



**Уральский  
федеральный  
университет**

имени первого Президента  
России Б.Н.Ельцина

**Институт радиоэлектроники  
и информационных  
технологий — РТФ**

**И. А. СПИЦИНА  
К. А. АКСЁНОВ**

# **ПРИМЕНЕНИЕ СИСТЕМНОГО АНАЛИЗА ПРИ РАЗРАБОТКЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ИНФОРМАЦИОННЫХ СИСТЕМ**

**Учебное пособие**



Министерство образования и науки Российской Федерации  
Уральский федеральный университет  
имени первого Президента России Б.Н. Ельцина

И. А. Спицина  
К. А. Аксёнов

**Применение системного анализа  
при разработке пользовательского интерфейса  
информационных систем**

Учебное пособие

Рекомендовано методическим советом  
Уральского федерального университета  
для студентов вуза, обучающихся  
по направлению подготовки  
09.03.01 «Информатика и вычислительная техника»

Екатеринбург  
Издательство Уральского университета  
2018

УДК 004.45:004.7(075.8)  
ББК 32.973.1я73+32.971.35я73  
С72

Рецензенты:

Отдел динамических систем института математики и механики им. Н. Н. Красовского Уральского отделения Российской академии наук (ИММ УрО РАН), (зав. отделом проф., д-р физ.-мат. наук *А. М. Тарасьев*);

доц., канд. техн. наук *В. Ф. Ярчук* (начальник программно-технологического отдела ООО «ТЭКСИ-Консалтинг»).

Научный редактор — проф., д-р техн. наук *Л. Г. Доросинский*

**Спицина, И. А.**

С72 Применение системного анализа при разработке пользовательского интерфейса информационных систем : учеб. пособие / И. А. Спицина, К. А. Аксёнов. — Екатеринбург : Изд-во Урал. ун-та, 2018. — 100 с.

ISBN 978-5-7996-2265-7

В издании отражены аспекты разработки и проектирования визуального интерфейса пользователя информационных систем. Основное внимание уделено разработке интерфейсов и архитектуре программного обеспечения с использованием метода системного анализа и автоматизированных средств проектирования (CASE-средств). Описана технология проектирования программного обеспечения и пользовательского интерфейса. Пособие содержит примеры, иллюстрирующие материал. Предназначено для студентов дневной и заочной форм обучения направления 09.03.01 «Информатика и вычислительная техника».

Библиогр.: 11 назв. Табл. 11. Рис. 43.

УДК 004.45:004.7(075.8)  
ББК 32.973.1я73+32.971.35я73

ISBN 978-5-7996-2265-7

© Уральский федеральный  
университет, 2018

---

# Оглавление

---

<b>Список основных сокращений</b> .....	4
<b>Предисловие</b> .....	5
<b>Глава 1. Анализ и проектирование интерфейса ПО</b> .....	6
1.1. Принципы проектирования пользовательских интерфейсов информационных систем.....	6
1.1.1. Роль аналитика при проектировании пользовательского интерфейса .....	6
1.1.2. Этапы разработки пользовательского интерфейса.....	7
1.1.3. Модели реализации и ментальные модели .....	8
1.1.4. Анализ прототипа ПИ на возможные проблемы .....	9
1.1.5. Особенности восприятия человеком информации .....	11
1.1.6. Особенности интеллектуальных мультиагентных систем.....	14
1.2. Вопросы качества пользовательского интерфейса программного обеспечения .....	20
1.2.1. Концепции качества интерфейса .....	20
1.2.2. Рекомендации по использованию концепций.....	24
1.2.3. Оценка удобства использования пользовательского интерфейса ....	25
1.3. Технология проектирования архитектуры программного обеспечения и пользовательского интерфейса .....	27
<b>Глава 2. Разработка интерфейса программного обеспечения</b> .....	36
2.1. Проектирование и создание прототипа пользовательского интерфейса приложения .....	36
2.2. Проектирование пользовательского интерфейса на этапе высокоуровневого проектирования .....	44
2.3. Разработка функций приложения, позволяющих взаимодействовать с папками и файлами .....	55
2.4. Работа с методами сериализации и десериализации объектов .....	62
2.5. Документирование .....	67
2.6. Применение технологии WPF для разработки интерфейса пользователя .....	73
2.7. Разработка WPF-приложения для работы с данными .....	83
2.8. Оценка эффективности пользовательского интерфейса по критерию скорости на основе модели GOMS .....	91
<b>Заключение</b> .....	97
<b>Список библиографических ссылок</b> .....	98

---

## Список основных сокращений

---

CASE	— Computer Aided Software Engineering, автоматизированная разработка ПО
GOMS	— Goals, Operators, Methods, and Selection Rules
MSF	— Microsoft Solutions Framework
RUP	— Rational Unified Process
WPF	— Windows Presentation Foundation
UML	— Unified Modeling Language, унифицированный язык моделирования
ИС	— информационная система
MAC	— мультиагентная система
МППР	— мультиагентные процессы преобразования ресурсов
ОТС	— организационно-технические системы
ПИ	— пользовательский интерфейс
СА	— системный анализ
ТЗ	— техническое задание

---

## Предисловие

---

Учебное пособие посвящено вопросам проектирования и разработки пользовательского интерфейса (ПИ) программного обеспечения (ПО), в нем рассмотрены современные подходы к разработке и проектированию интерфейсов информационных систем. Данные подходы используют в своей основе системный анализ при исследовании и формализации процессов предметной области, а также методы, применяемые в интеллектуальных системах и системах автоматизации проектирования программного обеспечения (CASE-средствах).

Характерной особенностью разрабатываемого и активно применяемого программного обеспечения является его социальная направленность, что является следствием активного распространения смартфонов, планшетов и различных портативных устройств на мировом рынке. Визуальный графический интерфейс у конечного пользователя (не специалиста в области информационных технологий) непосредственно ассоциируется с самим программным обеспечением (в том числе его математическим, лингвистическим и алгоритмическим обеспечением), поэтому актуальной задачей является разработка эффективного пользовательского интерфейса.

---

# Глава 1.

## Анализ и проектирование интерфейса ПО

---

### 1.1. Принципы проектирования пользовательских интерфейсов информационных систем

---

#### 1.1.1. Роль аналитика при проектировании пользовательского интерфейса

**П**ользовательский интерфейс (ПИ) представляет собой совокупность программных и аппаратных средств, осуществляющих взаимодействие пользователя с информационной системой. ПИ включает в себя систему меню, диалоговые формы, сообщения об ошибках, справочную систему и т. п.

Поскольку ПИ является частью информационной системы, то, с одной стороны, аналитику необходимо представлять базовые принципы проектирования ПИ, а с другой стороны, он должен быть вовлечен в этот процесс [1].

Как известно, при разработке ПИ следует учитывать компьютерную компетентность пользователя. Аналитик участвует в анализе потенциальных пользователей ИС, а затем преобразует полученную информацию в требования к ПИ. В частности, аналитик может:

- проанализировать предпочтения пользователей (цвет, расположение элементов и т. п.) в существующих интерфейсных решениях и преобразовать их в нефункциональные требования к ПИ;
- наблюдать за работой с прототипом ПИ или просмотром эскизов экранов, а затем сформулировать замечания и предложения по улучшению ПИ;
- проанализировать ПИ на возможность беспрепятственного выполнения всех функций, необходимых пользователю.



Таким образом, чтобы первое общение пользователя с информационной системой не стало последним, аналитику необходимо участвовать в определении требований, которые охватывают не только функции разрабатываемой системы, но и ПИ, чтобы повысить качество готового продукта.

Кратко рассмотрим инструментарий, который может использовать аналитик в своей деятельности.

Диаграмма вариантов использования языка UML (Unified Modeling Language) может быть применена для формализации функциональных требований к системе, в том числе для описания взаимодействия пользователей с проектируемой системой [2]. Следует отметить, что изобразительных средств этой диаграммы недостаточно для подробного описания требований, поэтому ДВИ дополняют текстовыми сценариями. С их помощью можно уточнить или детализировать последовательность действий, совершаемых системой при выполнении ее вариантов использования.

Шаблон «Проволочная диаграмма Visio» может быть использован для быстрого и простого создания прототипов форм приложения. Проволочные диаграммы позволяют обсудить общие решения по юзабилити и дизайну ПИ.

### **1.1.2. Этапы разработки пользовательского интерфейса**

Пользовательский интерфейс представляет собой набор средств диалога, взаимодействия программы (машины) с человеком. Именно поэтому ПИ является тем элементом ИС, по которому складывается первое общее впечатление о ней. Можно выделить следующие проблемы, которые возникают при разработке ПИ:

- отсутствие четких требований к ПИ в ТЗ, что приводит к его многочисленным переделкам;
- отсутствие точной структуры ПИ, что влечет за собой невозможность планирования работ;
- изменение и развитие бизнес-процессов, что приводит к необходимости адаптации ПИ, которую не всегда возможно реализовать.

Для решения этих проблем необходимо использовать адекватные методологические подходы.

Создание ПИ включает в себя следующие крупные этапы:

- исследование, анализ и определение общих требований к ПИ;
- определение сценариев использования и пользовательской модели интерфейса;
- разработка прототипа ПИ;
- реализация ПИ;
- тестирование и оценка качества ПИ.

В зависимости от выбранной модели жизненного цикла программного обеспечения эти этапы могут повторяться циклически [3]. На первых двух этапах следует продумать структуру приложения, определить основные требования к ПИ. На этапе создания прототипа показываются основные аспекты функционирования ИС и базовые подходы к ПИ. Созданный прототип позволяет обсудить базовые решения с пользователем до этапа разработки. На этапе тестирования, к которому рекомендуется подключать пользователей, можно выявить не только ошибки программирования, но и оценить, насколько ПИ отвечает потребностям и ожиданиям пользователей. Итерационный подход позволяет устранять выявленные недостатки на последующих итерациях разработки.

### 1.1.3. Модели реализации и ментальные модели

Различия между образами мышления пользователей, аналитиков и разработчиков — причина многих проблем, связанных с неудовлетворенностью результатом проектирования ПИ [4].

При работе с ИС пользователь, конечно же, не знает всех сложностей ее функционирования, он создает упрощенную мысленную схему, которая называется ментальной моделью. Для предсказания поведения интерфейса системы пользователь применяет свою ментальную модель, то есть *ментальная модель* — это представление пользователя о процессе взаимодействия с ИС.

*Модель реализации* — это представление о том, как реально работает ИС.

Очевидно, что в ИС наблюдаются существенные расхождения между моделью реализации и ментальной моделью, поскольку сложность реализации достаточно высокая, и пользователь

не может определить, что действительно делает программа в ответ на его действия.

Разрыв между реализацией и представлением пользователя служит источником третьей модели, возникающей при обсуждении ПИ, — модели представления. *Модель представления* — это способ демонстрации пользователю тех функций ИС, которые выбрал проектировщик.

Пользовательские интерфейсы и схемы взаимодействия ИС, спроектированные с точки зрения программиста, будут близки к модели реализации, но малопонятны пользователю. Следует разрабатывать такую модель представления, которая будет наиболее близка к ментальной модели. Для построения эффективной модели представления следует привлекать к работе аналитика. Он может использовать следующие способы сбора информации о работе пользователей:

- анализ их задач;
- интервью с настоящими и потенциальными пользователями;
- посещение мест их работы;
- отзывы клиентов;
- тесты по пригодности.

При этом стоит учитывать, что пользователи обычно описывают то, что они делают, а не то, что им хотелось бы делать, то есть их работа не всегда оптимальна.

#### **1.1.4. Анализ прототипа ПИ на возможные проблемы**

Разработка прототипа ПИ и его анализ позволяют на начальных этапах проектирования проверить предлагаемые концепции ПИ и осуществимость требований. Кроме того, есть возможность обнаружить проблемы ПИ до того, как они станут критическими. Чтобы обнаружить проблему, нужно проанализировать следующие моменты [5]:

- учитывает ли разрабатываемый ПИ все особенности устройств ввода—вывода информации, используемых пользователем;
- позволяют ли выбранные интерактивные элементы организовать ввод и вывод информации, которые будет соответствовать требованиям к ПИ;

- учитывают ли выбранные технологии и методы ведения диалога ИС с пользователем:
  - степень активности пользователя при взаимодействии (автоматический режим или перехват управления программой на себя, программные помощники);
  - степень учета ситуации (контекстные подсказки, меню дальнейших событий или объектов, запоминание типичных путей диалога);
  - устойчивость, терпимость к ошибкам пользователя путем исправления или недопущения типичных ошибок;
  - дублирование вручную отдельных функций системы и дополнительные контрольные процедуры работы отдельных режимов;
  - настройка ПИ на различный уровень подготовки пользователя;
  - степень адаптивности ПИ под предпочтения пользователя;
  - настройка ПИ на специфику задачи (новый формат данных, изменение набора объектов, дополнение атрибутов объектов);
- оптимально ли размещена информация и управляющие элементы на экранных формах; при расположении визуальных компонентов на форме учитывают:
  - алгоритм работы пользователя с данными;
  - баланс между «детальностью—обобщенностью» вывода информации;
  - необходимость выделения важной информации на экране;
  - необходимость четкого определения основных и вспомогательных блоков информации;
  - необходимость выделения на форме полей со статическими и динамическими данными;
- способы формирования обратной связи между пользователем и ИС:
  - показ актуального состояния системы, режима работы системы и режима взаимодействия;
  - вывод отдельных, важных для выполняемой функции данных и показателей;
  - отражение действий пользователя;
  - ясность и информативность сообщений системы;

- учитываются ли при проектировании панелей меню и инструментов следующие моменты:
  - логическая и смысловая группировка пунктов;
  - фиксированная позиция панелей на экране;
  - ограничение на ширину списка выборов и шагов (глубины) меню;
  - использование привычных названий, широко распространенных икон-пиктограмм, традиционных икон-символов и аккуратное введение сокращений;
  - размещение наиболее часто используемых пунктов в начале списка;
- используются ли средства ориентации и навигации:
  - варьирование степени детализации рассматриваемых объектов;
  - быстрый поиск в списке или таблице;
  - указание на дополнительно существующую информацию и способ ее получения;
  - использование средств листания и прокрутки;
- учитываются ли при создании форм для ввода данных:
  - использование одного или нескольких механизмов ввода в рамках режима;
  - определение способов ввода данных (таблицы, списки, простая форма, меню и пр.);
  - минимизация объема ввода;
  - выделение редактируемых обязательных и необязательных, а также не редактируемых полей;
  - использование механизмов быстрого ввода (по умолчанию, сокращения, с продолжением и пр.);
  - выделение введенной или отредактированной информации.

### **1.1.5. Особенности восприятия человеком информации**

При разработке ПИ следует учитывать особенности восприятия человеком информации, поскольку это влияет на производительность человека. Их нельзя натренировать. Исследованиями этих свойств человека занимаются когнитивная психология

и эргономика [6]. В работе приводятся практики проектирования ПИ, основанные на знаниях о человеческой психологии.

Одна из особенностей человека заключается в том, что в любой момент времени только один предмет может находиться во внимании человека. Это может быть либо объект реального мира, либо мыслительный процесс. Предмет, на котором сосредоточено внимание человека, называется *локусом* его внимания. Перечислим особенности человеческого восприятия, которые связаны с локусом внимания.

1. При периодическом переключении внимания (например, с рабочей области документа на уведомления об ошибках) эффективность работы снижается. Это связано с тем, что при смене локуса теряется связанная с ним «оперативная» информация, которая содержится в кратковременной памяти. Соответственно, при возвращении к прежнему локусу эту информацию необходимо каким-то образом восстанавливать.

