МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**)»

филиал «РКТ» МАИ в г. Химки Московской области

Специальность 09.02.07 «Информационные системы и программирование»

ОТЧЁТ

по циклу лабораторных работ

по курсу «Технология разработки ПО»

Вариант 24

Выполнил:

Студент группы ИСП-41-19

\_\_\_\_\_\_\_\_\_\_\_\_/Шепелёв С.А./

Проверила: Жилина Т.А.

Химки, 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**)»

филиал «РКТ» МАИ в г. Химки Московской области

Специальность 09.02.07 «Информационные системы и программирование»

«УТВЕРЖДАЮ»

Руководитель лабораторных работ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жилина Т.А.

\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

ЗАДАНИЕ

на цикл лабораторных работ по курсу

«Технология разработки ПО»

студенту группы ИСП-41-19

Шепелёву С.А.

1) Исходные данные к лабораторной работе определяются согласно индивидуальному варианту: вариант 24.

2) Перечень вопросов, подлежащих разработке в ходе выполнения лабораторных работ, приведен в Приложении к заданию.

3) Срок завершения цикла лабораторных работ: 30 декабря 2022 г.

4) Дата выдачи задания: 21 октября 2022 г.

5) График выполнения цикла лабораторных работ:

|  |  |  |  |
| --- | --- | --- | --- |
| Лабораторная работа | Срок выполнения | Итоги проверки | |
| Отметка о выполнении | Подпись преподавателя |
| 1. Анализ требований к СПО |  |  |  |
| 2. Проектирование СПО |  |  |  |
| 3. Создание сложной структуры данных |  |  |  |
| 4. Разработка СПО с использованием принципов SOLID и паттернов проектирования |  |  |  |
| 5. Вызов ассемблерных функций из языка высокого уровня |  |  |  |
| 6. Организация доступа к данным путем объектно-реляционного отображения |  |  |  |
| 7. Внедрение структурной обработки исключений |  |  |  |
| 8. Оценка эффективности функционирования СПО |  |  |  |
| 9. Документирование СПО |  |  |  |

Задание приняла к исполнению:

\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / С.А. Шепелёв

**Лабораторная работа №1**

**«Разработка технического задания»**

1. Основные сведения

Программный документ «Техническое задание» разрабатывается в соответствии с ГОСТ 19.201—78. Техническое задание содержит совокупность требований к программному средству и может использоваться как критерий проверки и приемки разработанной программы, поэтому достаточно полно составленное (с учётом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком техническое задание является одним из основополагающих документов проекта. Умение грамотно создавать техническое задание на разработку программного продукта определяет профессиональный уровень программиста и избавляет его от претензий со стороны заказчика.

Техническое задание – документ, в котором формулируются основные цели разработки, требования к программному продукту, определяются сроки и этапы разработки и регламентируется процесс приемно-сдаточных испытаний. В формулировании технического задания участвуют представители как заказчика, так и исполнителя.

В основе этого документа лежат исходные требования заказчика, результаты выполнения предпроектных исследований и т.п.

Разработка технического задания выполняется в такой последовательности:

1) устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных;

2) определяют перечень результатов, их характеристики и способы их представления;

3) уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В соответствии с ГОСТ 19.201—78 программный документ «Техническое задание» содержит следующие разделы.

Введение.

1. Основание для разработки.
2. Назначение разработки.
3. Требования к программе или программному изделию.
4. Требования к программной документации.
5. Технико-экономическое обоснование.
6. Стадии и этапы разработки.
7. Порядок контроля и приемки.
8. Приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять некоторые из них.

* 1. Техническое задание на разработку программного обеспечения АИС «Электронный каталог CD-дисков»

Введение

Целью разработки программного продукта – создания программного решения, реализующего приложение автоматизированной рабочей системы «Электронный каталог CD-дисков». Данное приложение является хорошим решением для составления списка дисков, регулировать состав каталога, оно позволяет формировать в удобной и простой форме формировать полный список дисков, например, в личной коллекции.

1. Основание для разработки

Основанием для разработки программного продукта послужило выданное задание в филиале «РКТ» МАИ.

1. Назначение разработки

Программный продукт в первую очередь предназначен для упрощения процесса составления и упорядочения, обновления дискового ассортимента в магазинах, специализированных на продаже дискового носителя информации.

1. Требования к программе
   1. Требования к функциональным характеристикам

* Пользователь должен иметь возможность вносить данные в БД;
* Должна быть предусмотрена возможность редактировать, удалять и сохранять записи в базу данных;
* У программы должен быть предусмотрен графический интерфейс взаимодействия с данными каталога.
  1. Условие эксплуатации

Программу можно эксплуатировать по прямому её назначению – для составления полного каталога магазина дисков, либо же использовать в личных целях (например, составить каталог дисков личной коллекции).

* 1. Требования к надежности
* В коде программы необходимо предусмотреть обработку исключительных ситуаций;
* Во избежание «падений» программы при неверном вводе данных пользователем должны быть введены всплывающие окна, предупреждающие пользователя о совершённых ошибках, и предотвращающих выход программы из строя.
  1. Требования к составу и параметры технических средств

Для использования программы потребуется компьютер со следующими аппаратными характеристиками:

* 2-х ядерный процессор архитектуры x86, amd64 и выше;
* 1 Гб ОЗУ и больше;
* Наличие видеочипа с 128 мб видеопамяти и выше;
* 150 мб свободного дискового пространства;
* Операционная система: Windows 8.1 и выше;
* Операционная система должна быть 64-разрадная.
  1. Требования к хранению и транспортированию

Файлы проекта должны храниться на резервном жёстком диске либо на флэш накопителе.

* 1. Специальные требования

ПО должно иметь дружественный интерфейс, ориентированный на пользователя любого уровня пользования ПК.

1. Требования к программной документации

Документация должна иметь описание программного продукта, а также руководство по эксплуатации программного продукта для пользователя.

1. Технико-экономическое обоснование

Разрабатываемое программное решение позволит оптимизировать процесс составления каталога дисков, тем самым облегчив ему задачу составления каталога и при этом сэкономить время пользователя.

**Лабораторная работа №2.**

**«Применение структурного подхода в анализе требований и определении спецификаций программного обеспечения»**

2.1 Основные сведения

Разработка любого программного обеспечения начинается с анализа требований к будущему программному продукту. В результате анализа получают спецификации разрабатываемого программного обеспечения: выполняют декомпозицию и содержательную постановку решаемых задач, уточняют их взаимодействие и определяют эксплуатационные ограничения. В процессе определения спецификаций строят общую модель предметной области как некоторой части реального мира, с которой будет тем или иным способом взаимодействовать разрабатываемое программное обеспечение, и конкретизируют его основные функции.

Методологии структурного анализа и проектирования, основанные на моделировании потоков данных, обычно используют комплексное представление проектируемого программного обеспечения в виде совокупности следующих моделей:

* диаграммы переходов состояний (SDT — State Transition Diagrams), характеризующие поведение системы во времени;
* функциональные диаграммы (SADT — Structured Analisis and Design Technique);
* диаграммы потоков данных (DFD — Data Flow Diagrams), описывающие взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе;
* диаграммы «сущность— связь» (ERD — Entity—Relationship Diagrams), описывающие базы данных разрабатываемой системы.

