საქართველოს ტექნიკური უნივერსიტეტი
1922 წლიდან

Грузинский Технический Университет

Сургуладзе Гия, Нарешелашвили Гулбаат

Основы программной инженерии

(Лекции и курсовой проект)

„სტუ-ს IT კონსალტინგის ცენტრი"

Грузинский Технический Университет

Сургуладзе Гия, Нарешелашвили Гулбаат

ОСНОВЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ

(Лекции и курсовой проект)

Утверждено:
редакционной коллегией
научного центра
ИТ Консалтинга ГТУ

Тбилиси – 2023

УДК 004.5

Излагается курс лекции по дисциплине „Основы программной инженерии”. Рассматриваются базовые теоретические вопросы, связанные снимы практические задачи визуального программирования, терминология и инструментальные средства их реализации на базе платформы Ms Visual_Studio.NET_Framework (2022). Лекции охватывают историчкские этапы создания и развития программной инженерии, включая парадигмы программирования, модели жизненного цикла ПО и вопросы менеджмента компьютерных систем. В приложении книги представлены образцы курсовых проектов по данной дисциплине, в частности, для Windows- и Web- аппикации с реляционными базами данных. Учебное пособие предназначено для студентов, изучающих курсы информатики по программной инженерии, информационным системам управления и информационным технологиям, а также для читателей, интересующихся вопросами создания ПО компьютерных стстем.

Рецензенты:

Самхарадзе Р. – Профессор каф. Программной инженерии, Д.т.н.

Туркия Е. – К.т.н. по Информатике. Зав. отд. Национального Банка Грузии

© ГТУ „ Научный центр IT-Консальтинга”,
2023 ISBN 978-9941-8-5590-0

Введение

В книге изложены теоретические и практические вопросы основ программной инженерии.

На основе изучения данного материала и выполнения конкретных лабораторных работ [1], студент сможет самостоятельно вывполнить курсовой проект с целью реализации программного обеспечения конкретного объекта (например, библиотека, университет, супермаркет и др.) на платформе MsVisual Studio. NET.

Целью курсового проекта является изучение процессов построения (девелопмента) Windows- и Web- апликаций в дисциплине "Основы программной инженерии", выработка навыков практического применения методов и средств новейших информационных технологий, с учетом требований международных стандартов. Овладение современными объектно-ориентированными и унифицированными визуальными методами и инструментами.

Студент индивидуально (или несколько студентов в команде) получит конкретное задание (задачу) для выполнения проекта из какой-либо проблемной области с целью проектирования компьютерной информационной системы управления и ее программной реализации.

Проблемной областью может быть объект любой отрасли, например, просвещения: университет, факультет, кафедра, школа и т.д.; любой департамент министерства, сфера сельского хозяйства; производство сельскохозяйственной продукции, ферма и т.д.; банковская система: управление вкладами клиентов, департамент кредитов и т.д.; сфера бизнесса и коммерции: департамент маркетинга, складское хозяйство, супермаркет и т.д.

В книге в качестве примера рассматривается задача-автоматизированная система приема нового сотрудника и начисления зарплаты.

Для решения задачи необходимо:

1) Для задачи приема в университет нового сотрудника построить UseCase диаграмму: роли и функций; определить функции (методы) и их выполнение (роли), а также блок- схему выполнения конкретной бизнес-задачи.

2) Определить формы документов и информацию:

- *исходную* (нормативную: должности, зароботная плата, шкала удержаний);

- *оперативную* (ежедневные или месячные табели учета, графики отпусков, бюллетени);

- *выходную* (ведомость перчисления на счета сотрудников);

3) Определить структуру базы данных SQL Server-а (или др.), построить ER диаграмму с таблицами (Tables), заполнить в них несколько строк (для эксперимента);

4) Разработать интерфейсы пользователей (рабочие формы) в интегрированной среде Visual Studio.NET Framework на базе Windows Forms Application (или ASP,Net Web Application) и языке C#;

5) Разработать алгоритм процесса выполнения конкретной задачи (например, расчета зароботной платы или др.);

6) Разработать методы связи задач интерфейсов пользователей с базой SQL Server-а (или др.) и выбора/модификации с соответствующими кодами языка C#;

7) Создать для иллюстрации стандартные требования, которые будут расположены на клавише Button интерфейса;

8) Провести модульное тестирование построенной системы на конкретных данных (Unit testing);

9) Подготовить демонстрационную версию системы, презентационный PowerPoint-файл (минимум с 7-ю слайдами) и документальные материалы (Word-файл минимум 15-20 страниц, включая титульный лист, рисунки, основной текст и список использованной литераеуры).

Глава 1

История развития Программной инженерии

Разработка программного обеспечения (Software Engineering, Development) относится к классу особенно сложных систем. На сегодняшний день существует большое многообразие программных систем. Если рассмотреть их классификацию, то можно выделить два больших направления: системные программы (Systems Software) и прикладные программы (Application Software), которые в свою очередь подразделяются на подмножества специального вида (рис.1.1).

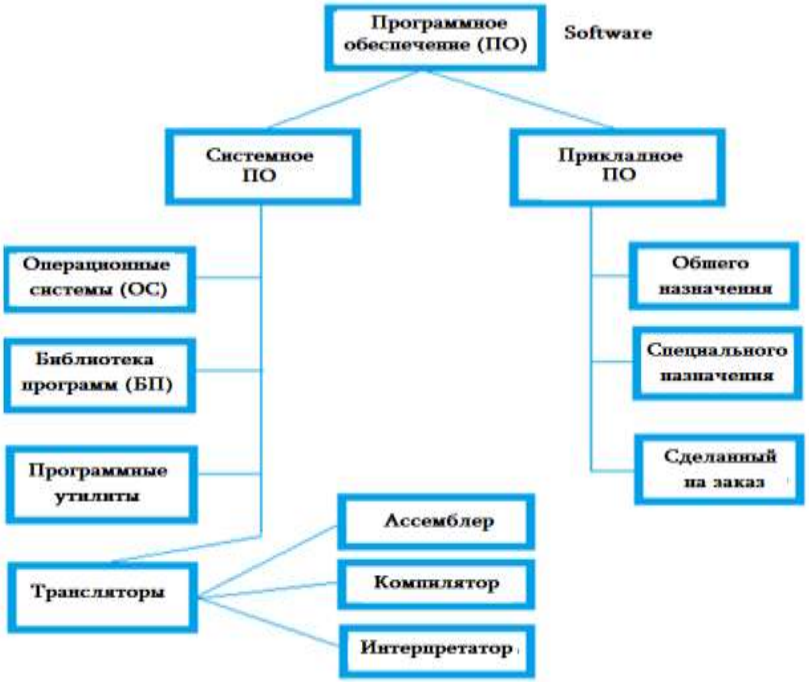


Рис.1.1. Классификация программного обеспечения

Объектом исследования в данной книге являются вопросы менеджмента программного пакета. В частности, разработка информационного, модельного и программного обеспечения с целью автоматизации бизнесс-процессов корпоративных организаций на базе интеграции новых технологий и методов информационной безопасности.

Современные методологии менеджмента программных систем на фоне развития *программных парадигм*, значительно отличаются друг от друга, что однозначно зависит от многих объективных и субъективных факторов.

Последователи этого направления, теоретики и программисты-практики стараются обосновать истинность своих убеждений и идеально представить ту или иную концепцию. Например, Унифицированный язык моделирования (UML - Г. Буч, И. Джакобсон, Д. Рамбо и др.) или Гибкое программирование (Agile) на примере экстремального программирования (Б. Кент, Д. Мартин и др.). Есть и такие, которые предлогают компромиссный подход – гибридный вариант (С. Амблер, Б. Румпе и др.) [5,6,11, 19,20].

В Грузии это научно-практическое направление с 2000 г. развивают профессора ГТУ: Г. Гогичаишвили, Г. Сургуладзе, Е. Туркиа, Т. Сухиашвили [4].

Вкратце рассмотрим историю развития технологий программирования (Программной инженерии) прикладных программных систем.

В первую очередь надо пояснить, для решения какой проблемы появились *гибкие методологии*. Суть проблемы состоит в том, что начиная с первых программных проектов до

сегодняшнего дня, разработка программного обеспечения была и остается трудно прогнозируемым, а часто неудачным делом.

Создание большого количества программных проектов заканчивается нарушением сроков и перерасходом бюджетных средств, а разработанные программы не отвечают требованиям заказчика или приносят мало пользы бизнесу.

Указанные проблемы являются основными признаками кризиса программного обеспечения. Несмотря на большие интеллектуальные усилия для преодоления кризиса, на сегодняшный день не найдено универсального решения (как часто говорят “серебрянная пуля”– метафора Фреда Букса) [21].

Суть подхода к *структурному управлению проектами* заключается в выполнении 10-ти этапной работы:

1) Визуальное представление цели; обращение внимания на приз;
2) Разработка списка задач, которые надо выполнить;
3) Должен существовать только один лидер;
4) Прикрепление исполнителей к задачам;
5) Управление будущими результатами, расчет резервов для ожидаемых ошибок, выроботка запасных позиций (управление рисками);
6) Применение соответствующего стиля руководства;
7) Знание того, что происходит;
8) Информирование исполнителей о том, что происходит;
9) Повторение этапов 1-8 до 10 этапа;
10) Приз (награда).

Выполнение данных этапов гарантия того, что проект успешно завершится. Используются термины: метод “Водопада”, итерация, планирование, риски и т.д.

1.1. Гибкие методы

Это современный ответ программной индустрии - как должны выпоняться проекты, чтобы с большой вероятностью они благлполучно завершились и принесли удовлет- ворение всем заинтересованным сторонам, и в первую очередь заказчику и команде исполнителей.

Окинем взор на историю развития программного обеспечения. Развитие языков программирования и разработка больших программных систем началось с 1950-ых годов. Начальные подходы к программированию были слабо формализованы и предстовляли процесс с наименованием „Code-and-Fix“ (кодирование и исправление).

Во время этого подхода разработка программного обеспечения начиналась непосредственно с кодирования (без всякого предварительного планирования, анализа требований и без проектирования). После этого найденные в коде проблемы (дефекты, несоответствие требованиям и т.д.) исправлялись в коде непосредственно вручную.

Со временем стало ясно, что для разработки больших, стабильных программных систем необходимы более осмысленные и формализованные подходы. Для решения проблемы надо обратить внимание к этому времени уже хорошо развитой сфере человеческой деятельности, которая связана со сложными производственными процессами. Например,

системотехника (System Engineering), проектирование и другие инженерные дисциплины.

Так возникло новое направление – программная инженерия (Software Engineering) в 1960 г., а в качестве фактического стандарта на протяжении многих лет укоренились так называемые *инженерные методологии* разработки программного обеспечения. Часто эти методологии называют основанными на планировании (Plan-driven), так как процесс разработки программного обеспечения является детерминированным инженерным процессом, который может быть запланирован от начала и до конца и затем выполнен согласно плану с применением формальных инженерных подходов.

Основным вариантом построения *жизненного цикла программы* использовалась модель "Водопада", которая предусматривает анализ требований, однократное выполнение проектирования, кодирования, тестирования и других фаз (рис.1.2).

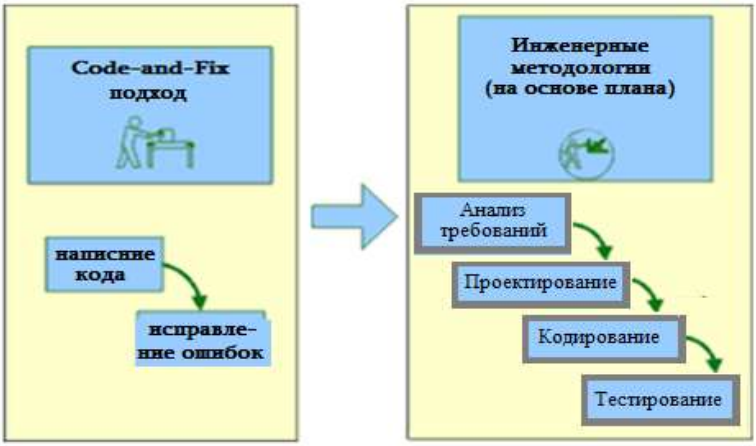


Рис.1.2. Переход на инженерные методологии

Кроме здесь рассмотренных двух подходов, существовали и другие варианты жизненного цикла, такие как модифицированные водопадные и итерационные модели. Здесь рассматриваются только основные, которые массово используются на практике.

1.2. Проблемы инженерных методологий

Переход на инженерные методологии и модель водопада действительно был шагом вперед. Он внес определенный порядок и организованность в процесс разработки, а также исключил многие проблемы подхода Code-and-Fix.

Однако инженерные методологии не смогли решить все проблемы и в первую очередь из-за заложенных в них внутренних конфликтов. Дело в том, что первоначально они создавались согласно образцам других инженерных дисциплин и совсем не учитывали особенности программного обеспечения, как уникального продукта (например, создание java программного продукта и производство холодильника).

Уникальность конфликта программного обеспечения проявляется как минимум двумя основными причинами:

1) **Недостаточное учитывание роли человеческого фактора.**

Инженерные методологии рассматривали процесс разработки программного обеспечения как выполненную абстрактным исполнителем последовательность шагов, т.е. основное внимание уделялось только тому, что должно быть сделано, а не на то, кто должен сделать. В действительности, большинство проблем программного проекта имеют социальный характер, а не технологический. Именно участники

проекта и их взаимоотношения являются основными факторами, определяющими успех проекта. Как показывают исследования в этой области, производительность лучших девелоперов компании в 10 раз выше, чем производительность девелоперов низкого уровня и в 2,5 раз выше, чем девелоперов среднего уровня. Таким образом, основными факторами, влияющими на производительность, являются не средства разработки и методологии, а качество рабочего места, психологическая обстановка в коллективе, эффективные коммуникации и другие социальные факторы.

2) **Несоответствие водопадного жизненого цикла природе программного обеспечения.**

Здесь не рассматриваются детали водопадной модели. Необходимо отметить только принципиальный момент, что любой программный продукт практически является уникальным и новым. Его разработка происходит в неопределенных условиях и попытка того, чтобы были учтены все факторы в начале разработки проекта (детальное составление требований, полное составление дизайна системы и др.), заранее обречены на провал. Недостатки водопадной модели не только усложняют разработку программной системы, но и сужают взаимодействие людей. Например, в начале разработки проекта составленные и согласованные требования часто становятся причиной раздора заказчика и разработчика на стадии реализации, когда необходимо внесение объязательных изменений.

Безусловно, существуют проекты, для которых водопадный жизненный цикл приемлем, однако для большинства

проектов приведенные выше проблемы являются актуальными и их неучет приводит к непринятию проекта.

В заключении можно сказать, что *инженерные методологии не смогли преодолеть кризис* в разработке программного обеспечения. Большое количество неудачных проектов в распорядок дня поставило вопрос поиска альтернативных подходов, что в дальнейшем осуществилось.

1.3. Легкие методологии:

Изучение проблем инженерных методологий разработки програмного обеспечения вызвало создание новых альтернативных подходов. Особенно быстро этот процесс распостранился во второй половине 90-ых годов, когда большую популярность приобрели ***легкие*** (или легкого веса) ***методы,*** такие как Scrum, DSDM, Crystal и др.

Несмотря на некоторые различия, сточки зрения практического применения, эти методы схожи тем, что в отличие от тяжелых и сверх меры формализованных инженерных методологий, они дают адаптивный, итерационный и ориентированный на человека подход.

Авторы ***легких подходов*** : М. Фоулер, К. Бек, А. Кауберн и др. на основе совместных встреч и консультаций (например, в 2001 г. в Сноуборде, штат Юта), пришли к такому решению, что этому направлению дать наименование “Agile” (гибкое, быстрое). Был создан документ “Манифест гибкой разработки программных систем” (с 4-мя пунктами). В дальнейшем к нему добавился список принципов (с 12-ю пунктами), который детально раскрывал манифест [6, 22].

1.4. Современные платформы и языки программирования

Платформа и язык программирования разные понятия. *Платформа* относится к аппаратно-программной технологии компьютера.

Компьютерная платформа определяется аппаратным решением ее процессора и шин и непосредственно связана с операционной системой (например, Unix, MsWindows, Linux).

Операционная система (ОС) имеет интерфейсы для программирования апликаций (API), совместимостью которых опреднляется семейство операционных систем. Таким образом, *программная платформа* есть операционная система, с помощью которой реализуются прикладные программные пакеты, т.е. программные апликации пользователей.

Язык программирования то инструментальное средство, на котором записываются программные платформы (операционные системы), а также программные апликации пользователей. Например, в 1972 году была записана на языке ***C*** первая сетевая операционная система ***Unix*** и создана новая известная "высокой надежностью" платформа, которая и сегодня занимает ведущее место на рынке программной инженерии [11].

История развития реализованных на вычислительных машинах языков программирования начинается с 1950 года (рис.1.3).

До сегодняшнего дня создано более 2000 языков, из них только "сильные" достигли 21-го века и до сих пор развиваются.

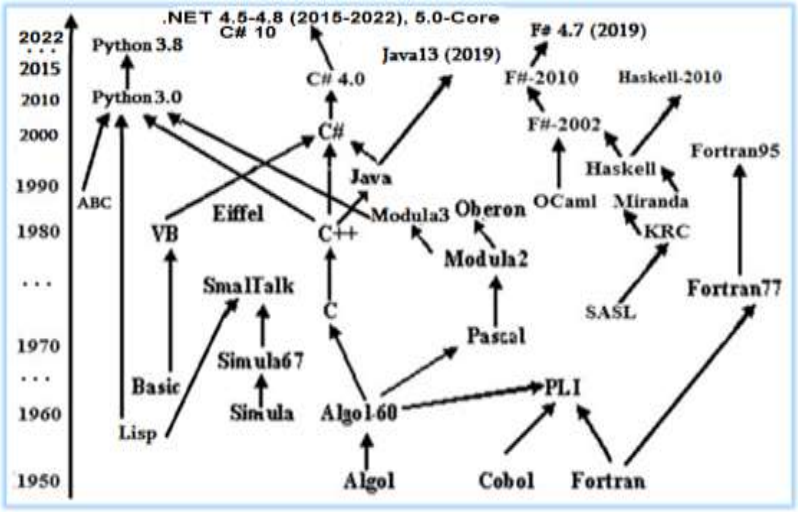


Рис.1.3. История развития языков программирования

На конкурентном рынке языков программирования господствуют потомки языка С – (C++, Java, C#). Этому способствовала универсальность и антивирусная надежность операционной системы Unix (по сравнению с MsWindows).

Из рисунка видна эволюция языков программирования (из-за большого объема информации здесь не все представлены) с точки зрения “парадигмы программирования”.

Парадигма программирования есть единство понятий и концепций, которое определяет ***стиль*** написания программы. Она вносит в язык свойственные изменения.

На сегодняшний день известны несколько стилей программирования: операторный, функциональный, императивный, логический, структурный, объектно-ориентированный, визуальный, компонентный, субъективный и т.д. [11].

Например, язык С известен структурным стилем программирования, языки C#, C++, Visual Basic, Java объектно-ориентированного стиля и т.д. Haskell и F# метафункционные языки, которые обладают объектно-ориентированным стилем, поэтому является актуальным их использование в университетах Америки и Европы. В версии Ms Visual Studio .NET 2010 майкрософт впервые представил язык F#, использование которого будет иметь в дальнейшем успех при решении математических и инженерных задач.

На платформе Майкрософта VS.Net-2022 можно создавать различные программные аппликаций, а именно Desktop-, Web-, мобильные и интегрированные приложения. Здесь также создается аппикаций на языках Python, JavaScript, PHP, ASP и т.д. (рис.1.4).

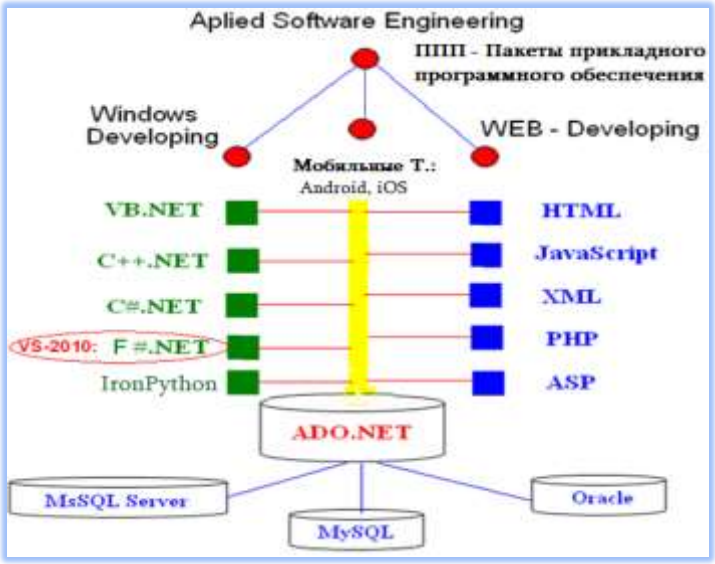


Рис.1.4. Инструментные средства Майкрософта

Ms Visual Studio.NET Flamework программная платформа, которая расположена между операционной системой и прикладной программной апликацией (рис1.5).

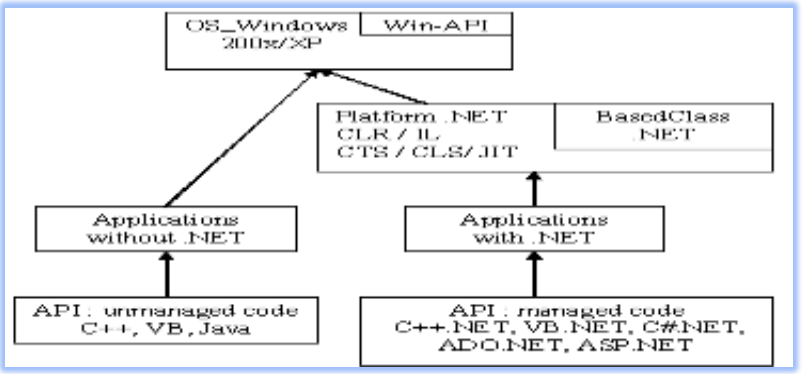


Рис.1.5. Концепция NET платформы

Целью использования этих интегрированных программных пакетов является моделирование, конструирование и реализация сложных систем с использованием унифицированной программной концепции.

Windows-система непосредственно работает с программными приложениями API (Application Programming Interface), написанными на C++, VB, C# и других языках, и которые реализованы как неуправляемые коды (unmanaged code). При этом она работает с программными приложениями, реализованными как управляемые коды (managed code), написанными на языках C#.NET, C++.NET, VB.NET и т.д. и управляемыми NET-платформой. Под управлением понимается то, что эти коды заработают непосредственно с помощью .NET-а, управляются их процессы и потоки данных, им предоставляются необходимые дополнительные ресурсы и т.д.

В принципе, .NET платформа выполняет определенные функции “операционной системы” и гибко функционирует с Windows-ом. На этом рисунке надо обратить внимание на блок NET платформы, в котором основной подблок Based Class.NET является библиотекой базовых классов платформы (большинство написаны на языке C#). Он полностью объектно-ориентирован, состоит из обьектов, в каждом из которых реализованы определенные группы методов, например, отображение окон и форм (Windows GUI), взаимодействие с файлами данных (ADO.NET), организация веб-сайтов и подключение к сети интернет (ASP.NET) и др.

1.5. Архитектура .NET Framework

На рис. 1.6 показана архитектура .NET Framework.

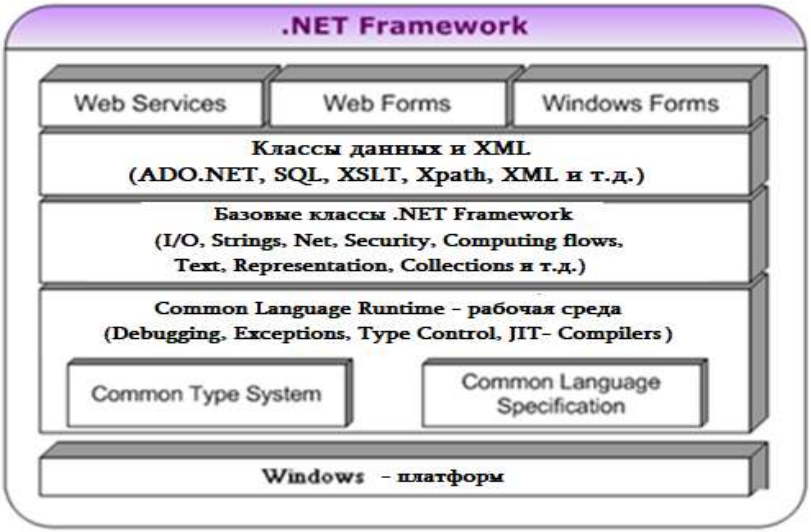


Рис.1.6. Архитектура .NET Framework

В этом же блоке показана рабочая среда .NET-runtime платформы (в которой выполняется программа), т.е. CLR (Common Language Runtime). Ее также называют общей областью выполнения. Это программное обеспечение для выполнения прикладных программ пользователя.

CTS (Common Type System) – система общих типов, на основе которой NET-платформа обеспечивает совместимость разных языков программирования. CTS также описывает правила определения классов пользователей.

IL (Intermediate Language) – язык промежуточных преобразований. Программы, исходные коды которых написаны, например на языках C#, C++ или VB в .NET-е, компилятор эти управляемые коды переводит в промежуточный язык IL, которые затем CTS быстро компилирует в машинные (dll - Dynamic-link library) коды. Таким образом, объектные коды с помощью IL языка получаются так, что в них не зафиксировано на каком языке записан исходный код.

Общая спецификация языка CLS (Common Language Specification), есть та минимальная спецификация стандартов, которая обеспечивает обращение к кодам из любого языка .NET-а. Все компиляторы этих языков имеют поддержку CLS-а.

JIT (Just-In-Time) есть фаза компиляции промежуточного кода в машинный код. Наименование указывает на то, что компиляция осуществляется только тех отдельных частей кода, которые необходимы для выполнения программы в данный момент времени. В .NET Framework-е используется 2-х этапная компиляция:

– 1-этап: компиляция в MSIL язык;

– 2-этап: “Just-In-Time” компиляция непосредственно в процессе выполнения.

MSIL – язык типа ассемблера, который не зависит от компьютера. Он выполняется везде, где установлен CLR.

ASPX страница отличается от HTML наличием в ней серверных элементов управления, которые описываются спец тегами. На рис.1.7 показана схема компиляции пользовательской апликации на .NET платформе.

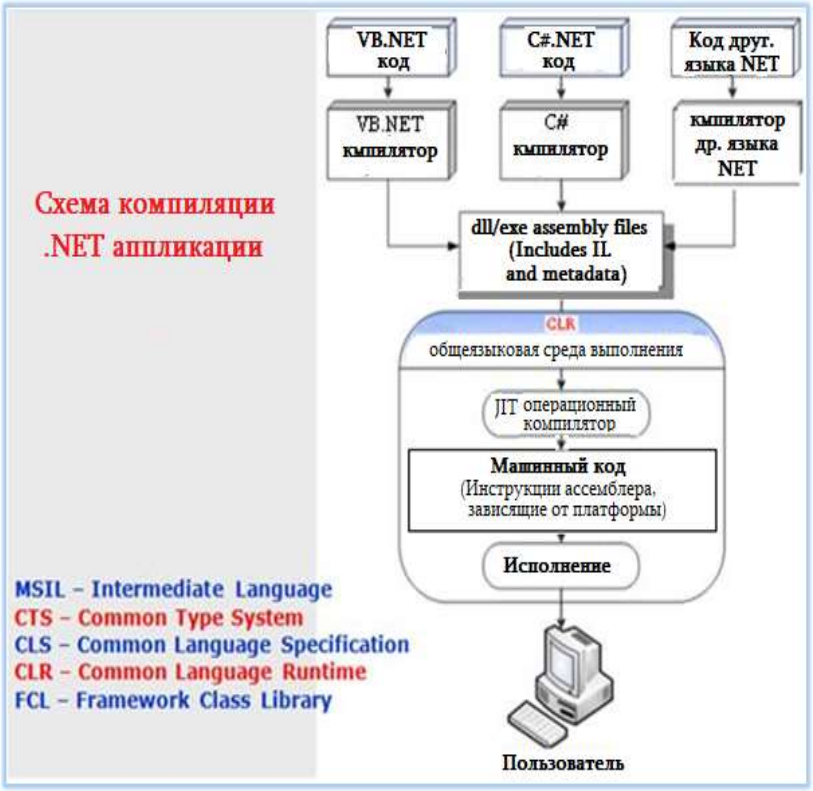


Рис.1.7. Компиляция программы на .NET платформе

Язык C# (“си-шарп”) является одним из мощных представителей объектно-ориентированных языков, который был специально создан для .NET–платформы и совместим с современными версиями Windows–а и интернета. На этом языке реализованы большинство базовых классов. Как известно, язык C++ компилируется в ассемблерный код, а язык C# – в промежуточный IL язык.

Назначением языка IL является осуществление платформенной и языковой независимости в объектно-ориентированной обстановке. Java также обеспечивает платформенную (Windows, Unix, Linux) независимость, однако на этапе выполнения его байт-кода он интерпретируется (а IL – компилируется).

1.6. Парадигмы программирования

Основой эволюционного развития языков программирования считают ***парадигмы*** программирования. В предыдущем параграфе были рассмотрены методы программирования, стили, модели и язык. Эти вопросы содержательно пересекаются с понятием парадигма.

Существует разные версии определения “парадигм программирования”. Например:

– “Парадигма есть стиль программирования для описания действия программиста” (Д. Бобров) [23];

– “Парадигма есть модель или подход для решения проблемы” (Б. Шрайвер) [24];

– “Парадигмы являются правилами классификации языков программирования в соответствии с определенными условиями, которые можно проверить” (П. Вегнер) [25].

– “Это способ концептуализации того, что значит ‘производство вычислений’ и как должна быть ‘структурирована и организована’ решаемая на компьютере задача” (Т. Бади) и др. [26].