2. При пристальном сосредоточении внимания все события вне локуса могут игнорироваться или просто оставаться незамеченными.

Отсюда следуют следующие рекомендации:

- если необходимо несколько проверок, то проверять нужно до первой возникшей ошибки;
- если сообщение об ошибке имеет большое значение, то нужно предусмотреть запись информации об ошибке в лог-файл.
- чем более критической является задача, тем меньше вероятность того, что пользователь заметит предупреждения относительно тех или иных потенциально опасных действий. Предупреждающее сообщение с наибольшей вероятностью может остаться незамеченным именно в тот момент, когда информация, содержащаяся в нем, имеет наибольшую ценность.

При совершении ежедневно повторяющихся действий человек перестает уделять внимание их выполнению — действия совершаются неосознанно, то есть у человека формируются привычки. С одной стороны, привычка позволяет человеку быстрее выполнять рутинные операции. ПИ следует разрабатывать так, чтобы он развивал у пользователей такие привычки, которые позволяют упростить ход работы. Следовательно, не стоит кардинально

изменять размещение пунктов меню и порядок выполнения функций в новых версиях системы без достаточных на то оснований.

С другой стороны, привычные действия пользователя могут принести вред. Типичный пример этого, который приводится в литературе, — подтверждение выполнения команды в диалоговом окне. Если при выполнении рутинных операций все время появляется диалоговое окно, то у пользователя вырабатывается привычка не вчитываться в текст сообщения, а сразу нажимать определенную кнопку. В конце концов возникнет ситуация, когда пользователь, не задумываясь, подтвердит команду, и это приведет к негативным последствиям. Отметим, что данный пример показывает неудачное интерфейсное решение, а не «глупость» пользователя. Если такое действие пользователя на самом деле критично для бизнес-процесса, то необходимо обеспечить его обратимость. Кроме того, можно также разрабатывать несколько вариантов решения задачи пользователя — тогда локус внимания пользователя смещается с самой задачи на выбор варианта.

Пользователь не может в своей работе избежать формирования привычек, и наличие в интерфейсе нескольких режимов может помешать этому. Режимы — это состояния интерфейса, в которых один и тот же жест пользователя интерпретируется по-разному. Жест — действие или последовательность действий, которые человек не разделяет на составляющие, а выполняет как бы единым движением. Один и тот же жест может вызывать разные действия в разных состояниях интерфейса. Интерфейс называется модальным, если в нем есть состояния, которые человек не осознает во время жеста, но в которых этот жест интерпретируется по-разному. Модальность может стать источником ошибок при взаимодействии с программой, поэтому при разработке ПИ следует ее избегать. Можно предложить следующие решения:

- разделение жестов (разные действия вызывают разные кнопки, клавиши и т. п.);
- использование «квазирежимов» (пользователю необходимо удерживать некоторую клавишу или кнопку для работы в таком режиме, следовательно, пользователь сосредоточен на выполнении этого действия, осознает, в каком режиме находится ПИ, и не допустит модальной ошибки);

- изменение сценария работы пользователя (изменяют, что и в каком порядке выполняет пользователь в разных состояниях интерфейса).

### 1.1.6. Особенности интеллектуальных мультиагентных систем

На основе результатов исследований современных ученых в области распределенных компьютерных систем, сетевых технологий решения проблем и параллельных вычислений сформировалось новое направление — мультиагентные системы.

*Агент* — это аппаратная или программная сущность, способная действовать в интересах достижения целей, поставленных перед ней владельцем и (или) пользователем [7]. Агенты обладают набором следующих свойств:

- адаптивность (агент обладает способностью обучаться);
- автономность (агент работает как самостоятельная программа, ставя себе цели и выполняя действия для их достижения);
- сотрудничество (агент может взаимодействовать с другими агентами несколькими способами, например, играя роль поставщика или потребителя информации или одновременно обе эти роли);
- способность к рассуждениям (агенты могут обладать частичными знаниями или механизмами вывода, например, знаниями, как приводить данные из различных источников к одному виду. Агенты могут специализироваться на конкретной предметной области);
- коммуникативность (агенты могут общаться с другими агентами);
- мобильность (способность к передаче кода агента с одного сервера на другой) [7].

Существуют следующие типы агентов: реактивные, интеллектуальные, гибридные.

Реактивный агент выбирает из нескольких правил «ситуация—действие» наиболее подходящее. Интеллектуальный агент имеет цели и использует общие ограниченные ресурсы и знания о внешнем мире для решения поставленных перед ним задач. Гибридный агент сочетает в себе возможности первых двух.



Интеллектуальная МАС представляет собой множество интеллектуальных агентов, распределенных в сети, которые отслеживают необходимые данные и взаимодействуют друг с другом для достижения поставленных перед ними целей. Можно выделить несколько важных причин взаимодействия агентов: совместимость целей (общая цель), отношение к ресурсам, необходимость привлечения недостающего опыта, взаимные обязательства [7].

С точки зрения программирования агенты представляют собой программы, которые способны действовать самостоятельно от лица пользователя. Агентный подход и методы искусственного интеллекта сейчас активно применяются в различных веб-сервисах, электронной коммерции, электронной медицине, электронном правительстве, распределенных системах. Уровень развития разработки программного обеспечения (ПО) за последние десятилетия существенно шагнул вперед — многие методы и подходы, применяемые в интеллектуальных и экспертных системах, естественным образом влились в методы разработки ПО и активно применяются в них.

При проведении системного анализа ОТС, к которым относятся системы взаимодействия человека и машины, обычно описывают следующие составляющие: миссию, виденье, стратегии, внешние процессы, внутренние процессы (производственные процессы, бизнес-процессы и т.д.). Применение мультиагентного подхода и модели МППР позволяет по-новому взглянуть на ОТС с точки зрения динамических систем, основанных на знаниях, и также позволяет уделить внимание следующим элементам:

- моделям человека (или моделям лица, принимающего решения), их знаниям, моделям поведения (процессам принятия решений);
- моделям координации и взаимодействия агентов;
- динамической составляющей процессов;
- рассмотрению отношений миссии, видения, стратегий, целей, ключевых показателей деятельности и мероприятий с помощью методики стратегического управления — системы сбалансированных показателей.

При построении иерархической модели на каждом уровне вводятся свои представления о системе и элементах. Элемент  $k$ -го уровня является системой для  $(k-1)$  уровня. Продвижение

от уровня к уровню имеет строгую направленность, определяемую стратегией проектирования — дедуктивную нисходящую «сверху вниз» (top–down) или индуктивную восходящую «снизу вверх» (bottom–up).

В контексте формализации процессов дедуктивную нисходящую стратегию проектирования используют в нотациях IDEF0, IDEF3, DFD, EPC, а также при построении иерархических моделей динамических процессов (агрегатах, сетях Петри, расширенных сетях Петри). Индуктивную восходящую стратегию используют в системных графах высокого уровня интеграции.

Для описания иерархической структуры мультиагентного процесса преобразования ресурсов (рис. 1.1) были использованы системные графы высокого уровня интеграции:

$$\begin{aligned} \vec{PR}_{L=i}^{\Sigma} &= \{Sender^m \cup Op^m \cup Receiver^m \cup Junction^m \cup Agent^m\}_{L=i}; \\ \{PR_{L=j}^{p_i}; p_i = 1, \dots, n_{L=j}^p\}_{j=2, \dots, i}; \{Relation_{AB}^{mk}\}_{L=i} &> . \end{aligned}$$

Граф  $i$ -го уровня интеграции образуется в результате поэтапной интеграции графов  $\vec{PR}_1^{\Sigma}, \vec{PR}_2^{\Sigma}, \dots, \vec{PR}_{i-1}^{\Sigma}$  с образованием на каждом  $j$ -м этапе множества  $\{PR_{L=j}^p; p=1, \dots, n_{L=j}^p\}$  процессов (под-процессов)  $j$ -го уровня интеграции,  $L$  — уровень интеграции. Элементы множества мультиагентного процесса преобразования ресурсов  $\{Sender^m \cup Op^m \cup Receiver^m \cup Junction^m \cup Agent^m\}_{L=i} \subset \subset \{Sender^m \cup Op^m \cup Receiver^m \cup Junction^m \cup Agent^m\}_{L=i-1} \subset \dots \subset \subset \{Sender^m \cup Op^m \cup Receiver^m \cup Junction^m \cup Agent^m\}$  и множества ресурсных отношений  $\{Relation_{AB}^{mk}\}_{L=i} \subset \{Relation_{AB}^{mk}\}_{L=i-1} \subset \dots \subset \{Relation_{AB}^{mk}\}$  системного графа  $\vec{PR}_{L=i}^{\Sigma}$  представляют собой элементы процесса преобразования и ресурсные отношения между элементами, а также элементы  $Sender^m \cup Op^m \cup Receiver^m \cup \cup Junction^m \cup Agent^m$  и ресурсные отношения  $Relation_{AB}^{mk}$  системного графа  $\vec{PR}^{\Sigma}$  нулевого уровня интеграции, не вошедшие при поэтапной интеграции ни в один процесс  $PR_{L=j}^p$ .

С точки зрения динамического моделирования в имитации участвуют только те элементы, которые в результате применения дедуктивной стратегии СА являются элементарными и в дальнейшем не детализируются. При использовании аппарата системных

графов на первом шаге построения модели (нулевой уровень интеграции) динамической системы получаем все необходимые данные для имитации.

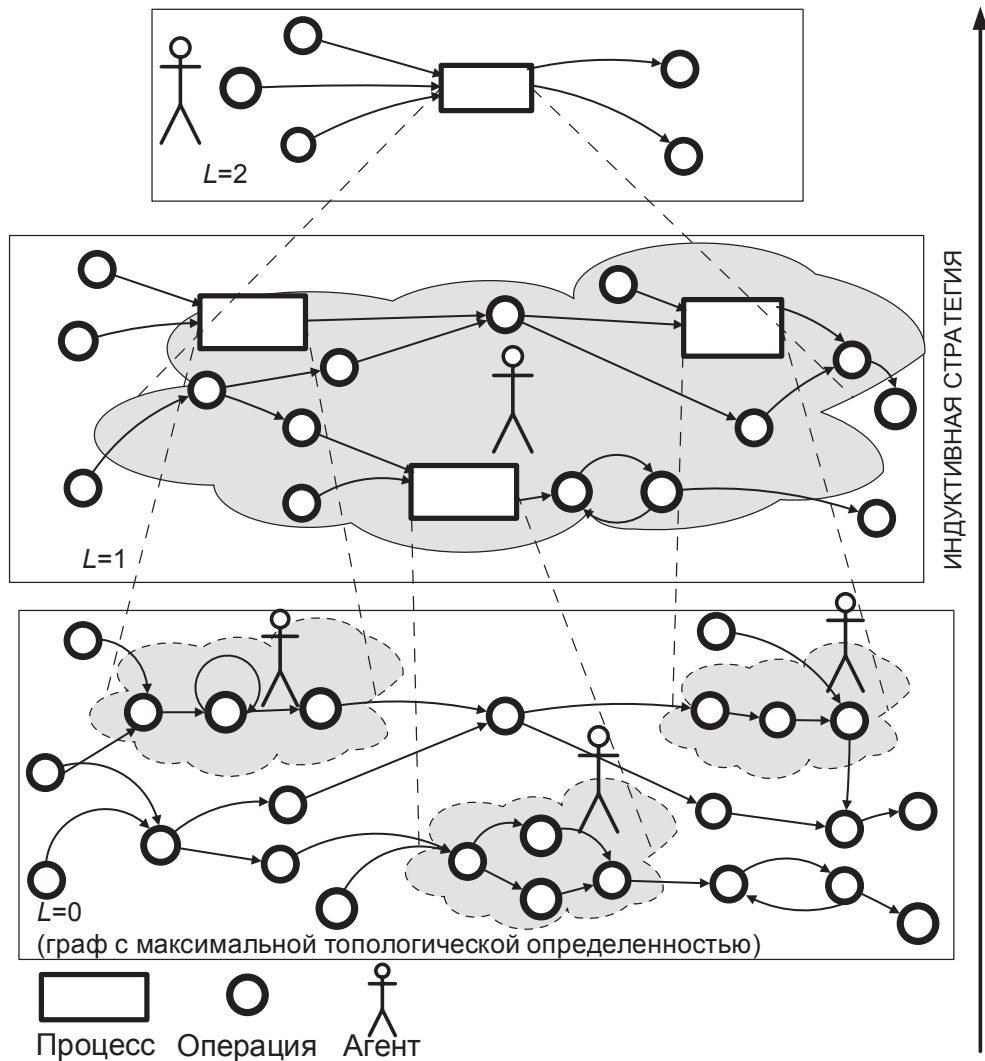


Рис. 1.1. Иерархическое представление МППР

В целом, существующие подходы к проектированию сложных систем можно разделить на два больших класса:

- структурный (системный) подход или анализ, основанный на идее алгоритмической декомпозиции, где каждый модуль

системы выполняет один из важнейших этапов общего процесса;

- объектный подход, связанный с декомпозицией и выделением не процессов, а объектов, при этом каждый объект рассматривается как экземпляр определенного класса.

Объектно-ориентированный подход, возникший как технология программирования больших программных продуктов, базируется на основных элементарных понятиях:

- объекты и классы как объекты, связанные общностью структуры и свойств;
- классификация как средство упорядочения знаний;
- иерархии с наследованием свойств;
- инкапсуляция как средство ограничения доступа;
- методы и полиморфизм для определения функций и отношений.

ООП имеет систему условных обозначений и предлагает набор моделей для проектирования сложных систем. Широкое распространение объектно-ориентированных языков программирования (C++, CLOS, Smalltalk, G2 и т. п.) успешно демонстрирует жизнеспособность и перспективность этого подхода. Этот подход успешно применяется и в CASE-средствах не только для проектирования программ, но и для моделирования бизнес-процессов.

На рис. 1.2 представлена концепция дуальной стратегии проектирования при проектировании функциональной структуры для экспертной системы помощи специалиста по развитию мультисервисной сети связи.

Основанием для прекращения агрегирования и дезагрегирования является полное использование словаря терминов, которым владеет эксперт, при этом число уровней является значимым фактором успешности структурирования. В части задачи проектирования пользовательского интерфейса ПО одной из важных задач становится задача выбора и согласования применяемых метафор.

При проектировании ПИ интеллектуальных систем следует учитывать возможность:

- формирования непрограммирующим пользователем произвольного запроса к системе;
- преобразование внутреннего представления результата обработки в формат, понятный непрограммирующему пользователю;

- использования когнитивной графики, то есть пользователю показываются графические образы, которые соответствуют текущему состоянию процесса или работы ИС (например, системы управления оперативными процессами).