2.2 Диаграммы переходов состояний

Диаграммы переходов состояния (SDT) демонстрируют поведение разрабатываемой программной системы при получении управляющих воздействий.

Под управляющими воздействиями, или сигналами, в данном случае понимают управляющую информацию, получаемую системой извне, например, управляющими воздействиями считают команды пользователя и сигналы датчиков, подключенных к компьютерной системе. Получив такое управляющее воздействие, разрабатываемая система должна выполнить определенные действия, а затем или остаться в том же состоянии, или перейти в другое состояние, зафиксировав некоторые изменения в системе.

Главное предназначение этой диаграммы — описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма переходов состояний представляет динамическое поведение сущностей на основе спецификации их реакции на восприятие некоторых конкретных событий. Системы, которые реагируют на внешние воздействия других систем или пользователей, иногда называют реактивными. Если такие воздействия инициируются в произвольные случайные моменты времени, то говорят об асинхронном поведении модели. Несмотря на то, что диаграммы чаще всего используются для описания поведения отдельных экземпляров классов (объектов), они также могут применяться для спецификации функциональности других компонентов моделей, таких как варианты использования, действующие лица, подсистемы, операции и методы. На рисунке 1 представлена диаграмма переходов состояний для разрабатываемой информационной системы.

Процесс реализации данной диаграммы для ИС «Электронный каталог CD-дисков» состоит из нескольких этапов:

1. Для начала необходимо определить основные состояния, взаимодействия (условия перехода), выполняемые действия, возможные переходы
2. Начальным состоянием (обозначается чёрным кружком) является поступление запроса на добавление введённых данных в базу данных (БД). От начального состояния следует совершение действия на получение строки запроса. Схематично действие обозначается стрелочкой, ведущей к промежуточному состоянию;
3. Промежуточным состоянием является проверка строки запроса. Исполнение процесса зависит от того, какое из заданных условий выполняется. Если проверены не все строки, то данный процесс повторяется до тех пор, пока все строки не будут проверены (на схеме действие обозначается круговой стрелкой исходящей и ведущей в одно место — промежуточное состояние)
4. Если все строки проверены, и некоторые поле ввода – пустое, от промежуточного состояния проверки строки действие переходит к состоянию ожидания. Если в процессе ожидания пустое поле было заполнено входными данными, но при этом остаются не заполненные поля ввода, данный процесс будет совершаться до тех пор, пока все поля не будут заполнены. Когда все поля будут заполнены данными, программа переходит в промежуточное состояние добавление данных в БД. После записи данных процесс окончательно завершается.
5. Если же изначально все поля будут заполнены данными, процесс ожидания пропускается и сразу же система переходит к выполнению операции добавления данных в базу данных.

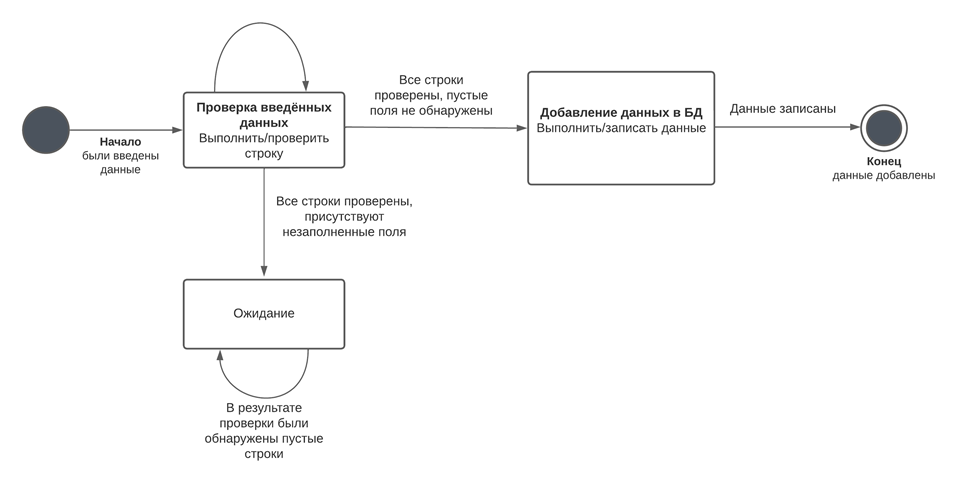


Рисунок 1 — Диаграмма переходов состояний SDT.

2.3 Функциональная диаграмма

Функциональные диаграммы (SADT) — отражают взаимные связи функций разрабатываемого программного обеспечения. Они создаются на ранних стадиях проектирования систем, для того чтобы помочь проектировщику выявить основные функции и составные части проектируемой программной системы и, по возможности, обнаружить и устранить существенные ошибки.

Процесс реализации данной диаграммы состоит из нескольких этапов:

1. Определяются функции системы и управляемая информация
2. Функции представляются блоками. Обрабатываемая информацию поступает в блок, то есть добавляется стрелка в левую часть блока, в верхнюю и нижнюю
3. Результат работы блока, т.е. функции, выдаётся с правой стороны блока, с исходящей стрелкой, идущей в другой блок с левой стороны.
4. Новый блок является дочерним для предыдущего. Функциональная диаграмма разрабатываемого программного продукта представлена на рисунке 2.

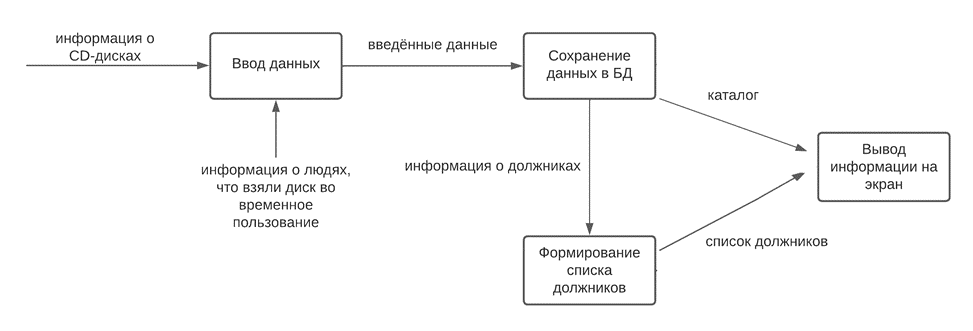


Рисунок 2 — Функциональная диаграмма.

2.4 Диаграммы потоков данных

Диаграмма потоков данных (DFD) — основное средство моделирования функциональных требований к системе, проектируемой или реально существующей. В основе модели лежат понятия внешней сущности, процесса, хранилища (накопителя) данных потока данных. Источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам; те, в свою очередь, преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям — потребителям информации.

Построение диаграммы потоков данных выполняется по следующим правилам:

1. Выделяются основные сущности и процессы в ИС с назначением для них соответствующих управляющих элементов диаграммы.
2. Каждый процесс должен сопровождаться как минимум одним входным и одним выходным потоком;
3. В каждое хранилище должен впадать как минимум один поток данных и как минимум один — вытекать;
4. Данные, хранимые в системе, должны проходить через процесс;
5. Каждый процесс диаграммы потоков данных должен вести либо к другому процессу, либо к хранилищу данных.