Таким образом, если обобщить различные определения парадигмы, можно сказать, что парадигма программирования есть *начальная концептуальная схема поставленной проблемы и ее решения, является грамматическим инструментом описания фактов, явлений и процессов.*

Основные парадигмы программирования следующие:

- **Императивное программирование** (imperative programming): определяет вычисления в виде последовательности команд (инструкций), с помощью которых меняется состояние программы. Командой полученные данные хранятся в памяти и доступны. Такие языки:Assembler, Algol, Fortran, C, C++ и др.;
- **Процедурное программирование** (procedural programming): определяет шаги, которые должна выполнить программа, чтобы достичь желаемого положения. Последовательно выполняемые операторы можно поместить в подпрограммы. Это парадигма соответствует традиционной архитектуре компьютера, которую предложил фон-Нейман. Такими языками являются: C, Pascal, C++ и др.;
- **Декларативное программирование** (declarative programming):определяет логику вычислений без определения его

потоков управления. Т.е. дается спецификация решения поставленной задачи (единство требований и параметров), что должно быть результатом. Здесь не видно, как должен быть получен этот результат (в императивном программировании это видно). Примером таких языков являются SQL, HTML и др.;

- **Функциональное программирование** (functional programming): рассматривает вычисления как результат выполнения математических функций на основе исходных данных или других функций. Оно не учитывает хранение состояния программы (как это было в императивном). Такими языками являются: Lisp, APL, Haskel, C++, F#.NET и др.;
- **Объектно-ориентированное программирование** (object-oriented programming, OOP): методология, которая рассматривает программу как единство объектов, каждый из которых является экземпляром определенного класса. Класс *инкапсулирован*, имеет имя, свойства (члены класса), методы (функции класса), инициирование которых осуществляется поступившим извне сообщением (вызванным определенным событием). Классы создают *наследственную иерархию* и имеют *свойство полиморфизма. Объектно-ориентированный язык не является алгоритмическим языком !*Такими языками являются: C++, Java, C#, Python, Smalltalk, Ruby, Eiffel, PHP и др. В книге детально будут рассмотрены языки OOP;
- **Событийно-ориентированное программирование** (event-driven programming): осуществляет выполнение программы на основе событий. Это может быть действие пользователя (мышь, клавиатура), сообщение, полученное от другой программы или

от операционной системы и т.д. Задачей программы является анализ события и выбор соответствующего обработчика (event handler). Такими языками являются: Javascript, ActionScript, Visual Basic, Elm идр.;

- **Логическое программирование** (Logic programming): основывается на автоматическом доказательстве теорем и разделе дискретной математики, который изучает принципы логического вывода. Примерами языков являются язык Planner и производные от него Popler, Conniver и Qlisp, также язык Prolog и его дети Mercury, Visual Prolog и Oz;
- **Программирование, базирующееся на автоматах** (Automata-based programming): является парадигмой программирования, при использовании которой программа или ее фрагмент подразумеваются как модель какого-либо формального автомата. При автоматическом программировании могут быть применены *конечные автоматы*, *сети Петри* или другие *графовые модели*.

Список приведенных здесь парадигм может быть рассширен и больше конкретизирован, использование каждой из которых служит для решения задач соответствующего класса. Надо отметить, что некоторые современные языки программирования являются носителями *мультипарадигмных* свойств, т.е. в них использованы несколько стилей. Например, C++, Java, C#, Python являются носитнлями объектно-ориентированного, структурного, функционального, императивного и др. парадигм (Рис.1.8).

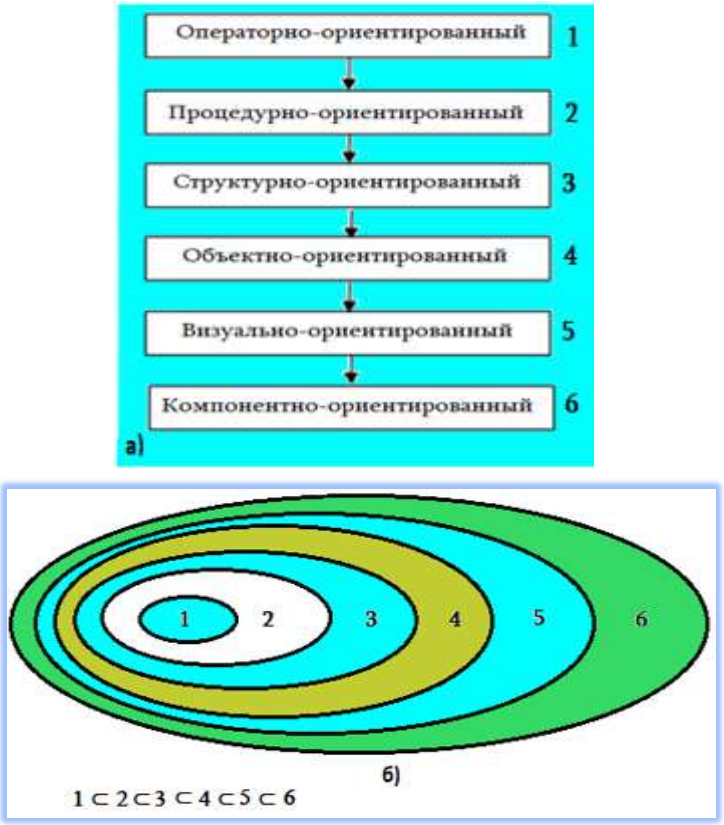


Рис. 1.8

Из диаграммы на рисунке (а) можно сказать, что парадигматический язык нижнего уровня включает в себя парадигматические возможности верхнего уровня (б). Так, например, язык структурного программирования полностью охватывает операционную и процедурную парадигмы. Кроме того, язык объектно-ориентированного программирования включает в себя структурную парадигму. Компонентно-

ориентированный язык основан на создании библиотеки повторно используемых компонентов, реализованных на основе классов и объектов.

Можно на языке С структурного программирования написать объектно-ориентированные программы, однако это значительно усложнит программный код.

Рассмотрим суть некоторых парадигм программирования и их значение.

Например, парадигма структурного программирования: на рис. 1.9 показаны наглядные общие схемы неструктурированной (а) и структурированной (б) программ. Основное внимание здесь уделяется отказу от использования оператора безусловного перехода GOTO [27].

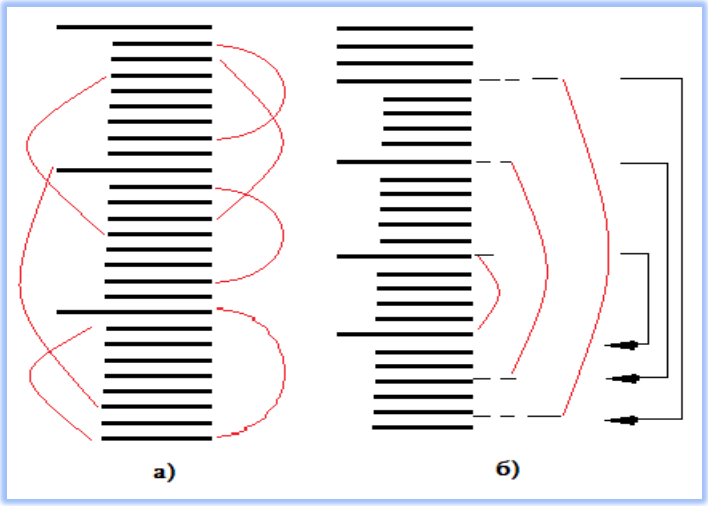


Рис. 1.9. Общие схемы программ:
Неструктурированный (а) с GOTO и структурированный (б) с вложенными циклами

Вместо этого следует использовать структурные элементы: *циклы и логические операторы ветвления*. Среди прочих требований стоит отметить использование *технологии нисходящего программирования* и *принцип разделения программы на модули*, а также *смысловые имена* переменных и констант программы (рис.1.10). Программное приложение становится гибким, простым в управлении и удобочитаемым для программистов.

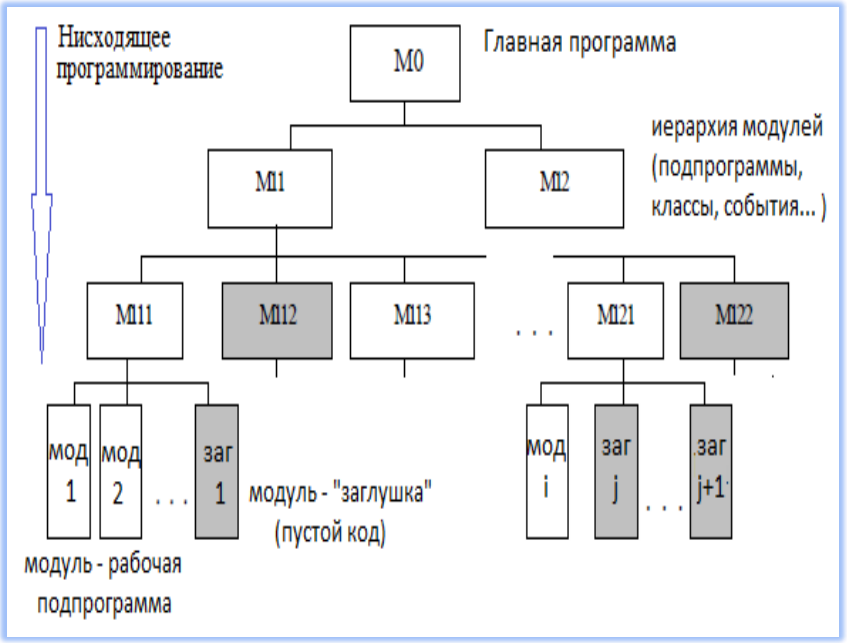


Рис.1.10. Пример модели программной аппликации со структурной парадигмой

Глава 2

Основы менеджмента программных систем

Разработка программных систем (проектов) во многих случаях рассматривается как труд коллектива специалистов, для удовлетворения требований пользователей автоматизации их рабочих процессов (бизнесс-процессов).

Как и любая коллективная работа, эта также требует определенной организации, в частности менеджмента (управление в социальных и организационных системах). Это длительный процесс, который связывает специалистов по разработке программных продуктов с прозводителями.

Основным элементом разработки программных систем является, с одной стороны изучение требований пользователей и с другой, обеспечение обратной связи, с помощью которой осуществляется процесс разработки программ. Решение таких задач возложено на руководителя т.е. менеджера проекта.

Исходя из размеров и сложности системы, проект может иметь одного персонального менеджера (**а** - небольшой проект) или служба менеджмента (**б** - большой проект), или главный менеджер и менеджеры направлений (**в** - для особенно больших проектов) (рис.2.1). Как видно из рисунка, менеджмент создания системных проектов с точки зрения организации распределения труда, очень разнообразный и зависит как от целей конкретного проекта,его сложности и сроков исполнения, так и от делегирования заданий коллетиву (распределение выполняемых функций) и применения отличительных моделей.

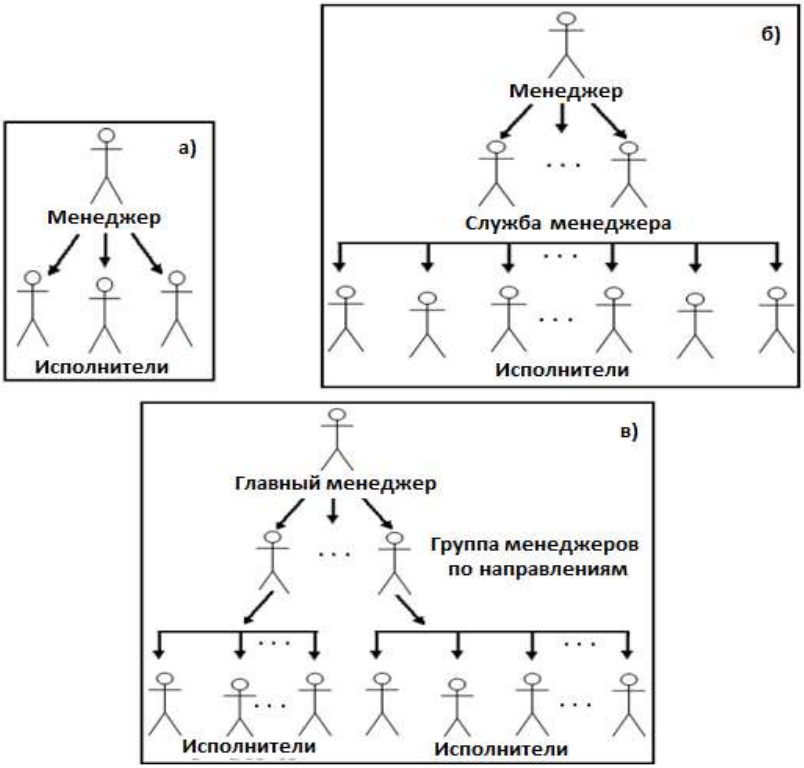


Рис. 2.1. Схемы организации менеджмента проекта

Например, в маленьких группах программистов (2-:10 человека), применяются принципы “быстрой разрвботки” программных систем (agile development), которые в современной теории программирования известны как методологии программирования”. В этом случае задачи, возложенные на главного менеджера, распределяются на программстов группы, которые сами определяют вопросы планирования, выполнения

и контроля заданий. В дальнейшем мы вернемся к методам экстремального программирования.

Перечисление всех вариантов организации менеджмента программных систем невозможно. Часто эти работы планируются и проводятся исходя из традиций существующего коллектива (команд).

Для менеджмента программных проектов всегда надо учитывать два аспекта:

- *управление проектом,* как процесс прозводства программного продукта для достижения определенной цели;
- *руководство* людьми, участвующими в проекте.

Главной и постоянной задачей менеджмента разработки программной системы является последовательная реализация-развитие проекта для достижения поставленной цели. Кроме тех способов и методов, с помощью которых можно решить задачу, необходимо также решение вопроса контроля эффективности распределения и использования ресурсов проекта.

Под ресурсами традиционно подразумеваются *время, финансы, технические средства* и *производственный потенциал кадров*.

Менеджер проекта всегда должен учитывать такие организационные и производственные контексты как *интересы заказчика проекта* и *исполнителя,* многообразие сфер деятельности и другие критерии.

Различают системные требования и требования пользователей:

- С*истемные требования* - это детальное описание системных функций и ограничений, которые часто называют

функциональными спецификациями. Они являются основой при оформлении контракта между заказчиком и исполнителем программной системы.

- *требования пользователей* - это описанные на естественном языке функции (бизнес-процессы) и диаграммы, которые выполняются системой, а также наложенные на нее ограничения (бизнес-правила).

Иногда также рассматривают *системные проектные спецификации*, которые являются обобщенным описанием структуры программной системы. Они являются основой для более детального проектирования системы и ее дальнейшей реализации.

Таким образом, *программный проект* можно рассмотреть как процесс реализации системы, который полностью удовлетворяет определенным ограничениям. Все ограничения формируются не одновременно, часто некоторые из них появляются в процессе выполнения или даже на этапе применения программной системы. Поэтому менеджер должен учитывать то обстоятельство, что ему придется работать над проектом для достижения окончательного результата в неопределенных условиях.

2.1. Функции менеджера ПО

Какие задачи приходится решать менеджеру при работе над проектом с точки зрения целесообразности и эффективности ? Рассматриваются два вопроса:

- кто участвует в разработке проекта ?
- какие этапы проходит проект при его осуществлении ?

Первый вопрос выделяет функции менеджера в команде разработчиков проекта. Здесь уделяется внимание двум аспектам: при управлении и руководстве.

С точки зрения управления участники проекта абстрактные действующие агенты, которые выполняют возложенные на них функции, используют определенные ресурсы и получают определенные результаты. Такое функциональное рассмотрение дает возможность взаимнозаменять участников проекта и это приводит к понятию "роли".

С точки зрения руководства главной задачей является создание деловой, согласованной и дружеской команды, которая может выполнить проект. Основным требованием команды является компетентность ее участников и кваллификация. Однако этого недостаточно. Кроме этого необходимо сформулировать правила командной работы, взаимодействия с руководством и заказчиком.

Создание таких взаимодействий и их управление является задачей менеджера проекта, как руководителя команды. Он устанавливает внешние и внутренние контакты (имеется ввиду взаимодействие с заказчиком программной системы и с руководством собственной организации).

Вопросы выполнения и развития этапов программной системы являются задачей менеджера проекта как руководителя. Он распределяет функции во времени между членами команды, определяет сложность решаемых задач, характер их коллективного решения, осуществляет контроль исполнения работ.

2.2. Жизненный цикл разработки ПО и типы ресурсов

Весь процесс планирования, проектирования и реализации программного продукта связан с понятием *жизненного цикла программного обеспечения.*

- Ж*изненный цикл ПО* есть процесс конструирования программного обеспечения компьютерной системы. Требование к программному проекту определяет цель и регламент данного процесса;
- Существуют *главные, производственные и вспомогательные ресурсы* :

- *главные ресурсы: кадры* (разработчики ПО, исполнители), а также *время* выполнения программного проекта (сроки) и *финансы*, которые выделяются для выполнения проектного задания;

- *производственные ресурсы:* технические средства, также программы, которые применяются в виде инструментальных средств, прототипов или вставляемых компонентов;

- *вспомогательные ресурсы:* обогрев, электроэнергия, рабочие комнаты, технические и организационные средства, которые непосредственно с разработкой проекта не связаны, но необходимы для его успешной реализации;

- *человеческие ресурсы* для хорошего проекта стабильны, не меняются в процессе выполнения проекта, при плохой организации работ имеет место текучка кадров, которая плохо влияет на окончательное качество проекта;

- *время* – его правильное планирование главная задача менеджера;

- *финансы* – основной затратный ресурс, который всегда ограничен. В частности, он используется не только в виде зароботной платы, но и для приобретения технических и инструментальных средств;

От менеджера программного проекта требуется *составление* оптимального *баланса использования ресурсов и решение задач его выполнения.* Эти задачи касаются менеджера всех видов проектов. На него воздействует специфика проекта и условия его выполнения.

Существуют множество подходов, способов, методик и технологий поддержки менеджмента. Но их использование непростая задача и требует наличия определенных знаний и подготовки. Также программные проекты часто являются индивидуальными и большое значение имеет практический опыт менеджера и исполнителя.

Таким образом, при рассмотрении задач менеджера в процессе разработки программных продуктов выделяются три взаимосвязанных направления:

1) *Функции,* которые необходимы для успешного развития проекта. Здесь объязательно надо выяснить для данного проекта *какие роли нужны* сотрудникам;

2) *Планирование проекта и контроль выполнения* в соответствии с жизненным циклом разрабатываемой программной системы;

3) *Формирование команды* проекта (кадры).

2.3. Формирование команды проекта: роли и функции

Как было отмечено, в процессе проектирования и реализации программных систем важное место занимает *эффективное использование людских ресурсов*. В начале разработки каждого нового конкретного проекта формируется *команда исполнителей*, руководство которой возложено на *менеджера*. На рис. 2.2 показаны типичные роли и функции команды создания программного обеспечения.

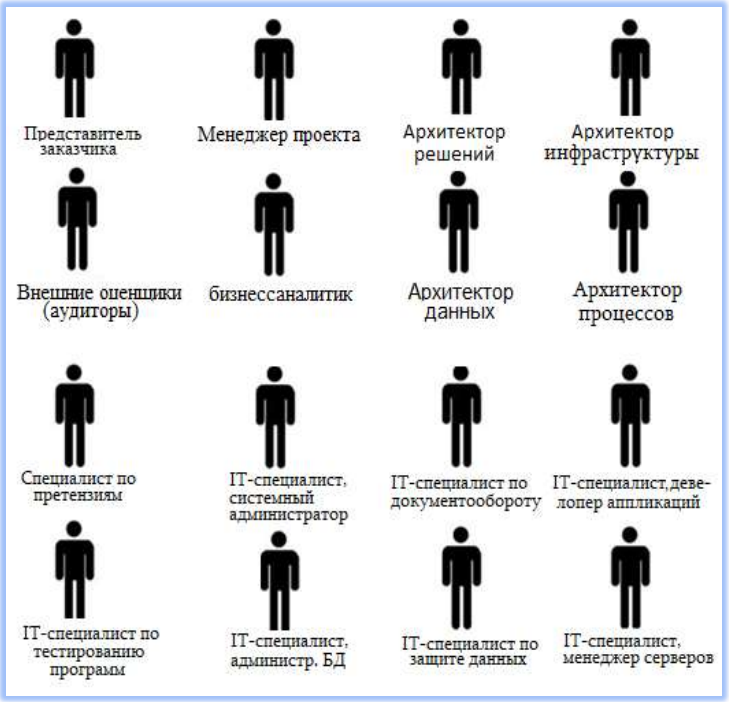


Рис. 2.2. Роли участников команды разработки ПО

- *Архитектор решения:* является специалист, который понимает бизнесс-цель, для достижения которой создается ПО, обладает специальными знаниями для ведения конкретного бизнеса, в частности процесса обработки претензий. Архитектор решений хорошо разбирается в существующей IT инфраструктуре, он отвечает за установление соответствия между новыми бизнесс-целями и техническими решениями с существующей IT-структурой. У него есть знания о SOA (сервис-ориентированные апликации) и Web-службах;
- *Специалист обработки претензий:* в нашем случае это представитель страховой компании, который управляет делами рассмотрения претензий, ведет администрирование полисов и устанавливает внешние репорты, которые необходимы для удовлетворения претензий. Для общения с внешними оценщиками (агентами) с целью передачи заданий он использует телефон, факс или электронную почту;
- *Внешние оценщики (аудиторы)*: являются специалистами, предоставляющими обслуживание, которые не являются штатными служащими страховой компании. Они работают в аудиторных фирмах и как независимые эксперты оценивают объекты страхования (например, разбитый автомобиль). Аудитор, получивший задание от специалиста по обработке претензий, идет к объекту, который надо оценить и оценивает объем ущерба. Репорт результатов отправляет по назначению;
- *Менеджер проекта*: сотрудничает с пользователями организации, бизнессаналитиками, аудиторами, следит в рамках бюджета исполнения требований;

- *Ведущий семинаров* : им может быть менеджер проекта, бизнессаналитик или специалист процессов, у которого есть опыт проведения презентаций. Он применяет Ms Visio или Rational Rose или Enterprise Architect, PowerPoint и т.д.;
- *IT-специалист (девелопер апликаций)*: разработчик приложения применяемой области или разработчик апликаций должен глубоко разбираться в бизнесс компонентах, которые входят в предложенное решение. Он должен запрограммировать эти компонеты для нового приложения с использованием, например C#, Java или других инструментов, а также должен уметь работать с форматами данных, которые существуют в системе;
- *Архитектор инфраструктуры*: его задача определение стандартов IT-инфраструктуры и для стратегического развития бизнесса установление IT–целей. Он хорошо разбирается в современных IT технологиях, в вопросах их надежности и производительности обработки требований;
- *Архитектор данных*: его касается моделирование данных корпорации, определение базы данных и ее таблиц. Он хорошо должен понимать бизнесс и должен быть проффесионалом в реляционных (и NoSQL) базах данных;
- *Архитектор по безопасности*: решает вопросы защиты данных и безопасности. Отвечает за конфиденциальность данных;
- И др.- в зависимости от типа и размеров проектируемой проблемной области.

2.4. Менеджмент проектов ПО

Деятельностью менеджера является *достижение цели выполнения проекта*, которая рассматривается как согласование таких параметров, как ареал деятельности, сроки и финансы. Существуют различные аналитические и графические способы интерпретации взаимосвязей этих параметров.

- Одна из упрощенных схем показана на рис. 2.3 в виде так называемого "Треугольника менеджмента проектов" [28].

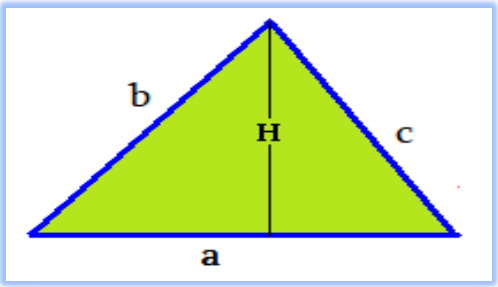


**Рис. 2.3. Треугольник менеджмента проекта,
где a – финансы (Расходы), b - ареал деятельности (Кадры),
c – время (Сроки выполнения)**

Задачу менеджмента программной системы можно объяснить следующим образом: необходимо такое согласование *ареала деятельности* (производительность кадров для выполнения определенной работы), *времени* (сроки выполнения работ) и *финансов* (использование ресурсов), которое удовлетворит требования *качества.*

Установление такого баланса в силу сложности выполняемых процессов и их неоднородности, довольно трудно. Поэтому менеджер целенаправленно выбирает один

или два параметра, а остальные считает зависимыми от выбранных. Поэтому треугольник получает такую интерпретацию: “Хорошо-Быстро–Дешево? – Выберите среди них два”.

На рис.2.3 варьированием значений длин (переменных) сторон треугольника получаются различные результаты. В качестве инвариантного показателя взаимосвязи этих параметров принимается площадь треугольника:

$$S = a \times H/2,$$

которая для конкретного проекта и конкретной команды исполнителей неизменная (константа).

Отсюда вытекает, если в треугольнике в качестве исходных будут выбраны две стороны, тогда вычисление третьей всегда возможно посредством площади и высоты.

- Сотрудник института управления проектами (*PMBOK* – Project Management Body of Knowledge) Р. Видеман ввел 4-й параметр – ***качество*** (рис.2.4).

Рис. 2.4

Получена четырехвершинная звездная схема в которой, несмотря на лучшие возможности содержательного описания, потеряла принцип простаты и важный инвариантный параметр – площадь.

- Решение укзанной проблемы стало возможным переходом в трехмерное пространство, в виде объема тетраэдра (рис.2.5).

Рис. 2.5

- На основе ввода третьего измерения Д. Мараско удалось построение “Проектной пирамиды” (рис. 2.6), которая является вероятностью успеха проекта (Probability of Success).

Основание пирамиды четырехугольник, стороны которого: *скорость (Speed)*, *экономичность (Frugality)*, *качество (Quality)* и *сфера деятельности (Scope)*.

Рис. 2.6

Высоте пирамиды соответствует *вероятность успеха проекта*. Эти пять параметров рассматриваются как переменные, значения которых связаны с инвариантом условий развития проекта силами команды его разработчиков. Инвариантом принят объем пирамиды.

Указанной моделью более реалистично наблюдение за деятельностью менеджера проекта в процессе выполнения и корректировки проекта. В виду того, что *вероятность успеха проекта* меняется во времени, а объем пирамиды постоянен, тогда изменение высоты пирамиды объязательно вызовет изменение переменных основания пирамиды. Справедливо и обратное утверждение, что вероятность успеха проекта зависит от площади основания.

Таким образом, так называемые “Треугольник менеджмента”, “Пирамида менеджмента” и т.д. являются иллюстрационными моделями, с помощью которых возможно глубокое исследование процесса проектирования программных систем.

Обобщенная схема методической работы менеджера заключается в следующем: с начала определяется проектная деятельность, затем выясняется какую корректировку надо выполнить и какие средства существуют для этого.

Результаты работы менеджера многообразны, однако они заключены в рамки с точки зрения установления баланса для достижения цели проекта.

2.5. Базовая модель жизненного цикла программной системы

Жизненный цикл программного обеспечения есть процесс конструирования компьютерной программной системы. Требования к программному обеспечению определяют цель и регламент этого процесса.

На рис. 2.7 показана традициооно полученная модель жизненного цикла программной системы. Она состоит из 2-х фаз и 7-и этапов.

Каждый этап характеризуется соответствующей *ролью менеджера и исполнителей проекта*. Если расмотреть *объектно-ориентированную* технологию центра IBM компании, то можно выделить довольно полный список *типичных ролей*, которые служат для выполнения и развития программных проектов [2-4].

Это роли участников проекта из форм программного обеспечения и корпораций заказчиков, которые влияют на как формирование задач проекта, так и на выделение соответствующих ресурсов и на выполнение работ для выполнения проекта и его развития.

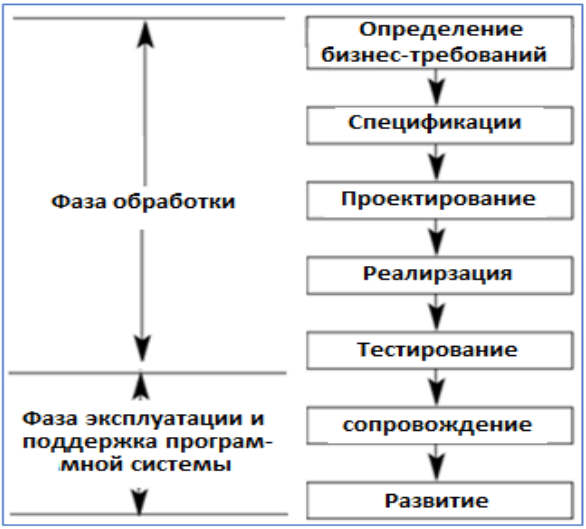


Рис. 2.7. Базовая модель жизненного цикла

Характеристика ролей включает перечень выполняемых организационных и производственных функций:

❖ *Заказчик (Customer)* – которому подчиняется команда проектировщиков или другой представитель организации, которому поручено подписание проекта, контроль выполнения и получение результатов;

❖ *Планировщик ресурсов (Planner)* – предоставляет и координирует требования к проекту в организации, где разрабатывается проект, а также с точки зрения организации развивает и сопровождает план выполнения проекта;

❖ *Менеджер проекта (Project Manager)* – ответственный за выполнением всего проекта, распределением заданий и ресурсов, прием результатов работ согласно графика выполнения, соответствие результатов требованиям. Он

активно сотрудничает с заказчиком и планировщиком ресурсов;

❖ *Руководитель проекта (Team Leader)* – осуществляет техническое руководство командой в процессе выполнения проекта. В больших проектах возможно существование нескольких руководителей в соответствии с подкомандами, которые ответственны за отдельные задачи;

❖ *Архитектор (Architect)* – ответственнен за проектирование архитектуры системы, согласовывает развитие работ, связанных с проектом;

❖ *Проектировщик подсистемы (Designer)* – несет ответственность за проектирование подситем или категории классов, определяет вопросы реализации и интерфейса с другими подсис- темами;

❖ *Эксперт предметной области (Domain Expert)* – несет ответственность за изучение и моделирование сопровождающих бизнесс-процессов исследуемой области, осуществляет поддержку проекта для решения задач в данной области;

❖ *Разработчик (Developer)* – осуществляет реализацию компонент проекта, владеет методами и создает специфические классы, пишет коды и тестирует их, создает программный продукт;

❖ *Разработчик информационного обеспечения (Information Developer)* – создает сопровождающие документы программного обеспечения, также материалы для инсталяции системы, инструкции для пользователя;

❖ *Специалист интерфейса пользователя (Human Factors Engineer)* – несет ответственность за легкое и удобное

применение системы, Работает с заказчиком, чтобы убедится, что интерфейс пользователя удовлетворяет требованиям;

❖ *Специалист по тестированию (Tester)* – проверяет функциональность продукта, качество и эффективность. Состовляет и применяет контрольные тесты для каждой фазы проекта;

❖ *Библиотекарь (Librarian)* – отвечает за создание общей библиотеки проекта, которая содержит все рабочие продукты проекта, а также соответствие этих продуктов стандартам;

С помощью языка унифицированного моделирования UML (Unified Modeling Language) построена UseCase диаграмма (рис. 2.8), соответствующей проблемной среды (рис. 2.9).