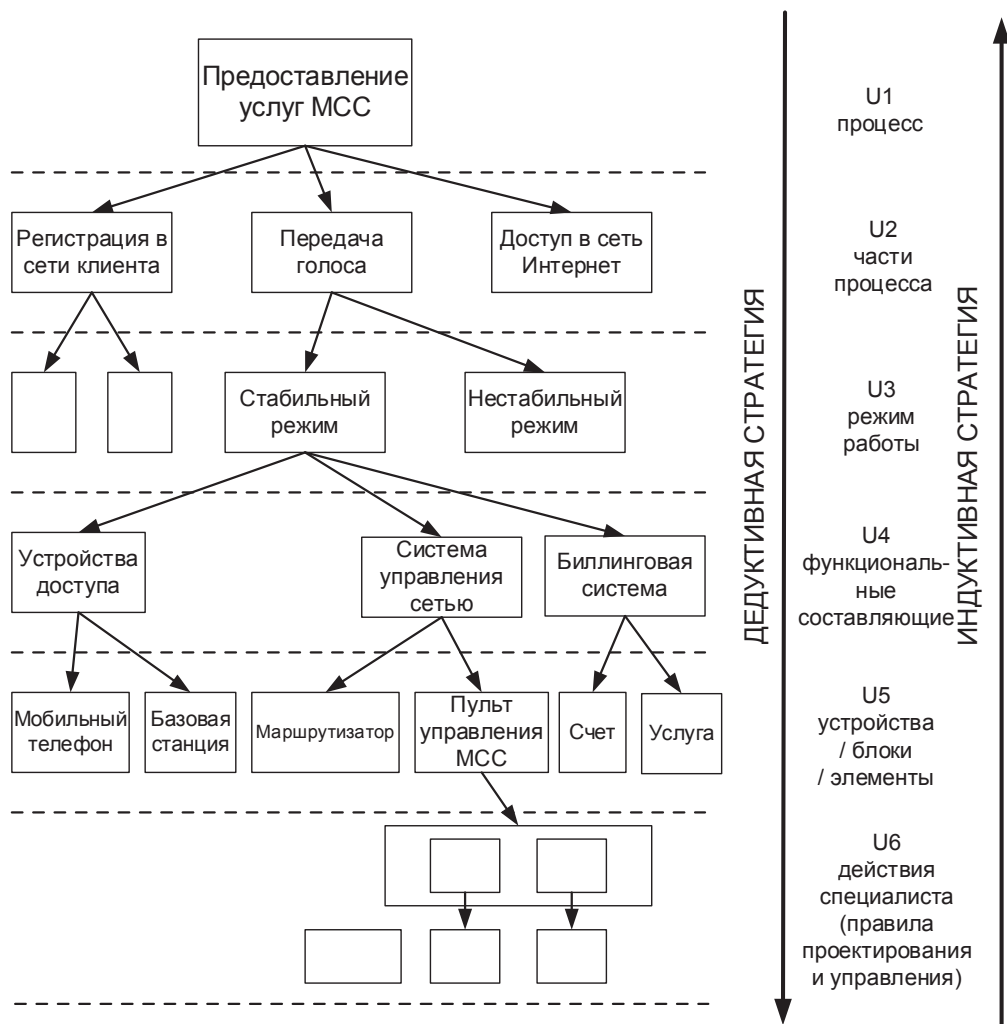


Рис. 1.2. Дуальная стратегия проектирования

## **1.2. Вопросы качества пользовательского интерфейса программного обеспечения**

---

### **1.2.7. Концепции качества интерфейса**

При разработке ПИ очень важно понимать, что хорошим ПИ считается:

- удобный, простой в использовании;
- эргономичный;
- интуитивно понятный;
- имеющий коммерческий успех интерфейс [8].

Однако такие критерии очень непрофессиональны, поскольку они строго не определены. Рассмотрим различные концепции качества ПИ, которые позволят получить более строгие критерии качества ПИ.

#### **Эргономические показатели**

Любой интерфейс имеет эргономические показатели качества, и исследователи постарались собрать их все в одну систему. Очевидно, что таких систем показателей можно получить достаточно большое количество. Наиболее распространенной является система показателей Шнейдермана, которая включает:

- скорость работы пользователя;
- количество человеческих ошибок;
- субъективную удовлетворенность;
- скорость обучения навыкам работы с интерфейсом;
- степень сохраняемости этих навыков при неиспользовании продукта.

Некоторые из этих показателей предметны и точны. К примеру, понятно, как сравнивать интерфейсы по скорости или количеству ошибок. Эти показатели позволяют четко сформулировать доработки ПИ, например, оптимизировать его по скорости работы пользователя.

В качестве недостатка показателей Шнейдермана можно указать конфликт показателей. На практике скорость работы пользователя с определенного значения почти всегда начинает конфликтовать со скоростью обучения, то есть проектирование быстрого

интерфейса приводит к уменьшению скорости обучения и обратно. Показатель сохраняемости навыков оперирования довольно расплывчат и не всегда актуален.

Обычно из четырех показателей Шнейдермана используют только два, например, скорость работы и количество операторских ошибок. Эта система показателей не дает рекомендаций, какие именно два показателя выбрать в каждом конкретном случае и по какой шкале их следует оценивать. Несмотря на недостатки показателей Шнейдермана, их можно использовать для общения с другими участниками проекта разработки и оценки ПИ.

### **Дизайн, ориентированный на пользователей**

Следующей концепцией качества интерфейса как по сложности, так и по времени своего появления является дизайн, ориентированный на пользователей (User Centered Design). Суть этой концепции в следующем: если хорошо изучить потенциальных пользователей (уровень их подготовки, знаний о предметной области и физиологических особенностей), а затем оптимизировать интерфейс под них, то такой ПИ будет хорошим. Следовательно, согласно этой концепции отношение пользователей к интерфейсу является главным показателем качества интерфейса.

К сожалению, немногие проекты могут позволить себе глубокие и дорогие исследования особенностей пользователей. Кроме того, пользователи способны адаптироваться к системе, поэтому непривычный ПИ через некоторое время может стать хорошим.

### **Дизайн, ориентированный на задачи пользователей**

Исследователи, занимающиеся вопросами качества ПИ, для устранения недостатков предыдущей концепции предложили дизайн, ориентированный на задачи пользователей (Task Centered Design). Согласно этой концепции, хорош тот интерфейс, в котором эффективно выполняются задачи пользователей. Под задачей понимается некоторая последовательность действий пользователя, которые в свою очередь состоят из операций.

Выше говорилось, что задачи могут быть решены несколькими способами. При разработке ПИ по этой концепции необходимо выбрать наиболее эффективное решение задачи и обеспечить ее выполнение с помощью ПИ.

Очевидно, что дизайн, ориентированный на задачи пользователей, включает в себя предыдущую концепцию. Также можно указать дополнительные достоинства:

- эффект от ПИ, разработанного по этой концепции, может быть оценен экономически. Если пользователь с помощью нового ПИ стал работать быстрее, то повысилась его производительность труда;
- разработку ПИ по этой концепции легче планировать, поскольку число задач пользователей конечно и более предсказуемо.

Но у дизайна, ориентированного на задачи пользователей, есть и заметный недостаток: такой ПИ не позволяет определить, какое именно число решаемых системой задач будет необходимым и достаточным, что существенно усложняет проект.

### **Дизайн, ориентированный на цели пользователей**

Недостатки предыдущей концепции позволяет решить следующая — дизайн, ориентированный на цели пользователей (Goal Directed Design). Согласно этой концепции, пользователи совершают определенные действия для достижения своих целей. Задача разработчиков ПИ — проанализировать цели пользователей, то есть конечные результаты их деятельности. Это поможет лучше понять, как реализовывать задачи пользователя в ПИ. Для этого следует составить список мотивов целевых пользователей и сравнить их со списком задач. Анализ поможет лучше оценить адекватность и полноту списка задач, а также упорядочить их по значимости.

На сегодняшний день дизайн, ориентированный на цели пользователей, позволяет на основе системного подхода создавать ПИ, удовлетворяющий пользователей.

### **Юзабилити**

Рассмотренные выше подходы обладают общим недостатком — они уделяют внимание только одному аспекту интерфейса. При описании более общего подхода к качеству ПИ используют термин юзабилити.

Определение юзабилити (из стандарта ISO 9241–11):



- usability — the extent to which a product can be used by specified users to achieve specified goals with efficiency, effectiveness and satisfaction in a specified context of use;
- *юзабилити* — степень эффективности, трудоемкости и удовлетворенности, с которыми продукт может быть использован определенными пользователями при определенном контексте использования для достижения определенных целей и (или) мотивов.

Важным в юзабилити-подходе является акцент на среде, то есть определенном контексте использования ПИ. Очевидно, что среда, в которой пользователи взаимодействуют с системой, может значительно влиять на интерфейсное решение.

Как правило, при упоминании юзабилити также говорят об эвристических правилах Якоба Нильсена. Более подробно о них рассказано в разделе 2.5.

### **Концепция деятельности**

Концепция деятельности основана на психологической теории деятельности А. Н. Леонтьева. Суть концепции сводиться к следующему:

- 1) проводим анализ деятельности — логическую организацию информационного материала, действий и операций, ведущих к достижению цели;
- 2) на основе этой информации сначала составляется сценарий взаимодействия пользователя с системой, а затем прототип будущего интерфейса.

Задачи и подзадачи пользователя определяют интерфейсные окна и режимы взаимодействия с ними. Операции — способ выполнения задачи — отображаются в виде интерактивных графических элементов, предполагающих определенный способ взаимодействия с ними. Особое внимание уделяется тому, какую деятельность должна поддерживать разрабатываемая система.

Данная концепция не получила широкого практического применения из-за отсутствия формализованных методологий.

### 1.2.2. Рекомендации по использованию концепций

Для разработки хорошего интерфейса в самом начале следует сформулировать и записать, какие эргономические показатели являются важными для этого ПИ. В конце разработки следует проанализировать полученные результаты.

В процессе разработки необходимо периодически в определенной последовательности отвечать на вопросы, заранее сформулированные на основе рассмотренных концепций. Первая группа вопросов:

- 1) есть ли возможность ускорить взаимодействие пользователя с этим интерфейсом;
- 2) можно ли определить в этом интерфейсе места, которые способствуют совершению пользователем ошибок; если такие места есть, то постараться их изменить;
- 3) проанализировать возможность обучения навыкам оперирования интерфейсом; знания, необходимые пользователю для успешного взаимодействия с этим интерфейсом, — может ли пользователь получить эту информацию из интерфейса?

Эти три вопроса необходимо задавать себе по очереди. Если ответы показывают, что интерфейс нужно изменять, то после модернизации нужно задать эти вопросы снова. Если на все три вопроса удалось дать отрицательный ответ, то переходим к следующей группе вопросов:

- оптимизирован ли интерфейс с точки зрения информации о пользователях;
- удовлетворяет ли разработанный интерфейс известные цели пользователя;
- совместим ли этот интерфейс со средой, в которой работают пользователи;
- эффективно ли разработанный интерфейс решает задачи пользователей?

Такой алгоритм действий позволит разработать эффективный ПИ.

### 1.2.3. Оценка удобства использования пользовательского интерфейса

Существует целый ряд подходов, позволяющих оценить удобство пользовательского интерфейса. Чтобы выявить проблемы в разработанном пользовательском интерфейсе, необязательно ждать его внедрения и тестирования. На разных этапах можно применить разные методики оценки.

Юзабилити-контроль — оценка качества ПО по характеристикам юзабилити. При эвристическом контроле требуется некоторое количество экспертов (юзабилити-специалистов, HCI-специалистов), которые оценивают интерфейс по некоторому набору критериев, например, эвристик Нильсена. Такой контроль позволяет найти основные проблемы, с которыми, скорее всего, столкнется пользователь. В качестве недостатка этой методики можно указать значительные затраты на привлечение достаточного количества экспертов. Также до завершения разработки можно оценивать, насколько интерфейс удовлетворяет целям пользователей. Для этого периодически необходимо отвечать на два вопроса:

- 1) знает ли пользователь, что делать на следующем шаге;
- 2) если знает, то приближается ли он к решению своей цели?

На более поздних этапах разработки целесообразно применять юзабилити-тестирование. Юзабилити-тестирование — измерение качества разработанной ИС по характеристикам юзабилити. Все методы можно разбить на две большие группы: методы непосредственно тестирования интерфейса группой пользователей и методы без тестирования, основанные на формальных расчетах. Все эти методы одинаково применимы как для оценки интерфейса традиционных ИС, так и web-приложений.

Выбор группы методов зависит, в основном, от того, насколько осуществимо непосредственное тестирование на той или иной стадии выполнения проекта, и от количества материальных и временных ресурсов, которые предусмотрены на тестирование. При этом следует учитывать не только стоимость самого проектирования и разработки качественного и удобного пользовательского интерфейса, но и возможных финансовых потерь из-за недостаточной проработанности ПИ или неудовлетворенности пользователя при его эксплуатации.

Перечислим методы непосредственного тестирования интерфейса группой пользователей:

- экспертная оценка;
- коридорный тест;
- тест пяти секунд;
- карточная сортировка;
- анкетирование;
- удаленное тестирование;
- А/В тестирование.

*Коридорный тест* позволяет оценить конкретные интерфейсные решения с помощью своих коллег. Для этого необходимо нарисовать или напечатать макет ПИ и опросить находящихся рядом людей. Они помогут определить, понятно ли спроектированы пункты меню (ясно ли, как нужно решать задачу пользователя) или содержит ли диалоговая форма всю необходимую информацию для выполнения операции. Конечно, при этом опрашивается нецелевая аудитория вашей ИС, но такое тестирование позволяет оценить, понятна ли ваша модель представления другим.

*Тест пяти секунд* позволяет проверить, что именно увидел или запомнил человек после просмотра макета формы (страницы) в течение пяти секунд. Таким образом можно оценить, позволяет ли пользователю предлагаемое интерфейсное решение быстро понять, сможет ли он достичь своей цели.

*Карточный тест* используется для оценки информационной структуры сайта или структуры главного меню ИС. Для его проведения необходимо подготовить бумажные карточки с названиями пунктов существующей информационной структуры. В одном случае участникам теста предлагается разложить карточки в группы и дать каждой группе общее название, в другом случае участники получают карточки с названиями пунктов меню или материалов и готовый список основных групп, в которые нужно поместить выданные карточки, то есть либо полностью составить информационную структуру, либо найти в существующей место для новой информации. Для контроля работы следует попросить участников сортировки в конце теста найти информацию в той структуре, которую они сами создали. Главное достоинство этого метода — построение информационной структуры сайта или структуры главного меню ИС самими пользователями, а не раз-

работчиками на основании своих предположений. При сложной структуре или специфической тематике приложения применять этот метод нецелесообразно.

*Анкетирование* может быть использовано на всех этапах жизненного цикла ИС: начиная от анализа потребностей пользователей и заканчивая дополнением к юзабилити-тестированию. Следует подготовить перечень вопросов, которые позволят понять требования к ИС, либо список прилагательных, которыми пользователи могли бы охарактеризовать свое впечатление от ИС.

*А/В-тестирование* позволяет оценивать количественные показатели работы двух вариантов ПИ, а также сравнивать их между собой. Для его проведения необходимо:

- выбрать нужные характеристики для оценки существующего варианта ПИ;
- разработать новый вариант ПИ, который будет повышать выбранные характеристики;
- проверить результаты.

*Удаленное тестирование* может быть модерируемым и немодерируемым. Оно позволяет привлечь к тестированию необходимую целевую аудиторию, причем эти люди территориально могут находиться в любом месте. При модерируемом тестировании исследователь непосредственно следит за действиями пользователя и может получить от него обратную связь. При немодерируемом тестировании для сбора информации о процессе используется специальный инструментарий.

### **1.3. Технология проектирования архитектуры программного обеспечения и пользовательского интерфейса**

---

Проектирование ПИ нужно рассматривать как одну из составляющих процесса проектирования ИС. Существует несколько различных технологий проектирования программного обеспечения: Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), гибкие технологии (Agile). Вне зависимости от выбора технологии проектирования ПО разработка начинается с обследования предметной области. На основании результатов обследо-

дования строится описание (модель) бизнес-процессов верхнего уровня на языке IDEF0. Для детализации бизнес-процессов нижнего уровня используются стандарты моделирования IDEF0, IDEF3. При проектировании информационной системы большой интерес представляет анализ информационных потоков, который удобно проводить с использованием методологии DFD (нотация Гейна—Сарсона).