Результат построения диаграммы потоков представлен на рисунке 3

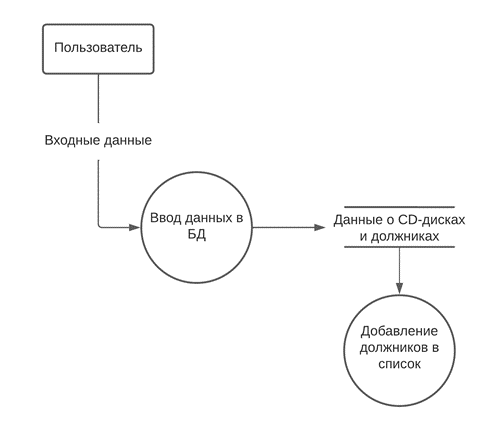


Рисунок 3 — Диаграмма потоков данных

2.5 Диаграмма «Сущность-Связь»

ER-модель данных обеспечивает стандартный способ определения данных и отношений между ними. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области.

Пример разработки диаграммы «Сущность-Связь описан ниже:

1. Выделяются основные сущности и связи;
2. Перечисляются объекты, которые могут быть полезны при проектировании данных организации, которым соответствуют сущности;
3. Между сущностями назначаются связи в зависимости от их назначения;
4. В зависимости от сущностей указываются типы установленных связей.

Диаграмма «Сущность-Связь» для разрабатываемого программного решения представлена на рисунке 4.

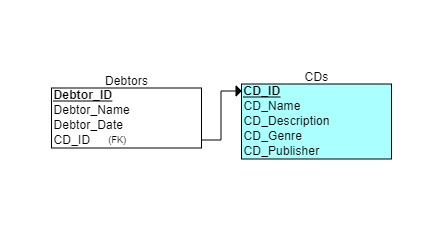


Рисунок 4 — Диаграмма «Сущность-Связь»

3 Проектирование программного обеспечения при структурном подходе

3.1 Структурная схема

Структурная схема – это схема, определяющая основные функциональные части изделия, их назначение и взаимосвязи. Результат уточнения структуры может быть представлен в виде структурной схемы, которая даёт достаточно полное представление о проектируемом программном обеспечении.

Для программного продукта ИС «Электронный каталог CD-дисков» структурная схема будет состоять из самой информационной системы, подразделяющаяся на две подсистемы:

1. Подсистема работы со списком дисков. Данная подсистема отвечает за работу с данными о хранящихся в каталоге дисках. В ней осуществляются такие функции как: удаление данных о диске, добавление нового диска в список, редактирование информации о диске.
2. Подсистема работы со списком должников. Данная подсистема позволяет создавать запись о новом должнике, а также удалять и редактировать запись о должнике.

Структурная схема ИС «Электронный каталог CD-дисков» представлена на рисунке 5.

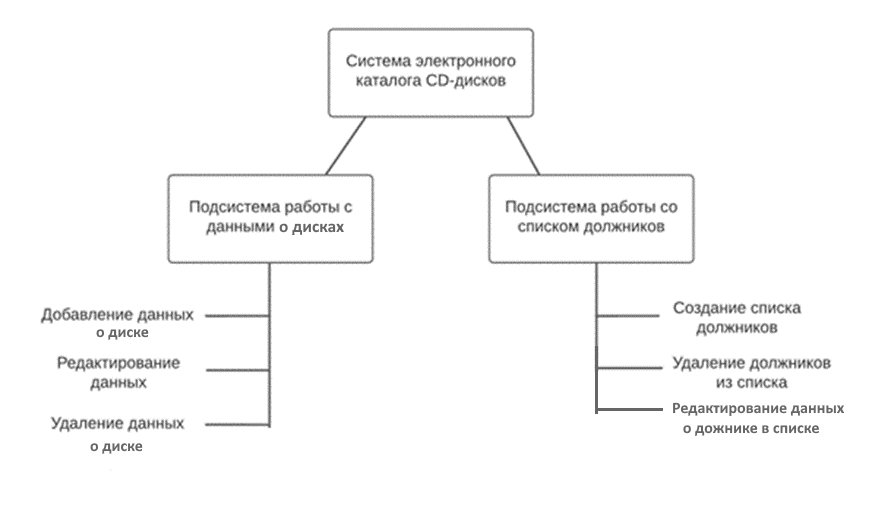


Рисунок 5 — Структурная схема

3.2 Функциональная схема

Функциональная схема — это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств.

Для ИС функциональная схема реализуется по следующему алгоритму:

1. Начало работы, т.е. инициализация всех процессов. Действие изображается прямоугольником с закругленными краями
2. Следующий этап — выбор варианта действия — добавление данных в базу данных посредством ввода в специальное поле некоторой информации. Действие изображается параллелограммом.
3. В зависимости от выбранного варианта действия, программа выполняет функцию проверки: если в поле отсутствуют данные для добавления, программ сообщит об этом пользователю, после чего ему нужно будет повторно произвести ввод данных. Действие, выполняющее проверку условия, изображается ромбом.
4. После происходит запись введённых данных в базу и проверка наличия диска в списке должников. В данном случае есть два варианта поведения системы:

* Если диск не отдан во временное пользование, то происходит завершение работы алгоритма проверки и схема завершается концом.
* Если диск был отдан в долг, он попадает в список должников вместе с именем того человека, который одолжил его, и запись об этом попадает в базу данных.

Функциональная схема ИС «Электронный каталог CD-дисков» представлена на рисунке 6.

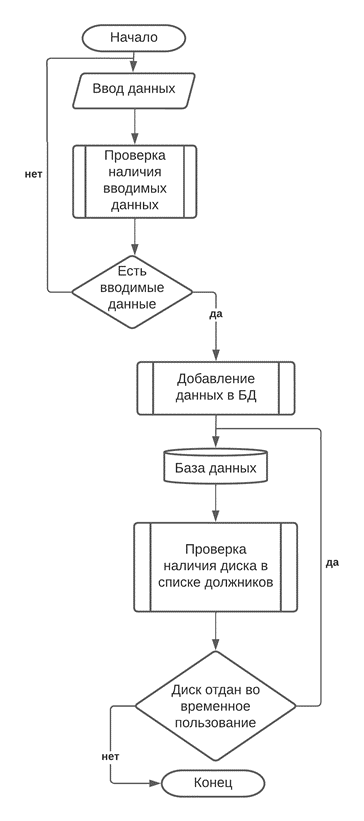


Рисунок 6 — Функциональная схема

4 Применение объектно-ориентированного подхода в анализе и проектировании программного обеспечения

4.1 Диаграммы вариантов использования

Диаграммы вариантов использования позволяют наглядно представить ожидаемое поведение программной системы.

Главное назначение диаграммы вариантов использования заключается в формализации функциональных требований к системе и возможности согласования полученной модели с заказчиком на ранней стадии проектирования.