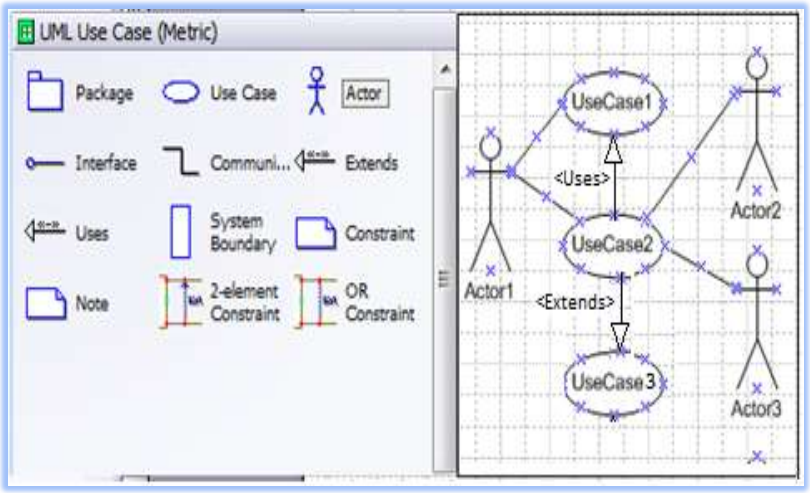


Рис. 2.8. Графический инструмент Ms VISIO - UML методологии для моделирования UseCase диаграмм (роли и функции)

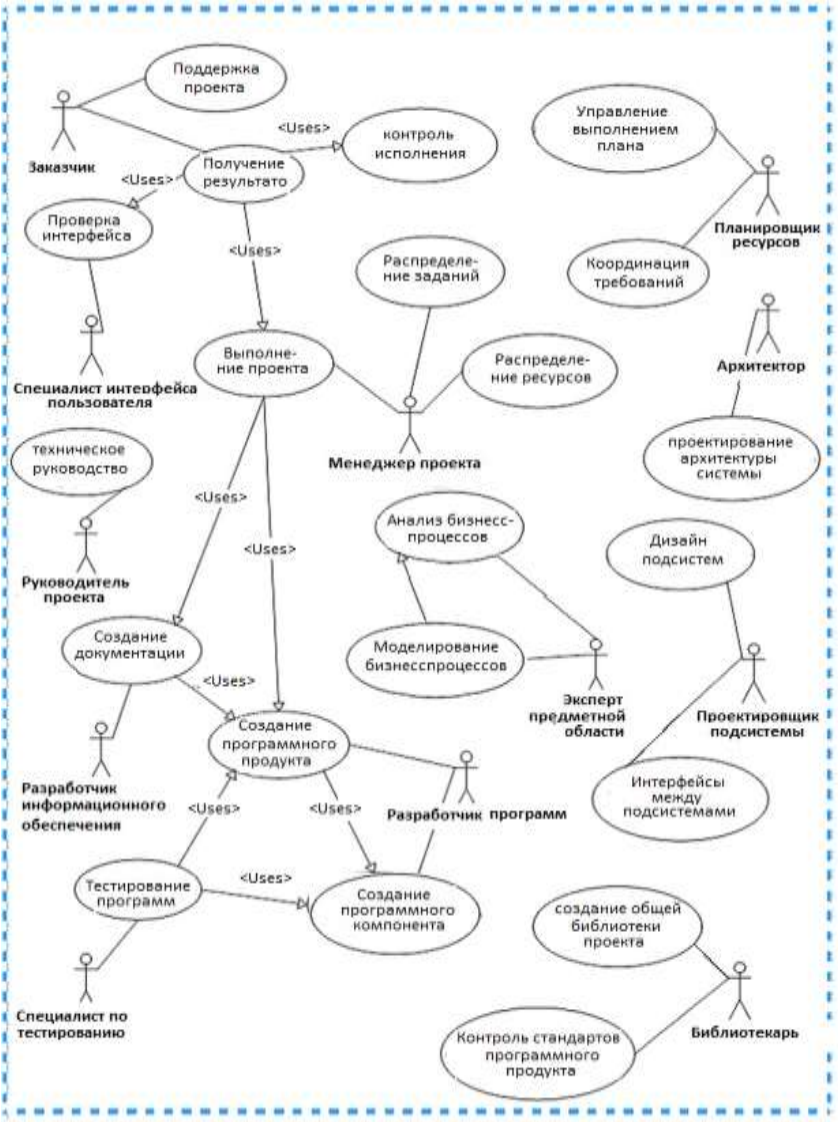


Рис. 2.9. Фрагмент UseCase диаграммы процессов менеджмента создания ПО

2.6. Классическая итерационная модель

Как было отмечено, жизненный цикл программного обеспечения – процесс построения программного обеспечения, этапы и регламент которого определяют функциональные и нефункциональные требования программной системы.

Мы рассмотрели традиционно полученную 7–ми этапную модель жизненого цикла программной системы (базовая модель). Возникает вопрос – может ли быть использована такая модель для менеджмента процесса построения любой программной системы: организации планирования, учета контроля и т.д. ?

Из-за ограничения указанной модели (например, здесь на каждом этапе не видны наглядно функции ролей исполнителей проекта), ее нельзя считать совершенной, хотя может быть принята как исходная (первоначальная) модель. Ее использование возможно для выполнения простых проектов, где не нужны итерационные процессы.

Итерация является возвращением к предыдущим шагам (этапам) в процессе создания программного продукта для проведения процедур корректировки.

Сложные проекты объязательно требуют проведение многочисленных итерационных процессов на всех этапах жизненного цикла программной системы как с целью исправления ошибок предыдущих этапов, так и из-за внесения изменений требований в условия эксплуатации системы.

Общая схема *жизненного цикла классической итерационной модели* показана на рис. 2.10.

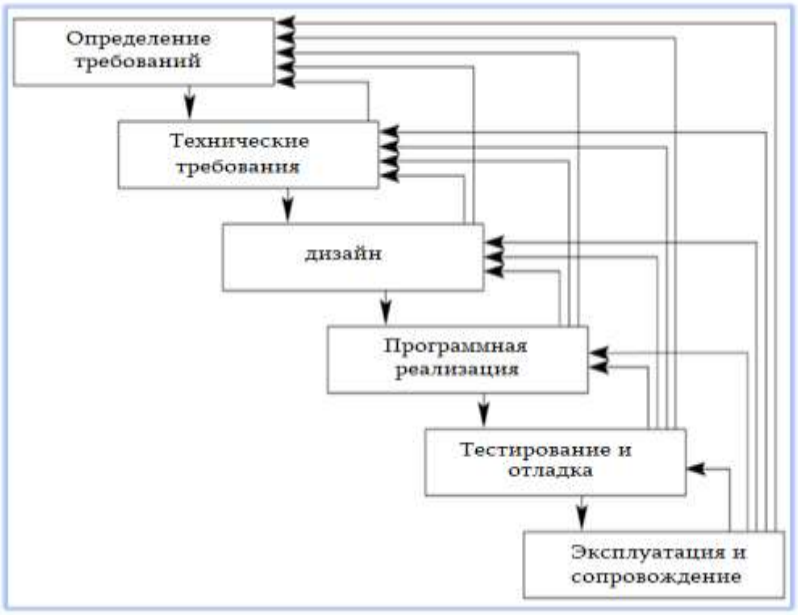


Рис. 2.10. Классическая итерационная модель

Стрелками (вверху) указаны итерационные процессы на предыдущем этапе, что вызвано целью исправления соответствующих ошибок и неопределенностей. Традиционные методы стараются минимизировать такие процессы, т.е. строго требуют, чтобы все было точно определено и исключают обратные итерации, что *можно считать их недостатком*.

Методы итерационного развития программных проектов актуальны, так как они не говорят о совершенстве этапов и не исключают их итерационного развития, совершенствование их функциональных и интерфейсных возможностей посредством расширения.

Таким образом, классическая модель жизненного цикла программной системы приемлема, однако в пределах одной итерации и важным допущением: все, что раньше было полезно, сохранится.

Идея сохранения старого кода, без потери его эффективности, полностью связана с объектно-ориентированным программированием и проектированием, применением CASE (Computer-Aided Software Engineering – Автоматизация программирования) инструментов (к этому вопросу вернемся позже).

Целью итерационного расширения является повышение гибкости системы, обеспечение возможностей ее адаптации во время изменения условий проекта и программной системы. Внесение итерационных шагов позволяют повысить адаптацию проекта, так как при испольэовании такого подхода в проект легче вносить изменения. Такая концепция противоречит традиционным методологиам построения программных систем, если в процессе разработки и архитектуре не будут учтены механизмы адаптации.

В обьектно-ориентитрованном подходе этот механизм более развит, поэтому он может быть принят основой всей идеалогии развития программных проектов.

Новые методологии программных систем в целом остаются объектно-ориентированными, однако стараются освобождения этой методологии от строгих требований.

Например, *метод экстремального программирования* (Agile методология) считает допустимым изменение результатов первичной декомпозиции системы (дизайна системы) в процессе выполнения проекта в результате уточ- нения требований пользователей.

Такое мировозрение требует создание специального варианта менеджмента программного проекта.

2.7. Каскадная модель (Waterfall model)

Каскадная модель сравнительно строгая разновидность классической итерационной модели, которая является иллюстративным примером того, как можно *минимизировать итерацию*. Ее также называют Waterfall model.

Для каскадной модели характерно: Завершение каждого этапа (как в классической модели) проверкой полученных результатов, чтобы исключить как можно больше проблем разработки программы; Циклическое повторение пройденных этапов (как в классической модели).

Мотивация каскадной модели связана *с управлением качества программного обеспечения*. В связи с этим уточняется смысл этапов, некоторые из них структурируются (спецификация требований и реализация). На рис. 2.11 показана схема каскадной модели, которая построена как модификация классической итерационной модели.

В каждом блоке (этапе) указано действие, которым заканчивается блок. Здесь тестирование не выделено отдельным этапом, оно считается препятствием, которое надо преодолеть, чтобы закончился этап, также, как и другие похожие действия (например, обзоры – документы, в которых описываются системные требования и они должны быть согласованы и утверждены заказчиком).

Результат проектирования верифицируется (проверяется), обеспечивает или нет принятая структура системы и механизмы реализации выполнение специфических функций.

Реализация контролируется путем тестирования, а после интеграции компонент в системе проводится аттестация для комплексной наладки. Т.е. осуществляется проверка-фиксация реализованных функций системы, описание ограничений реализации и т.д.

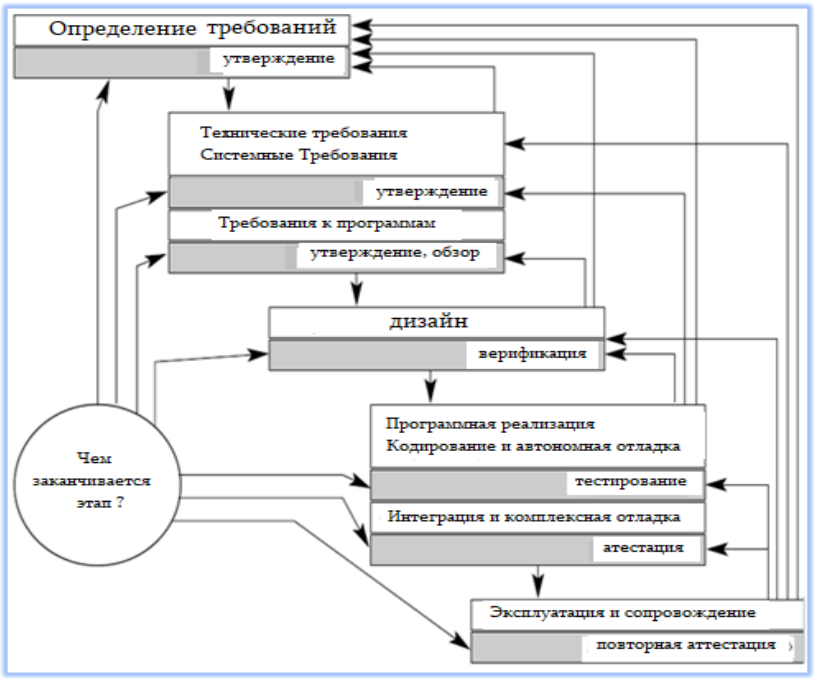


Рис.2.11. Каскадная модель (waterfall model)

В каскадной модели верификация и аттестация приписаны различным этапам:

Верификация отвечает на вопрос - правильно или нет построена программная система (т.е она проверяет соответствие спецификации и ее проводят проектировщики и программисты);

Аттестация (валидация) отвечает на вопрос – правильно ли работает программная система она рассматривается во время эксплуатации системы и о ее готовности заключение готовят специалисты – заказчики).

В *процессе эксплуатации и сопровождения* выясняется, насколько хорошо соответствует система требованиям потребителей, т.е. проводится повторное тестирование.

После каждой проверки возможен возврат проектировщиков на любой пройденный этап, кторый показан обратными стрелками. С целью минимизации обратных шагов в каскадной модели проектировщики применяют строгую проверку (в литературе это известно под именем *строгой каскадной модели*). Отметим ее характерные моменты жизненного цикла:

Точное распределение работ, заданий и ответственности между разработчиками этапов и их проверяющими, которые дают добро на переход к следующему этапу;

Существование малых циклов между соседними этапами, в результате которых достигается выполнение комплексного задания.

Использование рассмотренных методов жизненного цикла программных систем не является универсальным средством и зависит от конкретного объекта и от множества других факторов.

Разница между водопадной и итеративной моделями:

- в водопадной модели пользователь участвует в решении требований на этапе коммуникации;
- в инкрементной модели фаза коммуникации происходит в каждой итерации, поэтому пользователь участвует в каждой итерации.

2.8. Спиральная модель

Спиральная модель жизненного цикла разработки программной системы является комбинацией модели итерационного процесса и водопадной модели. Главное отличие в том, что в спиральной модели особое внимание уделяется *анализу рисков и управлению* (что отсутствует в водопадной модели). Анализ рисков включает возможность технической реализации, оценки и мониторинга, а также идентификацию рисков менеджмента, таких как задержка графика и превышение затрат.

В конце первой итерации, после тестирования программного продукта, заказчик проводит валидацию программного обеспечения и передает соответствующую информацию исполнителю (обратная связь). Основные этапы: анализ, проектирование, реализация, тестирование, внедрение-эксплуатация в циклическом эволюционном развитии (рис.2.12).

Рис.2.12. Спиральная модель с 4-мя фазами

Версии программной системы (Vers_1,2,3,...) создаются в виде прототипов, она развивается в виде спирали и приближается к требованиям заказчика – потребителя.

В процессе тестирования и эксплуатационного внедрения участвуют системные аналитики и заказчик, что дает возможность оперативно определить необходимые изменения для следую- щей версии.

2.9. MSF модель

Методология Microsoft Solution Framework (MSF) разработки программной системы объединяет свойства упоря-дочения и гибкости водопадной и спиральной моделей (рис.2.13) [2,4].

Рис.2.13. Фазы проекта

Базовые принципы следующие:

1) Единный взляд на проект. В начале и заказчик и команда проектировщиков имеют свои взгляды на цели и задачи проекта, и что должно быть достигнуто при работе над проектом;

a. Успех проекта невозможен без единного взгляда заказчика и команды проектировщиков, так как из-за недорозумений цель остается недостигнутой.

b. MSF методология для этого учитывает выделение отдельного этапа (фазы) – “разработка концепции”.

2) Выявление гибкости – готовность к изменениям. В отличии от каскадной модели MSF подход основан на принципе беспрерывного изменения свойств проекта;

3) Концентрирование на бизнес-принципах. Разрабатываемый продукт должен принести определенную пользу для конечного пользователя.Так как программный продукт может принести пользу только после внедрения, поэтому MSF подход в жизненном цикле учитывает фазу внедрения;

4) Поощрение свободных взаимоотношений.MSF модель процессов предполагает открытый и свободный обмен информацией как между исполнителями проекта, так и между всеми лицами, заинтересованными в проекте (stekeholders). Это снижает недопанимание между ними, снижает непредвиденные расходы. Поэтому MSF предусматривает систематический анализ процесса работы в устанновленном пункте времени, в котором участвуют заказчики и исполнители.

Основными новшествами MSF являются следующие [2]:

1) Акцент на внедрении IT-решения;

2) Модель процесса, объединяющая спиральную и водопадную модели;

3) Особая организация команды – не иерархическая, а как группа равных, но выполняющих разные функции (роли) работников;

4) Техника управления компромиссами.

Детальное ознакомление с моделью MSF можно в [2].

2.10. Модель Гантера: “Фазы и функции”

В моделях жизненного цикла программных продуктов отражены производственные функции, которые выполняют проектировщики. Эти функции должны быть связаны с контрольными элементами управления проектом, т.е. с этапами жизненного цикла. Они проводятся на протяжении всего периода разработки проекта с разной интенсивностью.

Модель Гантера *фазы и функции* является основой для построения развитого жизненного цикла, в котором отображены организационные и технические производственные функции. Кроме того, модель Гантера учитывает также итерации.

Модель должна быть основой организации взаимодействия разработчиков проекта. Одной из ее *целей* является *поддержка функций менеджера*. А это требует в проекте расположить контрольные точки, которые отображают организацинно-временные рамки проекта, организационно–технические производственные функции, выполняемые во время развития проекта. У модели Гантера есть два измерения:

– *фазовое,* которое отображает этапы выполнения и сопровождающие события;

– *функциональное*, в котором видно, какие производственные функции выполняются в процессе развития проекта и какова их *интенсивность* на каждом этапе.

На схеме модели Гантера ось абсцисс отображает развитие проекта, она является осью времени. На ней расположены контрольные точки (к.т.) и наименования событий (рис.2.14).

Рис. 2.14. Модель Гантера

- *Этап исследования* – начинается, когда необходимость *разработки* признана руководством проекта (к.т.-0), и заключается в том, что для проекта обосновываются необходимые ресурсы (к.т.-1) и Устанавлены требования к разрабатываемой программе (к.т.-2);
- *Анализ осуществимости* – начинается на *этапе исследования*, когда определены исполнители проекта (к.т.-1), и завершается утверждением требований (к.т.-3). Цель *этапа* – определить возможность *конструирования* изделия с технической точки зрения (достаточно ли ресурсов, квалификации и т.п.), будет ли изделие удобно для практического испольования; решение вопросов экономической и коммерческой эффективности;
- *Конструирование* – начинается обычно на *этапе анализа осуществимости*, как только документально зафиксированы предварительные цели проекта (к.т.-2), и заканчивается утверждением проектных решений в виде официальной спецификации на *разработку* (к.т.-5);
- *Программирование* – начинается на этапе конструирования, когда становятся доступными основные спецификации на отдельные компоненты программы (к.т.-4), но не ранее утверждения соглашения о требованиях (к.т.-3). Совмещение данной фазы с заключительным этапом конструирования обеспечивает оперативную проверку проектных решений и некоторых ключевых вопросов разработки. Цель этапа – Разработка компонентов программ для их дальнейшей сборки в приложение. Он завершается, когда разработчики заканчивают документирование, отладку и компоновку и передают изделие службе, выполняющей независимую оценку результатов работы (независимые испытания начались – к.т.-7);

- *Оценка* – является буферной зоной между началом испытаний и практическим использованием программы. Этап начинается, как только проведены внутренние (силами разработчиков) испытания аппликации (к.т.-6) и заканчивается, когда подтверждается готовность изделия к эксплуатации (к.т.-9);
- *Использование* – начинается ближе к концу этапа оценки, когда готовность программы к эксплуатации проверена и может организовываться передача текущей версии аппликации на распространение (к.т.-8). Этап продолжается, пока приложение находится в действии и интенсивно эксплуатируется. Он связан с внедрением, обучением, настройкой и сопровождением, возможно, с модернизацией программ. Этап заканчивается, когда разработчики прекращают систематическую деятельность по сопровождению и поддержке данного программного изделия (к.т.-10);

Модель фазы–функции не описывает возможности итеративного возврата на предыдущие этапы. Строгую каскадность в ней еще можно усмотреть в изображении перекрытия *этапов*. Но для отражения произвольных возвратов модель в том виде, в котором она предлагается Гантером, непригодна. Если попытаться развить модель Гантера с целью учета *итеративности* в том смысле, как она понимается в классической итеративной и каскадной моделях, то, очевидно, придется предусмотреть *расщепление линии жизненного цикла*, как показано на рис. 2.14.

Но это влечет за собой и расщепление матрицы интенсивностей выполняемых *функций*: было бы необоснованно считать, что интенсивности при возвратах сохраняются. В

целом, *по* мере продвижения *разработки* к ее завершению, они должны уменьшаться.

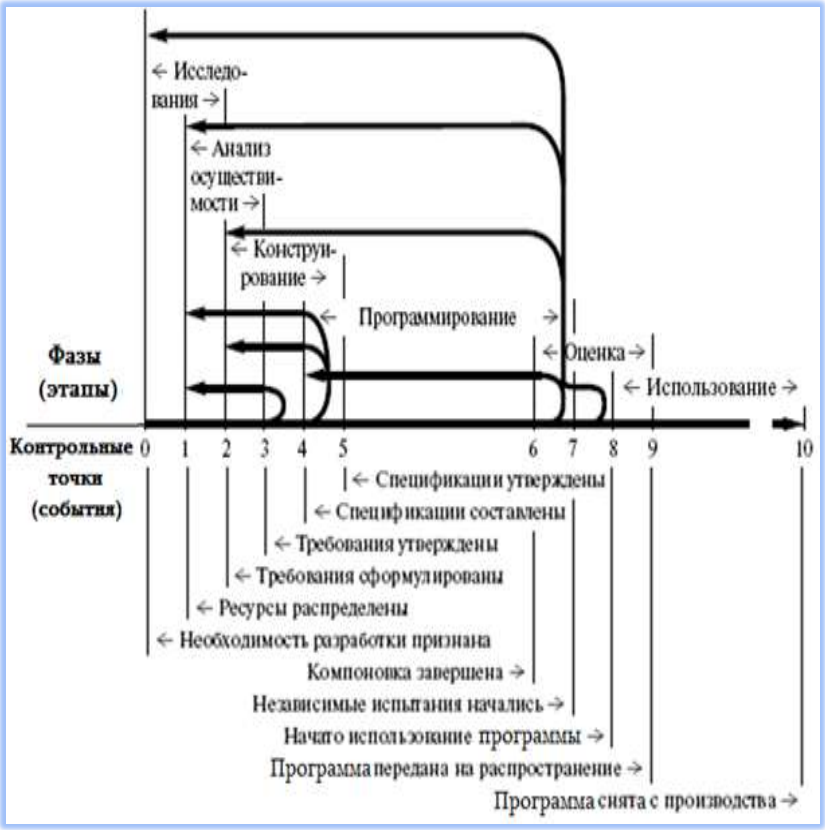


Рис. 2.14. Итеративность в модели Гантера

Таким образом, матрица интенсивностей приобретает новое измерение, отражающее итеративный характер развития проекта.

Расщепление линии жизненного цикла (приостановка основного процесса, переход к точке продолжения, в которой возобновляются указанные на схеме действия), выполнение процесса до точки *расщепления* и слияние *линий жизненного цикла*, которое понимается как возобновление основного процесса – эта схема процедур допускает, что приостановка может осуществляться в любой момент – нужно только позаботиться о мерах, которые в будущем обеспечат корректное слияние.

Итеративность неизбежна при разработке сложных программных продуктов, а потому ее планирование целесообразно.

2.11. Методология DevOps и инструментальные средства

Agile и DevOps методологии выполняют взаимодополняющие роли. Например, автоматизированное конструирование и тестирование программной системы, непрерывная интеграция и непрерывное предоставление [11, 12]. Agile можно считать механизмом устранения коммуникационных недостатков между заказчиком и девелоперами, а DevOps ориентирована на устранении существующих недостатков между девелоперами и IT операциями /инфраструктурами.

DevOps методология объединяет операции девелопмента (Dev) и информационных технологий (Ops) программного обеспечения (рис.2.15).

Рис.2.15. DevOps методология

Целью данной методологии является сокращение времени цикла создания программной системы и предоставление программного обеспечения высокого качества [3]. Кроме этого, DevOps уделяет внимание размещению (deployment) разработанного программного обеспечения по задачам.

DevOps является концепцией командной работы, поэтому не существует одного конкретного инструмента для его осуществления. Наоборот, существует совокупность нескольких инструментов ("цепочка DevOps инструментов"-рис.2.16), которые хорошо отражают аспекты девелопмента и предоставления заказчику программной системы [12]:

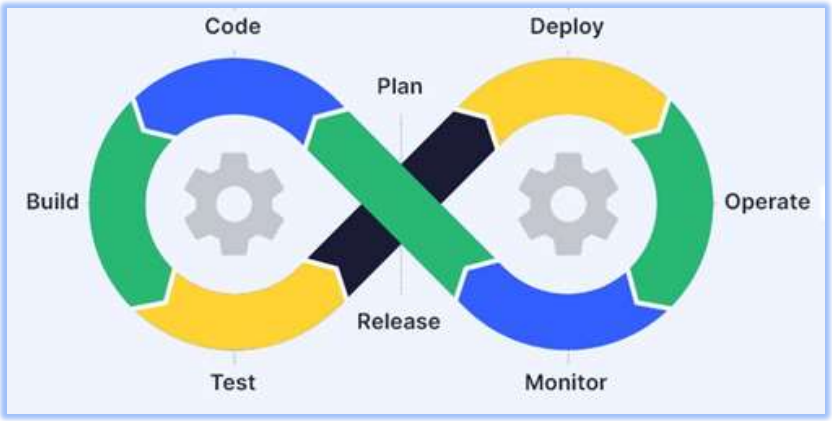


Рис.2.16

1) Coding – девелопмент и анализ кода, инструмент менеджмента исходного кода (Source Code Management) [13];

2) Building - инструмент непрерывной интеграции (Continuous Integration), статус конструирования [14];

3) Testing – инструмент непрерывного тестирования (Continuous Testoing), который обеспечивает быструю обратную связь в соотвествии с бизнес рисками [15];

4) Packaging – репозитарий артефактов (хранилище данных – data warehouse), предварительное размещение апликаций.В хранилище хранятся программные пакеты и их метаданные, журнал изменений для менеджмента системы [16];

5) Releasing – менеджмент изменений, утверждение выпуска версии (релиза), автоматизация версии апликации (Application-release automation - ARA) [17];

6) Configuring – конфигурация и менеджмент инфраструктуры, инфраструктура как инструмент кода (Infrastructure as code – IaC) [18];

7) Monitoring – мониторинг производительности (быстродействия) апликаций, опыта работы с конечным пользователем (Application Performance Management - APM). APM старается выявить проблемы при выполнении комплексных программ ;

8) И др.

Глава 3
Унифицированные методолдогии разработки программных проектов

Методологии жизненного цикла разработки ПО (SDLC), которые мы обсуждали в предыдущей главе, в основном создавались со второй половины прошлого века и развиваются до сих пор [29]. Эти итеративные и инкрементальные модели являются *„тяжеловесными“* методами (например, Waterfall model), характеризующиеся строгими ограничениями на процессы планирования и изменений.

В последнее десятилетие века стали развиваться *„облегченные“* методологии. Такие как например: RAD – быстрая разработка приложений, XP – экстремальное программирование, RUP – Rational Unified Process, UML – Unified Modeling Language, Agile – гибкие методы разработки ПО и.т.д..

С начала 21 века стали быстро развиваться UML и Agile методологии разработки программного обеспечения.

UML – используется для крупных проектов, а Agile – для небольших и быстрых проектов среднего размера. Оба подхода считаются актуальными и важными направлениями на сегодняшний день. Также было создано третье направление – управления программными проектами, основанное на их совместном, компромиссном использовании. Например, „UML диаграммы для Agile Software групп“ (Tool Visual Paradigm).

В этой главе мы подробно обсудим эти вопросы, поскольку они очень важны для объектно-ориентированного подхода к анализу, проектированию, тестированию и внедрению современных программных систем [5,6].

3.1. Унифицированный язык моделирования - UML

UML (Unified Modeling Language) – это графический язык, используемый для создания чертежей программного обеспечения [5]. UML можно описать как язык визуального моделирования общего назначения для визуализации, спецификации, построения и документирования программной системы. Хотя UML обычно используется для моделирования программных систем, он не ограничивается этими границами.

В основе языка UML лежат принципы объектно-ориентированного подхода (классы, объекты, инкапсуляция, наследование, полиморфизм, абстракция и т.д.). Язык учитывает методологию жизненного цикла программных продуктов. Его инструментальные средства включают CASE технологии (например, Enterprise Architect – совместим с Visual Studio.NET для C#, VB, a Visual Paradigm – для языка Java и т.д.). Это вопросы автоматизации программирования. Из построенной модели (диаграммы) классов можно сгенерировать программный код в соответствии с указанным языком.

На рис.3.1 показана классическая модель UML технологии (4-этапа), с 4-я парами диаграмм. Из них 4 статические модели и 4 динамические.

Рассмотрим кратко пример инструментального средства (Tool Ms Visio) языка UML, которое широко применяется для создания этих диаграмм (до этапа программирования).

Можно использовать также и другие CASE инструменты языка UML, например, Enterprise Architect фирмы SparX или Ms Visual Studio .NET Framework 4.7.2 (2015) (Templates: Modeling Projects).

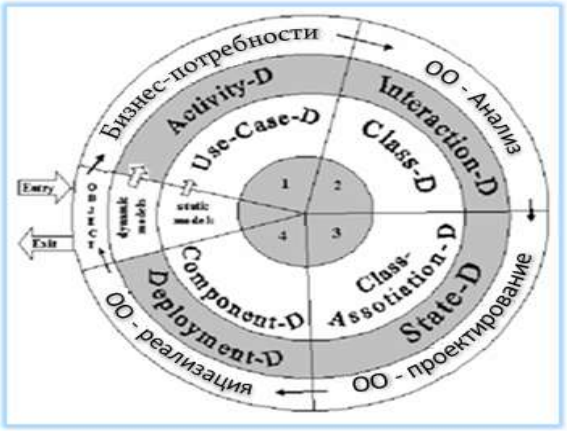


Рис. 3.1. Классическая модель UML методологии

3.1.1. UseCase диаграмма

Первая диаграмма, которую надо построить с помощью UML методологии, есть диаграмма прецендентов (UseCase diagram). Она является схемой взаимосвязи ролей (Actors) и функций (Actions) (рис.3.2). Здесь UseCase1 соответствует первой роли, а в выполнении UseCase2 участвуют все три роли.