Рассмотрим технологию проектирования ПИ для ИС «Расписание» при использовании CASE-средства BPsim.SD.

Диаграмма DFD, описывающая процесс преобразования информации от ее ввода в систему до выдачи потребителю, демонстрирует отношения между процессами преобразования информации. Она представлена на рис. 1.3.

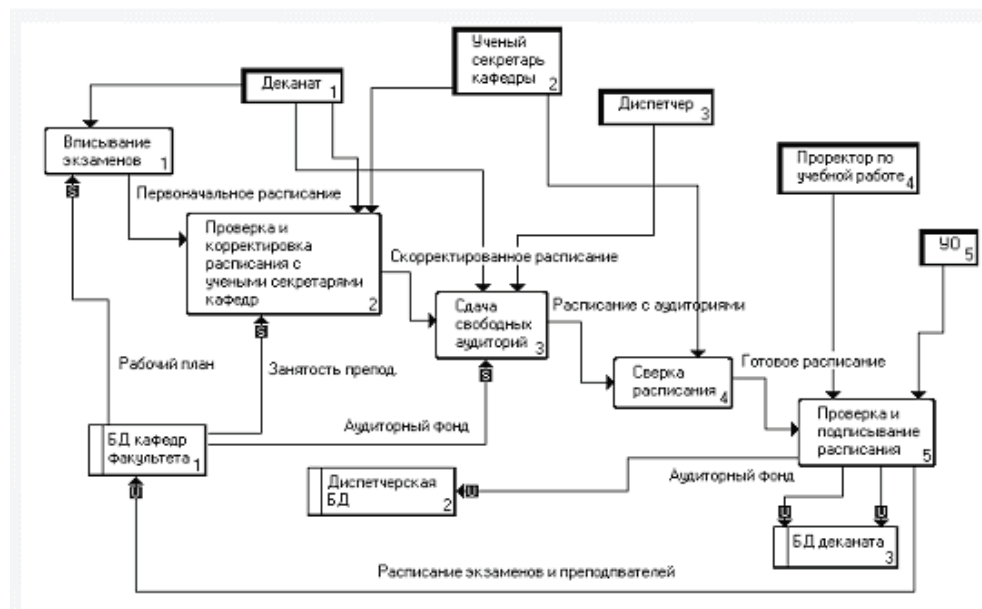


Рис. 1.3. Процесс формирования расписания экзаменов в BPsim.SD

Диаграммы, представляющие результаты проведенного структурного анализа, являются основой для объектно-ориентированного проектирования. Использование объектно-ориентированного подхода повышает уровень унификации разработки и пригодность для повторного использования. Информационные системы становятся более компактными, что означает не только уменьше-

ние объема программного кода, но и удешевление проекта за счет использования предыдущих разработок. Примером метода объектно-ориентированного анализа и проектирования является язык UML, представляющий собой набор диаграмм (диаграммы прецедентов, последовательности, классов и т. д.).

Переход от диаграмм стандартов IDEF0, IDEF3 и DFD (см. рис. 1.3) к диаграмме прецедентов можно осуществлять в автоматическом (рис. 1.4) или полуавтоматическом (рис. 1.5) режиме. При этом каждой функции на этих диаграммах ставится в соответствие прецедент, каждой внешней сущности (DFD) или механизму (IDEF0, IDEF3) — пользователь (актер, действующее лицо) [15].

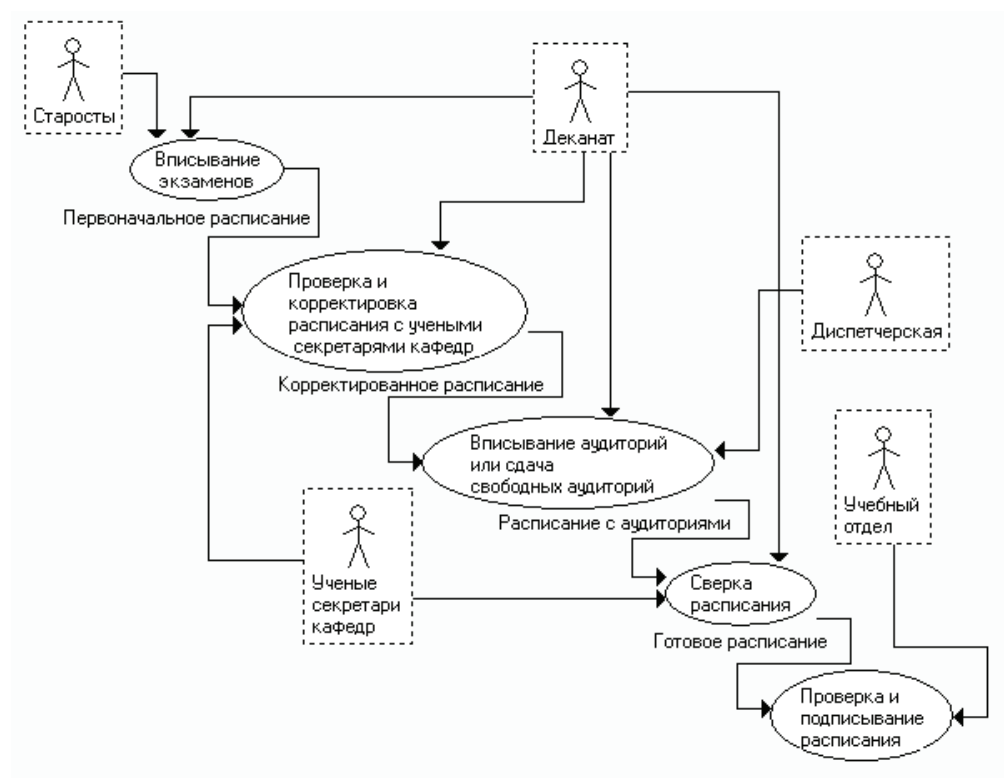


Рис. 1.4. Диаграмма прецедентов процесса формирования расписания экзаменов в BPsim.SD

Полуавтоматический режим позволяет автоматизировать проектирование больших и сложных диаграмм, выделить из них от-

дельные функциональные модули или рассмотреть работу отдельных пользователей с информационной системой. Пример диаграммы прецедентов для пользователя «деканат» (сотрудник деканата) приведен на рис. 1.5.

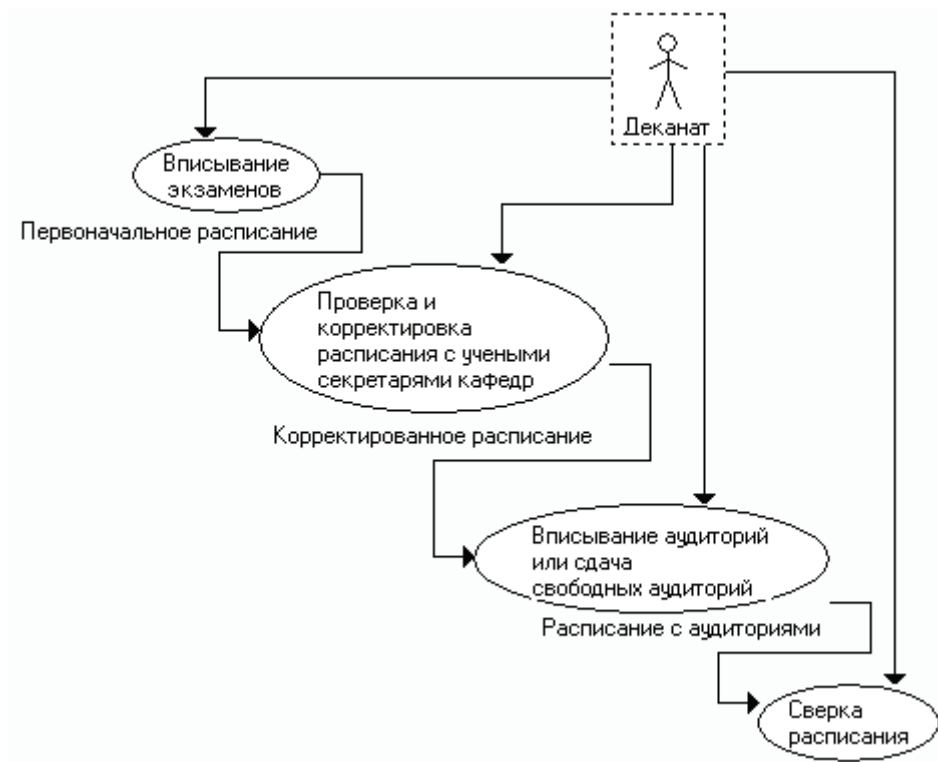


Рис. 1.5. Диаграмма прецедентов пользователя «деканат» процесса формирования расписания экзаменов в BPsim.SD

Диаграммы последовательности отражают временную последовательность событий, происходящих в рамках определенного варианта использования. На диаграмме отображаются ряд объектов (пользователей, оконных форм) и те сообщения, которыми они обмениваются между собой. Для каждого варианта использования диаграммы прецедентов (рис. 1.4 и 1.5) с помощью диаграмм последовательностей описывается порядок выполнения операций и смены экранных форм информационной системы. На рис. 1.6 приведена диаграмма последовательности для варианта использования «Вписывание экзаменов».



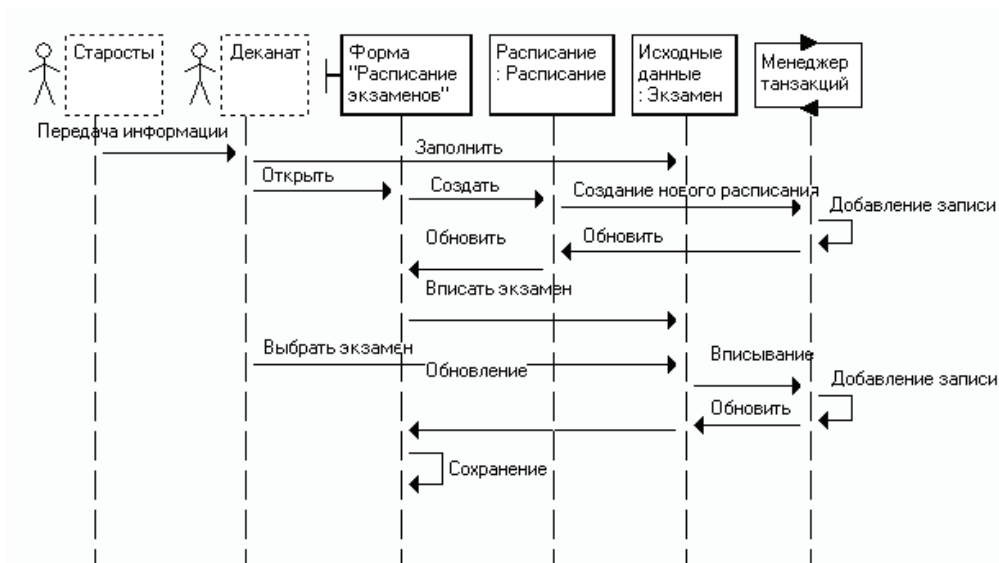


Рис. 1.6. Диаграмма последовательности выполнения операций вписывания экзаменов в расписание

Для каждого объекта диаграммы последовательности определяется класс, описываются атрибуты, типы, связи, которые существуют между ними, операции и ограничения, которые накладываются на связи между этими классами. При проектировании пользовательского интерфейса разрабатываемой системы каждый объект диаграммы последовательности привязывается к одному из четырех типов пакетов:

- бизнес-объекты;
- управление — управляющие объекты;
- границы — интерфейсы взаимодействия между подсистемами и графические интерфейсы;
- актеры.

Система проектирования пользовательского интерфейса позволяет по существующим диаграммам последовательности и классам спроектировать внешний вид разрабатываемой информационной системы.

За основу берется объект — граница, содержащий список экранных форм, необходимых конкретному пользователю (или нескольким пользователям) для выполнения рассматриваемой задачи (варианта использования информационной системы).

В соответствии с типами и другими свойствами бизнес-объектов, описанных в привязанном к данному объекту классе, выбираются наиболее подходящие компоненты для отображения соответствующих данных на формах информационной системы. Разработчику интерфейса информационной системы предлагается для каждого атрибута бизнес-объекта выбрать из предложенных системой проектирования нужный компонент и разместить его на форме. После создания и описания форм каждого прецедента получаем проект ПИ разрабатываемой информационной системы, который впоследствии может быть изменен. В рассмотренном нами примере для бизнес-объекта «Расписание» создан класс «Расписание» (рис. 1.7), для которого определены необходимые атрибуты и выполняемые операции.

Проектирование формы ввода расписания заключается в размещении пользователем на шаблоне формы компонентов, необходимых для отображения всех атрибутов, связанных с данной формой классов. На рис. 1.8 приведен пример спроектированной формы ввода расписания.

В дальнейшем спроектированные формы дорабатываются программистами. На рис. 1.9 приведен пример разработанной по описанному выше проекту вкладки ИС формирования расписания экзаменов.