Для разработки диаграммы вариантов использования соблюдать некоторую последовательность действий:

* Определить пользователя (или группу пользователей). В случае с разрабатываемым продуктом – Пользователь ИС.
* Настроить отношения ассоциации: «Пользователь» — «Добавление данных о дисках в БД», «Редактирование данных о дисках», «Составление списка должников» и «Редактирование списка должников».

Диаграмма вариантов использования представлена на рисунке 7.

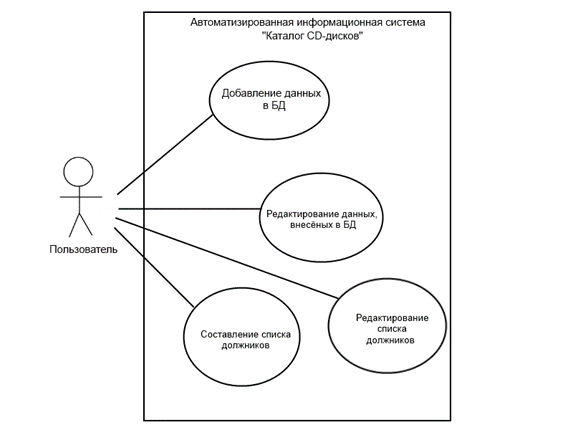


Рисунок 7 — Диаграмма вариантов использования

4.2. Диаграммы последовательности.

Диаграмма последовательности системы — графическая модель, которая для определенного сценария варианта использования показывает динамику взаимодействия объектов во времени.

Для построения диаграммы последовательности ИС необходимо:

* Идентифицировать каждое действующее лицо или объект;
* Из описания варианта использования определить множество системных событий и их последовательность для каждого действующего лица или объекта;
* Изобразить системные события в виде линий со стрелкой на конце между линиями жизни действующих лиц и системы, а также указать имена событий и списки передаваемых значений.

Для лица «Пользователь» системные события связаны с запросом на составление списка должника, составление каталога дисков.

Диаграмма последовательности к разрабатываемому программному продукту представлена на рисунке 8.

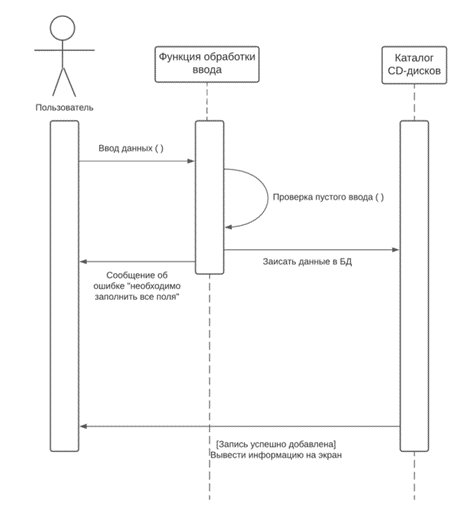


Рисунок 8 — Диаграмма последовательности.

4.3 Диаграмма деятельности

Диаграмма деятельности позволяет подробно иллюстрировать отдельный вариант использования и его сценарии. В зависимости от степени детализации диаграмма деятельности может использоваться на разных этапах разработки. На этапе анализа требований и уточнения спецификаций диаграммы деятельностей позволяют конкретизировать основные функции разрабатываемого программного обеспечения.

Алгоритм проектирования диаграммы деятельности для ИС:

1. Определяются действия по сценарию выполнения программы

* Начало;
* запрос добавление данных в БД;
* проверка наличия пустого ввода (если в поле ввода ничего нет, то программа выводит на экран окно с ошибкой; если в поле ввода есть данных, то они добавляются в базу данных);
* завершение процесса.

Диаграмма деятельности ИС «Электронный каталог CD-дисков» представлена на рисунке 9.

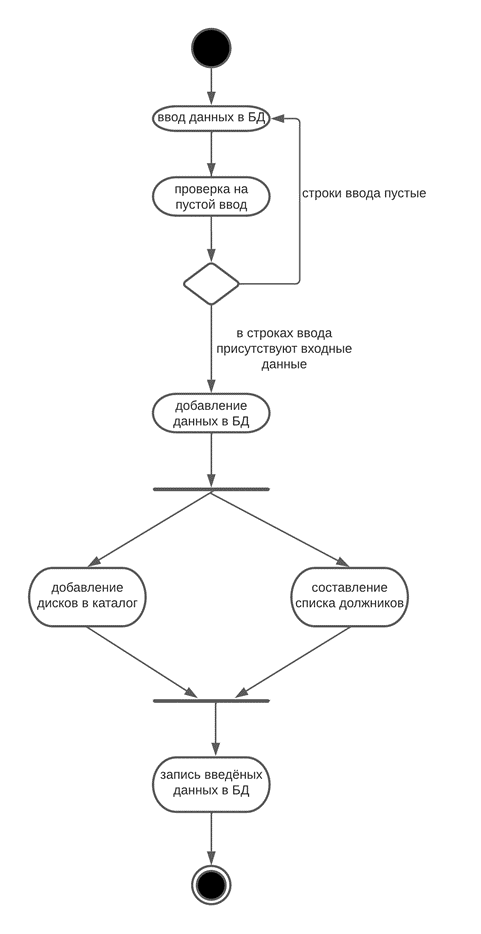


Рисунок 9 — Диаграмма деятельности

1. Разработка прототипа программного обеспечения

Прототипы программного обеспечения необходимы для проектирования главных окон интерфейса. В прототипах схематично указываются элементы интерфейса, для максимально ясного представления того, как должна выглядеть программа по итогу разработки. Сами прототипы довольно примитивны. Иными словами, прототип есть ни что иное, как схематичный набросок будущего интерфейса.

Для информационной системы «Электронный каталог CD-дисков» прототип интерфейса всех страниц будет выглядеть следующим образом.

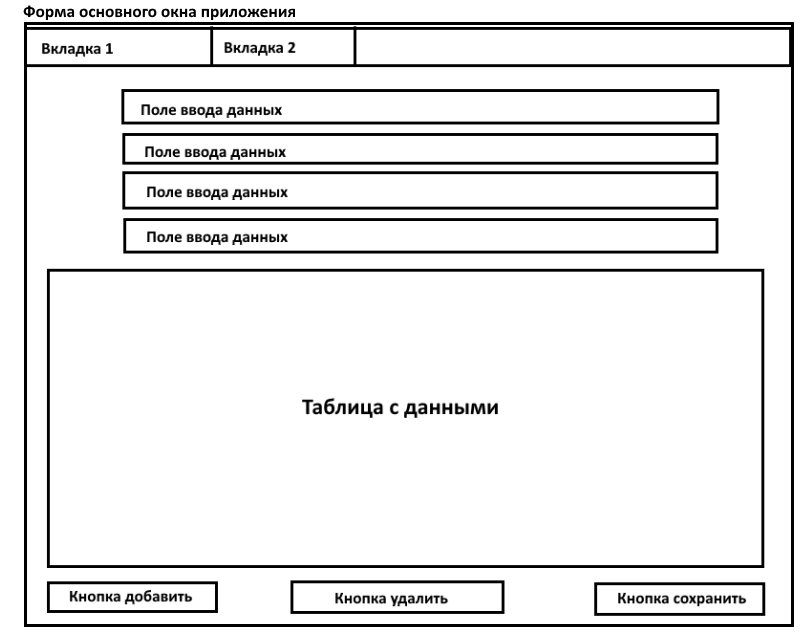


Рисунок 10 — Прототип основного окна приложения

Для главного окна шаблон формы будет иметь единую концепцию, представленную на рисунке 10. Форма будет состоять из следующих элементов:

1) В верхней части формы будет располагаться вкладки, между которыми будет возможность переключаться для взаимодействия с разными таблицами.