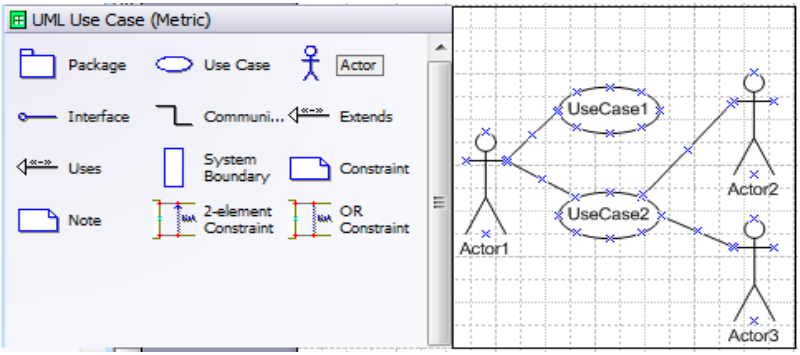


Рис.3.2.Интерфейс построения UseCase диаграммы

3.1.2. Activity диаграмма

На основе изучения, анализа и структурной формализации бизнес-процессов и бизнес-правил проектируемого объекта будет построена диаграмма активностей. Она является конкретной функцией конкретной роли (ролей), которая состоит из последовательно или параллельно выполняемых суб-действий. Они иерархически расположенны в пространстве и во-времени. Схема имеет один вход и несколько взможных окончаний, процедуры разветвления или соединения, определяемые бизнес-правилами, исходную, промежуточную или итоговую документацию и т.д. (рис.3.3 а-в).

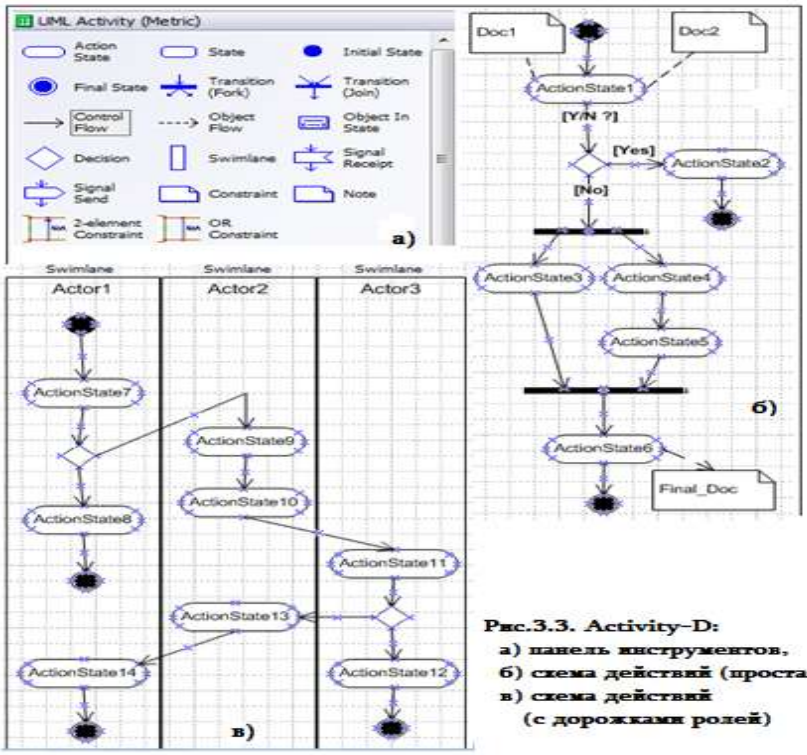


Рис.3.3. Activity-D:
а) панель инструментов,
б) схема действий (простая),
в) схема действий
(с дорожками ролей)

3.1.3. Краткий обзор этапов методики UML

На *первом этапе* строятся UseCase и Activity диаграммы. В них отражены роли сотрудников проектируемого объекта (Actors) и их функции (Actions). Эти две модели предполагают ознакомление, изучение объекта, и они нужны для того, чтобы исполнитель проекта мог установить бизнес-потребности объекта исследования.

На *втором этапе* проводится объектно-ориентированный анализ, строятся Sequence и Collaboration диаграммы. Они описывают сценарий работы конкретной *роли* с компьютерной системой для выполнения *конкретной функции* (задачи). Отдельные действия сценария бизнес-задачи расположены *последовательно* во времени. Здесь также видна последовательность обмена информацией и сообщениями между человеком и компьютерной системой. На этом этапе отдельные классы (имя класса, атрибуты, методы – инкапсулированная структура) представляются в виде статической модели. Диаграмма последовательности использует эти классы в сценарии.

На *третьем этапе* осуществляется объектно-ориентированное проектирование (процесс дизайна), строятся Class-Association и State диаграммы. *Схема отношений* классов строится с использованием специальных отношений, таких как наследование, ассоциация, реляционные и функциональные отношения между классами. *Диаграмма состояний* строится только тогда, когда в логической схеме процесса могут существовать два или более возможных продолжения (в

варианте if… else… then). Каждое продолжение моделируется отдельной State-схемой.

Последний, *четвертый этап* касается задачи построения диаграмм Components компьютерной системы и Deployment. Компоненты – это набор программных модулей, которые программируются девелоперами и тестируются тестировщиками. После получения положительных результатов разработчики системы разместят отдельные рабочие программы на компьютерах пользователей соответствующих подразделений объекта. Физически распределенная система логически объединена в единый пакет и управляется системным администратором. На крупных объектах также может существовать роль администратора базы данных.

После внедрения компьютерной системы в производство начинается этап эксплуатации. Как традиционно принято в моделях жизненного цикла программных систем, и здесь методология UML учитывает процесс поддержки системы. То есть, если в процессе эксплуатации была обнаружена ошибка в работе программы, либо было изменено какое-либо бизнес-процесс или бизнес-правило, то девелоперы системы продолжают работу по устранению указанных неисправностей и выполнению новых задач (на основе контракта поддержки системы).

Примечание: **методология UML будет подробно изучена в отдельном курсе (в следующем семестре).**

3.2. Agile методология программирования и методы девелопмента апликаций

Гибкая методология программирования (Agile Software Methodology) рассматривает различнные Agile методы, такие как Extreme Programming (XP), Scrum, Kanban/Lean, Agile Unified Process (AUP), Dynamic Systems Development Method (DSDM), Disciplined Agile Delivery (DAD), Feature-Driven Development (FDD), Test-Driven Development (TDD), Rapid Application Development(RAD) и Scaled Agile Framework (SAFe).

Здесь мы детально рассмотрим концепции гибкого программирования, их принципы и некоторые методы. Особенно широко используемые (XP, Scrum) и экономные методы с визуальными свойствами (Kanban/Lean) [6,8,11].

3.2.1. Принципы экстремального программирования

С февраля 2001 года официально была определена новая методология разработки программных систем Agile (гибкая). Вместо хаотичного подхода (code-and-fix) и строго формализованных инженерных методологий, начало развиваться семейство *гибких* методологий, которые основаны на принципах манифеста разработки программного обспечения и его реализации (рис.3.4) [6].

По степени формализации гибкие методологии занимают промежуточное положение между рассмотренными выше подходами, т.е. они не так строго формализованы, как инженерные и не так хаотичны (бессистемный подход), как code-and-fix

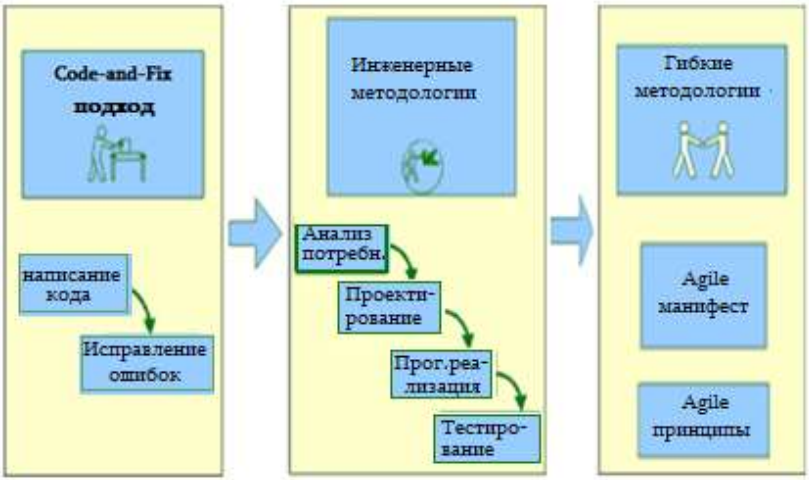


Рис.3.4. Переход на Agile методологию

Идеи, заложенные в Agile метологию не новы. Итерационная разработка и особая роль человеческого фактора и другие идеи были известны и до февраля 2001 года, но только с этого времени впервые были предложены в виде рекомендаций, как прктически проверенная альтернатива традиционным подходам разработки программных систем

Отличие Agile методологии от UML хорошо видно из ее манифеста, принятого в 2001 году, на первом официальном сборе сторонников этого направления.

Agile Manifesto состоит из четырех основных концепций и 12 принципов.

Левая часть концепции содержит понятия и аспекты, которые имеют большее значение в разработке программного обеспечения, чем те, которые помещены в правую часть:

– Люди и отношения важнее процессов и инструментов;

– Рабочий продукт важнее обширной исчерпывающей документации;

– Сотрудничество с заказчиком важнее условий, оговоренных в договоре;

– Быть готовым к изменениям важнее, чем придерживаться первоначального плана.

Принципы:

1) Удовлетворенность клиентов за счет раннего и непрерывного предоставления ценного программного обеспечения;

2) Приветствие изменения требований даже на последнем этапе работы (это повышает конкурентоспособность конечного продукта);

3) Частая поставка рабочего ПО заказчику (ежемесячно, еженедельно или чаще);

4) Частый, ежедневный контакт заказчика с поставщиком на протяжении реализации проекта;

5) Над проектом работают мотивированные люди, которым обеспечены необходимые условия работы, поддержка и доверие;

6) Рекомендуемый способ передачи информации - личные беседы (с глазу на глаз);

7) Работающее программное обеспечение — лучший показатель прогресса. Целью проектов является создание программной системы, а не планов и документации. Оценивая

производительность приложения, можно объективно измерить ход проекта;

8) Спонсоры, поставщики и клиенты должны иметь возможность поддерживать постоянный темп на неопределенный срок;

9) Постоянное внимание к совершенствованию технического мастерства (навыков) и удобному дизайну;

10) Простота - искусство без лишней работы. Не нужно принимать сложные универсальные решения, если нет явной необходимости;

11) Лучшие технические требования, дизайн и архитектура создаются самоорганизованной командой;

12) Постоянная адаптация к изменяющимся обстоятельствам. Итеративный жизненный цикл основан на управлении с обратной связью, важным элементом которого является анализ результатов, реализация обратной связи и улучшение процесса.

3.2.2. Scrum – фреймворк гибкого метода

В сфере программной индустрии предстовителем Agile методологии является Scrum метод. Впервые его упомянули японцы, как новый подход для разработки новых сервисов и продуктов (не только для программных продуктов). Основная идея метода заключалась в работе малочисленной универсальной команды, которая разрабатывает все фазы проекта. Аналогия делалась с игрой регби, когда единная команда передвигается вперед и назад в соответствии с передачей мяча.

➢ **Общее описание.** Метод дает возможность разработать проект гибко (быстро) командой из 5-9 человек. Процесс разработки носит итерационный характер и дает большую свободу команде. Вместе с этим метод очень прост и его легко изучить, поэтому часто применим на практике (рис.3.5).

Рис.3.5.Шаги Scrum метода

В начале определяются потребности ко всему продукту. Затем из них выбираются самые актуальные и состовляется первый (последующий) план итерации. В периоде итерации план не изменяется (это стабилизирует процесс разработки), его продолжительность 2-4 недели. Итерация заканчивается созданием рабочей версии продукта, которая может быть передана заказчику, продемонстрирована даже с минимальными функциональными возможностями.

После этого анализируются результаты и корректируются требования к продукту. Это дает возможность не только уточнить функции ПО, но и позволяет заказчику пользоваться им. Затем планируется следующая итерация и все повторяется.

Внутри итерации работает вся команда, Scrum тут не определяет роли. Синхронизация между менеджментом и заказчиком осуществляется после завершения итерации. Процесс итерация может оборваться только в исключительном случае.

- **Роли.** В Scrum методе есть *три роли*:

- *Владелец продукта (Product Owner)* – это менеджер продукта, который представляет интересы заказчика. Его объязанность определение исходных требований (BacklogProduct) к продукту, их своевременная корректировка, установление приоритетов, сроков сдачи и др. Он непосредственно не участвует в выполнении итерации;
- *Scrum*-мастер (*Scrum Master)* – обеспечивает максимальную производительность и продуктивность команды как для выполнения Scrum-процесса, так и для решения хозяйственных и административных задач.В частности, его задача-защита команды от внешних воздействий во время итерации;
- *Scrum*-команда (*Scrum Team)* – группа, состоящая из 5-9 независимых, инициативных программистов-девелоперов. Первая задача группы – определение для итерации реально достижимого и для проекта приоритетного задания (на основании Project Backlog-а и при активном участии владельца продукта и Scrum-мастера). Вторая задача- это объязательное выполнение задания в установленные сроки и требуемого качества. Очень важно, что команда сама участвует в процессе постановки заданий и их выполнении. Здесь согласованы свобода и ответственность, хорошо отражена дисциплина обязанностей.

➢ **Практики.** В системе Scrum определены следующие практики:

- *Sprint Planning Meeting.* Встреча для планирования Sprint–а (это короткая дистанция, т.е. итерация). В начале владелец продукта, Scrum-мастер, команда, также представитель заказчика и другие заинтересованные лица определяют, какие требования из Project Backlog приоритетные и какие должны быть выполнены в пределах данного спринта. Формируется Sprint-Backlog. Затем Scrum-мастер и Scrum-команда определяют, как должна быть достигнута цель из Sprint-Backlog-а. Для его каждого элемента определяется список задач и оценивается их трудоемкость;
- *Daily Scrum Meeting* –ежедневное, 15-тиминутное Scrum совещание, цель которого определение того, что было сделано после предыдущего совещания, корректировка рабочего плана в соответствии с сегодняшним днем и определение путей решения существующих проблем. Каждый член Scrum-команды отвечает на три вопроса: что сделал после предыдущего совещания, какие у него проблемы и что должен сделать к следующей встрече. В совещании может участвовать любое заинтересованное лицо, но принятие решения имеют только члены Scrum-команды. Это правило оправдано, так как они берут объязательство достижения цели итерации. Вмешательство постороннего лица снимает с них ответственность за итог;
- *Sprint Review Meeting.* Встреча для обзора Sprint-а осуществляется после окончания каждого спринта (рис.3.6).

Рис. 3.6. Scrum-метод со Sprint-шагами

В начале Scrum-команда демонстрирует продукт, который осуществился во время данного спринта. Будут приглашены все *заинтересованные лица заказчика*.

Владелец продукта определяет, какие требования из Sprint Backlog-а были выполнены и вместе с командой и представителем заказчика определяет, как лучше распределить приоритеты в последующем спринте Sprint-Backlog-а. Вторая часть встречи посвящается анализу предыдущего спринта, который проводит Scrum-мастер. Scrum-команда анализирует положительные и отрицательные результаты, полученные на последнем спринте, делает выводы и принимает важные решения о дальнейшей работе.

Scrum-команда определяет также пути для повышения эффективности будущих работ.

В дальнейшем цикл повторяется.

3.2.3. Унифицированный процесс Agile девелопмента

Rational Unified Process (RUP) – это гибкая методология разработки программного обеспечения. RUP делит жизненный цикл проекта на четыре этапа. На каждом из этапов все шесть основных дисциплин разработки включают бизнес-моделирование, требования, анализ и проектирование, программирование, тестирование и развертывание-внедрение.

Agile Unified Process (AUP) основан на урезанной версии Rational Unified Process (RUP). AUP использует гибкий подход, который фокусируется как на более широком жизненном цикле, так и на итерациях на каждом этапе для предоставления дополнительных выпусков (релизов) с течением времени.

Подводя итог, Agile и Unified Process стремятся обеспечить прозрачность и наглядность процесса управления проектами. Самая большая разница между ними заключается в том, что Unified Process фокусируется на структуре, а Agile – на скорости и гибкости.

Visual Paradigm предоставляет мощную платформу, которая позволяет разработчикам программного обеспечения систематически создавать качественное программное обеспечение быстрее, лучше и дешевле за счет использования возможностей автоматизированного процесса Scrum, отслеживаемых визуальных моделей UML и огромного хорошо интегрированного гибкого набора инструментов.

Глава 4

Разработка программной аппликации компьютерной системы (применительно для Курсового проекта)

В качестве предметной сферы рассмотрим „Университет", а проблемной области „Управление учебным процессом на факультете". С учетом объектно-ориентированного подхода и этапов жизненного цикла ПО, например, с использованием (частично) UML методологии, получим следующие фазы:

- Определение бизнес-потребностей объекта,
- ОО-анализ и проектирование системы,
- программная реализациия проекта,
- тестирование,
- внедрение программной системы (возможно поэтапное) на факультете.

С целью определения бизнес-потребностей проектируемой системы следует определить „Роли и Функции“ (т.е. для кого строится ПО и для каких задач (функций).

Первый этап работы группы „системных инженеров“ является исходной точкой или постановкой задачи нашего курсового проекта. В процессе активно участвует бизнес-аналитк и менеджер проекта, как члены команды построения ПО. В следующей части этой главы представлены образец оформления курсового проекта и результаты поэтапного движения разработчика (или команды) от постановки задачи к созданию программного продукта.

Условно, мы должны создать проект компьютерной системы „Учебный процесс“ ВУЗ-а на платформе VS.NET, используя Windows Forms App приложение и БД Sql Server.

Здесь представлена форма титульного листа проекта:

Грузинский Технический Университет
Факультет информатики и систем управления

Департамент Программной инженерии

Наименование темы:

Разработка программного обеспечения учебного процесса факультета

Студент группы: ######
Фамилия Имя

Руководитель курсового проекта проф. Фамилия И.

Тбилиси – 2023

4.1. Определение бизнес-потребностей объекта

Учебный процесс ВУЗ-а включает в себя такие понятия (классы и объекты), как: студент, преподаватель, академическая_группа, академический_курс (предмет), преподаватель, аудитория, лекция, лаборатория, факультьет, кафедра и др.

На рис.4.1 представлены Роли, участвующие в учебном процессе и их Функций.

[Вопрос к итоговому экзамену: ***(a)*** *– Постановка задачи: Анализ бизнес-потребности объекта проектирования*]

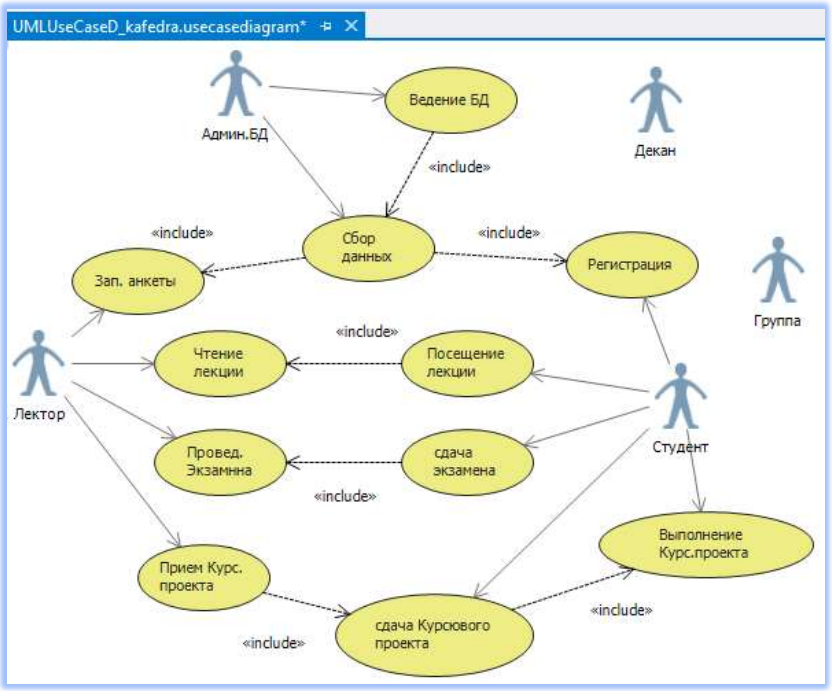


Рис.4.1. Моделирование объекта исследования:
Университет → Кафедра → Студент
[Диаграмму можно построить в Word, MsVisio или Vis.Stud.NET 2015]

Каждая роль имеет свои функции (овалы), которые отражают определенные бизнес-задачи организациии и могут быть запрограммированы (после проведения комплексного анализа соответствующего подразделения).

Связи между функциями (овалами: ассоциативные, агрегатные, реляционные и функциональные) отражают причинно-следственные связи между ними.

Какая функция какой роли должна быть детально проанализирована и запрограммирована разработчиками, определяет заказчик по согласованию с руководителем проекта. Последовательность программирования функций также зависит от них.

4.2. Создание структуры базы данных с таблицами

Рассмотрено создание базы данных MsSQL Server с пакетом Management Studio для задачи автоматизации процесса обучения. Мы построили пять таблиц (Tables) и ER-модель (рис. 4.2-4.9).

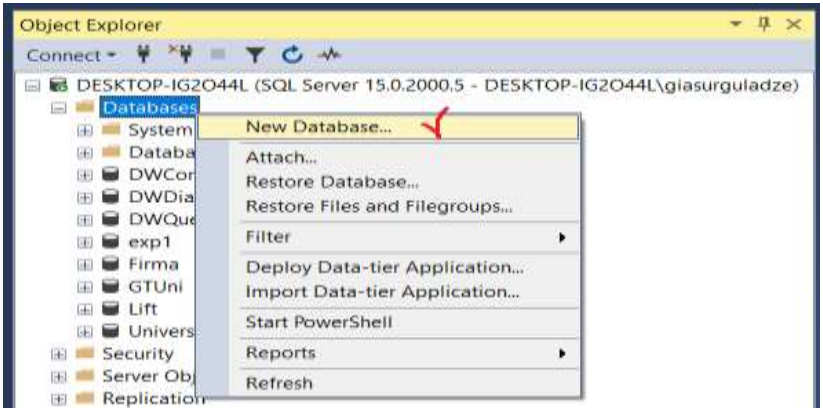


Рис. 4.2. Создание БД кафедры (Kaf)

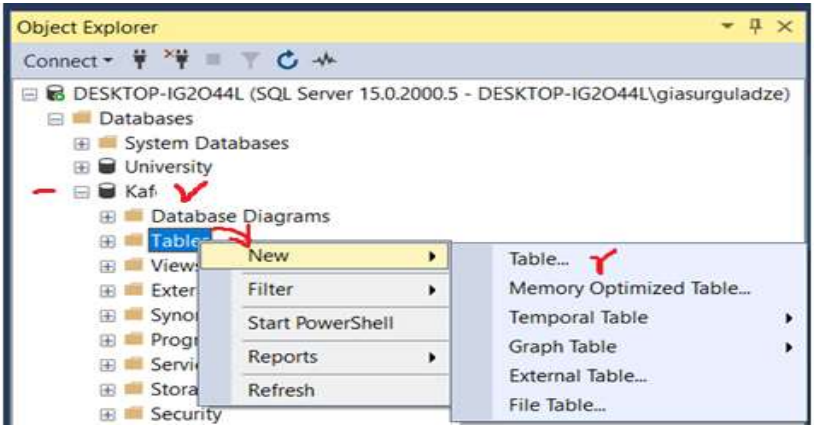


Рис.4.3. Создание таблиц БД

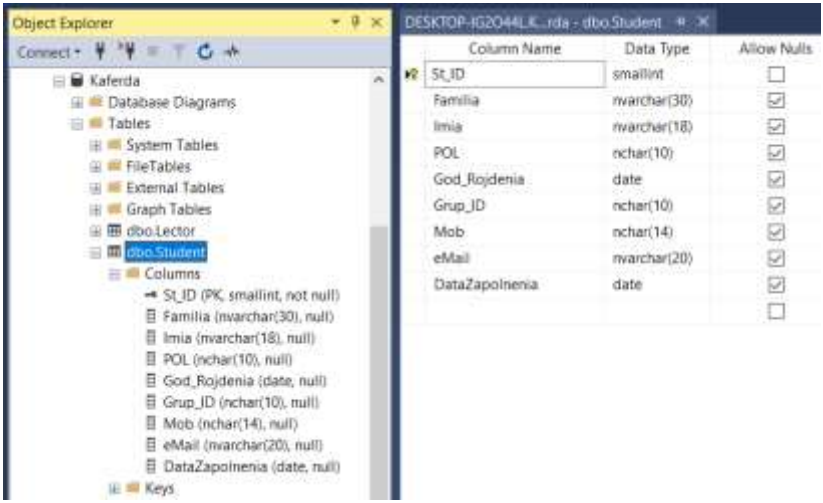


Рис.4.4. Создание таблицы Student

В правой части окна показаны атрибуты таблицы Student с соответствующими типами данных и длинами полей. Первичный ключ этой табицы - St_ID.

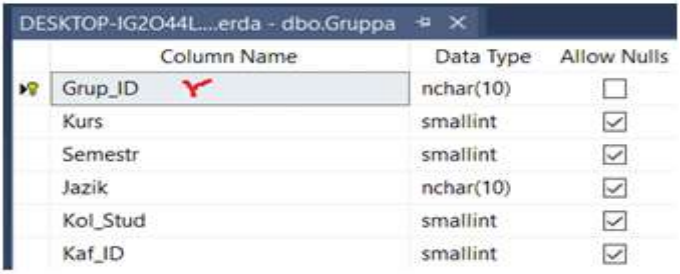


Рис.4.5. Таблица акад-группа (Gruppa)

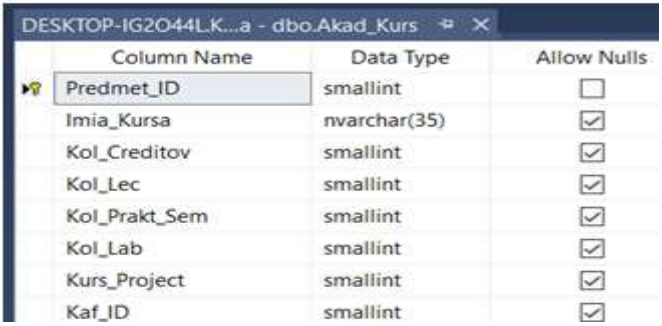


Рис.4.6. Таблица акад-курс (Predmet)

Рис.4.7. Таблица кафедра (Ksfedra)

DESKTOP-IG2O44L.Kaferda - dbo.Lector

Column Name	Data Type	Allow Nulls
Lector_ID	smallint	☐
Familia	nvarchar(30)	☑
Imia	nvarchar(15)	☑
POL	nchar(10)	☑
GodRojdenia	date	☑
Doljnost	nvarchar(20)	☑
Zarplata	money	☑
Mob	nchar(14)	☑
eMail	nvarchar(30)	☑
Kaf_ID	smallint	☑
DataZapolnenia	date	☑

Рис.4.8. Таблица лектор (Lector)

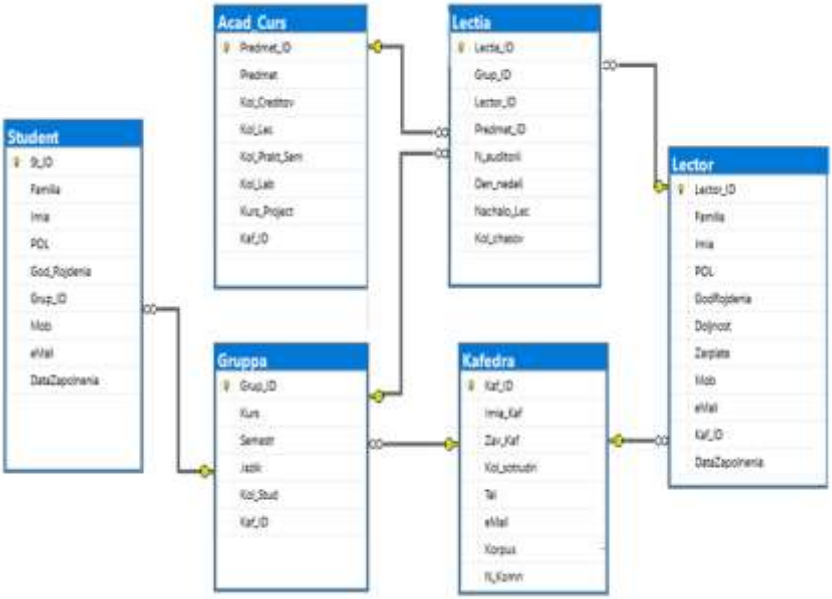


Рис.4.9. Диаграмма таблиц (ER-Model)

[Вопрос к итоговому экзамену: *(б) – Проектирование БД для Интерфейса пользователя*].

4.3. Создание нового проекта на C# для нтерфейса пользователя

В системе Visual Studio .NET Framework создаем новый проект (с именем и локацией).