Таким образом, в рамках процесса проектирования интерфейса будущей ИС BPsim.SD предлагает пользователю ряд возможностей:

1) описание бизнес-процессов, автоматизируемых создаваемым ПО, с помощью диаграмм стандарта DFD. Диаграммы DFD декомпозируемы до любого уровня детализации;

2) описание функций, выполняемых пользователями системы в рамках автоматизируемых процессов, с помощью диаграмм прецедентов. Допускается построение диаграмм прецедентов с «нуля», то есть создание новой диаграммы, или с помощью автоматизированного конвертирования из DFD, где выбранным из списка существующих на диаграмме DFD внешним сущностям ставятся в соответствие актеры, а функциям — прецеденты. Полученные конвертированием диаграммы редактируемы. Допускается строить неограниченное количество диаграмм прецедентов для каждой диаграммы DFD;

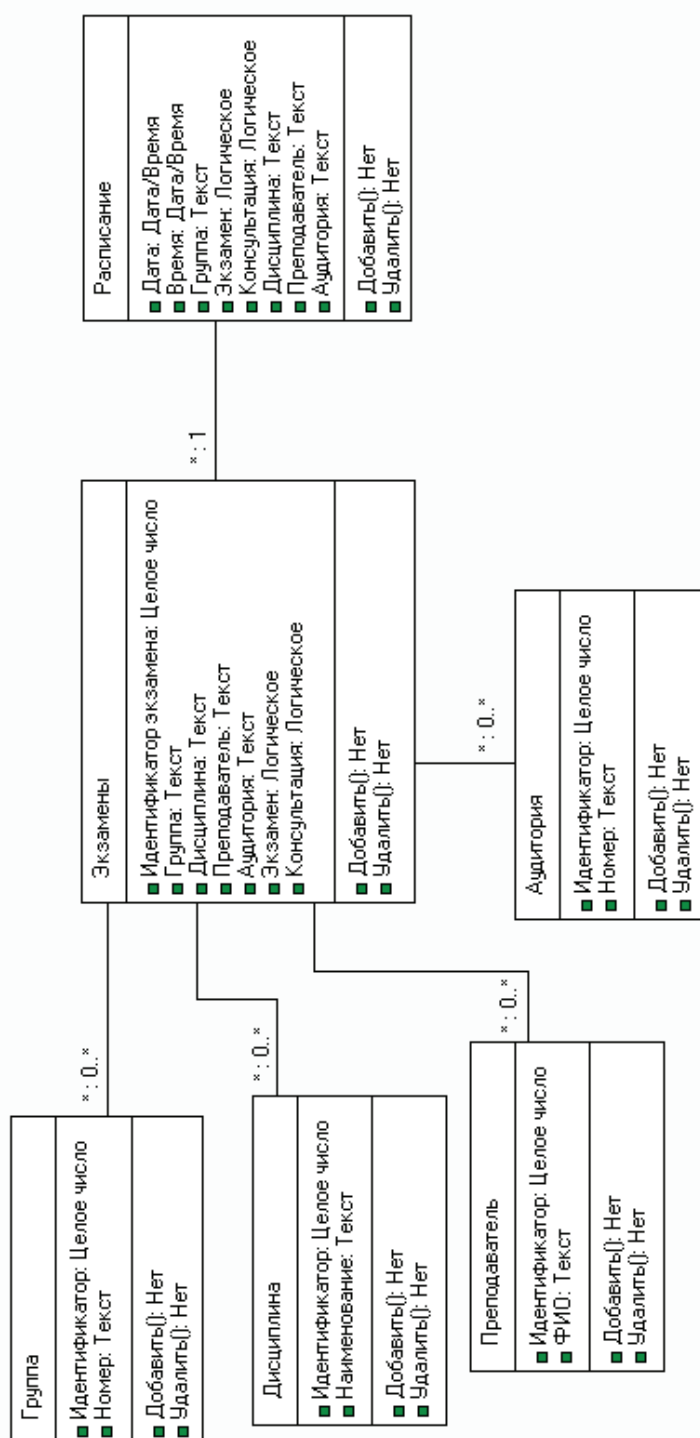


Рис. 1.7. Диаграмма классов в BPsim.SD

Расписание экзаменов

Дата начала  Дата окончания

Дата_нач	...	...	...	...	...	Дата_окон
Расписание	Расписание	Расписание	Расписание	Расписание	Расписание	Расписание

Группа	Дисциплина	Преподаватель	Аудитория	Экзамен	Консульт	Дата	Время
Экзамены.Г	Экзамены.Д	Экзамены.П	Экзамены.А	Экзамены.З	Экзамены.З		

☒ Экзамен/Консультация

Рис. 1.8. Спроектированная форма ввода расписания экзаменов

Расписание

База | Данные для расписания уч. занятий | Расписание занятий | Расписание экзаменов

Начало сессии: 12.01.2006 | Конец сессии: 23.01.2006 | Курс: 1 |

	12.01.2006 чт	13.01.2006 пт	14.01.2006 сб	15.01.2006 вс	16.01.2006 пн	17.01.2006 вт	18.01.2006 ср	19.01.2006 чт	20.01.2006 пт	21.01.2006 сб	22.01.2006 вс
X-15011					конс. 16:00 Р-116	Физика доц. Павлов О. Н. 09:00 Мт-325					
X-15031	Отечественная история асс. Ангина Е. Д. 12:00		Высшая математика асс. Мухомов А. В. 09:00								
X-15032	конс. 16:00 Мт-309	Отечественная история асс. Ангина Е. Д. 09:00 Мт-314									
X-15041											

№ группы | Предмет | Преподаватель | № аудитории | экзамен | конс.

X-15011	Отечественная история			+	+
X-15011	Математика			+	+
X-15011	Общая и неорганическая химия			+	+
X-15031	Отечественная история			+	+
X-15031	Высшая математика			+	+
X-15031	Общая и неорганическая химия			+	+
X-15031	Механика			+	+
X-15032	Высшая математика			+	+
X-15032	Теоретическая механика			+	+
X-15032	Химия			+	+

экзамен:  Время: 09:00

Рис. 1.9. Главное окно ИС формирования расписания экзаменов

3) для каждого прецедента, выполняемого пользователем, может быть дано описание последовательности элементарных операций в системе. Для этого предназначены диаграммы последовательности, которые могут быть созданы для любой функции диаграммы прецедентов. Система позволяет частично автоматизировать этот процесс: разработчику предлагается перенести на диаграмму последовательности выбранных актеров с диаграммы прецедентов. Для каждого объекта диаграммы последовательности определяется один из четырех типов по принадлежности к определенному пакету (границы, актеры, управление, бизнес-объекты);

4) создание диаграммы классов и сопоставление объектов диаграммы последовательности (кроме границ) с классами этой диаграммы;

5) проектирование визуальных форм моделируемого ПО (границ диаграммы последовательности): размещение на форме компонентов, привязка к компонентам методов и свойств классов и сохранение моделей форм в виде двух файлов: файл формы с расширением.dfm и файл программного модуля с расширением.pas. В дальнейшем данные модели могут быть переданы программисту для импорта в среду Delphi и дальнейшей проработки алгоритмов;

6) формирование отчетов о созданном проекте с изображениями спроектированных диаграмм и форм. Предусмотрен вывод отчетов на печать и экспорт в Word;

7) сохранение проекта на сервере и загрузка с сервера MS SQL Server для редактирования. При сохранении проекта пользователь присваивает ему уникальное имя, по которому впоследствии и осуществляется загрузка проекта.

---

## Глава 2.

# Разработка интерфейса программного обеспечения

---

### 2.1. Проектирование и создание прототипа пользовательского интерфейса приложения

---

**Н**а начальном этапе разработки необходимо решить следующие задачи:

- 1) выбрать и описать тему разрабатываемого приложения;
- 2) определить функции, которые будет выполнять приложение;
- 3) разработать диаграмму вариантов использования UML для приложения по выбранной тематике;
- 4) построить сценарии вариантов;
- 5) создать эскизы экранных форм.

#### Описание вариантов использования разрабатываемого предложения

Диаграмма вариантов использования языка UML (Unified Modeling Language) может быть применена для формализации функциональных требований к системе, в том числе для описания взаимодействия пользователей с проектируемой системой. Следует отметить, что изобразительных средств этой диаграммы недостаточно для подробного описания требований, поэтому ДВИ дополняют текстовыми сценариями. С их помощью можно уточнить или детализировать последовательность действий, совершаемых системой при выполнении ее вариантов использования.

Вариант использования представляет собой последовательность действий, выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом) [3].

На диаграмме вариантов использования языка UML рисуются актеры, варианты использования и связи (стрелки), которые их

соединяют. Пример простейшей диаграммы вариантов использования приведен на рис. 2.1.

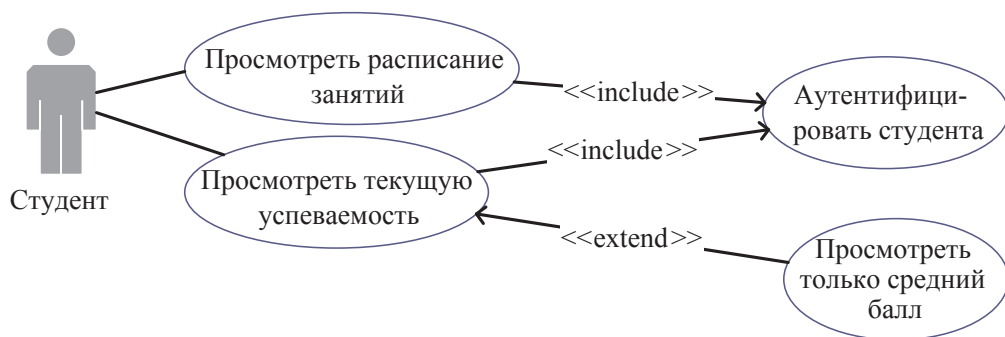


Рис. 2.1. Пример простейшей диаграммы вариантов использования

Актер (действующее лицо) — это роль, которую пользователь играет по отношению к системе [3]. При работе с программной системой все пользователи делятся на определенные группы. Каждая группа имеет определенные права, и может выполнять только разрешенные ей функции. Каждая такая группа описывается определенной ролью, например, клиент, администратор, оператор.

Связи, применяемые в диаграмме вариантов использования, могут быть четырех типов [2]:

1) связь ассоциации — связь между вариантом использования и актером, показывающая возможность использования актером прецедента (сплошная линия);

2) связь включения — связь между двумя вариантами использования, которая указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования (пунктирная линия со стрелкой и словом *include*);

3) связь расширения — связь между двумя вариантами использования, которая указывает, что один из них может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования (пунктирная линия со стрелкой и словом *extend*);

4) связь обобщения — связь между элементами диаграммы, показывающая, что один элемент является частным случаем другого элемента.

На рис. 2.2 показан пример диаграммы вариантов использования со связью обобщения.

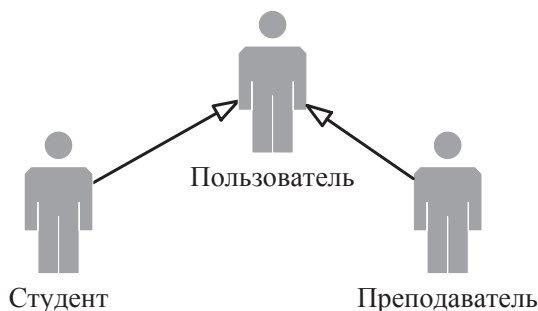


Рис. 2.2. Пример диаграммы вариантов использования со связью обобщения

Более подробная информация о диаграммах вариантов представлена в [1] и [2].

Следует отметить, что диаграммы вариантов использования не применяется для функциональной декомпозиции системы. Придумывать названия для вариантов использования следует с точки зрения пользователя, а не программиста.

При разработке прототипа пользовательского интерфейса следует дополнить каждую роль описанием по плану, приведенному ниже. Эта информация позволит предложить решения ПИ, которые наилучшим образом подойдут данной группе пользователей (например, ПИ для новичков и опытных пользователей будет отличаться). Возможно, с разрабатываемым вами ПИ будут работать люди с ограниченными возможностями. При описании роли не обязательно использовать все пункты этого плана:

- требуемые знания о предметной области, приложении, прочите знания;
- умения, то есть уровень мастерства в работе с системой;
- описание взаимодействия с приложением:
  - частота нахождения в данной роли;
  - регулярность;
  - непрерывность;
  - равномерность распределения работы во времени;
  - интенсивность (скорость взаимодействия);
- описание используемой информации:



- каковы источники входящей информации, предоставляемой пользователем;
- объем информации;
- сложность информации.
- критерии практичности интерфейса;
- функциональная поддержка: особые требования к функциональности;
- практический риск: оценка потенциальных убытков при работе с системой;
- аппаратные ограничения (если есть);
- среда (возможно условия, в которых будет работать пользователь, накладывают какие-то ограничения на интерфейсные решения).

Рассмотрим пример создания диаграммы вариантов использования UML с помощью среды разработки Visual Studio.

Заходим в среду разработки. В пункте меню выбираем «Архитектура» → «Создать схему...». В диалоговом окне «Добавление новой схемы» выбираем шаблон «Схема вариантов использования UML». Далее будет предложено создать проект модели — прописываем имя проекта и нажимаем ОК (рис. 2.3).

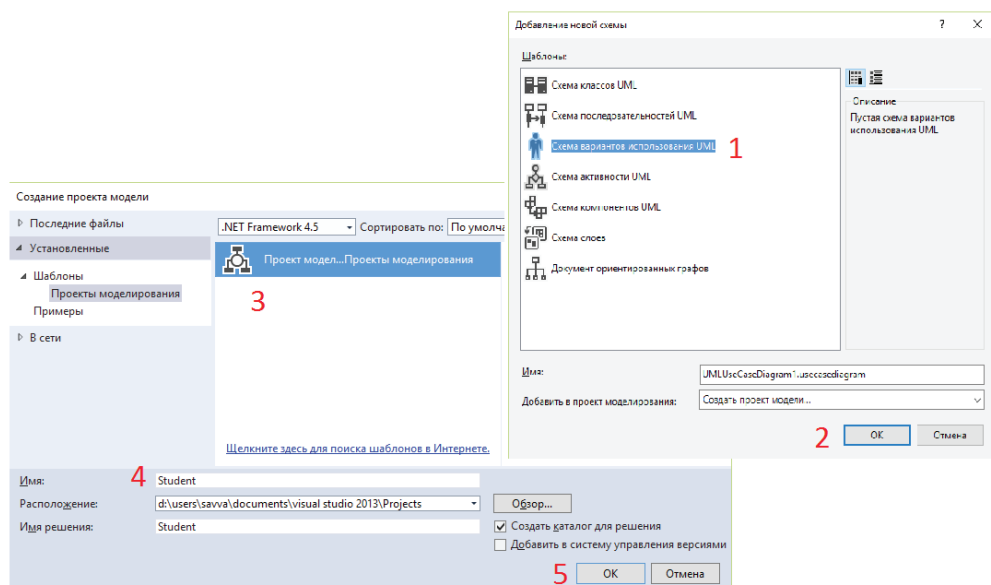


Рис. 2.3. Создание шаблона UML-диаграммы в Visual Studio

В появившемся окне необходимо открыть вкладку «Панель инструментов», где содержится весь необходимый инструментарий для проектирования приложений.

Сценарий варианта использования — текстовое описание, которое поясняет суть варианта использования. При проектировании его можно использовать для определения элементов интерфейса, реализующих отдельные действия сценария. Как правило, результат работы над сценарием оформляется в виде таблиц (см. табл. 2.1–2.3).

Таблица 2.1

### Аутентификация. Главный раздел

Вариант использования	Ввести логин и пароль
Роль	Пользователь
Цель	Зайти в систему под своей учетной записью
Краткое описание	Пользователь запускает приложение. В появившейся форме вводит логин и пароль, затем нажимает кнопку.
Тип	Базовый
Ссылки на другие варианты использования	Нет

После краткого описания варианта использования необходимо привести более детальную последовательность действий при типичном ходе событий (табл. 2.2).

Таблица 2.2

### Аутентификация. Раздел «Типичный ход событий»

Действия актера	Отклик системы	Элемент интерфейса
1. Пользователь запускает приложение	2. Приложение показывает на экране форму для ввода имени пользователя и пароля	Поле ввода логина — элемент TextBox. Поле ввода пароля — элемент MaskedTextBox. Кнопки «Вход» и «Отмена»
3. Пользователь вводит логин и пароль и нажимает кнопку «Вход»	4. Приложение закрывает форму авторизации и открывает главную форму приложения	
Исключение 1: неверно указаны логин или пароль	4.а. (см. Раздел «Исключения»)	

В табл. 2.3 описано исключение из типичного хода событий.

Таблица 2.3

**Аутентификация. Раздел «Исключения»**

Исключение 1	Действия актера	Отклик системы	Элемент интерфейса
Указаны неверно логин и (или) пароль	Пользователь вводит логин и пароль и нажимает кнопку «Вход»	4.а. На экране появляется сообщение «Логин или пароль введены неверно»	Стандартное диалоговое окно с текстом

Рассмотрим пример описания задания для системы тестирования. На рис. 2.4 показана диаграмма вариантов использования для этой системы.

Система тестирования нужна следующим заинтересованным лицам:

- обучаемому (студенту);
- составителю тестов (преподавателю);
- преподавателю, принимающему экзамен;
- сотруднику деканата, осуществляющему контроль за успеваемостью;
- администратору сети и баз данных учебного учреждения.

На начальном этапе создания системы мы можем ограничиться только двумя важными для нас ролями действующих лиц:

- студент (тестируемый);
- администратор (преподаватель, составитель тестов).