2) В блоке ввода данных будет находиться набор полей ввода, соответствующих каждому из столбцов таблицы.

3) Ниже блока ввода данных будет располагаться таблица, отображающая содержимое базы данных.

4) В нижней части формы будет располагаться блок манипуляции данными, состоящий из кнопок «Добавить данные», «Удалить данные», «Сохранить данные». Эти кнопки позволяют взаимодействовать непосредственно с данными в базе данных (БД): добавление новой записи, удаление существующей, запись изменения данных в БД.

1. Проектирование интерфейса пользователя

Следующий этап разработки информационной системы «Электронный каталог CD-дисков» — разработка готового интерфейса программы.

Для построения пользовательского интерфейса приложения требуется создать форму. С помощью средств Qt Designer есть возможность разработать необходимую форму.

Qt Designer – инструмент Qt, предназначенный для проектирования и построения графических пользовательских интерфейсов (GUI) с виджетами Qt.

Инструкция при разработке форм в «Qt Designer»:

1) Для создания главного окна необходимо выбрать «Main Window», а для создания диалогового окна нужно выбрать «Dialog without Buttons»;

2) Для настройки отображения элементов UI и настройки элементов взаимодействия пользователя непосредственно с программой используются виджеты.

Программа позволяет быстро создать необходимые формы интерфейса.

При создании форм представленных на рисунках 11 и 12 были использованы следующие объекты:

* QTableWidget — виджет, отображающий содержимое таблицы базы данных,
* QPushButton — представляет собой виджет кнопок, с которых можно взаимодействовать с программой. Есть возможность назначит на них выполнение определённых команд,
* QLabel — требуется для вывода текстовой информации в определенном участке формы приложения,
* QTabWidget — это панель вкладок и "область страниц", которая используется для отображения страниц, связанных с каждой из вкладок,
* QComboBox — представляет собой комбинированную кнопку и всплывающий список,
* QLineEdit — это однострочный текстовый редактор, в который можно записать текст,
* QDateEdit — виджет для редактирования дат.

На основе спроектированных прототипов создаются интерфейсы окон приложения. На экран выводится окно, содержащее основные управляющее элементы, поля для ввода и поле вывода информации.

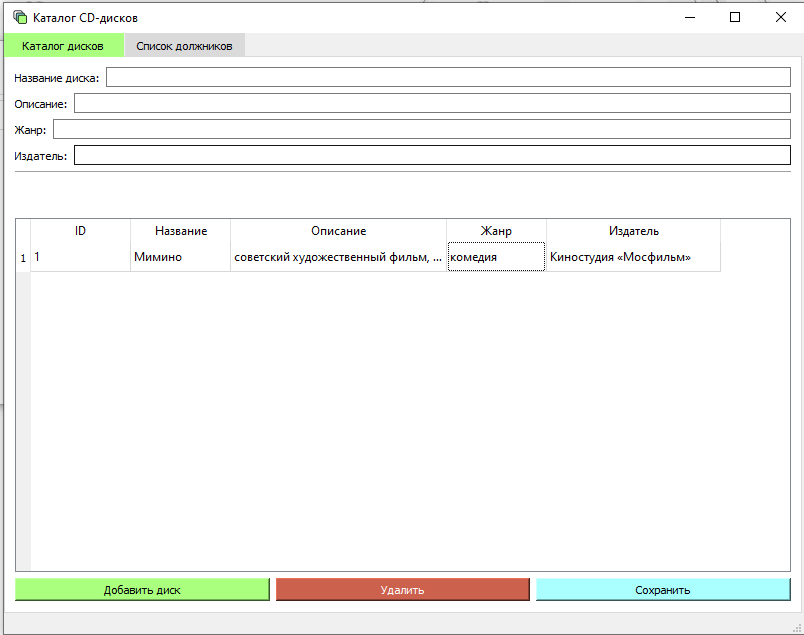


Рисунок 11 — Вкладка «Каталог дисков»

Кнопки «Добавить», «Удалить», «Сохранить» отвечают за операции добавления, удаления и сохранения данных.

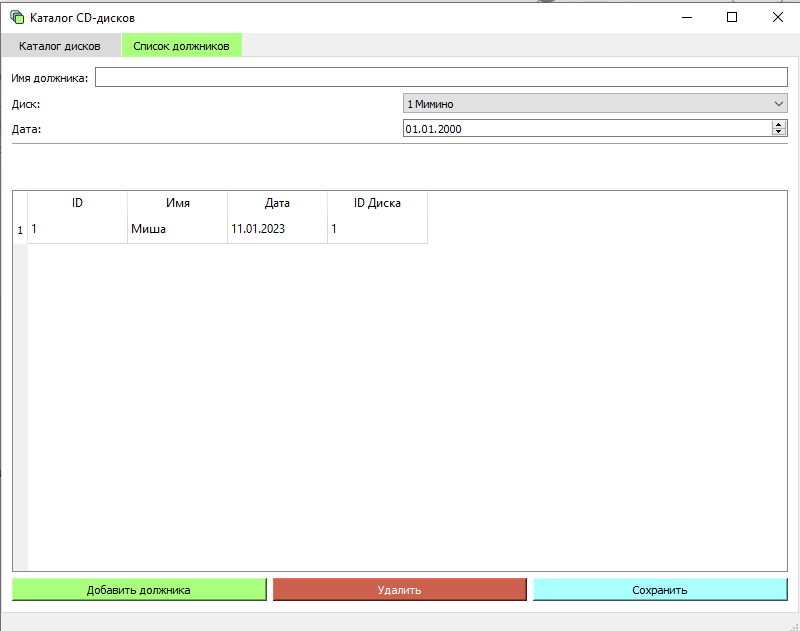


Рисунок 12 — Вкладка «Должники»

1. Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) – программирование, которое опирается на классы и объекты. Класс — описание типа объектов, т.е. их свойств, множества значений и действий, которые можно выполнять над объектами данного типа. Объект — экземпляр класса, конкретный представитель класса. Фактически, класс — тип данных, объект — переменная

В процессе создания информационной системы «Электронный каталог CD-дисков» были реализованы классы, представляющие собой отдельные составляющие всей общей системы, каждому классу назначен определенный функционал, организована связь между ними для чёткой и слаженной работы системы.

Для более эффективной работы системы принято разделять на отдельные модули — важные составляющие программы. Для информационной системы были реализованы:

1. модуль графического интерфейса программы,
2. модуль, обеспечивающий связь с базой данных с применением основных операций (добавление, сохранение, удаление).