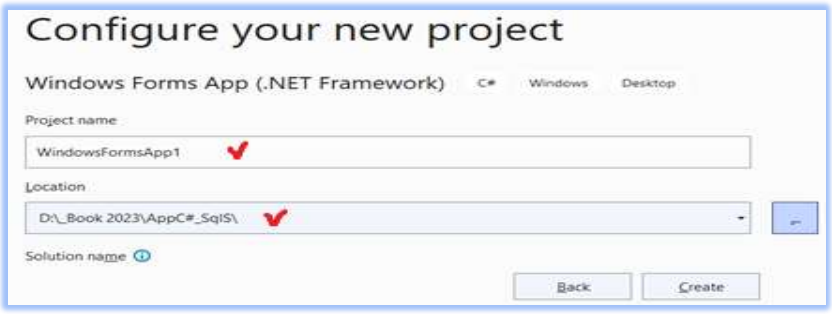


Рис.4.10. Создание проекета

В форме «Пользовательский интерфейс» проекта выберите источник данных (файл базы данных SQL Server), используя подключение (connection)

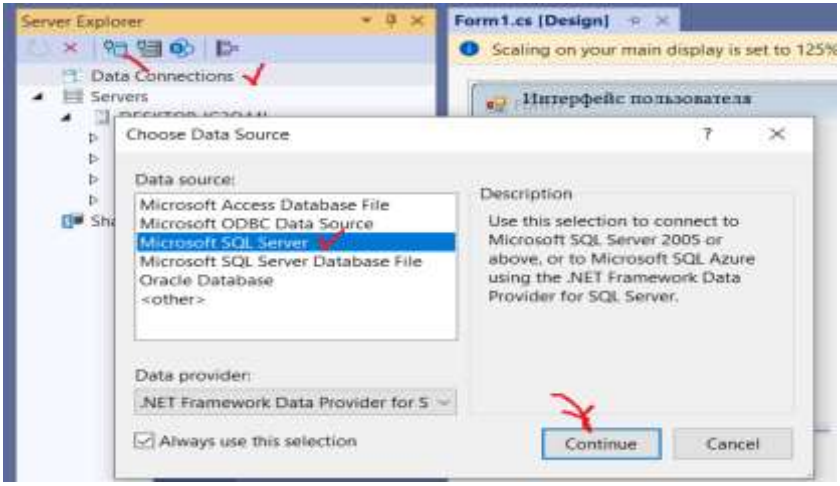


Рис.4.11. Выбор источника данных

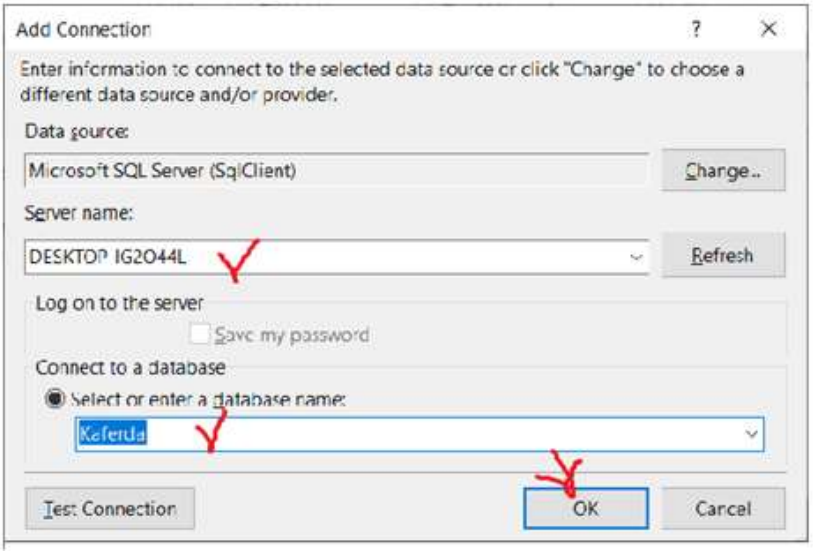


Рис.4.12. Выбор имя сервера и БД

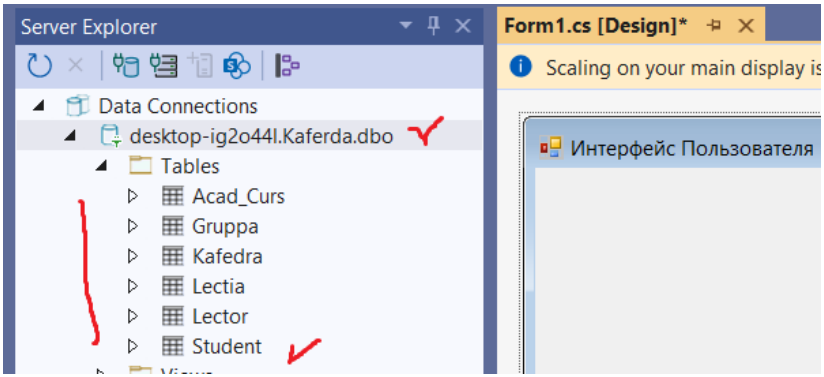


Рис.4.13. Результат выбора БД для проекта

[Вопрос к итоговому экзамену: *(б) – Подключение (Connecting) проекта к базе данных*].

4.4. Элемент DataGridView и определение параметров таблицы

Переместите элемент DataGridView с панели инструментов на форму и активируйте правый верхний угол маленькой стрелкой. Получаем рисунок 4.14.

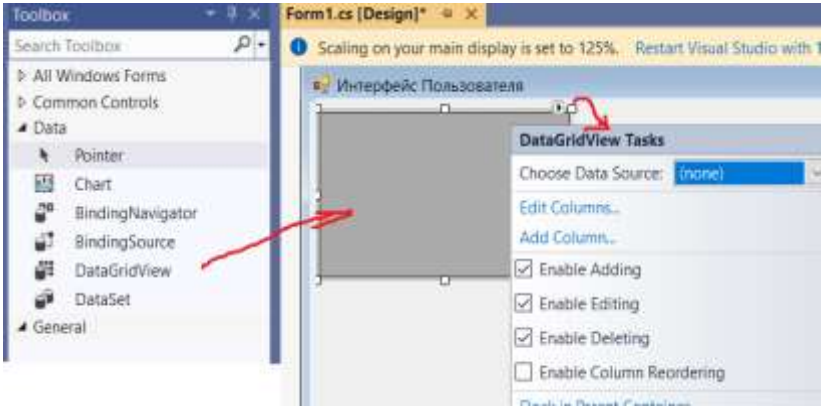


Рис.4.14. Форма с элементом DataGridView и определение параметров

На рисунке видно, что операции вставки, редактирования и удаления разрешены (флажки отмечены). Если мы нажмем кнопку выпадающего списка Choose Data Source, мы получим Рисунок 4.15.

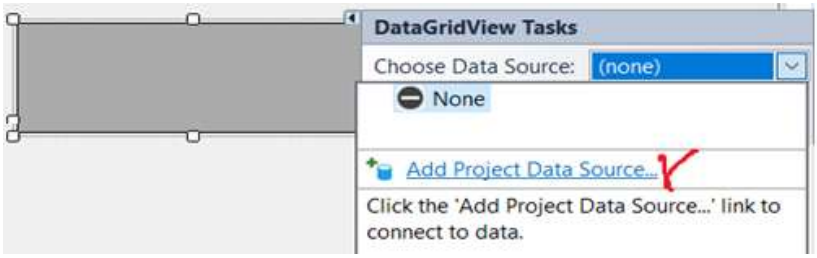


Рис.4.15. Добавление данных в таблицу DataGridView

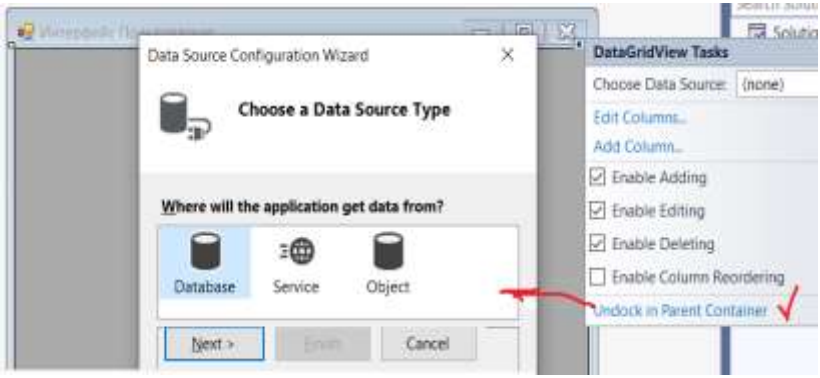


Рис. 4.16. Выбираем Database и Next

Каждый компьютер будет иметь свою собственную настройку подключения. Его можно определить из Server Explorer (в нашем случае это: desktop-ig2o44l.Kaf.dbo). Наконец, нажмите Next и Connection String сохраняется в файле конфигурации Аппликации. Затем будет вызвано окно выбора объекта базы данных (Рис. 4.17).

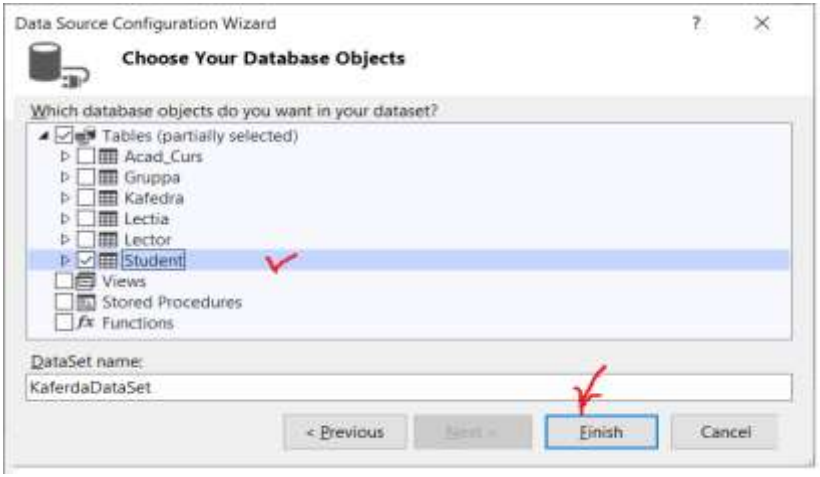


Рис. 4.17. Выбор объектов (например, Student)

На рис. 4.18 показано значение результата определения источника данных, в нашем случае это:

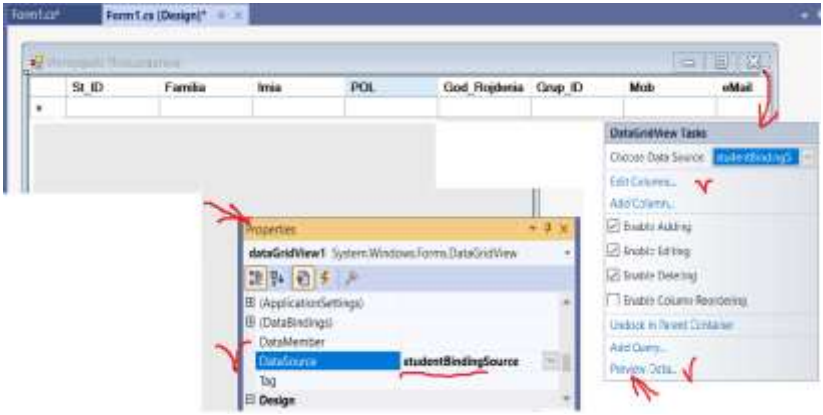


Рис. 4.18. Результат источника данных: „StudentBindingSource“

В окне DataGridViews Tasks Столбцы таблицы можно редактировать с помощью кнопки Edit Columns (Рис.4.19).

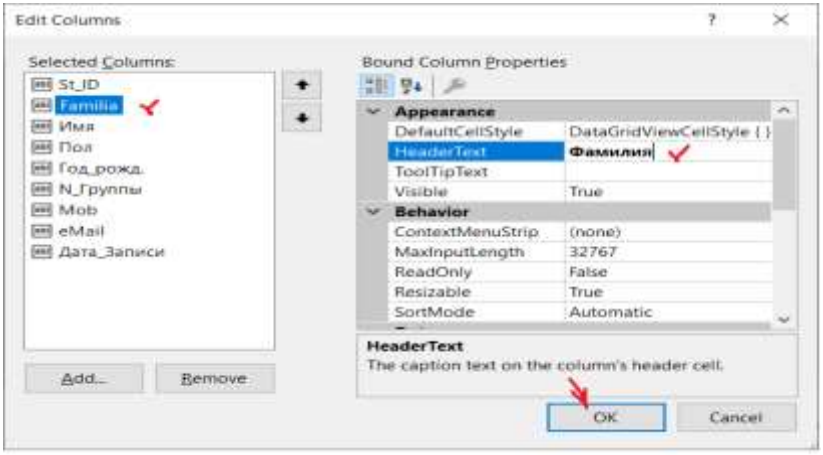


Рис.4.19. Edit Columns

Перетащим с панели инструментов три кнопки (Button 1, 2. 3), назовем их (рис.4.20). Добавим также BindingNavigator из раздела Data панели инструментов, в верхний-левый угол для будущего использования.

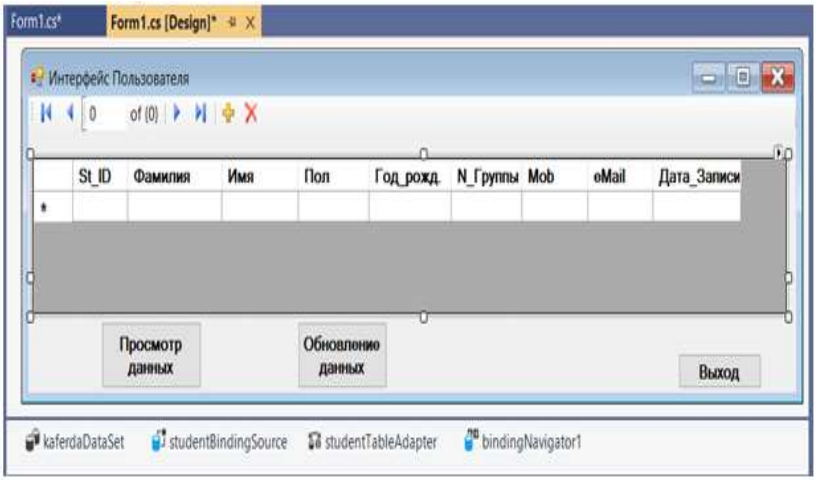


Рис.4.20. Фрагмент основного интерфейса проекта

[Вопрос к итоговому экзамену: *(б) – Отображение базовых данных в таблице интерфейса DataGridView*].

4.5. Программирование CRUD – операций

Теперь перейдем к программированию функций кнопок системного интерфейса, то есть к просмотру, вставке, изменению и удалению записей соответствующей таблицы базы данных SQL Server.

Сначала добавьте строку пространства имен:

using System.Data.SqlClient;

Далее объявим глобальные переменные (для работы с ADO.NET):

```
SqlDataAdapter sda;
SqlCommandBuilder scb;
DataTable dt;
```

Для кнопки «Просмотр данных» (button1) у нас будет следующий код:

```
private void Button1_Click(object sender, EventArgs e)
 {
   SqlConnection con = new SqlConnection("Data Source=DESKTOP-
        IG2O44L;  Initial Catalog = Kaferda; Integrated Security = True");
   sda = new SqlDataAdapter(@"SELECT St_ID, Familia, Imia,
       POL,God_Rojdenia, Grup_ID, Mob, eMail from Student", con);
   dt = new DataTable();
   sda.Fill(dt);
   dataGridView1.DataSource = dt;
  }
```

Запустив программу, если в ней нет ошибок, получается рисунок 4.21.

Код кнопки «Обновление данных» (Insert, Update, Delete = Вставить, Обновить, Удалить) показан ниже:

```
private void Button2_Click(object sender, EventArgs e)
  {
     scb = new SqlCommandBuilder(sda);
     sda.Update(dt);
  }
```

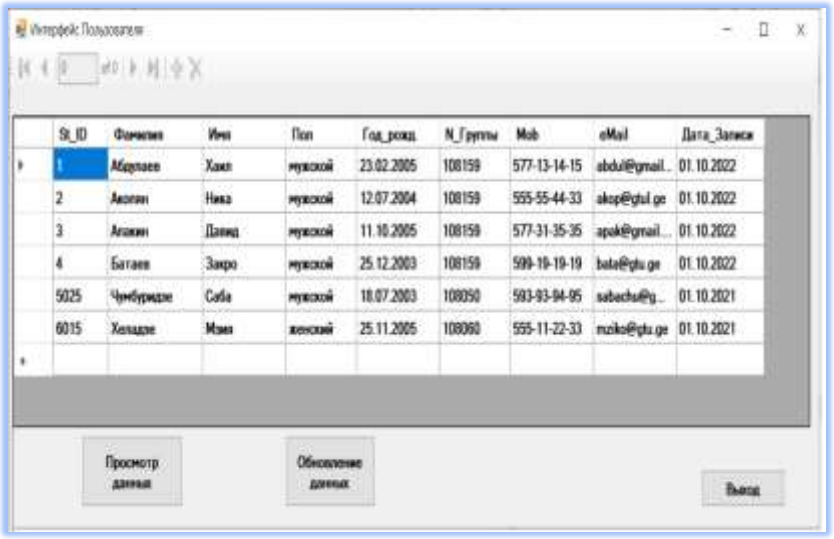


Рис.4.21. Результат по кнопке "Просмотр данных"

Полный код программы приведен в листинге.

```
//-- Листинг ---------------
  using System;
  using System.Collections.Generic;
  using System.ComponentModel;
  using System.Data;
  using System.Drawing;
  using System.Linq;
  using System.Text;
  using System.Threading.Tasks;
  using System.Windows.Forms;
  using System.Data.SqlClient;    // добавили !

  namespace WindowsFormsApp1
  {
     public partial class Form1 : Form
     {
        SqlDataAdapter sda;
        SqlCommandBuilder scb;
```

```
        DataTable dt;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        // TODO: This line of code loads data into the 'kaferdaDataSet.Student'
table.
            // You can move, or remove it, as needed.
            this.studentTableAdapter.Fill(this.kaferdaDataSet.Student);

        }

        //--Просмотр данных -------
        private void button1_Click(object sender, EventArgs e)
        {
         SqlConnection con = new SqlConnection("Data Source=DESKTOP-
IG2O44L;Initial" +
                " Catalog = Kaferda; Integrated Security = True");
            sda = new SqlDataAdapter(@"SELECT St_ID, Familia, Imia,
POL,God_Rojdenia,
                        Grup_ID, Mob, eMail from Student", con);
            dt = new DataTable();
            sda.Fill(dt);
            dataGridView1.DataSource = dt;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            scb = new SqlCommandBuilder(sda);
            sda.Update(dt);
        }

        private void button3_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

[Вопрос к итоговому экзамену: ***(в)*** *– Программирование методов обработки данных: Insert, Update, Delete C# коды*].

- Этап валидации ПО и внерения:

На рис. 4.22 показано изменение двух значений (Mob) в существующей таблице (новые числа показаны в кружке), затем нажатие 1-й и 2-й кнопок запишет эти измененные значения в базу данных (мы можем закрыть программу и начать заново).

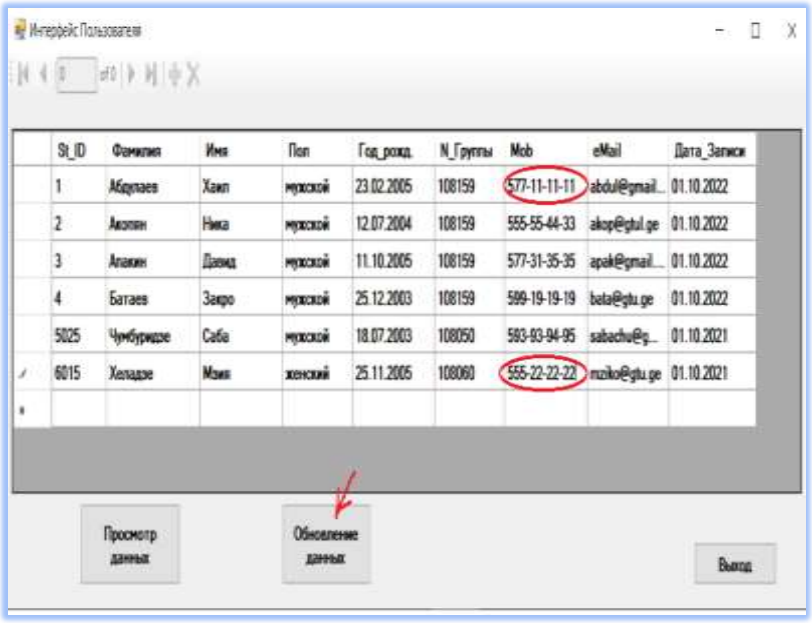


Рис.4.22. Обновление (Update) в поле Mob

Добавим новую строку с помощью Insert.

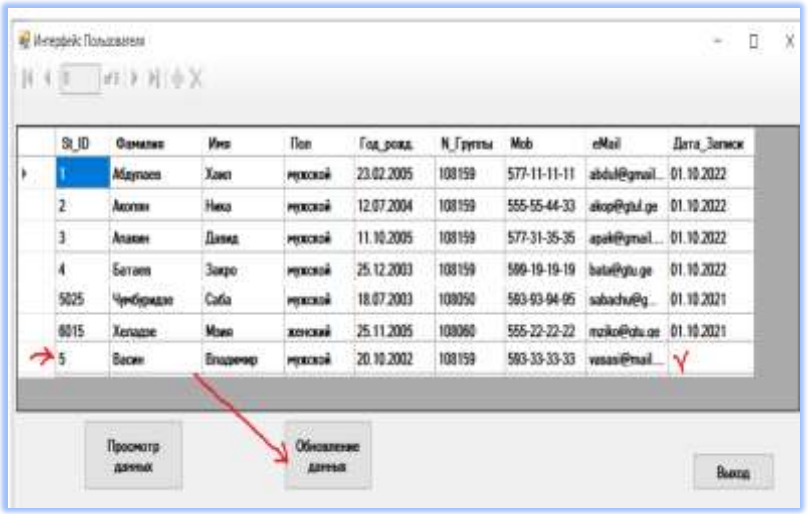
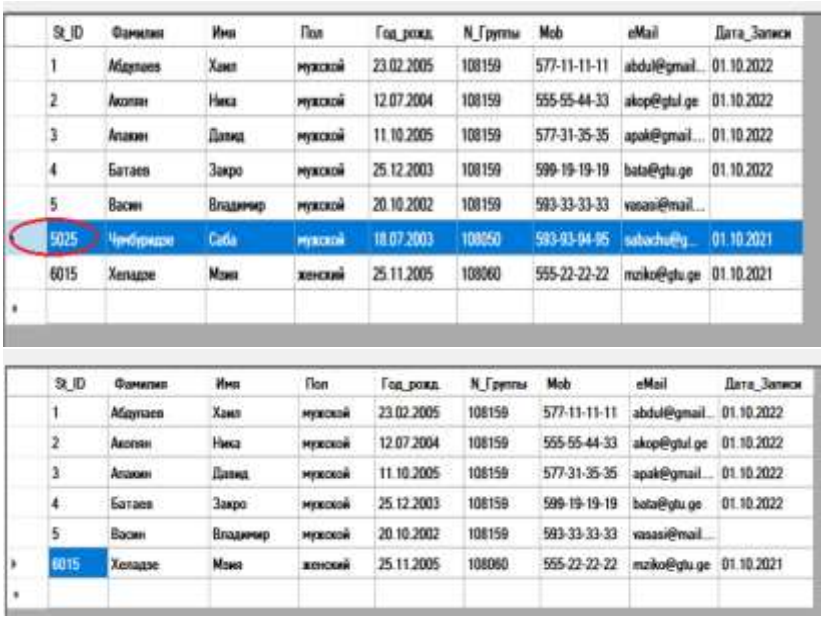


Рис.4.22. Обновление (Insert)

Рис.4.23. Запуск программы после обновления

Наконец, отображается операция удаления строки (Delete). Это осуществляется, например, выделением соответствующей строки «St_ID» и последующим нажатием кнопки «Удалить» (Delete) на клавиатуре компьютера (рис. 4.24).



St_ID	Фамилия	Имя	Пол	Год_рожд.	N_Группы	Mob	eMail	Дата_Записи
1	Абдулаев	Хамз	мужской	23.02.2005	108159	577-11-11-11	abdul@gmail...	01.10.2022
2	Акопян	Ника	мужской	12.07.2004	108159	555-55-44-33	akop@gtul.ge	01.10.2022
3	Апакин	Давид	мужской	11.10.2005	108159	577-31-35-35	apak@gmail...	01.10.2022
4	Батаев	Закро	мужской	25.12.2003	108159	599-19-19-19	bata@gtu.ge	01.10.2022
5	Васин	Владимир	мужской	20.10.2002	108159	593-33-33-33	vasasi@mail...	
5025	Чембуридзе	Саба	мужской	18.07.2003	108050	593-93-94-95	sabachu@g...	01.10.2021
6015	Хеладзе	Мзия	женский	25.11.2005	108060	555-22-22-22	mziko@gtu.ge	01.10.2021

St_ID	Фамилия	Имя	Пол	Год_рожд.	N_Группы	Mob	eMail	Дата_Записи
1	Абдулаев	Хамз	мужской	23.02.2005	108159	577-11-11-11	abdul@gmail...	01.10.2022
2	Акопян	Ника	мужской	12.07.2004	108159	555-55-44-33	akop@gtul.ge	01.10.2022
3	Апакин	Давид	мужской	11.10.2005	108159	577-31-35-35	apak@gmail...	01.10.2022
4	Батаев	Закро	мужской	25.12.2003	108159	599-19-19-19	bata@gtu.ge	01.10.2022
5	Васин	Владимир	мужской	20.10.2002	108159	593-33-33-33	vasasi@mail...	
6015	Хеладзе	Мзия	женский	25.11.2005	108060	555-22-22-22	mziko@gtu.ge	01.10.2021

Рис.4.24. Выделение и удаление строки

На этом мы завершили обсуждение задачи реализации C# аппликации с подключением к базе данных на основе визуальных элементов C#, в результате чего был построен фрагмент информационной системы для управления Кафедрой университета.

[Вопрос к итоговому экзамену: *(г) – Внедрение: Запустите программную аппликацию и представте результаты*].

Глава 5
Создание ASP.NET Web приложений

5.1. Создание интерактивной Web –страницы с помощью ASP.NET APP

Задача: на Ms Visual Studio .NET платформе с использованием ASP.NET построить новую Web – страницу, в которую пользователь занесет личные данные и перешлет на сервер [1,6-8].

- создать новую ASP.NET апликацию с именем проекта WebAppRegister5 (Рис.5.1);

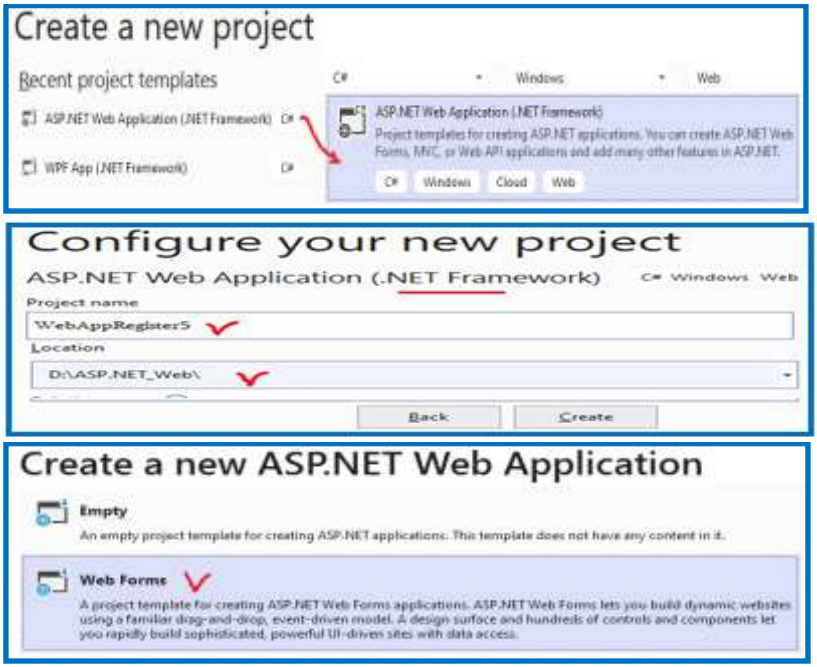


Рис.5.1

- получаем Solution Explorer (Рис.5.2 – левый). Заменим Default имя новым, в частности именем Registration (Рис.5.2 – правый).

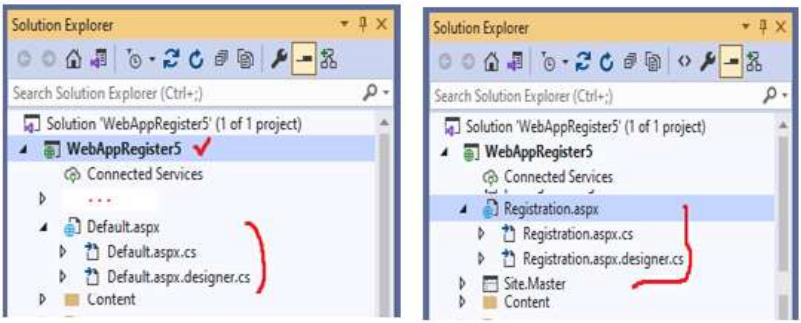


Рис.5.2

- 1-я строка Registration.aspx файла будет иметь вид, показанный на рис.5.2, где отображены имена проекта и программ .aspx и .aspx.cs (рис. 5.3);

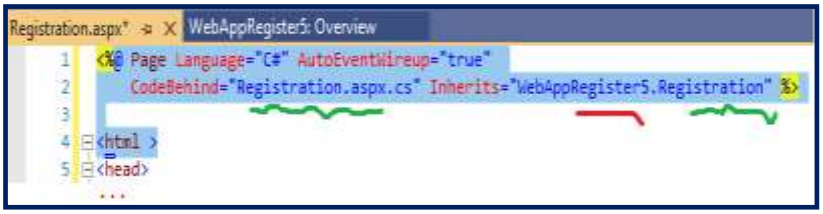


Рис. 5.3

- листинг исходного текста файла Registration.aspx.cs имеет следующий вид:

