообразности одного из предлагаемых вариантов по причине того, что по первому варианту себестоимость ИП была больше в сравнении со вторым, а капитальные вложения, соответственно, меньше, то расчёт был продолжен до момента вычисления расчётного срока окупаемости дополнительных капитальных вложений, а также расчёта приведённых затрат по каждому из вариантов.

Исходя из расчётов и построенного по ним графика, сделаем вывод, что, производя уже 25 УП за год, выгоднее становится вариант с ПО, так

как приведённые затраты для соответствующего количество производимых УП для этого варианта оказываются меньше.

Анализируя итоговые данные, выбираем для реализации второй вариант, то есть покупка оборудования вместе со встроенным ПО (CAM-системой), объясняя выбор тем, что расчётный срок окупаемости оказался намного меньше рассматриваемого нормативного срока окупаемости (0,4 и 5 лет, соответственно), а приведённые затраты по первому варианту оказались больше, чем по второму.

Действительно, экономия времени на создании УП нивелирует большие капитальные вложения на этапе инвестиционного периода УП.

3.5 Выводы по главе 3

Заключение

В результате проделанной работы стало ясно, что ничего не ясно...

Список использованных источников

1. *Murray, James D.* Encyclopedia of graphics file formats / James D Murray, William VanRyper // *Sebastopol: O'Reilly.* — 1996.

2. Scalable Vector Graphics (SVG) Tiny 1.2 Specification / Ola Andersson, Robin Berjon, Jon Ferraiolo et al. // *World Wide Web Consortium (W3C) recommendation.* — 2008. — Vol. 22.

3. *AutoCAD, Autodesk.* DXF Reference. — 2012.

4. *Уколов, Станислав Сергеевич.* Разработка алгоритмов оптимальной маршрутизации инструмента для САПР управляющих программ машин листовой резки с ЧПУ: диссертация на соискание ученой степени кандидата технических наук: 05.13.12: Ph.D. thesis / б. и. — 2021.

5. *Ascher, David.* Learning Python / David Ascher, Mark Lutz. — O'Reilly, 2004.

6. *Lutz, Mark.* Programming python / Mark Lutz. — "O'Reilly Media, Inc. 2001.

7. *Костин, Л.А.* Квалификационный справочник должностей руководителей, специалистов и других служащих / Л.А. Костин // *М: ЮРИТИ-ДАНА.* — 2014.

8. *Кларк, У.* Графики Ганта. Учёт и планирование работы. 5-е издание / У. Кларк // *Москва:«Техника управления.* — 1931.