Соответственно, основными прецедентами (вариантами использования) для нашей системы являются следующие:

1) прецедент для студента:

- П1 — пройти тестирование;

2) прецеденты для администратора:

- П2 — создать/изменить тест;
- П3 — просмотреть результаты тестирования;
- П4 — добавить/изменить пользователей и др.

Краткая форма описания содержит название варианта использования, его цель, действующих лиц, тип варианта использования (базовый, второстепенный или дополнительный) (см. табл. 2.4).

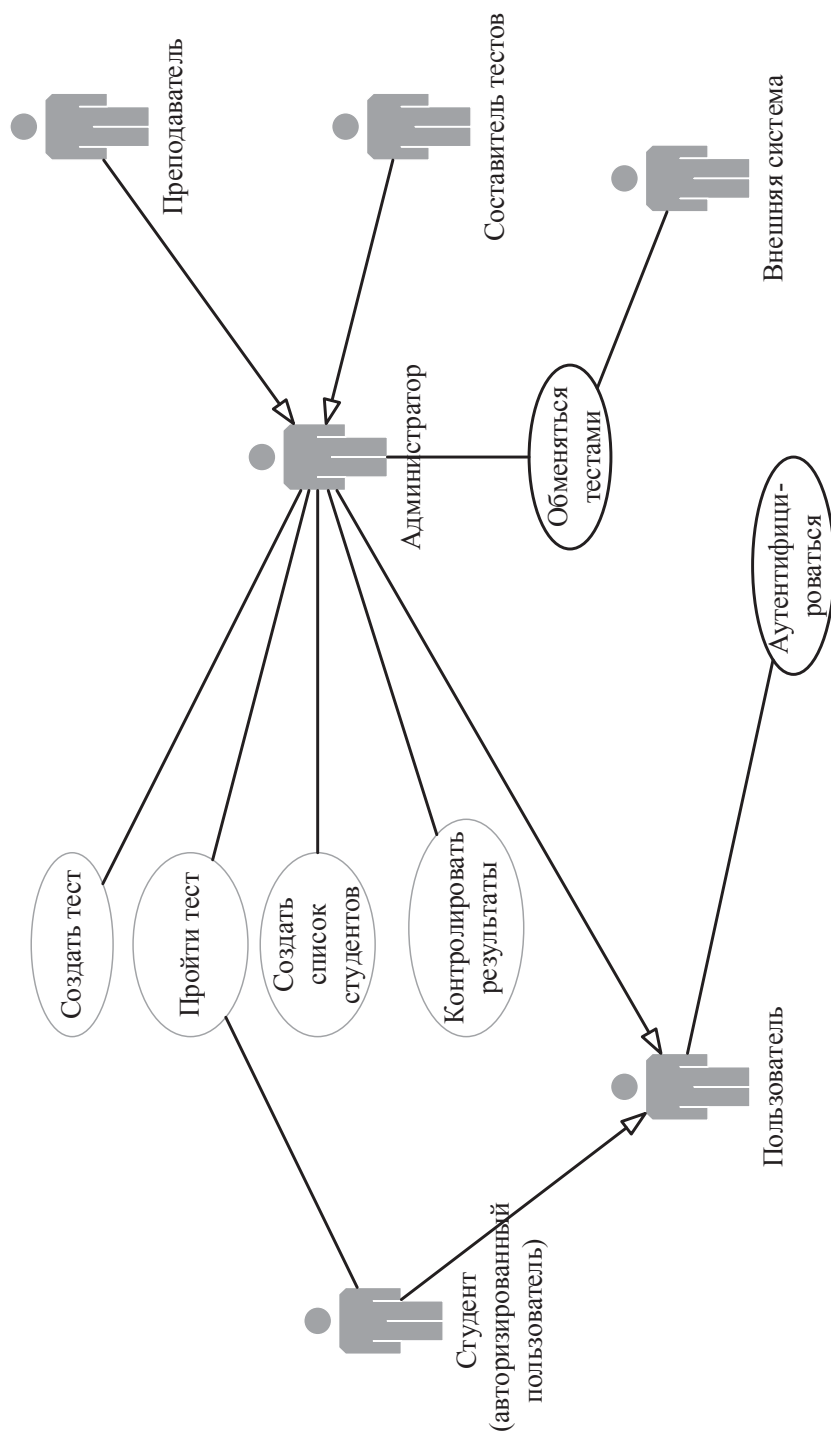


Рис. 2.4. Диаграмма вариантов использования для системы тестирования

Таблица 2.4

**Краткое описание варианта использования**

Название варианта использования	Прохождение тестирования
Роль	Студент
Цель	Прохождение тестирования, получение оценки
Краткое описание	Регистрация студента, запуск теста, выбор ответа или ввод ответа, завершение теста, получение оценки
Тип варианта	Базовый

Подробная форма описания варианта использования представлена в табл. 2.5.

Таблица 2.5

**Подробное описание варианта использования прохождения теста**

Действия актера	Отклик системы
Ввод своих данных (ФИО, группа) в текстовые поля формы, то есть регистрация в системе	Создание на диске файла с результатом тестирования и предложение выбрать тест
Выбор теста из списка	Запуск теста
Последовательные ответы на вопросы путем проставления отметок в специальных полях	Регистрация правильных и неправильных ответов
Завершение тестирования путем нажатия на кнопку «Завершить»	Подсчет процента правильных ответов
Ожидание результата	Демонстрация результата и предложение сохранить его
Сохранение результата или не сохранение результата путем нажатия на кнопку «Сохранить»	Если выбрано сохранение, система записывает результат в файл
Завершение работы путем нажатия на кнопку «Выход»	Завершение работы

**Описание задания**

Для закрепления материала студентам предлагается выполнить индивидуальные задания по проектированию и созданию прототипа ПИ. В начале работы необходимо определиться с тематикой разрабатываемого приложения. Выбранная тема должна быть интересна и понятна студенту. Разрабатываемое приложение должно обязательно содержать механизм аутентификации, главное меню, контекстное меню и табличное отображение информации.

После анализа выбранной темы и определения функций приложения необходимо разработать диаграмму вариантов использования языка UML.

Затем необходимо построить сценарии вариантов использования для более полного описания работы приложения. Сценарий варианта использования — текстовое описание, которое поясняет суть варианта использования, при разработке используется для определения элементов интерфейса, реализующих отдельные действия сценария. Результаты оформите в виде таблиц, примеры которых представлены в начале раздела 2.1. Для пояснения предлагаемых интерфейсных решений необходимо подготовить эскизы экранных форм. Эскизы могут быть сделаны вручную или с помощью программ, например, Visio.

## **2.2. Проектирование пользовательского интерфейса на этапе высокоуровневого проектирования**

---

*Высокоуровневое проектирование* — разработка на основе сценариев вариантов использования общей модели пользовательского интерфейса приложения с описанием структуры форм и переходов между ними.

Изучая пользовательские сценарии и диаграмму вариантов использования, можно выделить функциональные блоки приложения, а также операции, выполняемые пользователями в этих блоках, и объекты, над которыми эти операции делаются. После этого необходимо сгруппировать отдельные выделенные элементы, исходя из их логической связи. Таким образом, получается структура главного меню и других навигационных элементов (выпадающих меню, кнопок, инструментальных панелей).

При разработке меню следует придерживаться определенных стандартов — это ускорит обучение пользователя работе с новым приложением за счет использования им уже сформированных ранее привычек. Перечислим некоторые рекомендации (см. рис. 2.5):

- пункты меню, относящиеся к одной функциональной группе, следует отделять разделителем (черта или пустая строка);

- названия пунктов меню пишутся с большой буквы, желательно, чтобы название состояло максимум из двух слов;
- название меню, к которому относится каскадное меню, следует заканчивать стрелкой;
- в названиях меню можно использовать пиктограммы;
- недоступные пункты меню следует показывать другим цветом (обычно серым);
- недоступные пункты меню можно делать невидимыми.

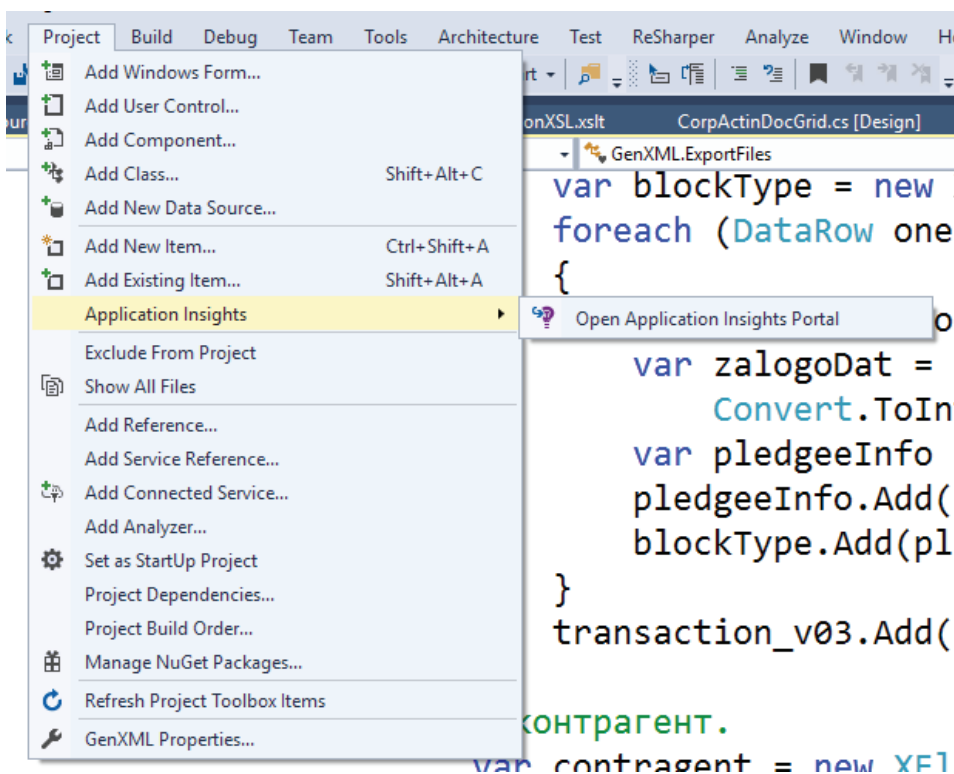


Рис. 2.5. Пример оформления пунктов меню

Для получения практических навыков по проектированию пользовательского интерфейса на этапе высокоуровневого проектирования необходимо решить следующие задачи:

- 1) ознакомиться с возможностями создания форм в Visual Studio C#;
- 2) разработать интерфейс приложения в соответствии с выбранной темой в разделе 2.1;

3) убедиться, что разработанный интерфейс не содержит типичных проблем, возникающих на этапе разработки прототипа ПИ (см. раздел 1.1.4).

Ниже приводится пример создания приложения Windows Forms в среде разработки Visual Studio 2015.

1. Запустить среду разработки Visual Studio.
2. Создать новое приложение, выбрав в меню File → New → Project. На экране появится следующая картинка (рис. 2.6).

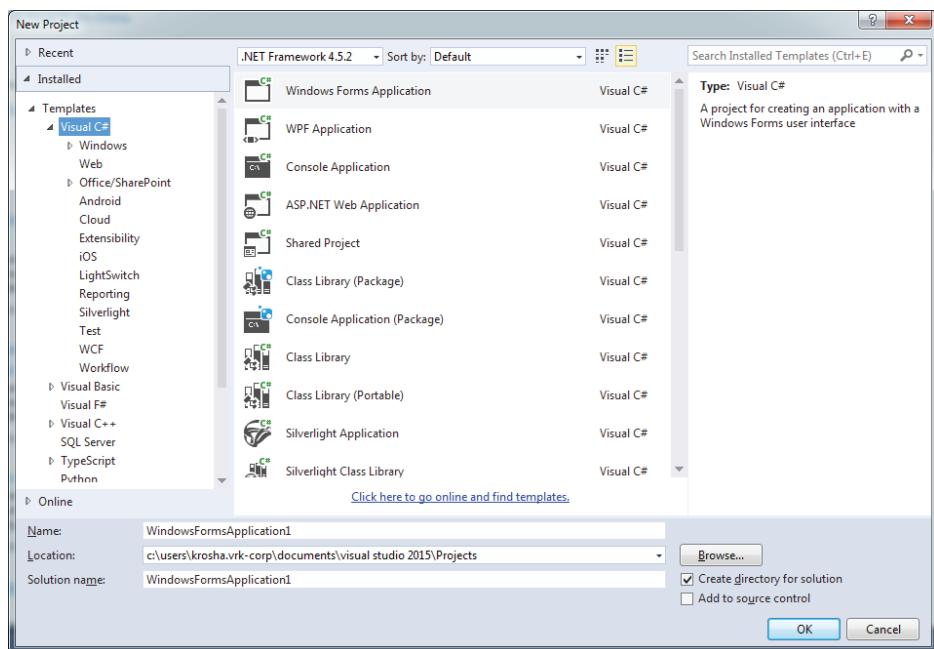


Рис. 2.6. Форма создания нового приложения

На форме необходимо выбрать шаблон (Template) — Visual C# → Windows → Windows Forms Application, а затем указать название приложения, место на диске, куда оно будет сохранено, и нажать ОК. После этого Visual Studio создаст и откроет проект (см. рис. 2.7).

Рабочее окно состоит из следующих частей:

- Toolbox («Панель инструментов») — располагается слева, содержит элементы управления, которые формируют пользовательский интерфейс (рис. 2.8). Вызвать ее также можно через меню View → Toolbox;



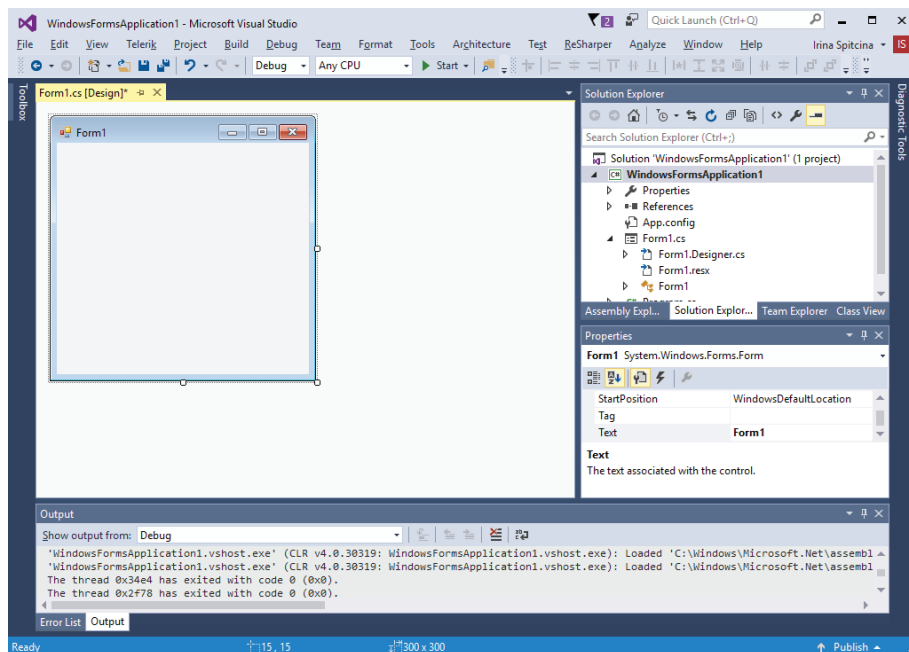


Рис. 2.7. Форма созданного приложения

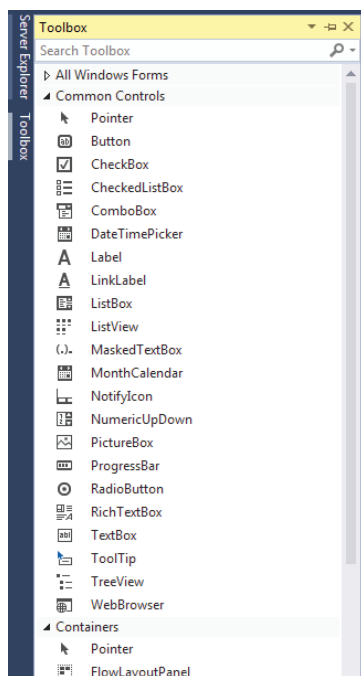


Рис. 2.8. Toolbox «Панель инструментов»

- окно Solution Explorer («Обозреватель решений») — располагается справа, содержит информацию о составе проекта: Properties — настройки проекта, Links — подключенные к проекту библиотеки, созданные и подключенные к проекту файлы исходных кодов (с расширением.cs) и подключенные к проекту формы (например Form1) см. рис. 2.9.