```
// ----- Листинг_5.1 Registration.aspx.cs.------
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace WebAppRegister5
{
  public partial class Registration : Page
  {
     protected void Page_Load(object sender, EventArgs e)
     {
     }
  }
}
```

- Макет регистрационной формы Web – страницы показан на рис.5.4.

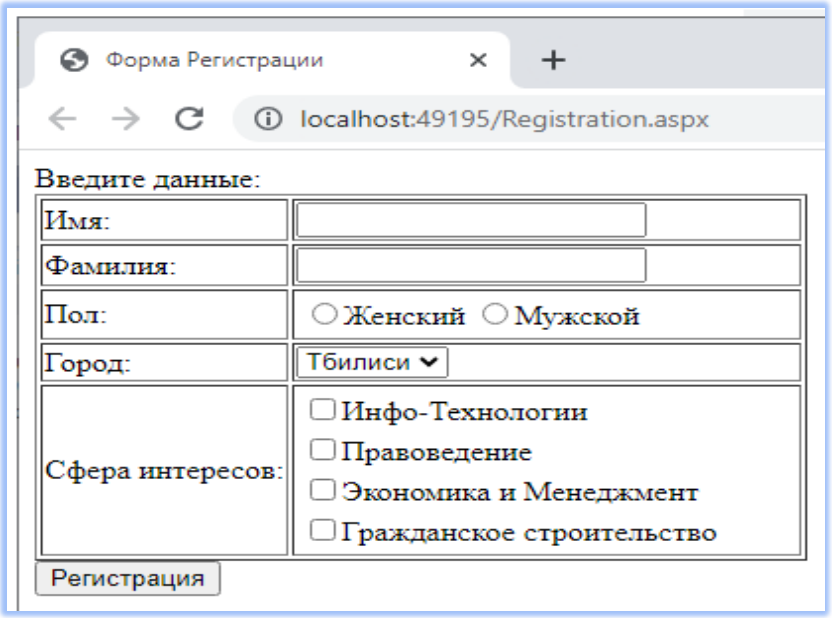


Рис. 5.4. Исходная форма в браузере

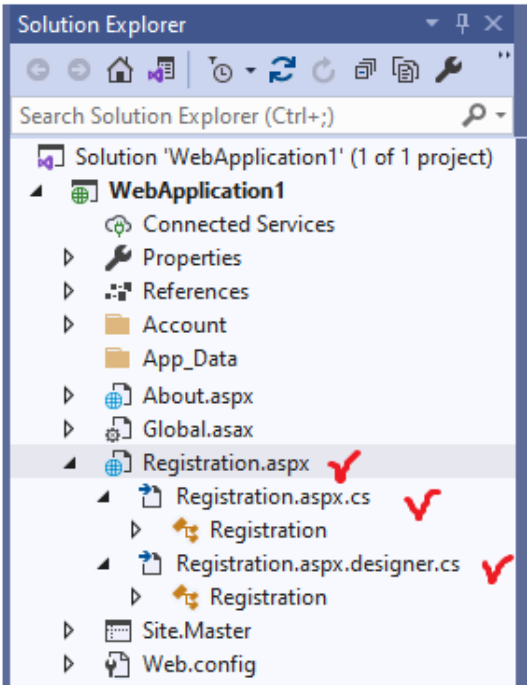


Рис. 5.5. Solution Explorer для WebApp Registration

На форме расположены элементы серверного управления form, asp:TextBox, asp:DropDownList, asp:CheckBoxList, asp:Button, asp:Label и т.д., которые отображены в листинге 5.2 файла Registration.aspx. Откроем Registration.aspx и внесем следующий код (или воспользуемся его прототипом, в книге):

```
<!----Листинг 5.2 - Registration.aspx ------------------------->
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="Registration.aspx.cs"
Inherits="WebApplication1.Registration" %>

<html >
```

```
<head>
  <title>Форма Регистрации</title>
    <style type="text/css">
        .auto-style1 {
            width: 253px;
        }
    </style>
</head>
 <body>
  <form method="post" runat="server" id="registration">
   Введите данные:
   <table border="1">
    <tr>
     <td>Имя:</td>
     <td class="auto-style1">
      <asp:TextBox id="FirstName"
            runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>Фамилия:</td>
     <td class="auto-style1">
      <asp:TextBox id="LastName"
            runat="server"></asp:TextBox></td>
    </tr>
    <tr>
     <td>Пол:</td>
     <td class="auto-style1"><asp:RadioButtonList id="Sex"
            runat="server" RepeatDirection="Horizontal">
       <asp:ListItem Value="Женский"></asp:ListItem>
       <asp:ListItem Value="Мужской"></asp:ListItem>
      </asp:RadioButtonList></td>
    </tr>
    <tr>
     <td>Город:</td>
     <td class="auto-style1"><asp:DropDownList id="City"
           runat="server">
       <asp:ListItem Value="Тбилиси"></asp:ListItem>
       <asp:ListItem Value="Кутаиси"></asp:ListItem>
       <asp:ListItem Value="Рустави"></asp:ListItem>
       <asp:ListItem Value="Батуми"></asp:ListItem>
```

```
        <asp:ListItem Value="Сухуми"></asp:ListItem>
        <asp:ListItem Value="Гори"></asp:ListItem>
        <asp:ListItem Value="Телави"></asp:ListItem>
       </asp:DropDownList></td>
     </tr>
     <tr>
           <td>Сфера интересов:</td>
      <td class="auto-style1">
       <asp:CheckBoxList id="Interests" runat="server">
        <asp:ListItem Value="Инфо-Технологии">
           </asp:ListItem>
        <asp:ListItem Value="Правоведение"></asp:ListItem>
        <asp:ListItem Value="Экономика и Менеджмент">
           </asp:ListItem>
        <asp:ListItem Value="Гражданское строительство">
           </asp:ListItem>
       </asp:CheckBoxList></td>

     </tr>
    </table>
    <asp:Button id="Register" runat="server"
Text="Регистрация" OnClick="Register_Click"></asp:Button>
    <br />
    <asp:Label id="Message" runat="server"></asp:Label>
   </form>
  </body>
</html>
```

// - - - Листинг 5.3 C#-кода -- - Registration.aspx.cs ----------

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication1
{
  public partial class Registration : System.Web.UI.Page
    {
```

```
    protected void Page_Load(object sender, EventArgs e)
       {

       }
 protected void Register_Click(object sender, EventArgs e)
    {
      System.Text.StringBuilder sb = new
                System.Text.StringBuilder();
      sb.Append("Ваши переданные данные:<br>");
      sb.AppendFormat("Имя: {0}<br>", FirstName.Text);
      sb.AppendFormat("Фамилия: {0}<br>", LastName.Text);
      sb.AppendFormat("Пол: {0}<br>", Sex.SelectedValue);
      sb.AppendFormat("Город: {0}<br>",
                              City.SelectedValue);
      sb.Append("Интересы: ");

      foreach (ListItem item in Interests.Items)
        {
          if (item.Selected)
               sb.AppendFormat("{0}, ", item.Value);
        }

      sb.Append("<br>Спасибо за регистрацию !");
      Message.Text = sb.ToString();
    }
  }
}
```

//-- Листинг 5.3 для C#-кода -- Registration.aspx.designer.cs ---

```
// <auto-generated>
//     This code was generated by a tool.
//
//     Changes to this file may cause incorrect behavior
// and will be lost if the code is regenerated.
// </auto-generated>
//------------------------------------------------------

namespace WebApplication1
{
```

```
    public partial class Registration
    {

        /// <summary>
        /// registration control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected
global::System.Web.UI.HtmlControls.HtmlForm registration;

        /// <summary>
        /// FirstName control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected
global::System.Web.UI.WebControls.TextBox FirstName;

        /// <summary>
        /// LastName control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected
global::System.Web.UI.WebControls.TextBox LastName;

        /// <summary>
        /// Sex control.
        /// </summary>
        /// <remarks>
```

```
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected
global::System.Web.UI.WebControls.RadioButtonList Sex;

        /// <summary>
        /// City control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected
global::System.Web.UI.WebControls.DropDownList City;

        /// <summary>
        /// Interests control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected
global::System.Web.UI.WebControls.CheckBoxList Interests;

        /// <summary>
        /// Register control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Button
Register;
```

```
        /// <summary>
        /// Message control.
        /// </summary>
        /// <remarks>
        /// Auto-generated field.
        /// To modify move field declaration from designer
file to code-behind file.
        /// </remarks>
        protected global::System.Web.UI.WebControls.Label
Message;
    }
}
```

Запустим приложение. В браузере появится форма Регистрации в виде сервиса. Вводим данные:

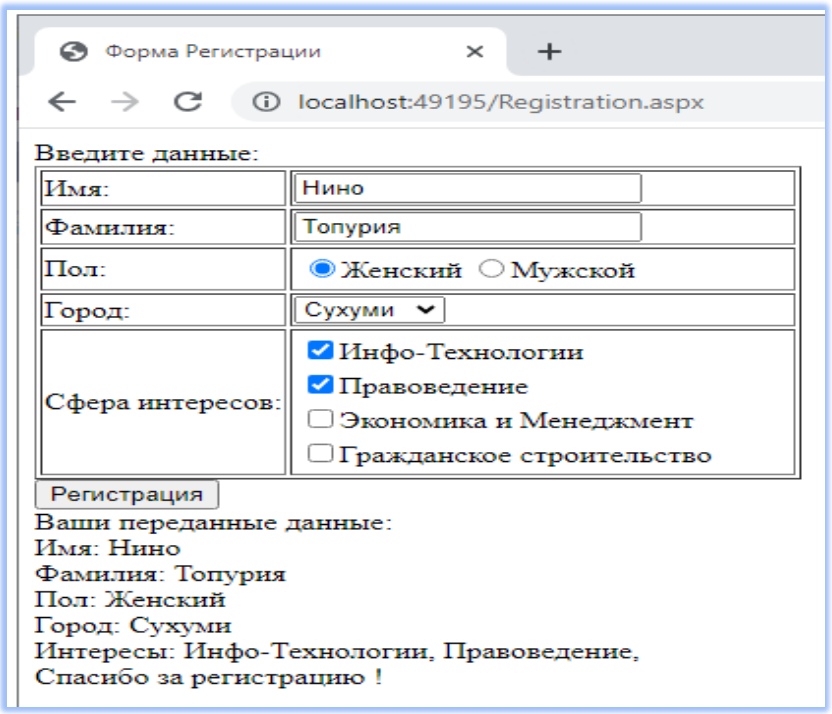


Рис. 5.6. Пример результата-1

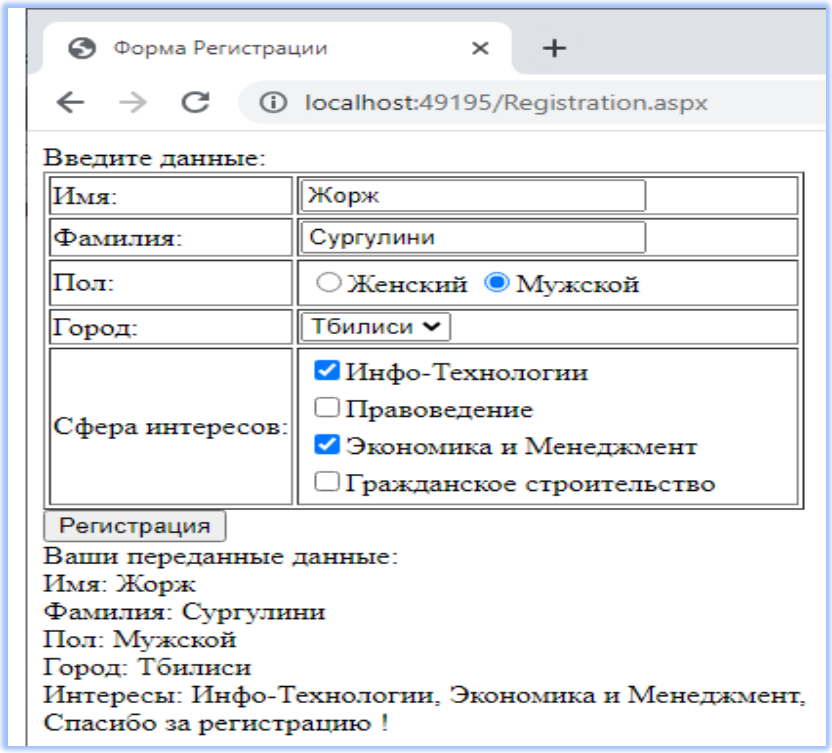


Рис. 5.7. Пример результата-2

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Сургуладзе Г., Нарешелашвили Г. Основы программной инженерии. (Уч.пособые - Лаб Практикум на visual C#). ГТУ. „IT-консальтинг центр". Тбилиси, 2022, -185 ст.

2. Долженко А.И., Глушенко С.А.. Программная инженерия. Учебное пособие. Ростов-на-Дону. 2017. -128 с.

3. Sommerville I. Softweare engineering, 11th ed., ISBN 978-0137035151. Addison-Wesley. 2016

4. Сургуладзе Г., Туркия Э. Основы менеджмента программных систем. ГТУ, "Технический ун-т". Тбилиси, 2016. -350 с.

5. Booch G., Jacobson I., rambaugh J. (1996). Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara,

6. Бек К. Шаблоны реализации корпоративных приложений. Экстремальное программирование: Пер. с англ. М.: Вильямс. 2008

7. Кознов Д.В. Введение в программную инженерию. 2009. http://www. intuit.ru /department/se/inprogeng.

8. Scrum.org. https://www.scrum.org/Resources

9. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих: Пер. с англ. - СПб.: БХВ-Петербург. 2013

10. Мартнн Р. Чистая архитектура. Искусство разработки программного обеспечения. СПб.: Питер, 2018. -352 с.: ил. - (Серия «Библиотека программиста»)

11. Сургуладзе Г. Методы и методологии компьютерного программирования (SP, OOP, VP, Agile, UML). ISBN 978-9941-1900-1. ГТУ, "ИТ-консалтинговый центр". Тб., 2019. -200 с.

12. DevOps. https://webmobtech.com/blog/what-is-devops-toolchain

13. Source Code Control System. Internet resource: https://de.wikipedia.org/wiki/Source_Code_Control_System

14. Fowler M. Continuous Integration. Internet resource: 2006. https://martinfowler.com/articles/continuousIntegration.html

15. The Relationship between Risk and Continuous Testing. 2014. https://www.stickyminds.com/interview/relationship-between-risk-and-continuous-testing-interview-wayne-ariola

16. Rahm E. Data Warehouses. Einführung. S.2, 2015. Vorlesungsskript, Universität Leipzig, Germany

17. Application Release Automation. Internet resource: https://en.wikipedia.org/wiki/Application-release_automation

18. Infrastructure as code. Internet resource: https://en.wikipedia.org/wiki/Infrastructure_as_code

19. Амблер С. (2005). Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста. Спб.: Питер.

20.Rumpe B. (2012). Agile Modellierung mit UML. Berlin, „Springer“. 2-te Auflage.

21.Закис А. Структурное руководство проектом. Серебряная пуля? http://citforum.ru/SE/project /silver_bullet/.

22. Fowler M. Continuous Integration. Internet resource: 2006. https://martinfowler.com/articles/continuousIntegration.html

23.Bobrow D.G. If Prolog is the answer, what is the question. 5-th Generation of Computer Systems, pp. 138-145, Tokyo, Japan, November ‘84. Inst. for New Generation Computer Technology (ICOT). North-Holland. 1984

24. Shriver B.D. Software paradigms. IEEE Software, 3(1):2, January '86. 1986

25. Wegner P. Concepts and paradigms of object-oriented programming. {OOPS messenger}, 1(1): 7-87, August '90. 1990

26. Budd T.A. Multy-Paradigm Programming in LEDA. Addison-Wesley, Reading, Massachusets. 1995

27. Dijkstra E. GOTO Statement Considered Harmful. Communications of the ACM, Vol.11, No.3, March 1968, pp.147-148. Assoc.for Computing Machinery, https://files.ifi.uzh.ch/rerg/arvo /courses/kvse/ueb ungen/Dijkstra_Goto.pdf

28. What Is the Project Management Triangle? Written by Coursera. 2023. https://www.coursera.org/articles/project-management-triangle

29. Half R. 6 Basic SDLC Methodologies: Which One Is Best? 2023. Internet resource: https://www.roberthalf.com/blog/salaries-and-skills/6-basic-sdlc-methodologies-which-one-is-best

Приложение-1

Грузинский Технический Университет

Факультет Информатики и систем управления

Департамент Программной инженерии

Курсовой проект

по предмету: Основы программной инженерии

Тема: АСУ „Библиотека”

Студенты:
Всеволод Пантелеев
Владимир Васин
Группа: 108159

Преподаватели:
Проф. Сургуладзе Г.
Проф. Нарешелашвили Г.

Тбилиси 2023

Оглавление

Над проектом работали:

Пантелеев Всеволод – разработчик программного обеспечения, создатель базы данных, конспектирование Word файла.

Васин Владимир – консультант по функционалу ПО, создатель презентации Powerpoint по данному проекту.

Введение

Проект был создан в рамках обучающего курса„Основы Программной инженерии“. Задачей проекта было использование навыков, наработанных в ходе курса. Проект создавался в VisualStudio.NETFramework. К файлу C# была подключена база данныхSQLServer. Програмный код был написан на основе запросов, классов,методов, объектов, делегатов, переменных, методов и свойств.

1. Основные понятия и термины ОО-программирования

Класс — шаблон для создания объекта.

Метод — это именованный блок кода, содержащий набор инструкций.

Переменная — это именнованную область памяти, в которой хранится значение определенного типа.

Делегаты — это объекты, которые хранят ссылку на методы.

Специальные поля, которые используют функции get и set, называющиеся свойства.

Также статические поля, методы, свойства, которые относятся ко всему классу и для обращения к подобным членам класса не нужно создавать экземпляр класса.

Полиморфизм — принцип ООП, который подразумевает наследование и переопределение классов.

2. Постановка задачи построения компьютерной системы контроля библиотеки

Наша программа позволяет хранить информацию о книгах и таких, как: дата, цена, название, адресс, а также подразумевает возможность фильтрации и сортировки.

Программа предназначается для пользователей любого уровня и может быть использована компаниями для ведения статистики.

Программа имеет следующие удобства и возможности:

- Добавление и редактирование записей.
- Сортировка по разным критериям.
- Поиск по интересующим данным.

2.1. Тип объекта и его основные фунции

Приложение для добавления, сортировки, поиска и удаления информации в базе данных.

2.2. Сотрудики объекта и их фунции

Данное приложение представляет собой связующее звено между обычным пользователем и базой данных.

Пользователь без специальной подготовки сможет легко задействовать это приложение для введения обширной БД для своего бизнеса.

2.3. Модель „Роли и функции“ (UseCase диаграмма)

На рисунке 1 представлена диаграмма прецедентов. Она показывает участвующие роли и их функции в в рамках данной задачи.

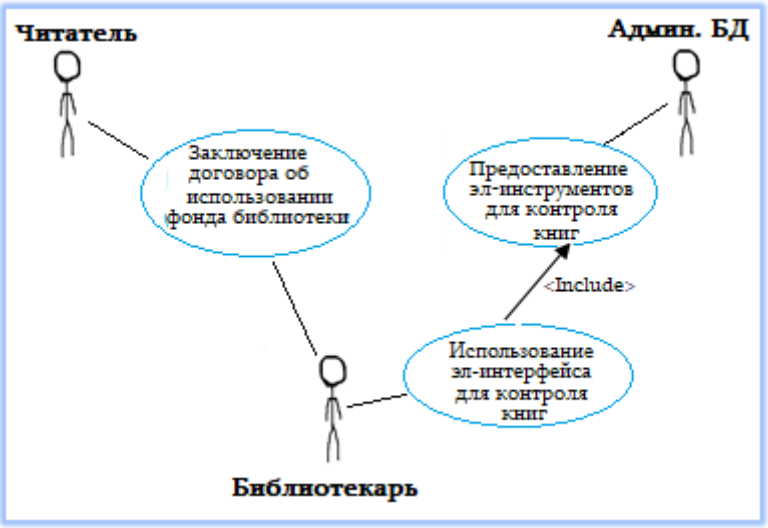


Рис.1. UseCase Diagramm (Роли и Функции)

<Include> – Связь между функциями.

Процесс „Предоставление..." предшествует процессу „Использование…" (т.е. без предоставления эл-инструментов администратором БД, библиотекарь не сможет использовать эл-интерфейс с функциями контроля БД книг).

3. Создание Базы Данных в DBeaver

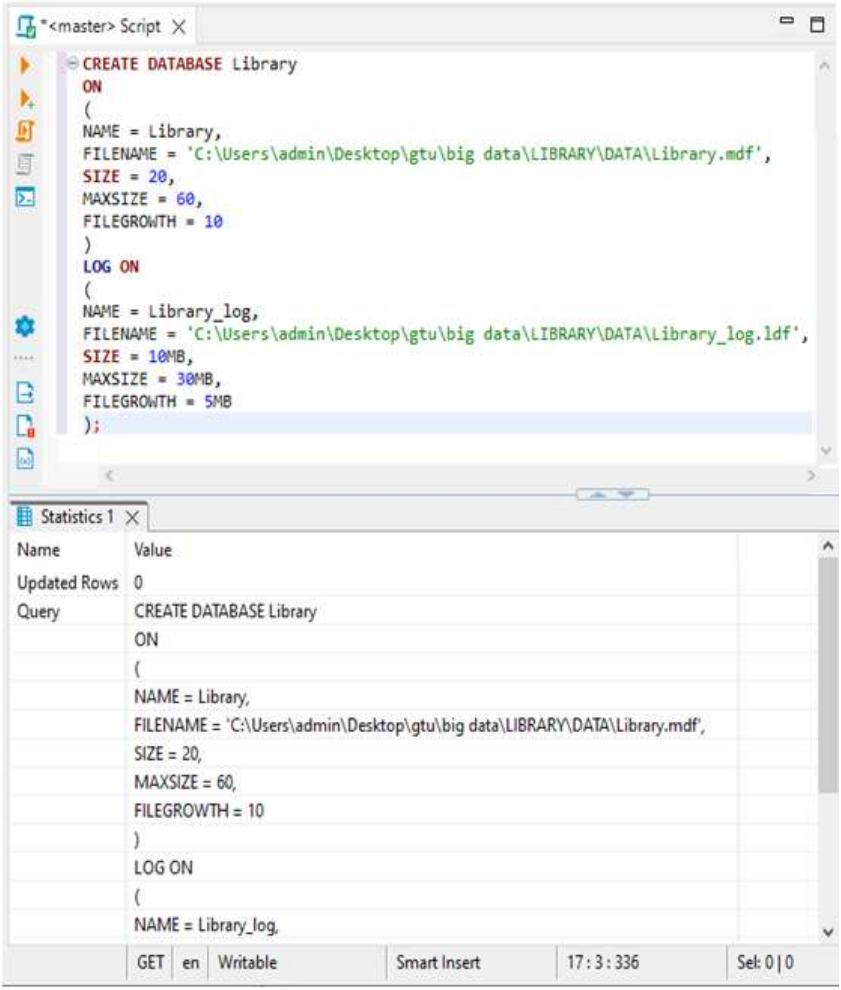


Рис. 2. Создание Базы Данных

3.1. Проектирование таблиц

1) Создание таблицы Books (Книги)

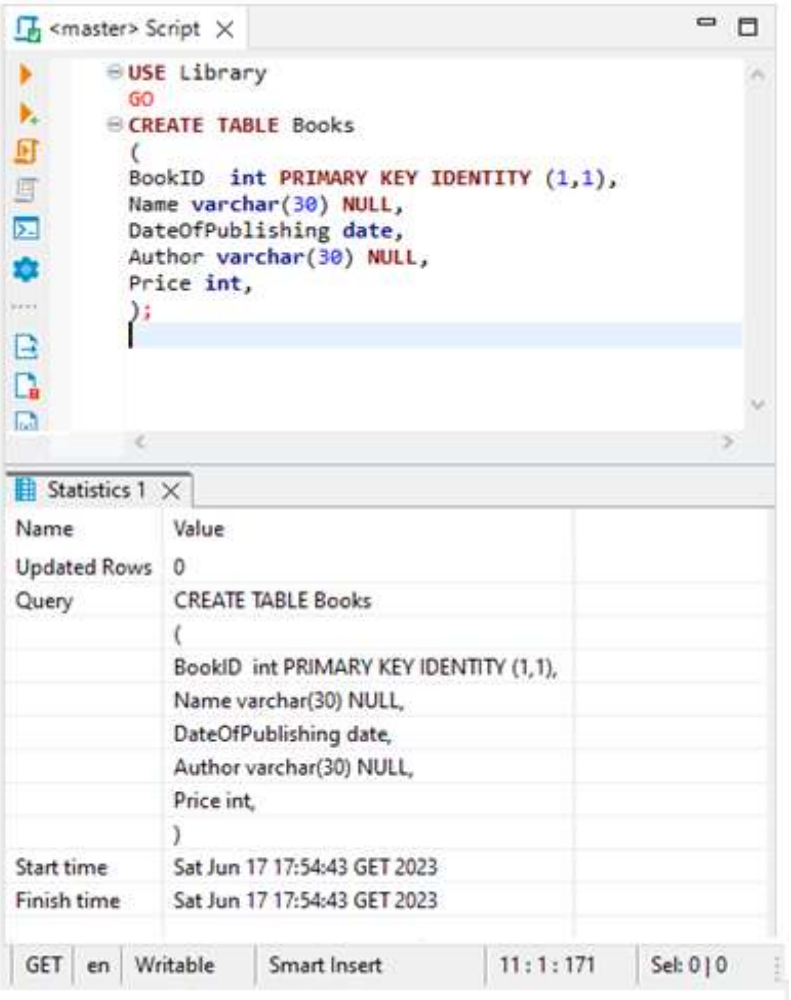


Рис.3.Таблица Books

2) Созданиетаблицы Readers (Читатели)

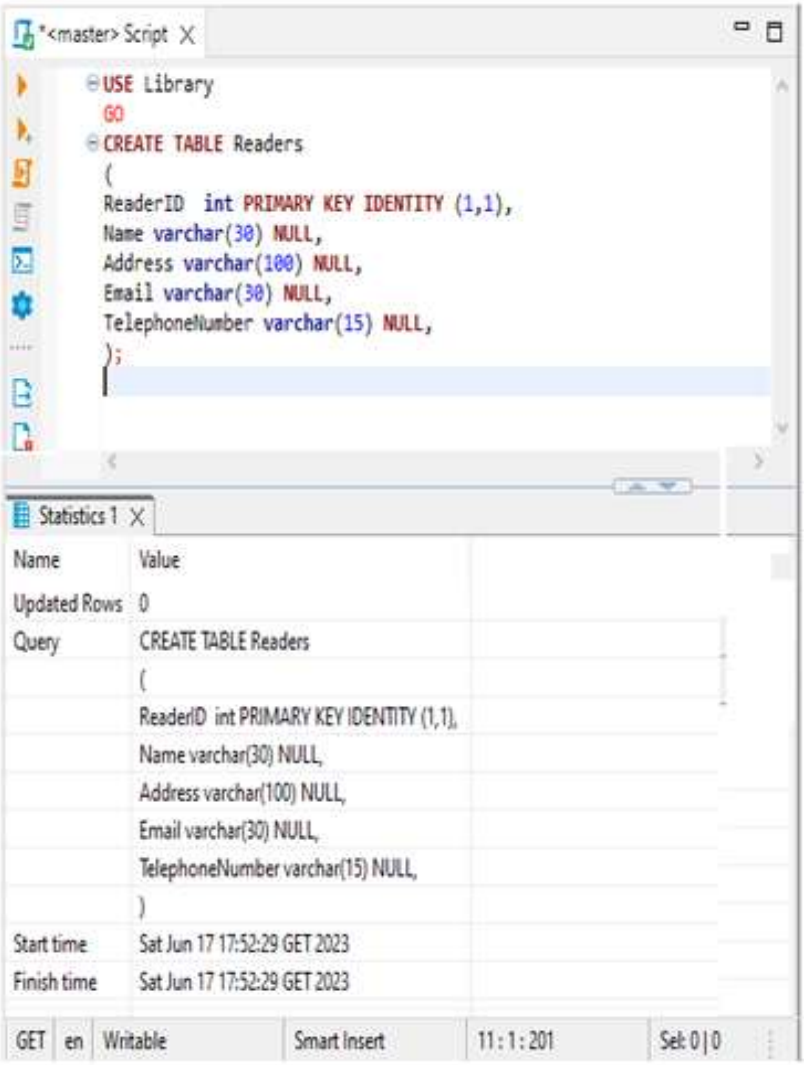


Рис. 4. Таблица Readers

3) Создание таблицы Control (Управление)

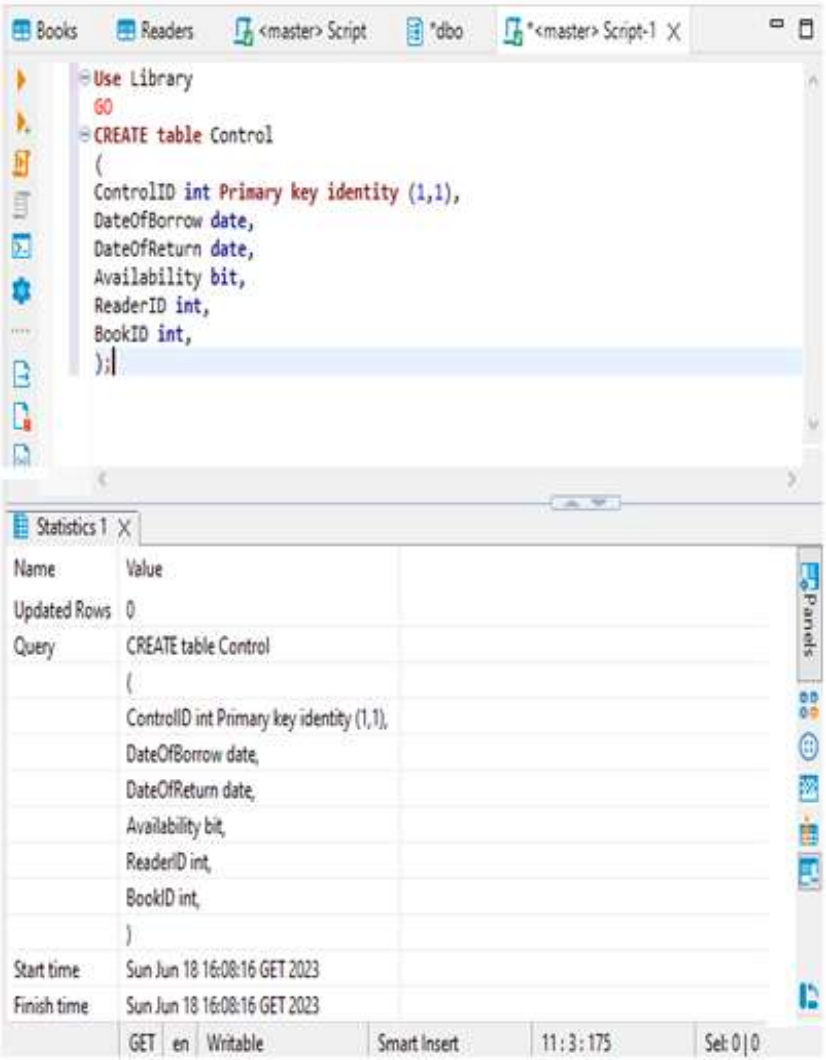


Рис. 5. Таблица Control

3.2. Создание концептуальной модели БД (DatabaseDiagram)