Для программной реализации всех модулей был применён описанный выше подход к программированию — объектно-ориентированное программирование (ООП). Каждый модуль программы является отдельным классом, функционал каждого модуля — метод класса, свойства каждого модуля — атрибут класса.

Для модуля, взаимодействующего с базой данных, был реализован класс Database(). Данный класс использует специальную библиотеку языка программирования Python — «sqlite», для и связи с базой данных. Методы данного класса:

1. Получение информации из базы данных:

* get\_from\_cds — получить информацию о дисках;
* get\_from\_debtors — получить информацию о должниках;
* get\_cds — получение id и название диска, необходимых для построения виджета QComboBox.

1. Добавление данных в таблицы:
   * add\_in\_cds — добавление данных в таблицу «Диски» («CDs»);
   * add\_in\_debtors — добавление данных в таблицу «Должники» («Debtors»).
2. Удаление данных из таблиц:
   * delete\_from\_cds — удаление данных из таблицы «Диски» («CDs»);
   * delete\_from\_debtors2 — удаление данных из таблицы «Должники» («Debtors»).
   * delete\_from\_debtors — удаление данных из таблицы «Должники» («Debtors») при удалении данных о диске.
3. Обновление данных в таблицах базы данных:
   * update\_cds — обновление данных таблицы «Диски» («CDs»);
   * update\_debtors — обновление данных таблицы «Должники» («Debtors»).

Аналогично для модуля графического интерфейса был разработан класс MainWindow(). Методы данного класса:

* \_\_init\_\_() — инициализация класса (окна), отвечающего за подключение к кнопкам, объявление переменных, заполнение таблиц, отрисовку вкладок QTabWidget;
* add\_debtor — добавление данных в таблицу «Должники»;
* update\_combobox — обновление комбобокса со списком дисков;
* save\_debtor — сохранение данных таблицы «Должники»;
* save\_cd — сохранение данных таблицы «Диски»;
* add\_cd — добавление данных в таблицу «Диски»;
* update\_draw\_cd — функция отрисовки таблицы «Диски»;
* update\_draw\_debtor — функция отрисовки таблицы «Должники»;
* draw\_cd — обновление таблицы «Диски»;
* draw\_debtor — обновление таблицы «Должники»;
* get\_from\_table — получение данных о количестве столбцов и строк таблиц;
* delete\_cd — удаление данных из таблицы «Диски»;
* delete\_debtor — удавление данных из таблицы «Должники».

1. Тестирование

Тестированием называется выполнение программы в целях обнаружения ошибок. Отладкой называется локализация и исправление ошибок. Тесты разрабатывают в определенном порядке. При этом по внешней спецификации различают тесты:

* для каждого класса входных данных;
* для граничных и особых значений входных данных.

Контролируется, все ли классы выходных данных при этом проверены, и добавляются при необходимости нужные тесты.

Для тестирования функционала информационной системы «Электронный каталог CD-дисков», с использованием библиотеки unittest, были реализованы два класса:

1. TestAddData – тестирование добавления данных;
2. TestDeleteAndSave — для тестирования функции удаления и сохранения данных в БД.

Для класса TestAddData реализованы следующие модули:

* setUp — инициализация класса, подключение ко всем модулям и класса;
* test\_add\_cd — тестирование функции добавления дисков в каталог. Имитируется процесс ввода данных в БД и нажатие кнопки «Добавить». После прохождения теста программа удаляет ранее введённые тестовые данные командой «self.db.delete\_from\_cds»;
* test\_add\_debtor — тестирование функции добавления человека, взявшего в долг диск, в список должников. Имитируется процесс ввода информации о должнике в БД и нажатие кнопки «Добавить». После прохождения теста программа удаляет ранее введённые тестовые данные командой «self.db.delete\_from\_debtors».

Для класса TestDeleteAndSave реализованы следующие модули:

* setUp — инициализация класса, подключение ко всем модулям и класса для тестирования их функций;
* test\_delete\_cd — тестирование функции удаления диска из каталога. Имитируется добавление нового диска в каталог (тестовые значения — «cd\_name», «description», «genre», «publisher») и нажатие кнопки «Сохранить». Затем имитируется нажатие кнопки «Удалить»;
* test\_delete\_deb — тестирование функции удаления человеке из списка должников. Имитируется добавление должника (тестовые значения — «name», «date», «1») и нажатие кнопки «Сохранить». Далее следует имитация нажатия кнопки «Удалить».

8 Создание инсталлятора программы

Создание инсталлятора программного продукта позволяет автоматизировать процесс установки программного продукта на компьютеры пользователей, предоставляя при возможность выбора различных сценариев установки и обеспечивая корректность его дальнейшей работы.

Первоначальным этапом в создании инсталлятора программы является компиляция кода программы в исполняемый файл с расширением “.exe”.

Язык Python не является компилируемый языком программирования, в нём не задумывалась возможность преобразования программ в исполняемый вид. Вместо этого, в языке используется интерпретатор, который переводит код языка в машинный и выполняет его на месте. Теоретически, можно взять интерпретатор Python и объединить его с программой и её компонентами, получив портативную программу, которую можно запускать без установки языка. Для того чтобы создать исполняемый файл из кода на языка Python необходимо воспользоваться пакетом PyInstaller, выполняющего все эти действия.

PyInstaller собирает в один пакет Python-приложение и все необходимые ему библиотеки следующим образом:

1. Считывает файл скрипта.
2. Анализирует код для выявления всех зависимостей, необходимых для работы.
3. Создаёт файл .spec, который содержит название скрипта, библиотеки-зависимости, включая те параметры, которые были переданы в команду PyInstaller.
4. Собирает копии всех библиотек и файлов вместе с активным интерпретатором Python.
5. Создает папку BUILD в папке со скриптом и записывает логи вместе с рабочими файлами в BUILD.
6. Создает папку DIST в папке со скриптом, если она еще не существует.
7. Записывает все необходимые файлы вместе со скриптом или в одну папку, или в один исполняемый файл.

Команда для установки pyinstaller через pip – «pip install pyinstaller». Для создания исполняемого файла в командной строке надо ввести: pyinstaller –onefile «название\_файла».py. Если же при открытии .exe файла не нужна консоль, выводящая дополнительную информацию о работе программы, тогда в команду надо добавить параметр -w: pyinstaller -w –onefile gui.py.

Для более комфортной работы, без необходимости прописывать команды в консоль, можно воспользоваться библиотекой «auto-exe-to-py». Для её установки нужно прописать в консоли команду «pip install auto-py-to-exe». После установки в консоль нужно написать команду «auto-py-to-exe», после чего появится окно, что представлено на рисунке 13, в котором нужно выбрать путь до файла .py и указать выходную директорию для выходного исполняемого файла .exe. После чего нажать кнопку «CONVERT .PY TO .EXE».