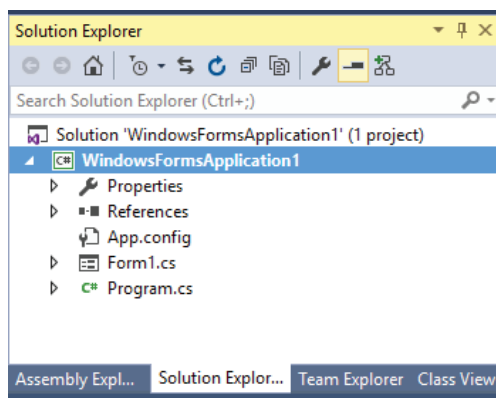


Рис. 2.9 Solution Explorer («Обозреватель решений»)

- окно Properties («Свойства») — окно, позволяющее просмотреть и настроить свойства и методы выбранного элемент управления или формы (см. рис. 2.10).

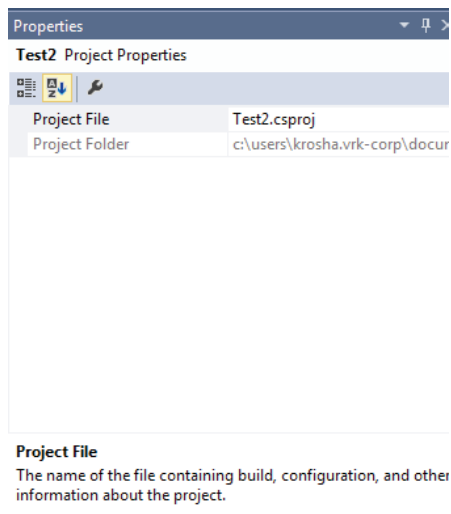


Рис. 2.10. Properties («Свойства»)

3. На основе выбранного шаблона Visual Studio появился каркас оконного приложения с главной формой Form1. Класс, который ее описывает, выглядит следующим образом (рис. 2.11):

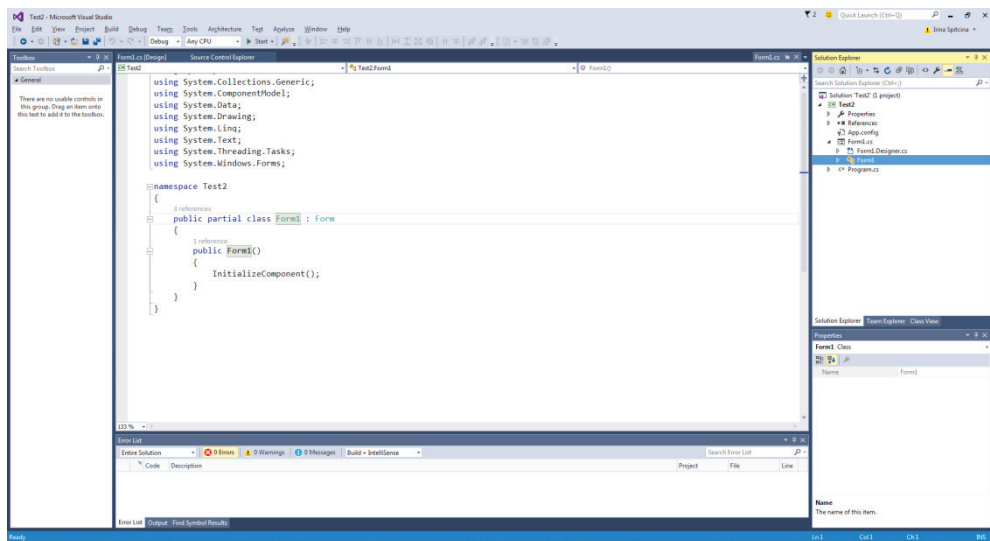


Рис. 2.11. Описание класса Form1

Существует несколько способов переключения между дизайном формы и описанием класса:

- 1) выбрать в Solution Explorer Form1 (описание класса) или Form1.cs (вид формы) (рис. 2.12);
- 2) выбрать в меню View → Code (F7) или View → Designer («Shift + F7»).

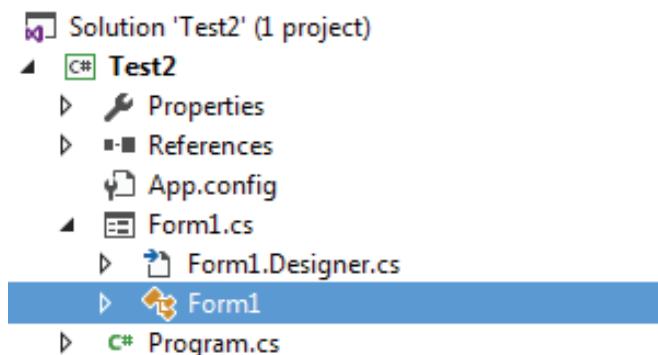


Рис. 2.12. Выбор в Solution Explorer описание класса Form1

Создание оконных приложений сводится к созданию всех необходимых диалоговых окон, а также к размещению на них необходимых элементов. В дальнейшем необходимо прописать обработку событий, создаваемых пользователем, и настроить технические аспекты работы программы.

4. Приведем пример размещения кнопки на форме и написания обработчика события Click (нажатие кнопки).

Найти на панели инструментов (Toolbox) нужный элемент Button («Кнопка») и переместить на форму (см. рис. 2.13).

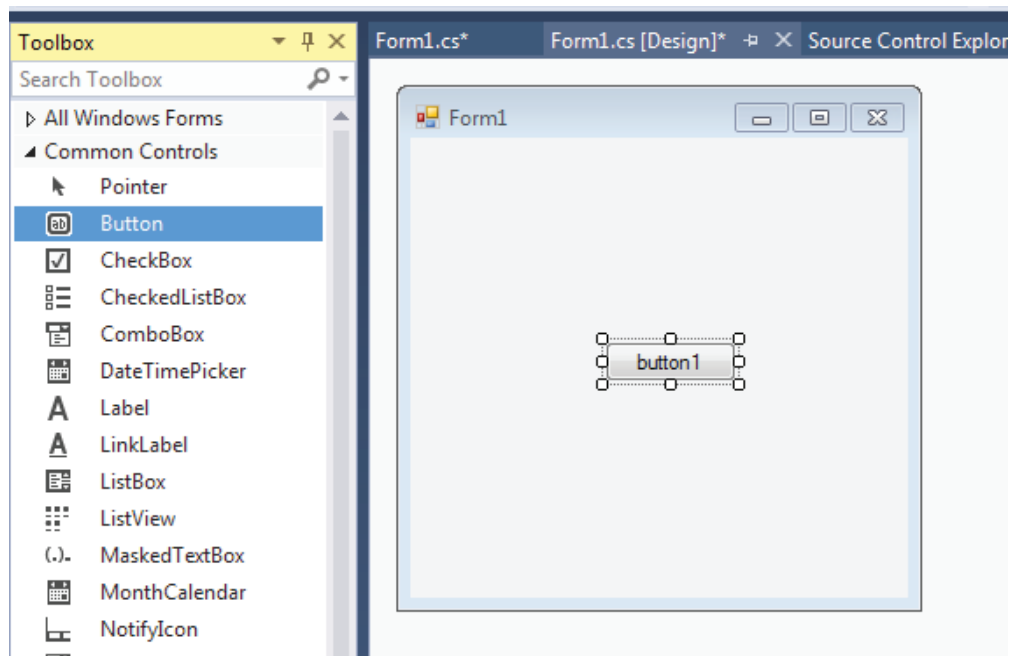



Рис. 2.13. Размещение кнопки на форме

В окне Properties («Свойства») найти свойство Text и изменить его значение на «Заккрыть» (рис. 2.14).

В окне Properties («Свойства») нужно переключиться на вкладку Event («События»), нажав кнопку , и найти событие Click (см. рис. 2.15).

Двойным щелчком мыши на событии создать его обработчик (см. рис. 2.16).

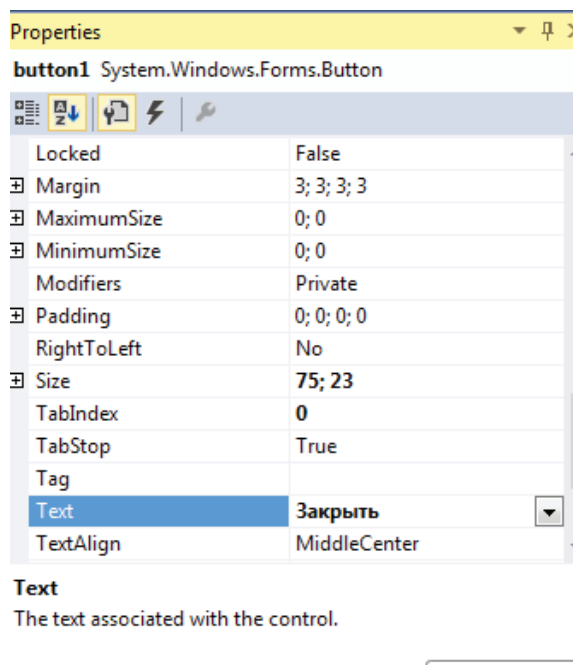


Рис. 2.14 Изменение свойства Text элемента управления button1

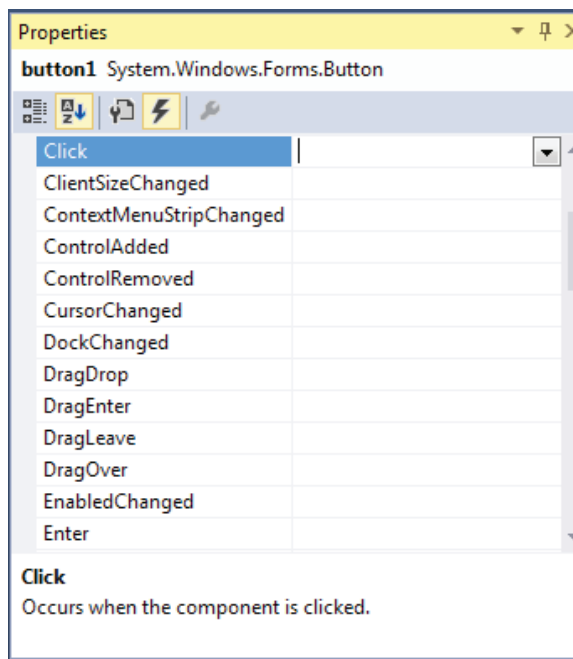


Рис. 2.15. Вкладка Event

```

namespace Test2
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
        }
        2
        1 reference
        private void button1_Click(object sender, EventArgs e)
        {
            //
        }
    }
}

```

Рис. 2.16. Обработчик события Click

Для закрытия формы вызовем метод Close ().

Запустим наше приложение клавишей F5 или кнопкой Start (см. рис. 2.17) и убедимся в работоспособности приложения.

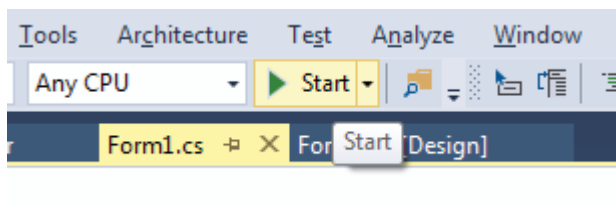


Рис. 2.17. Кнопка Start для запуска приложения

Изначально приложение по умолчанию создает одну форму и при запуске выводит ее. Когда же появляется несколько форм, необходимо осуществить переход между ними.

Создаем вторую форму: в «Обозревателе решений» нажимаем правой кнопкой мышки по проекту → «Добавить» → «Форма Windows» → ОК.

На первой форме Form1 создаем кнопку, как это было описано ранее. В конструкторе кликаем дважды по кнопке button1 (см. рис. 2.18), при этом автоматически создается событие button1\_Click.

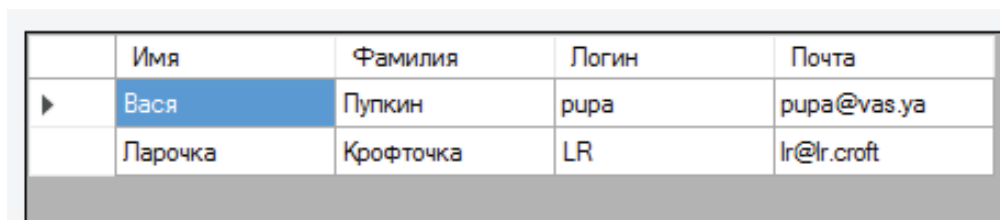
Для перехода необходимо создать экземпляр класса формы, которую мы хотим отобразить, и вызвать метод отображения.

```
private void button1_Click(object sender, EventArgs e)
{
    //создание экземпляра класса формы
    //на которую необходимо перейти
    Form2 form2 = new Form2();
    //вывести форму на экран
    form2.Show();
}
```

Рис. 2.18. Пример кода для открытия второй формы

В данном случае при запуске приложения и нажатии на кнопку button1 отобразится вторая форма, но обе формы будут активны, то есть работать можно будет и с первой формой, и со второй. Иногда требуется, чтобы при открытии второй формы первая блокировалась, и тогда вместо метода Show () должен вызываться метод ShowDialog ().

Для отображения табличных данных обычно используется компонент DataGridView (см. рис. 2.19).



	Имя	Фамилия	Логин	Почта
▶	Вася	Пупкин	pupa	pupa@vas.ya
	Ларочка	Крофточка	LR	lr@lr.croft

Рис. 2.19. DataGridView для таблицы «Студенты»

DataGridView на этом этапе лучше всего заполнить данными из конструктора формы (см. рис. 2.20).

Для закрепления полученных теоретических знаний студентам предлагается разработать интерфейс приложения в соответствии с выбранной и описанной темой в разделе 2.1. В результате должно получиться полностью оформленное приложение (не отработка функционала, а именно отображение всех объек-

тов, которые необходимы для решения поставленной задачи). Если в выбранной теме присутствуют какие-то табличные элементы, их необходимо оформить в виде `DataGridView` и заполнить набором статистических данных, которые предположительно там будут находиться.