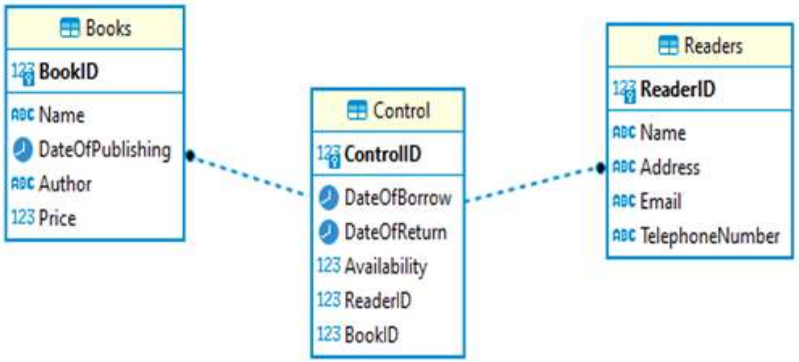


Рис. 6. DatabaseDiagram

4. Создание WinFormsApp проекта Library на платформе MsVisualStudio.NET Framework

4.1. Подключение SQL ДБ к проекту C#

```
public static SqlConnection GetConnection()
{  string sql = "Data Source=DESKTOP-JSDAFNJL\\SQLEXPRESS;
          Integrated Security=True;TrustServerCertificate=True";
    SqlConnection connection = new SqlConnection(sql);
    try
    {  connection.Open(); }
    catch (Exception ex)
    {
      MessageBox.Show("SQL Connection!\n" + ex.Message, "Error",
      MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    return connection;
}
```

Рис. 7. Листинг фрагмента программы подключения

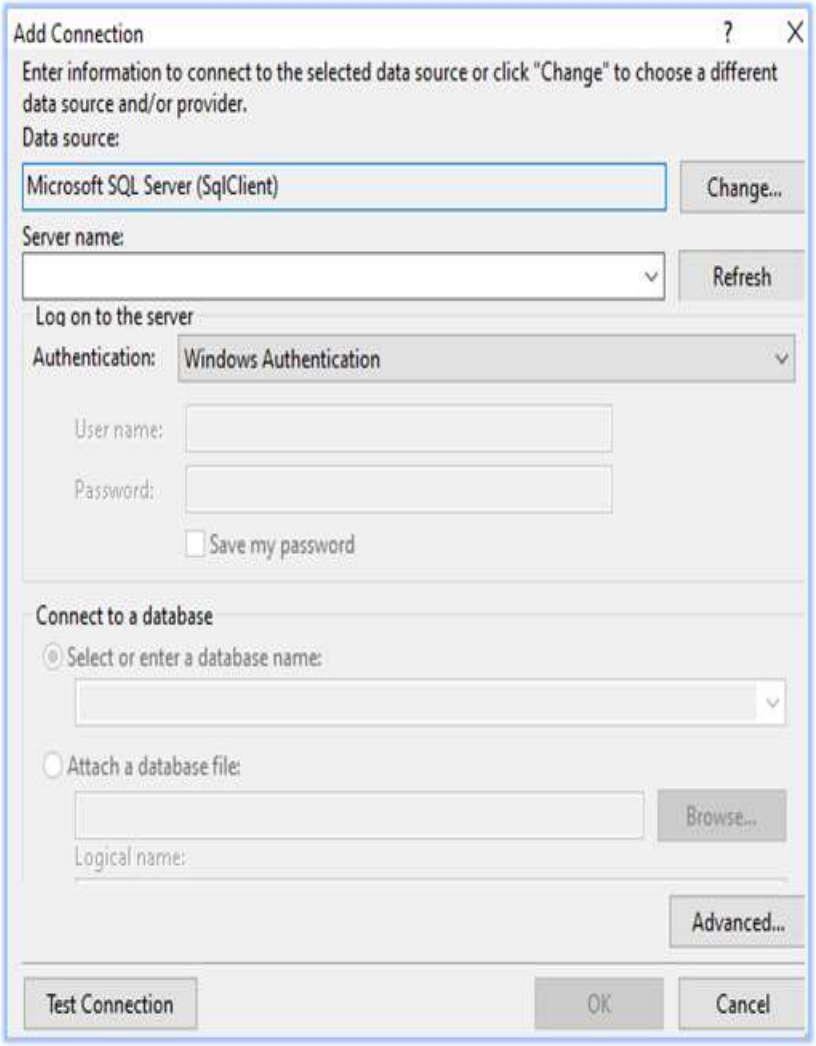


Рис.8. Подключение к базе данных в среде MicrosoftVisualStudio

4.2. Проектирование главной формы программы

Добавление таких элементов, как: Button, Label, DataGridView, TextBox, PictureBox, Panel

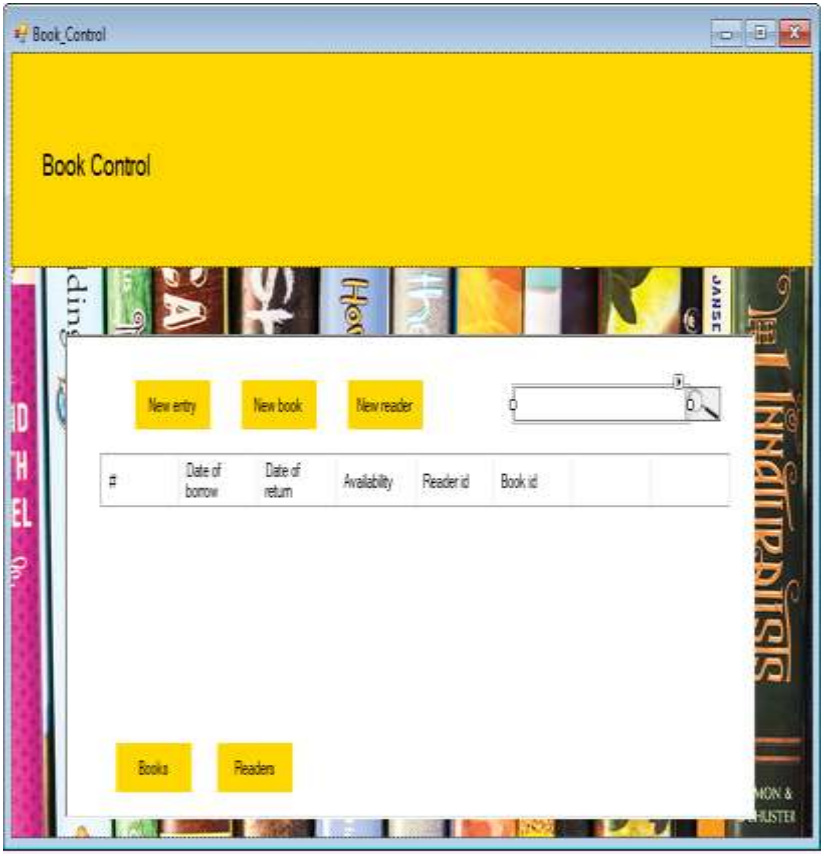


Рис. 9. Главная форма

5. Программирование CRUD операций

5.1. Программный код для кнопки CREATE

```
public static void AddEntry(ClassNewEntry std)
{
   string sql = "INSERT INTO Library.dbo.Control
      (DateOfBorrow, DateOfReturn, Availability, ReaderID, BookID)
      VALUES(@DateOfBorrow, @DateOfReturn, @Availability, @ReaderID, @BookID)";

   SqlConnection connection = GetConnection();
   SqlCommand cmd = new SqlCommand(sql, connection);
   cmd.CommandType = CommandType.Text;
   cmd.Parameters.Add("@DateOfBorrow", SqlDbType.VarChar).Value = std.DateOfBorrow;
   cmd.Parameters.Add("@DateOfReturn", SqlDbType.VarChar).Value = std.DateOfReturn;
   cmd.Parameters.Add("@Availability", SqlDbType.VarChar).Value = std.Availability;
   cmd.Parameters.Add("@ReaderID", SqlDbType.VarChar).Value = std.ReaderID;
   cmd.Parameters.Add("@BookID", SqlDbType.VarChar).Value = std.BookID;

   try
   { cmd.ExecuteNonQuery();
     MessageBox.Show("Added succesfully", "Information", MessageBoxButtons.OK,
                                          MessageBoxIcon.Information); }
   catch (SqlException ex)
   { MessageBox.Show("Not added\n" + ex.Message, "Information",
                     MessageBoxButtons.OK, MessageBoxIcon.Information);
   }
   connection.Close();
}
```

Рис. 10. Программный код для кнопки CREATE

5.2. Программный код для кнопки READ

```
public static void DisplayAndSearch(string query,
                                     DataGridView dgv)
{
    string sql = query;
    SqlConnection connection = GetConnection();
    SqlCommand cmd = new SqlCommand(sql, connection);
    SqlDataAdapter adp = new SqlDataAdapter(cmd);
    DataTable tbl = new DataTable();
    adp.Fill(tbl);
    dgv.DataSource = tbl;
    connection.Close();
}
```

Рис. 11. Программный код для кнопки READ

5.3. Программный код для кнопки UPDATE

```
public static void UpdateEntry(ClassNewEntry std, string id)
{
    string sql = "UPDATE Library.dbo.Control SET DateOfBorrow = @DateOfBorrow, DateOfReturn = @DateOfReturn,
                Availability = @Availability, ReaderID = @ReaderID, BookID = @BookID WHERE ControlID = @ControlID";

    SqlConnection connection = GetConnection();
    SqlCommand cmd = new SqlCommand(sql, connection);
    cmd.CommandType = CommandType.Text;
    cmd.Parameters.Add("@ControlID", SqlDbType.VarChar).Value = id;
    cmd.Parameters.Add("@DateOfBorrow", SqlDbType.VarChar).Value = std.DateOfBorrow;
    cmd.Parameters.Add("@DateOfReturn", SqlDbType.VarChar).Value = std.DateOfReturn;
    cmd.Parameters.Add("@Availability", SqlDbType.VarChar).Value = std.Availability;
    cmd.Parameters.Add("@ReaderID", SqlDbType.VarChar).Value = std.ReaderID;
    cmd.Parameters.Add("@BookID", SqlDbType.VarChar).Value = std.BookID;

    try
    {
        cmd.ExecuteNonQuery();
        MessageBox.Show("Updated succesfully", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (SqlException ex)
    {
        MessageBox.Show("Not Updated\n" + ex.Message, "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    connection.Close();
}
```

Рис. 12. Программный код для кнопки UPDATE

5.4. Программный код для кнопки DELETE

```
public static void DeleteEntry(string id)
{
    string sql = "DELETE FROM Library.dbo.Control
                        WHERE ControlID = @ControlID";

    SqlConnection connection = GetConnection();
    SqlCommand cmd = new SqlCommand(sql, connection);
    cmd.CommandType = CommandType.Text;
    cmd.Parameters.Add("@ControlID", SqlDbType.VarChar).Value = id;

    try
    {
        cmd.ExecuteNonQuery();
        MessageBox.Show("Deleted succesfully", "Information",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (SqlException ex)
    {
        MessageBox.Show("Not Deleted\n" + ex.Message, "Information",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    connection.Close();
}
```

Рис. 13. Программный код для кнопки DELETE

6. Финальное окно программы и результаты работы

6.1. Запуск exe – программы

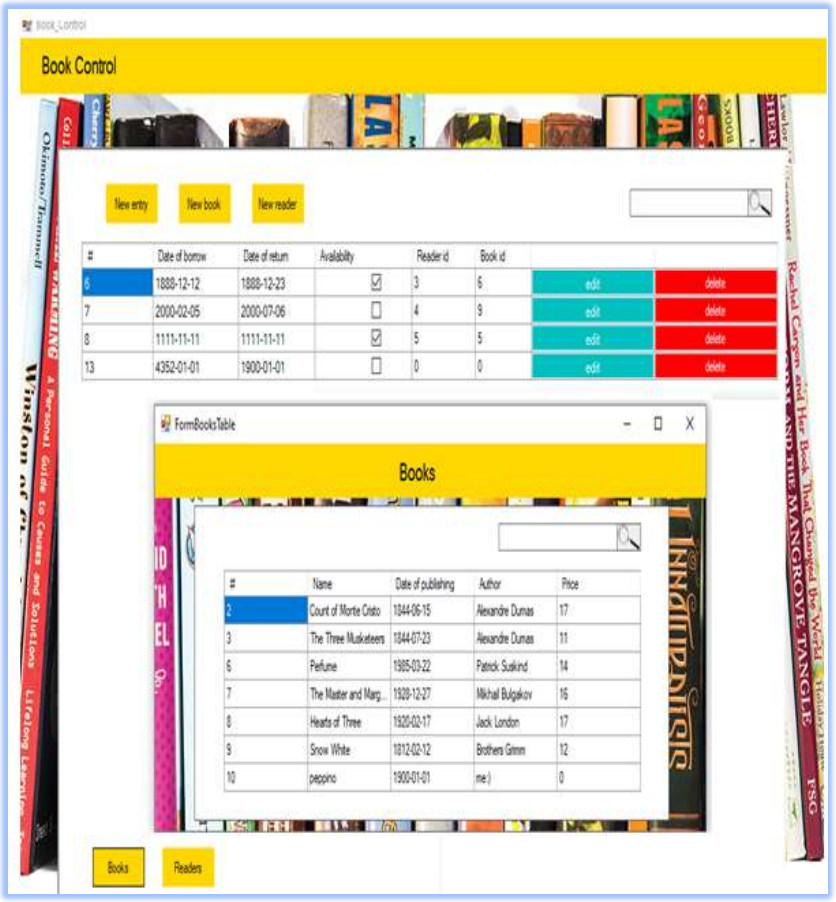


Рис. 14. Запуск программы

В результате проектирования программы получается список книг и читателей. У нас есть возможность добавления, редактирования, удаления и сортировки данных.

7. Руководство пользователя

В главной Форме у нас есть несколько кнопок:

Newentry- Добавить новую запись

Edit–Изменить запись

Delete–Удалить запись

Newbook-Добавить новую книгу

Newreader- Сохранить Изменения

Books– Открыть таблицу Books

Readers- Открыть таблицу Readers

Здесь пользователь имеет возможность редактировать список и открывать другие таблицы.

8. Заключение

Программа была успешно создана с использованием приобретенных знаний в процессе изучения учебного курса ООП, а также БД SQL Server и языка C#. Были использованы методы, классы, запросы, формы и т.д.

При ее создании был поставлен ряд проблем, которые были решены и зафиксированы в результате работы программы. Благодаря этому программа готова к использованию для работы в сфере книгообразующих предприятий.

Библиография (приложения-1):

1. Clean architecture by Robin Martin.[Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2018. –352 с.: –(Серия «Библиотека программиста»). Пер. с англ., ISBN 978-5-4461-0772-8

2. Microsoft Object-Oriented programming (C#): https://www.microsoft.com/en-us/d/visual-studio-test-professional-subscription/dg7gmgf0dst6?activetab=pivot%3aoverviewtab

3. METANIT Объектно-ориентированное программирование. Практика.Создание проекта библиотеки классов: https://metanit.com/sharp/tutorial/3.29.php

4. Professor Web. Основы объектно-ориентированного программирования. https://professorweb.ru/my/csharp/charp_theory/level3/3_1.php

5. itProger. Основы ООП: Уроки C#: Создание классов и объектов. https://itproger.com/course/csharp/14

6. Сургуладзе Г., Нарешелашвили Г.Основы программной инженерии (Лаб. Практикум на visualC#). ГТУ, „IT-консальтинг центр“. Тбилиси, -185 ст.

Приложение-2

ГРУЗИНСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информатики и систем управления

Департамент Программной инженерии

КУРСОВОЙ ПРОЕКТ

по учебному курсу„Основы программной инженерии“

Тема: Автоматизированная система управления Кино-сессиями

Студент: Ибадов Илькин

Группа: 108159

Преподаватели: Проф. Сургуладзе Г.
Проф. Нарешелашвили Г.

Тбилиси 2023

Оглавление

Введение

Кино сессионная программа – **это** представляющая кино сессионный бренд в одном или нескольких кинотеатрах и географических точках, имеющий обученный персонал, финансовые ресурсы, оборудованне в соответствии с требованиями кинотеатров.

Целью работы является разработка проекта компьютерной системы на языке программирования C# для клиентов (зрителей) и персонала кинотеатров, что значительно экономит время клиентов и повышает эффективность работы персонала.

Основные задачи проекта:

- Упрощение работы кинотеатров;
- Возможность онлайн заказа билетов;
- Определение условий скидок на стоимость билетов;
- Отмена заказа.

Для выполнения этого были использованы:

- Формирование классов и объектов. Полиморфизм. Наследование. Статические компоненты;
- Исключительные ситуации. Ввод-вывод информации. Делегаты. События;
- Работа с базой данных. Разработка программной аппликации для пользовательыкого интерфейса.

Глава 1

Анализ потребностей объекта автоматизации

1.1. Теоретические сведения

Первый кинотеатр, известный как "кинематограф", был создан братьями Люмьерами. Они открыли его в Париже, Франция, в 1895 году. Кинотеатр назывался "Синематограф Люмьер", и зрители могли посмотреть короткие фильмы братьев Люмьер на большом экране. Этот кинотеатр стал отправной точкой для развития кинематографа и кинопоказов по всему миру.

1.1.1. Описание деятельности

Управление кино-сессиями включает в себя организацию и координацию показа фильмов в кинотеатре. Это включает в себя следующие задачи: создание расписания сеансов, подбор фильмов для показа, определение времени начала и окончания сеансов, продажа билетов, контроль за доступностью мест в зале, обеспечение комфортного просмотра фильмов для зрителей, обработка платежей и учет посещаемости. Организаторы кино-сессий также могут работать с дистрибьюторами фильмов для получения необходимых лицензий на их показ и промоушн фильмов для привлечения зрителей. Кроме того, важным аспектом управления кино-сессиями является поддержание оборудования и технической инфраструктуры, такой как кинопроекторы, звуковая система и экраны, чтобы обеспечить высокое качество показа фильмов.

1.1.2. Основные функции кино-сессий

- Расписание киносеансов: Управление киносеансами требует разработки расписания, определяющего время начала и окончания каждого сеанса в течение дня. Расписание может быть составлено с учетом популярности фильмов, прогнозируемого спроса на определенное время и длительности фильмов.
- Выбор фильмов: Кинотеатры выбирают фильмы, которые они собираются показывать в течение определенного периода времени. Это может включать фильмы разных жанров и категорий, чтобы удовлетворить различные предпочтения зрителей.
- Билеты и продажи: Управление киносеансами включает работу с системами билетной продажи, чтобы обеспечить доступность билетов для зрителей. Это может включать продажу билетов через кассу в кинотеатре, онлайн-бронирование и покупку билетов через интернет или мобильные приложения.
- Поддержка технического оборудования: Управление киносеансами также включает обслуживание и поддержку технического оборудования в кинозале, такого как проекторы, звуковые системы и экраны. Регулярное обслуживание и проверка оборудования необходимы для обеспечения высокого качества показа фильмов.
- Контроль посещаемости и аналитика: Важным аспектом управления киносеансами является контроль посещаемости и

сбор данных о предпочтениях зрителей. Это позволяет анализировать данные, определить популярность фильмов и принимать решения по поводу продолжительности показа фильмов и их замены.

Глава 2

Проектирование информационой системы

2.1. Разработка инфраструктуры ПО

Разработка проекта была осуществлена с помощью платформы MsVisual Studio.NET. Была использована интегрированная среда построения приложения Windows Forms Application(.Net Framework) для пользовательского интерфейса на основе языка программирования C#, базы данных MsAccess и драйвера ADO.NET.

Windows Forms — это технология пользовательского интерфейса для .NET, набор управляемых библиотек, которые упрощают общие задачи приложений, такие как чтение и запись в файловую систему. При использовании среды разработки Visual Studio.NET возможно создавать интеллектуальные клиентские приложения Windows Forms, которые отображают информацию, запрашивают ввод от пользователей и взаимодействуют с удаленными компьютерами по сети.

В Windows Forms *форма* - это визуальная поверхность, на которой отображается информация для пользователя. Обычно, создается приложение Windows Forms, добавляя элементы управления из панели инструментов (ToolBox) на форме и

разрабатывая ответы (события - events) на действия пользователя, такие как щелчки мыши или нажатия клавиш.

Элемент управления представляет собой дискретный элемент пользовательского интерфейса, который отображает данные или принимает входные данные. Когда пользователь что-то делает с формой или одним из ее элементов управления, действие генерирует событие. Приложение реагирует на эти события с помощью кода (метода) и обрабатывает события, когда они происходят.

Windows Forms содержит множество элементов управления, которые можно добавлять в формы: элементы управления, отображающие текстовые поля, кнопки, раскрывающиеся списки, переключатели и даже веб-страницы.

Windows Forms имеет богатые элементы управления пользовательского интерфейса, имитирующие функции высокопроизводительных приложений, таких как Microsoft Office. При использовании элементов управления ToolStrip и MenuStrip можно создавать панели инструментов и меню, содержащие текст и изображения, отображать подменю и размещать другие элементы управления, такие как текстовые поля и поля со списком. Дизайнер предоставляет такие инструменты, как линии сетки и линии привязки, чтобы упростить выравнивание элементов управления. Наконец, если вам необходимо создать свои собственные пользовательские элементы пользовательского интерфейса, пространство имен System.Drawing содержит большой выбор

классов для визуализации линий, кругов и других фигур непосредственно в форме.

2.2 База данных MsAccess

Для использования данных я подключил базу данных MsAccess к проекту C#. Для этого мне понадобилось создать таблицы в MsAccess и заполнить строки данных из программы C# пользовательского интерфейса.

Таблицы, которые были созданы в MsAccess: *Фильмы, Сеансы и Заказы*.В таблицу **“Фильмы”** было вписаны данные о коде, названии, продолжительности и также описании фильма (Рис.1).

Фильмы

Код_фильма	Название_фильма	Продолжительность	Описание	Click to Add
1	Creed III	2	Still dominating the boxing world, Adonis Creed is thrivin	
2	The Mother	2	While fleeing from dangerous assailants, an assassin con	
3	Air	2	Sonny Vaccaro and Nike pursue basketball rookie Micha	
4	Asterix & Obelix: The Middle Kingdom	2	The Chinese princess Fu Yi arrives at the Gallic village to	
5	Murder Mystery 2	2	Now private detectives launching their own agency, Nick	
6	Operation Fortune: Ruse de Guerre	2	Special agent Orson Fortune and his team of operatives	
7	Chang Can Dunk	2	Chang, a 16-year-old Asian American high school studen	
8	Ghosted	2	Cole falls head over heels for enigmatic Sadie, but then r	
9	Ride On	3	A washed-up stuntman and his stunt horse become an o	
10	Dungeons & Dragons: Honor Among Thi	3	A charming thief and a band of unlikely adventurers emb	
11	Extraction 2	3	Back from the brink of death, commando Tyler Rake emb	
	(New)			

Рис.1. Таблица Фильмы

В таблицу **“Сеансы”** были записаны данные о коде сеанса, коде фильма, виде (утренний, дневной или вечерний), дате и начальной стоимости каждого вида сеанса.(Рис.2).

Сеансы

Код_сеанса	Код_фильма	Вид	Дата	Начальная_стоимость	Click to Add
1	1	Утренний	03.03.2023 15		
2	1	Дневной	03.03.2023 20		
3	1	Вечерний	03.03.2023 25		
4	2	Утренний	15.05.2023 10		
5	2	Дневной	15.05.2023 15		
6	2	Вечерний	15.05.2023 25		
7	3	Утренний	24.04.2023 9		
8	3	Дневной	24.04.2023 18		
9	3	Вечерний	24.04.2023 24		
10	4	Утренний	07.04.2023 13		
11	4	Дневной	07.04.2023 17		
12	4	Вечерний	07.04.2023 20		
13	5	Утренний	21.02.2023 10		
14	5	Дневной	21.02.2023 19		
15	5	Вечерний	21.02.2023 24		
16	6	Утренний	01.03.2023 11		
17	6	Дневной	01.03.2023 16		
18	6	Вечерний	01.03.2023 20		
19	7	Утренний	10.05.2023 15		
20	7	Дневной	10.05.2023 25		
21	7	Вечерний	10.05.2023 10		
22	8	Утренний	06.04.2023 15		
23	8	Дневной	06.04.2023 20		
24	8	Вечерний	06.04.2023 25		
25	9	Утренний	19.02.2023 12		
26	9	Дневной	19.02.2023 18		
27	9	Вечерний	19.02.2023 24		
28	10	Утренний	29.04.2023 17		
29	10	Дневной	29.04.2023 26		
30	10	Вечерний	29.04.2023 30		
31	11	Утренний	20.06.2023 20		
32	11	Дневной	20.06.2023 25		
33	11	Вечерний	20.06.2023 35		
(New)					

Рис.2. Таблица Сеансы

В таблицу **“Заказы”** было вписаны данные о коде заказа, коде сеанса, ряд, место и имя заказчика (т. е. клиента) (Рис.3).

Заказы

Код_заказа	Код_сеанса	Ряд	Место	Имя_заказчика	Click to Add
1	19		5	Viktor	
2	17		7	Maqamed	
3	18		6	Maqammed	
4	18		7	Ibragim	
5	18		8	Elmin	
6	18		9	Ali	
7	18		5	Anar	
8	19		6	David	
9	19		7	Zakhar	
10	113		7	Abdulla	
11	211		4	Amina	
12	211		5	Cavid	
13	211		6	Cavidan	
14	27		7	Narmin	
15	27		6	Narmina	
16	28		5	Huseyn	
17	28		6	Nadejda	
18	28		7	Sofia	
19	214		5	Zlata	
20	214		6	Sofya	
21	36		4	Taisiya	

Рис.3. Таблица Заказы

Все эти таблицы соединены между собой реляцией:

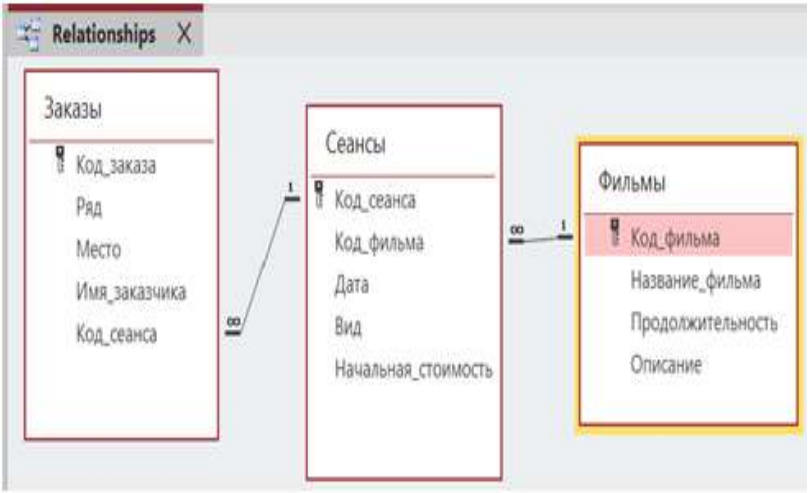


Рис.4. Реляция

2.3. Создание программного проекта

Для начала работы в Visual Studio .NET соответственно нужно запустить саму программу. Далее на открывающемся окне кликаем на “Create a new project” (Рис.5).

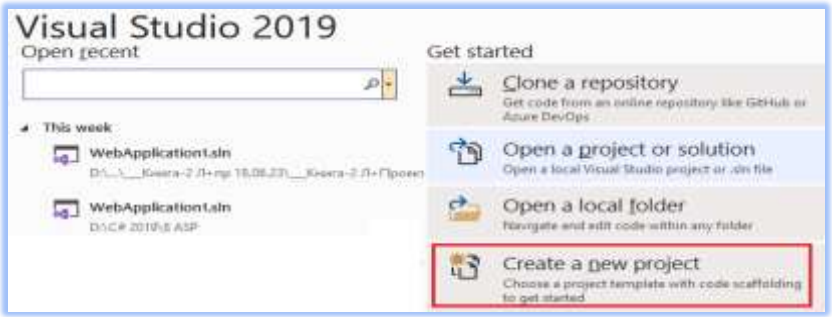


Рис.5. Экран запуска программы

На следующем окне в качестве типа проекта выберем приложение “Windows Forms App” (Рис. 6).

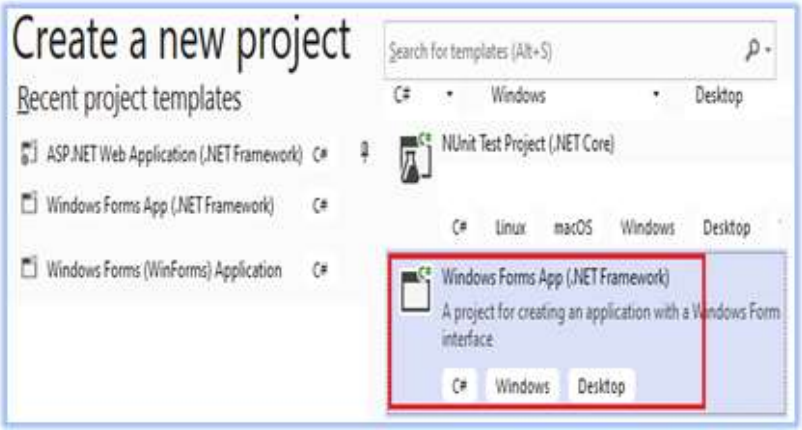


Рис. 6. ВыборплатформыПриложение Windows Forms (.Net Framework)

Далее в поле **Project Name** дадим проекту какое-либо название. В этом случае это *kinoshka*. А также укажем папку, где будет располагаться проект (Рис.7).