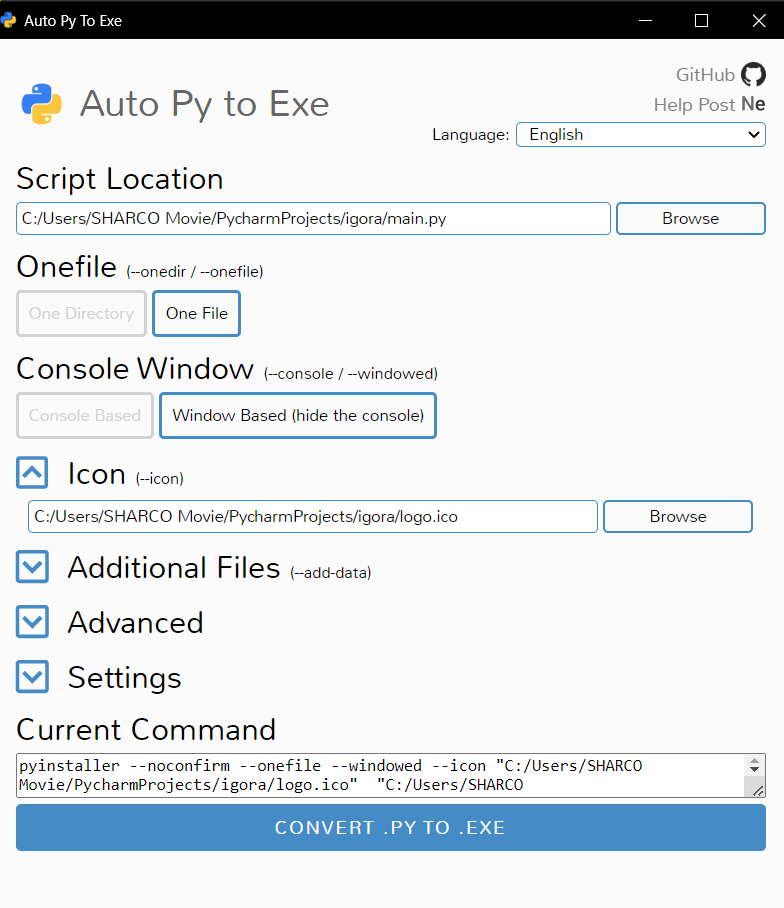


Рисунок 13 – Окно auto-py-to-exe.

В результате выполнения команды в папке, которая была указана для вывода будет находится исполняемый файл приложения. После создания исполняемого файла .exe нужно скопировать папки с формами и изображениями, чтобы директория выглядела так, как на рисунке 14.

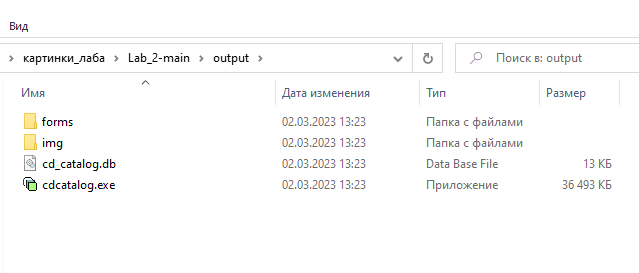


Рисунок 14 – Папка с исполняемым файлом

Следующим шагом необходимо создать инсталлятор приложения. Для этого нужно воспользоваться программой Inno Setup Compiler. После запуска программы в появившимся окне, представленном на рисунке 15, нужно выбирать пункт «Create a new script file using the Script Wizard». После чего нажать «OK» и «Next».

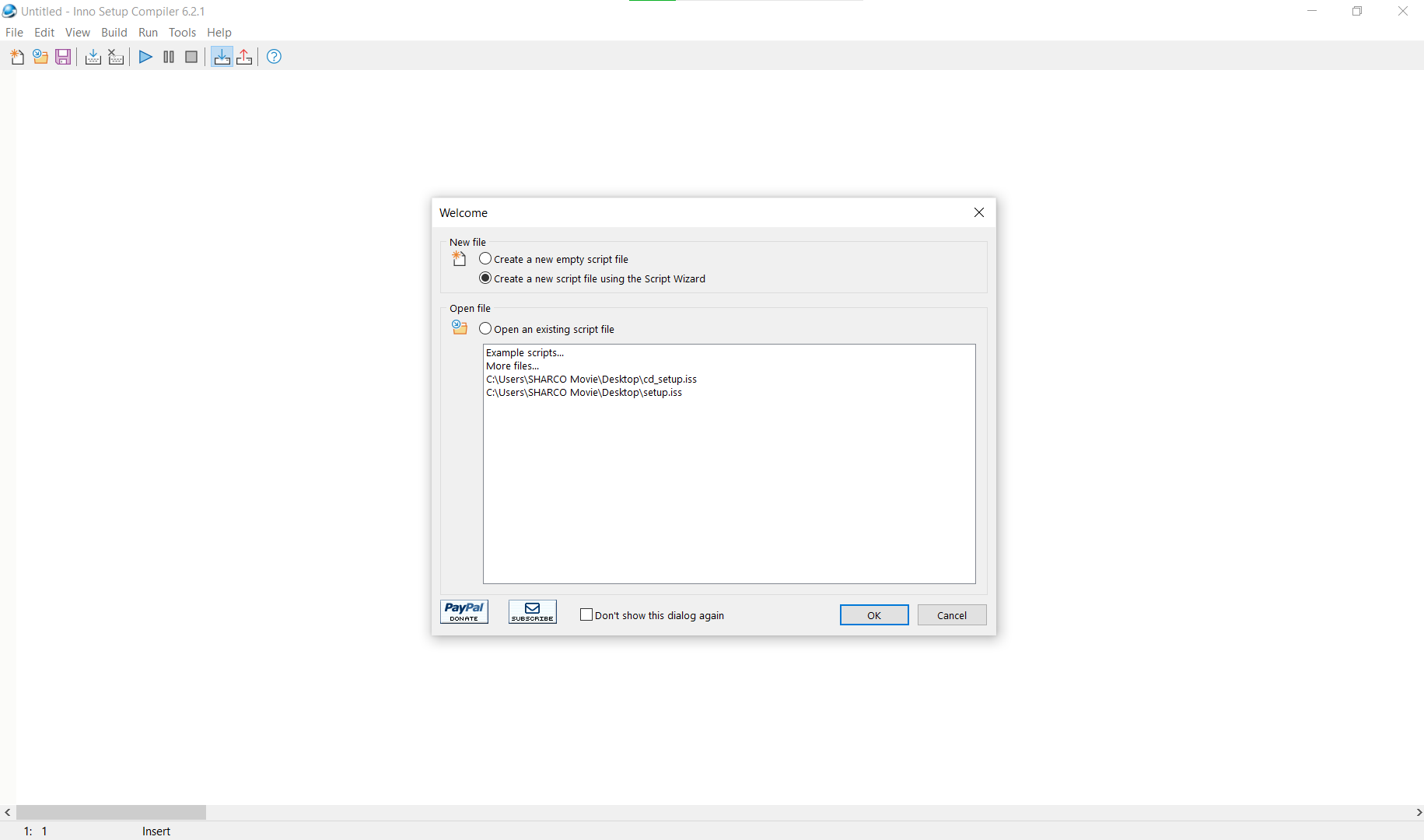


Рисунок 15 – Окно создания нового скрипта.