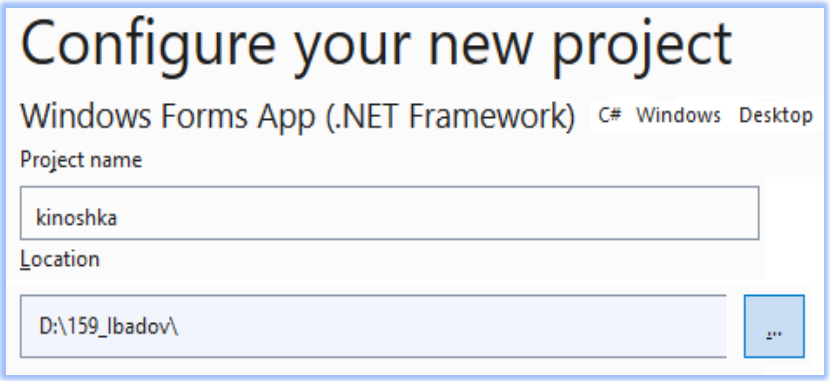


Рис. 7. Задается название проекту

2.3.1. Разработка формы AuthForm

Для разработки формы, был создан следующий код:

```
usingSystem;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace kinoshka
{
publicpartialclassAuthForm : Form
    {
publicAuthForm()
        {
            InitializeComponent();
        }

privatevoid button1_Click(object sender, EventArgs e)
      {
if (maskedTextBox1.Text == "admin")
          {
            AdminForm adminForm = new AdminForm(this);
                adminForm.Show();
this.Hide();
          }
      }

privatevoid AuthForm_Load(object sender, EventArgs e)
       {

       }
    }
}
```

В файле `AuthForm.cs` определен класс `AuthForm`, который является наследником класса `Form` из пространства имен `System.Windows.Forms`. Этот класс представляет форму аутентификации пользователей.

В конструкторе класса `AuthForm` вызывается метод `InitializeComponent()`, который инициализирует все элементы управления формы, такие как кнопки, текстовые поля и другие компоненты.

У класса `AuthForm` определены следующие методы:

1) `button1_Click`: Этот метод обрабатывает событие нажатия на кнопку с именем `button1`. Внутри метода проверяется, совпадает ли введенный текст в `maskedTextBox1` с "admin". Если условие выполняется, создается экземпляр класса `AdminForm` (предположительно окно для администратора) и отображается, а текущая форма скрывается методом `this.Hide()`.

2) `AuthForm_Load`: Этот метод обрабатывает событие загрузки формы. В данном случае, метод пустой и не содержит дополнительной функциональности.

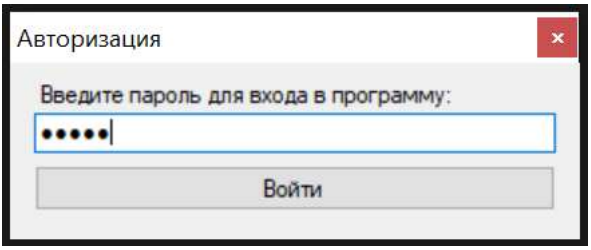


Рис.8. Авторизация в программу

2.3.2. Разработка формы AdminForm

Для разработки AdminFrom был написан код, листинг которого приведён в приложении-1.

В файле `AdminForm.cs` определен класс `AdminForm`, который является наследником класса `Form` из пространства имен `System.Windows.Forms`. Этот класс представляет форму для администратора.

У класса `AdminForm` определены следующие элементы:

1) `authForm`: Это поле класса типа `AuthForm`, которое хранит ссылку на экземпляр класса `AuthForm`, который представляет форму аутентификации. Это позволяет администратору закрыть форму аутентификации при закрытии формы администратора;

2) Конструктор `AdminForm`: Принимает экземпляр класса `AuthForm` в качестве параметра и сохраняет его в поле `authForm`. Он также вызывает метод `InitializeComponent()`, который инициализирует все элементы управления формы;

3) Метод `saveChanges()`: Сохраняет изменения, внесенные в данные таблицы заказов (`заказыTableAdapter`) и обновляет набор данных (`kinoshkadbDataSet`). Затем выполняется заполнение таблиц фильмов, сеансов и заказов с использованием соответствующих адаптеровADO.NET (`фильмыTableAdapter`, `сеансыTableAdapter`, `заказыTable-Adapter`);

4) Метод `AdminForm_Load`: Этот метод обрабатывает событие загрузки формы. В данном случае, метод пустой и не содержит дополнительной функциональности;

5) Метод `AdminForm_FormClosing`: Этот метод обрабатывает событие закрытия формы. Он закрывает форму аутентификации (`authForm`) при закрытии формы администратора;

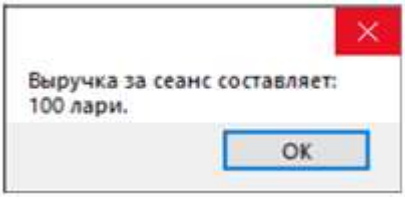
6) Методы обработчиков событий `управление Фильмами-ToolStripMenuItem_Click` и `управление Сеансами-ToolStripMenuItem_Click`: эти методы обрабатывают щелчки на соответствующих пунктах меню и создают и отображают соответствующие формы управления фильмами и сеансами;

7) Методы обработчиков событий `button3_Click`, `button4_Click`, `button1_Click` и `button2_Click`: Эти методы обрабатывают нажатия на соответствующие кнопки на форме и выполняют определенные действия, такие как сохранение изменений, добавление нового заказа, отображение информации от выручки и создание формы для рисования;

8) Метод `удалитьToolStripMenuItem_Click`: Этот метод обрабатывает событие щелчка на пункте меню "удалитьToolStripMenuItem". Он отображает диалоговое окно подтверждения удаления заказа и при подтверждении удаляет выбранные строки из таблицы заказов и сохраняет изменения;

9) Метод `AdminForm_VisibleChanged`: Этот метод обрабатывает событие изменения видимости формы. В данном случае, метод пустой и не содержит дополнительной функциональности;

10) Метод `AdminForm_Activated`: Этот метод обрабатывает событие активации формы. Он выполняет заполнение таблиц *фильмов, сеансов и заказов* с использованием соответствующих адаптеровADO>NET (`фильмыTableAdapter`, `сеансыTableAdapter`, `заказыTableAdapter`).



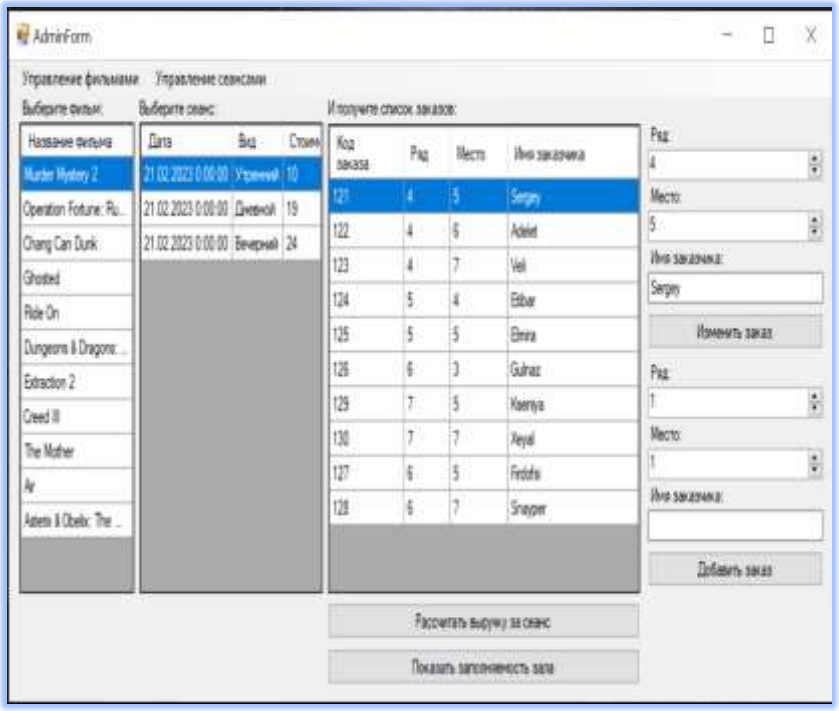


Рис.9. Главное окноAdminForm

2.3.3. Разработка формы FilmControlForm

Для разработки формы FilmControlForm был создан код, который представлен в приложении 2 (рис. 9).

В файле `FilmsControlForm.cs` определен класс `FilmsControlForm`, который является наследником класса `Form` из пространства имен `System.Windows.Forms`. Этот класс представляет форму для управления информацией о фильмах.

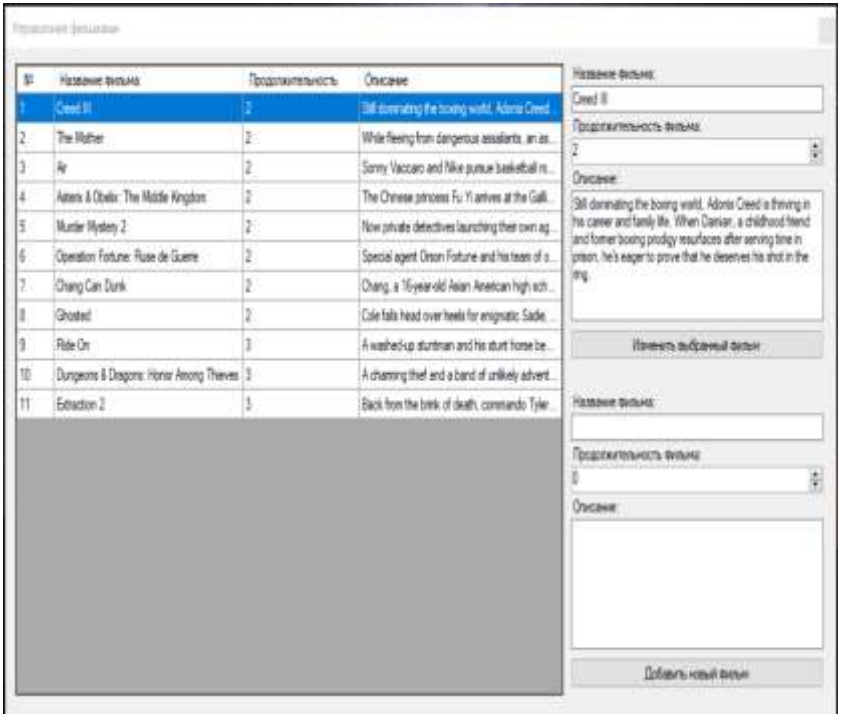


Рис.9. Главное окно Управление фильмами

У класса `FilmsControlForm` определены следующие элементы:

1) `adminForm`: Это поле класса типа `AdminForm`, которое хранит ссылку на экземпляр класса `AdminForm`, который представляет форму администратора. Это позволяет вернуться к форме администратора при закрытии формы управления фильмами;

2) Конструктор `FilmsControlForm`: Принимает экземпляр класса `AdminForm` в качестве параметра и сохраняет его в поле `adminForm`. Он также вызывает метод `InitializeComponent()`, который инициализирует все элементы управления формы;

3) Метод `saveChanges()`: Сохраняет изменения, внесенные в данные таблицы фильмов (`фильмыTableAdapter`) и обновляет набор данных (`kinoshkadbDataSet`). Затем выполняется заполнение таблицы фильмов с использованием адаптера `фильмыTableAdapter`;

4) Метод `FilmsControlForm_Load`: Этот метод обрабатывает событие загрузки формы. Внутри метода выполняется заполнение таблицы фильмов с использованием адаптера `фильмыTableAdapter`;

5) Метод `FilmsControlForm_FormClosing`: Этот метод обрабатывает событие закрытия формы. Он отображает форму администратора (`adminForm`) при закрытии формы управления фильмами;

6) Методы обработчиков событий `button1_Click` и `button2_Click`: Эти методы обрабатывают нажатия на

соответствующие кнопки на форме и выполняют определенные действия, такие как сохранение изменений и добавление нового фильма;

7) Метод `удалитьToolStripMenuItem_Click`: Этот метод обрабатывает событие щелчка на пункте меню "удалитьToolStripMenuItem". Он отображает диалоговое окно подтверждения удаления фильма и при подтверждении удаляет выбранные строки из таблицы фильмов и сохраняет изменения.

2.3.4. Разработка формы SessinsControlForm

Для разработки формы SessinsControlForm был создан код, который представлен в приложении 3.

В файле `SessionsControlForm.cs` определен класс `SessionsControlForm`, который является наследником класса `Form` из пространства имен `System.Windows.Forms`. Этот класс представляет форму для управления информацией о сеансах.

У класса `SessionsControlForm` определены следующие элементы:

1) `adminForm`: Это поле класса типа `AdminForm`, которое хранит ссылку на экземпляр класса `AdminForm`, который представляет форму администратора. Это позволяет вернуться к форме администратора при закрытии формы управления сеансами;

2) Конструктор `SessionsControlForm`: Принимает экземпляр класса `AdminForm` в качестве параметра и сохраняет его в поле `adminForm`. Он также вызывает метод `InitializeComponent()`, который инициализирует все элементы управления формы;

3) Метод `saveChanges()`: Сохраняет изменения, внесенные в данные таблицы сеансов (`сеансыTableAdapter`) и обновляет набор данных (`kinoshkadbDataSet`). Затем выполняется заполнение таблицы сеансов и таблицы фильмов с использованием соответствующих адаптеровADO.NET (`сеансыTableAdapter`, `фильмыTableAdapter`);

4) Метод `SessionsControlForm_Load`: Этот метод обрабатывает событие загрузки формы. Внутри метода выполняется заполнение таблицы сеансов и таблицы фильмов с использованием соответствующих адаптеров;

5) Методы обработчиков событий `button1_Click` и `button2_Click`: Эти методы обрабатывают нажатия на соответствующие кнопки на форме и выполняют определенные действия, такие как сохранение изменений и добавление нового сеанса;

6) Метод `удалитьToolStripMenuItem_Click`: Этот метод обрабатывает событие щелчка на пункте меню "удалитьToolStripMenuItem". Он отображает диалоговое окно подтверждения удаления сеанса и при подтверждении удаляет выбранные строки из таблицы фильмы-сеансы и сохраняет изменения;

7) Метод `SessionsControlForm_FormClosing`: Этот метод обрабатывает событие закрытия формы. Он отображает форму администратора (`adminForm`) при закрытии формы управления сеансами.

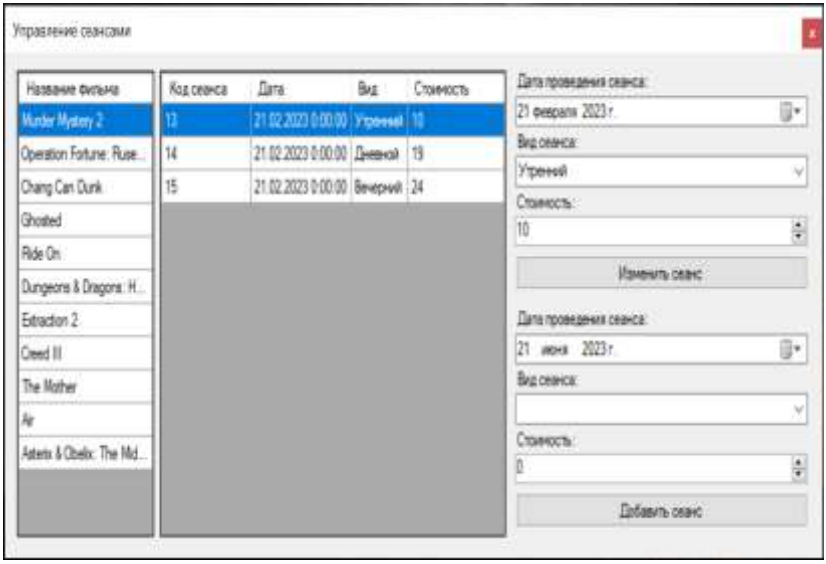


Рис.10. Окно Управление сеансами

2.3.5. Разработка формы DrawForm

Для разработки формы DrawForm был создан код, который представлен в приложении 4.

Этот код определяет класс `DrawForm`, который является формой для отображения зала кинотеатра с помощью рисования на холсте.

Основные точки в коде:

1) Класс `DrawForm` наследуется от класса `Form` из пространства имен `System.Windows.Forms`;

2) В конструкторе `DrawForm` передаются экземпляр класса `AdminForm` и двумерный массив `tmp`, который содержит информацию о занятости мест в зале кинотеатра;

3) Метод `DrawForm_Paint` - представляет обработчик события от рисовки формы. В этом методе происходит рисование зала кинотеатра на холсте (`e.Graphics`);

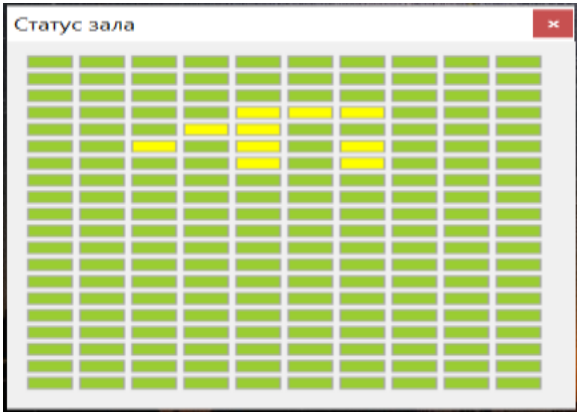


Рис.11. Окно о статусе зала

4) В цикле проходится по каждому месту в зале и в зависимости от значения в массиве `tmp` (1 - место занято, 0 - место свободно) заполняется прямоугольник соответствующим цветом (желтый для занятых мест и желтоЗеленый для свободных мест);

5) Метод `DrawForm_FormClosing` – представляет обработчик события закрытия формы. При закрытии формы происходит отображение формы `AdminForm`;

6) Метод `DrawForm_Load` – представляет обработчик события загрузки формы. В данный момент этот метод пуст и не содержит никакого кода;

7) Этот код позволяет отобразить зал кинотеатра с помощью рисования на форме `DrawForm`.

3. Заключение

Основной задачей было создать проект на основе Киноцентров. Соответственно для выполнения этой задачи были использованы программы:

1. Visual Studio (C#);
2. MS Access.

В программе “Visual Studio” на языке программирования C# были созданы приложения:

1. Для оформления фильма;
2. Для добавления данных в базу было использована “MSAccess”.

Главной целью было создание программы для помощи работникам и клиентам, чтобы работники могли быстро и эффективно оформлять покупку фильма для клиентов в кинотеатрах и сэкономить время.

В программе работники могут быстро рассчитать сумму денег за утренний, дневной и вечерний сеансы, также следить за количеством посетителей в кинозалах. Работники могут в программе добавлять или удалять фильмы, также можно добавлять дату, когда будет показываться фильм в этом кинотеатре и о чём этот фильм.

Список литературы:

1. Мартнн Р. Чистая архитектура. Искусство разработки программного обеспечения. СПб.: Питер, 2018. -352 с.: ил. - (Серия «Библиотека программиста»)

2. Язык C# и платформа .NET Core. https://learn.microsoft. com/ru-ru/dotnet/core/introduction (24.03.2023)

3. Одномерные массивы (Руководство по программированию на C#). https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/arrays/single-dimensional-arrays (07.04.2023)

4. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих: Пер. с англ. - СПб.: БХВ-Петербург. 2013

5. Подключение базы данных к C#. https://learn.microsoft .com/en-us/visualstudio/data-tools/connect-to-data-in-an-access-database-windows-forms?view=vs-2022

6. Сургуладзе Г., Нарешелашвили Г. Основы программной инженерии. (Уч.пособые - Лаб Практикум на visual C#). ГТУ. „IT-консальтинг центр". Тбилиси, 2022, -185 ст.

7. Долженко А.И., Глушенко С.А.. Программная инженерия. Учебное пособие. Ростов-на-Дону. 2017. -128 с.

8. Sommerville I. Softweare engineering, 11th ed., ISBN 978-0137035151. Addison-Wesley. 2016

9. MS Access и C# — работаем с базой данных из программы Windows Forms. https://vscode.ru/prog-lessons/ms-access-i-c-sharp-rabotaem-s-bd.html (12.03.2018)

Приложение 2.1

```
// ----  Листинг C# кода для AdminForm ----
usingSystem;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace kinoshka
{
publicpartialclassAdminForm : Form
    {
        AuthForm authForm;
publicAdminForm(AuthForm authForm)
        {
            InitializeComponent();
this.authForm = authForm;
        }

privatevoid saveChanges()
        {
            сеансыЗаказыBindingSource.EndEdit();

заказыTableAdapter.Update(kinoshkadbDataSet);
            kinoshkadbDataSet.AcceptChanges();
this.заказыTableAdapter.Fill(this.kinoshkadbDataSet.Зак
азы);
this.сеансыTableAdapter.Fill(this.kinoshkadbDataSet.Сеа
нсы);
this.фильмыTableAdapter.Fill(this.kinoshkadbDataSet.Фил
ьмы);
        }

privatevoid AdminForm_Load(object sender, EventArgs e)
```

```
        {


        }

privatevoid AdminForm_FormClosing(object sender,
FormClosingEventArgs e)
        {
this.authForm.Close();
        }

privatevoid
управлениеФильмамиToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            FilmsControlForm filmsControlForm = new
FilmsControlForm(this);
            filmsControlForm.Show();
this.Hide();
        }

privatevoid
управлениеСеансамиToolStripMenuItem_Click(object
sender, EventArgs e)
        {

            SessionsControlForm sessionsControlForm =
new SessionsControlForm(this);
            sessionsControlForm.Show();
this.Hide();
        }

privatevoid button3_Click(object sender, EventArgs e)
        {

foreach (Control c in panel1.Controls)
            {
foreach (Binding b in c.DataBindings)
```

```
                {
                    b.WriteValue();
                }
            }
this.saveChanges();
        }

privatevoid удалитьToolStripMenuItem_Click(object
sender, EventArgs e)
        {
if (MessageBox.Show("Вы действительно хотите удалить
этот заказ?\nЭто действие невозможно отменить.",
"Подтвердите удаление", MessageBoxButtons.YesNo,
MessageBoxIcon.Warning) == DialogResult.Yes)
            {
foreach (DataGridViewRow item in
dataGridView3.SelectedRows)
                {

сеансыЗаказыBindingSource.RemoveAt(item.Index);
                }
                saveChanges();
            }
        }

privatevoid button4_Click(object sender, EventArgs e)
        {
            kinoshkadbDataSet.СеансыRow tmp =
kinoshkadbDataSet.Сеансы.NewСеансыRow();
            tmp.Код_сеанса =
int.Parse(dataGridView1.SelectedRows[0].Cells[0].Value.
ToString());

kinoshkadbDataSet.Заказы.AddЗаказыRow(numericUpDown4.Va
lue.ToString(), numericUpDown3.Value.ToString(),
textBox2.Text, tmp);
this.saveChanges();
        }
```

```
privatevoid button1_Click(object sender, EventArgs e)
        {
if (dataGridView1.SelectedRows.Count > 0)
                MessageBox.Show("Выручка за сеанс
составляет:\n" + (dataGridView3.Rows.Count *
int.Parse(dataGridView1.Rows[dataGridView1.SelectedRows
[0].Index].Cells[3].Value.ToString())).ToString() + "
лари.");
        }

privatevoid button2_Click(object sender, EventArgs e)
        {
string[,] result = newstring[20,10];
for (int i = 0; i < 20; i++)
            {
for (int j = 0; j < 10; j++)
                {
string ch = " ";
foreach (DataGridViewRow r in dataGridView3.Rows)
                    {
if ((int.Parse(r.Cells[1].Value.ToString())-
1).ToString() == i.ToString() &&
(int.Parse(r.Cells[2].Value.ToString())-1).ToString()
== j.ToString())
                        {
                            ch = "1";
break;
                        }
                    }
                    result[i,j] = ch;
                }
            }
            DrawForm drawFrom = new DrawForm(this,
result);
            drawFrom.Show();
this.Hide();
        }
```

```
privatevoid AdminForm_VisibleChanged(object sender,
EventArgs e)
        {
        }

privatevoid AdminForm_Activated(object sender,
EventArgs e)
        {
this.заказыTableAdapter.Fill(this.kinoshkadbDataSet.Зак
азы);
this.сеансыTableAdapter.Fill(this.kinoshkadbDataSet.Сеа
нсы);
this.фильмыTableAdapter.Fill(this.kinoshkadbDataSet.Фил
ьмы);
}
    }
}
```

Приложение 2.2

```
// ---- Листинг C# кода для FilmControlForm -----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace kinoshka
{
publicpartialclassFilmsControlForm : Form
    {
        AdminForm adminForm;
publicFilmsControlForm(AdminForm adminForm)
```

```
        {
            InitializeComponent();
this.adminForm = adminForm;
        }

privatevoid saveChanges()
        {
            фильмыBindingSource.EndEdit();

фильмыTableAdapter.Update(kinoshkadbDataSet);
            kinoshkadbDataSet.AcceptChanges();
this.фильмыTableAdapter.Fill(this.kinoshkadbDataSet.Фил
ьмы);
        }

privatevoid FilmsControlForm_Load(object sender,
EventArgs e)
        {
this.фильмыTableAdapter.Fill(this.kinoshkadbDataSet.Фил
ьмы);

        }

privatevoid FilmsControlForm_FormClosing(object sender,
FormClosingEventArgs e)
        {
this.adminForm.Show();
        }

privatevoid button1_Click(object sender, EventArgs e)
        {

foreach (Control c in panel1.Controls)
            {
foreach (Binding b in c.DataBindings)
                {
                    b.WriteValue();
                }
```

```
            }
this.saveChanges();

        }

privatevoid button2_Click(object sender, EventArgs e)
        {


kinoshkadbDataSet.Фильмы.AddФильмыRow(textBox4.Text,
numericUpDown2.Value.ToString(), textBox3.Text);
this.saveChanges();
        }

privatevoid удалитьToolStripMenuItem_Click(object
sender, EventArgs e)
        {
if (MessageBox.Show("Вы действительно хотите удалить
этот фильм?\nЭто действие невозможно отменить.",
"Подтвердите удаление", MessageBoxButtons.YesNo,
MessageBoxIcon.Warning) == DialogResult.Yes)
            {
foreach (DataGridViewRow item in
dataGridView1.SelectedRows)
                {

фильмыBindingSource.RemoveAt(item.Index);
                }
                saveChanges();
}
        }
    }
}
```

Приложение 2.3

```
// ---- Листинг C# кода для SessionsControlForm ----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace kinoshka
{
publicpartialclassSessionsControlForm : Form
    {
        AdminForm adminForm;
publicSessionsControlForm(AdminForm adminForm)
        {
            InitializeComponent();
this.adminForm = adminForm;
        }

privatevoid saveChanges()
        {
            фильмыСеансыBindingSource.EndEdit();

сеансыTableAdapter.Update(kinoshkadbDataSet);
            kinoshkadbDataSet.AcceptChanges();
this.сеансыTableAdapter.Fill(this.kinoshkadbDataSet.Сеа
нсы);
this.фильмыTableAdapter.Fill(this.kinoshkadbDataSet.Фил
ьмы);
        }

privatevoid SessionsControlForm_Load(object sender,
EventArgs e)
{
```

```
// TODO: данная строка кода позволяет загрузить данные
в таблицу "kinoshkadbDataSet.Сеансы". При необходимости
она может быть перемещена или удалена.
this.сеансыTableAdapter.Fill(this.kinoshkadbDataSet.Сеа
нсы);
// TODO: данная строка кода позволяет загрузить данные
в таблицу "kinoshkadbDataSet.Фильмы". При необходимости
она может быть перемещена или удалена.
this.фильмыTableAdapter.Fill(this.kinoshkadbDataSet.Фил
ьмы);

}

privatevoid button1_Click(object sender, EventArgs e)
        {
foreach (Control c in panel1.Controls)
            {
foreach (Binding b in c.DataBindings)
                {
                    b.WriteValue();
                }
            }
this.saveChanges();
        }

privatevoid button2_Click(object sender, EventArgs e)
        {

            kinoshkadbDataSet.ФильмыRow tmp =
kinoshkadbDataSet.Фильмы.NewФильмыRow();
            tmp.Код_фильма =
int.Parse(dataGridView2.SelectedRows[0].Cells[0].Value.
ToString());

kinoshkadbDataSet.Сеансы.AddСеансыRow(dateTimePicker2.V
alue.Date.ToString(), comboBox2.Text,
numericUpDown2.Value.ToString(), tmp);
this.saveChanges();
```

```
        }

privatevoid удалитьToolStripMenuItem_Click(object
sender, EventArgs e)
        {
if (MessageBox.Show("Вы действительно хотите удалить
этот сеанс?\nЭто действие невозможно отменить.",
"Подтвердите удаление", MessageBoxButtons.YesNo,
MessageBoxIcon.Warning) == DialogResult.Yes)
            {
foreach (DataGridViewRow item in
dataGridView1.SelectedRows)
                {

фильмыСеансыBindingSource.RemoveAt(item.Index);
                }
                saveChanges();
            }
        }

privatevoid SessionsControlForm_FormClosing(object
sender, FormClosingEventArgs e)
        {
            adminForm.Show();
        }
    }
}
```

Приложение 2.4

```
// ---- Листинг C# кода для DrawForm -----
usingSystem;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace kinoshka
{
publicpartialclassDrawForm : Form
    {
        AdminForm adminForm;
string[,] tmp;
publicDrawForm(AdminForm adminForm, string[,] tmp)
        {
            InitializeComponent();
this.adminForm = adminForm;
this.tmp = tmp;
        }

privatevoid DrawForm_Paint(object sender,
PaintEventArgs e)
        {
int chairWidth = ((this.ClientSize.Width-20) / 10) - 4;
int chairHeight = ((this.ClientSize.Height-20) / 20) -
4;
int offsetX = 10;
int offsetY = 10;

for (int i = 0; i < 20; i++)
            {
for (int j = 0; j < 10; j++)
                {
if (tmp[i,j] == "1")
                    {

e.Graphics.FillRectangle(Brushes.Yellow, offsetX,
offsetY, chairWidth, chairHeight);
                    }
else
                    {

e.Graphics.FillRectangle(Brushes.YellowGreen, offsetX,
offsetY, chairWidth, chairHeight);
```

```
                    }

e.Graphics.DrawRectangle(Pens.DarkGray, offsetX,
offsetY, chairWidth, chairHeight);
                    offsetX += chairWidth + 4;
                }
                offsetX = 10;
                offsetY += chairHeight + 4;
            }

        }

privatevoid DrawForm_FormClosing(object sender,
FormClosingEventArgs e)
        {
            adminForm.Show();
        }

privatevoid DrawForm_Load(object sender, EventArgs e)
{

        }
    }
}
```

Georgian Technical University

Surguladze Gia, Nareshelashvili Gulbaat

BASICS OF SOFTWARE ENGINEERING
(Lectures and course project)

ISBN 978-9941-8-5590-0

Передан в производство 29.06 2023 г. Размер офсетной бумаги 60X84 1/16. Условная печатная форма 10,6. Тираж 50 экз.

«Научный центр ИТ-консалтинга» ГТУ,

Тбилиси, М. Костава 77

ISBN 978-9941-8-5590-0