Открывается меню настроек приложения. В нём указывается название приложения, версию приложения, имя автора (или название компании, команды, группы), а также веб-страница приложения или его создателей (при наличии).

Следующее окно — указание пути к исполняемому файлу, а также добавление различных ресурсов к инсталлятору проекта.

* В поле «Application main executable file» указывается путь к проекту.
* В поле «Оther application files» указывается папка с файлами необходимые для работы приложения. Окно настройки ресурсов инсталлятора представлено на рисунке 16.

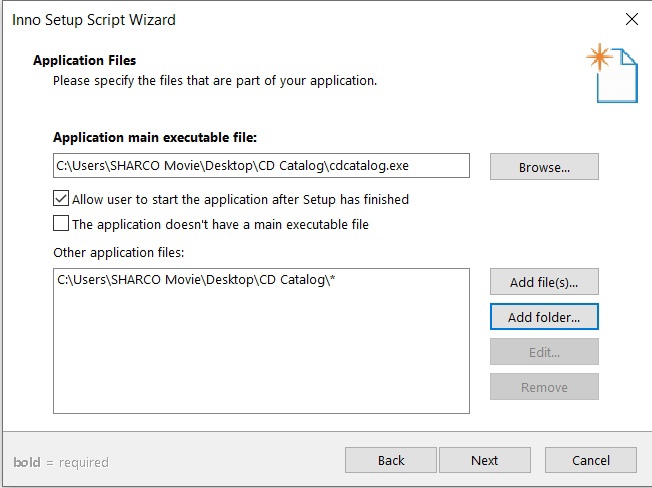


Рисунок 16 — Меню добавления файлов проекта.

В следующем окне необходимо снять галочку с поля «Associate a file type to the main executable».

На следующем окне, представленном на рисунке 17, нужно оставить всё без изменения и нажать кнопку «Next».

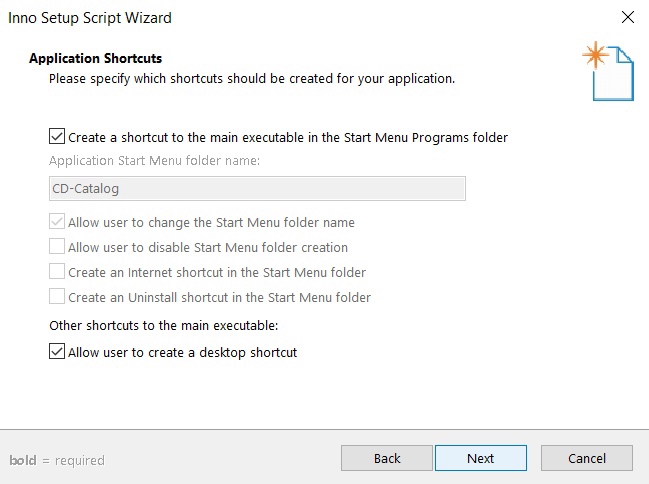


Рисунок 17 — Настройка ярлыков приложения

В окне на рисунке 18 добавляется (при необходимости) файл с текстом лицензионного соглашения конечного пользователя (EULA) и текстом сообщения перед (или после) установки приложения.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 18 — Настройка текстовых сообщений установщика

В Следующем окне, представленном на рисунке 19, всё оставляется без изменений.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 19 — Настройка режима установки

В следующем окне, представленном на рисунке 20, находится меню настройки языка инсталлятора. Здесь можно выбрать как один язык, так и несколько.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 20 — Выбор языка установки.

Далее указывается папка, в которую сохраняется созданный инсталлятор, указывается наименование инсталлятора, его иконка, а также пароль. Вид меню настройки представлен на рисунке 21.

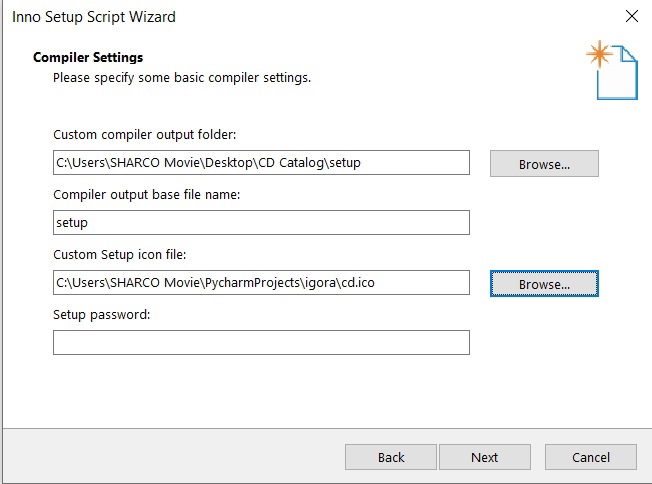


Рисунок 21 — Настройка папки выходного файла инсталлятора

На последнем этапе настройки инсталлятора, представленном на рисунке 22, настраивается использование директив «#define». Необходимо снять галочку с поля «Yes, use #define compiler directives» и нажать кнопку «Next», а затем кнопку «Finish».

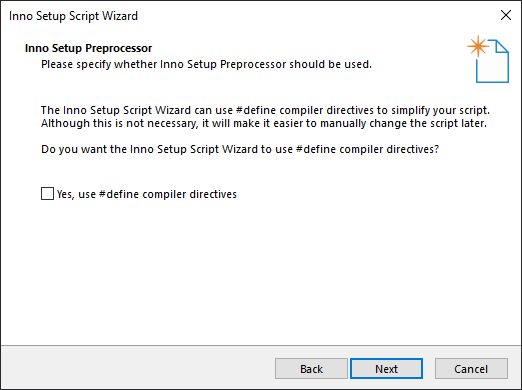


Рисунок 22 — Настройка директив #define

В появившимся окне нужно подтвердить исполнение сгенерируемого кода и нажать «Да», и дождаться компиляции.

В результате исполнения скрипта будет создан инсталлятор программы, показанный на рисунке 23.

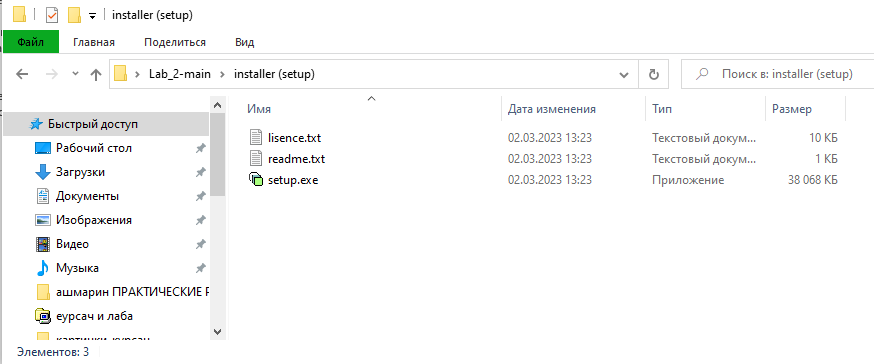


Рисунок 23 — Директория инсталлятора