ссылка 1

```
public Form1()
{
    InitializeComponent();
    int id = dataGridView1.Rows.Add();
    dataGridView1.Rows[id].Cells[0].Value = "Вася";
    dataGridView1.Rows[id].Cells[1].Value = "Пупкин";
    dataGridView1.Rows[id].Cells[2].Value = "пура";
    dataGridView1.Rows[id].Cells[3].Value = "пура@vas.ya";

    id = dataGridView1.Rows.Add();
    dataGridView1.Rows[id].Cells[0].Value = "Ларочка";
    dataGridView1.Rows[id].Cells[1].Value = "Крофточка";
    dataGridView1.Rows[id].Cells[2].Value = "LR";
    dataGridView1.Rows[id].Cells[3].Value = "lr@lr.croft";
}
```

Рис. 2.20. Заполнение `DataGridView` данными

При разработке интерфейса обязательно надо предусмотреть следующее:

- меню как минимум с тремя пунктами, два из которых — «О программе ...» и «Выход» (рис. 2.21);

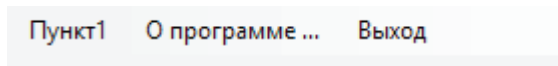


Рис. 2.21. Пример оформления меню

- не менее трех форм, с возможностью перехода между ними;
- в местах, где требуется список элементов (объектов), использовать контейнер `DataGridView` с контекстным меню (например, на удаление, добавление и изменение);
- аутентификация пользователей.



В приложении должно быть минимум две роли: простые авторизованные пользователи и администраторы. У администратора должна быть возможность просматривать списки зарегистрированных пользователей, оформленных в виде DataGridView, создавать нового пользователя, а также удалять уже существующих пользователей. Также администратор должен иметь возможность изменять пароли пользователей. Следует отметить, что на данном этапе необходимо спроектировать только интерфейсные формы для аутентификации и работы администратора. Их функционирование реализуется в разделе 2.4.

В конце работы следует убедиться, что разработанный прототип приложения:

- корректно отображается при разных условиях (размер экрана монитора, разрешение экрана);
- содержит корректно выбранные компоненты для ввода и отображения информации;
- содержит контекстные подсказки, контекстные меню;
- содержит информирующие и (или) предупреждающие сообщения для пользователей;
- имеет несколько возможностей вызова функций главного меню (панель инструментов, горячие клавиши).

Дополнительную информацию о разработке можно подчерпнуть из [9].

### **2.3. Разработка функций приложения, позволяющих взаимодействовать с папками и файлами**

---

Для получения практических навыков по разработке функций приложения, позволяющих взаимодействовать с папками и файлами, необходимо решить следующие задачи:

- 1) ознакомиться с классами, которые реализуют взаимодействия с папками и файлами;
- 2) доработать приложение, добавив функцию записи действий пользователя: основные действия пользователя и возникшие сообщения об ошибках должны сохраняться в лог-файле. Структура лог-файла разрабатывается студентами самостоятельно;

- 3) добавить функцию, которая позволит отображать дерево каталогов и файлов;
- 4) при размещении информации на форме использовать принцип золотого сечения.

В стандартном наборе компонента C# существуют не визуальные компоненты OpenFileDialog и SaveFileDialog, с помощью которых создаются диалоги открытия и сохранения файла.

Ниже перечислены основные свойства компонентов OpenFileDialog и SaveFileDialog:

- свойство AddExtension — разрешает или запрещает автоматическое добавление расширения, указанного в свойстве DefaultExt;
- свойство DefaultExt — расширение, принятое по умолчанию для автоматического добавления к имени файла при значении свойства AddExtension равным true;
- свойство CheckFileExists — используется для получения или установки значения, указывающего, отображать или нет диалоговое окно предупреждения, если пользователь указал в свойстве FileName имя файла, которого не существует в данной директории, и нажал кнопку «Открыть» при невыбранном кликом мышки файле. Если значение свойства CheckFileExists равно true, то вместо прерывания будет выдано сообщение, что такого файла нет, и исключения не вызывается;
- свойство FileName — имя файла по умолчанию для выборки, если была нажата кнопка ОК и не выбран кликом мышки файл в окне диалога;
- свойство CheckPatchExists — используется для получения или установки значения, указывающего, отображать или нет диалоговое окно предупреждения, если пользователь указал в свойстве FileName имя файла с несуществующим именем директории;
- свойства Filter, FilterIndex — фильтр для выбираемых файлов и индекс строки, отображаемой в выпадающем списке «Имя файла».

Например: значение свойства, заданного строкой Filter = “rtf файлы (\*.rtf)|\*.rtf|txt файлы (\*.txt)|\*.txt” и при FilterIndex = 1.

Данные настройки позволят установить выбор только текстовых файлов в формате rtf или txt. В выпадающем списке «Тип

файла» будут только две строки (рис. 2.22). По умолчанию будет установлен первый фильтр, то есть по rtf;

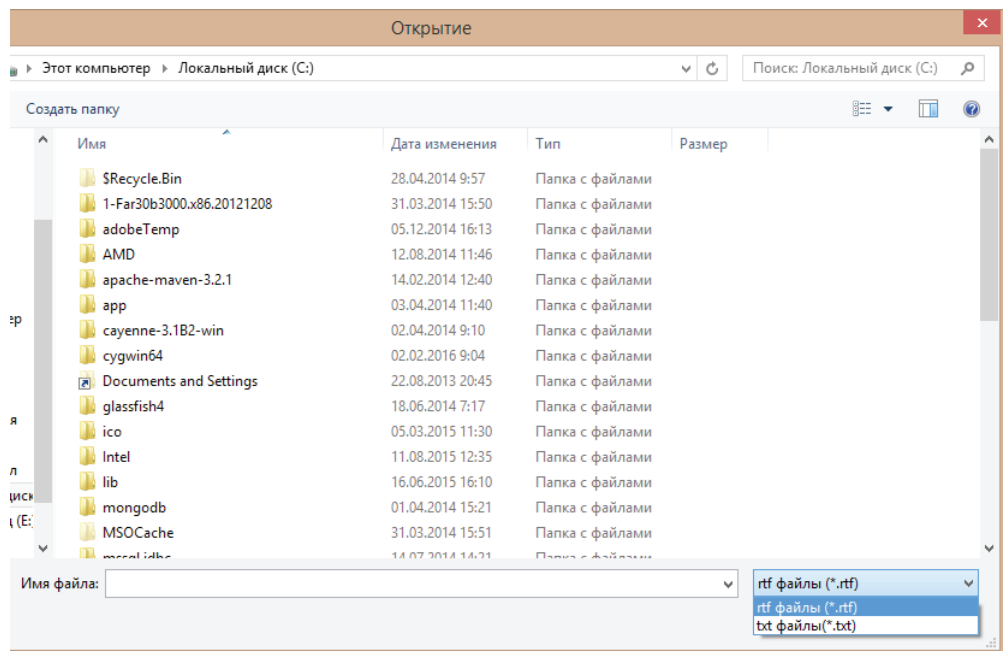


Рис. 2.22. Окно компонента OpenFileDialog с фильтром для выбираемых файлов

- свойство `InitialDirectory` — директория, которая выбирается при старте `OpenFileDialog`;
- свойство `MultiSelect` — при значении `true` позволяет выбрать мышкой при нажатой кнопке `Shift` или `Ctrl` несколько файлов и сохранить их имена в свойстве `FileNames` в виде массива строк;
- свойство `ReadOnlyChecked` — при значении `true` позволяет команде `OpenFile` открывать выбранные файлы только в режиме чтения;
- свойство `Title` — заголовок диалогового окна.

Основной метод компонент `OpenFileDialog` и `SaveFileDialog` — метод `ShowDialog()`, который позволяет открыть диалоговое окно.

Для работы с каталогами файлов существует компонент `FolderBrowserDialog`, который обеспечивает просмотр, создание, и вы-

бор рабочего каталога. Перечислим основные свойства компонента `FolderBrowserDialog`:

- свойство `RootFolder` задает одну из системных папок и смещает корень дерева просмотра на данную папку, делая недоступными для просмотра все другие папки, которые находятся выше выбранной. Так, задав в качестве свойства `Programs`, доступными для просмотра будут только папка меню «Программы» со всеми ее вложениями;
- свойство `SelectedPath` позволяет задать папку, отображаемую при старте как выбранную. Свойство имеет встроенный редактор (кнопку с тремя точками, появляющуюся при выборе свойства, нажатие на которую вызывает редактор «Обзор папок»), позволяющий обычным для Windows способом выбрать или создать папку;
- свойство `ShowNewFolderButton` разрешает показ кнопки «Создать новую папку» и создание папки из программы;
- свойство `Description` позволяет задать текст, который будет выведен над окном выбора папки.

Для визуального отображения структуры каталогов и файлов можно использовать компоненты `ListView` и `TreeView`.

Теперь кратко ознакомимся с классами для работы с файлами из пространства имен `System.IO`.

Класс `File` предоставляет статические методы для создания, копирования, удаления, перемещения и открытия одного файла, а также помогает при создании объектов класса `FileStream`.

Перечислим основные методы класса:

- `Delete (string)` — удаляет файл с указанным именем;
- `Exists (string)` — проверяет наличие файла в указанном каталоге, при наличии файла возвращает значение `true`;
- `Create (string)` или `Create (string, int)` — используется для создания файла с именем и в каталоге, определенным в строковом параметре, и с размером буфера, указанным во втором параметре. Методы возвращают объект типа `FileStream`. Исключение возникает, если диск закрыт для записи;
- `Open (string, FileMode, FileAccess, FileShare)` — открывает файл по указанному пути (первый параметр), второй параметр определяет необходимость создания файла, если он не существует, третий параметр — определяет допустимые

операции с файлом, четвертый параметр указывает опцию совместного использования. При открытии он создает объект типа `FileStream`. Обязательными параметрами являются только первые два;

- `CreateText (string)` — создает файл, который будет открыт для записи. Метод возвращает объект типа `StreamWriter`, который может быть использован для записи текста в файл. Исключение возникает, если диск закрыт для записи;
- `OpenText (string)` — используется для открытия существующего текстового файла и создания объекта типа `StreamReader`;
- `AppendText (string)` — создает объект типа `StreamWriter`, который можно использовать для добавления текста в кодировке UTF-8 в конец файла;
- `Copy (string, string, bool)` — копирует существующий файл, указанный в первом параметре, по новому адресу, указанному во втором параметре. При значении третьего параметра равном `false` перезапись файла при его наличии не происходит;
- `Move (string, string)` — переносит существующий файл, указанный в первом параметре, по новому адресу, указанному во втором параметре.

Класс `Path` выполняет операции для экземпляров класса `String`, содержащих сведения о пути к файлу или каталогу. Выполнение этих операций не зависит от платформы.

Перечислим основные методы класса:

- `GetFileName (string)` — возвращает имя файла вместе с расширением, но без пути.
- `GetFileNameWithoutExtension (string)` — возвращает имя файла без расширения и без пути;
- `GetExtension (string)` — возвращает расширение имени файла;
- `GetFullPath (string)` — возвращает полное имя файла;
- `GetDirectoryName (string)` — возвращает имя каталога для указанного файла.

Класс `Directory` предоставляет статические методы для создания, перемещения и перечисления в каталогах и вложенных каталогах.

Основные методы класса:

- `CreateDirectory (string)` — создает директорию с указанным названием;
- `Exists (string)` — проверяет существование директории с указанным названием.

Класс `FileInfo` — наследник абстрактного класса `FileSystemInfo` — используется для создания, копирования, удаления, перемещения и открытия файлов, и при использовании ряда методов создает объекты типа `FileStream`.

Класс `FileSystemInfo` содержит свойства и методы, общие для управления файлами и директориями.

Для закрепления теоретических знаний студент может выполнить как доработку возможностей уже разработанного ранее приложения (см. разделы 2.1 и 2.2), так и реализовать новое самостоятельное приложение.

Предлагается разработать функцию приложения, позволяющую просматривать содержимое директории и осуществлять переход по директориям, а также предоставлять полную информацию (всю, которая только имеется) о выделенном объекте (например, имя файла, дата создания, разрешения, владелец, расширение и т. д.).

Для любых текстовых документов и изображений реализовать вывод информации на экран. К примеру, при выделении текстового документа с расширением `.txt` в специальном окне должно отобразиться содержимое документа. После этого надо добавить в приложение настройку (с возможностью сохранения), которая позволит отображать выбранный файл в ассоциированном с ним приложении (например, файл с расширением `.doc` в приложении `Word`).

Затем нужно продумать структуру лог-файла приложения, сохранять все основные действия пользователя (выбор пунктов меню, нажатие на кнопки формы) и сообщения об ошибках в лог-файле.

Рассмотрим вопрос запуска стороннего приложения из своего для отображения выбранного файла в ассоциированном с ним приложении.

Метод `Process.Start (string)` запускает ресурс процесса путем указания имени документа или файла приложения и связывает ресурс с новым компонентом `Process`. Его можно использовать для открытия файла в ассоциированном приложении.

Определяя, как лучше разместить на форме элементы ПИ, можно использовать практические принципы построения ПИ:

- золотое сечение;
- «кошелек Миллера»;
- принцип группировки;
- «бритва Оккама» или KISS;
- «видимость отражает полезность»;
- «умное заимствование» [10].

Принцип *золотого сечения* был открыт еще в древности. Золотое сечение — это такое пропорциональное деление отрезка на неравные части, при котором весь отрезок так относится к большей части, как сама большая часть относится к меньшей. Если принять весь отрезок за единицу, то отношение его частей будет равно 1,618.

Золотой прямоугольник — это фигура с длинной стороной, которая равна короткой, умноженной на 1,618. Его можно использовать в макетах ПИ. Для этого необходимо узнать ширину и высоту формы, а затем поделить ее на части, пропорциональные золотому прямоугольнику. Далее его можно применять к кнопкам, панелям, изображениям и тексту.

Принцип *«кошелька Миллера»* говорит о том, что при группировке элементов ПИ (пунктов меню, кнопок, закладок и т. п.) следует учитывать возможность человека без усилий запоминать семь сущностей. Следовательно, в одной группе не должно быть больше  $7 \pm 2$  элемента.

Принцип *группировки* означает, что при размещении элементов ПИ на форме необходимо располагать их осмысленными группами (например, на рис. 2.21 (с. 54) пункты меню объединены в группы).

Принцип *«бритвы Оккама»* означает, что:

- любая функция приложения должна выполняться минимальным числом действий;
- логика этих действий должна быть очевидной для пользователя;
- движения курсора и глаз пользователя по экрану должны быть оптимизированы.

Принцип *«видимость отражает полезность»* состоит в том, что наиболее часто используемые элементы управления ПИ долж-



ны находиться на переднем плане и быть легко доступны пользователю — например, панели инструментов, которые меняются в зависимости от того, какие задачи решает пользователь.

Принцип *«умное заимствование»* означает, что при разработке своего ПИ следует использовать широко распространенные приемы дизайна, которые уже де-факто стали стандартом. Пользователю будет легче знакомиться с вашей разработкой.

## **2.4. Работа с методами сериализации и десериализации объектов**

---

Для получения практических навыков работы с методами сериализации и десериализации объектов необходимо решить следующие задачи:

- 1) ознакомиться с методами сериализации и десериализации объектов;
- 2) разработать приложение в соответствии с заданием, сохраняя сериализованные данные в файле;
- 3) реализовать механизм шифрования паролей пользователей с помощью алгоритма MD5;
- 4) провести анализ удобства выполнения одной функции разрабатываемого приложения с использованием закона Фитса.

Сериализация объекта — это процесс сохранения состояния объекта в последовательности байт. Десериализация — процесс обратного восстановления объекта из сохраненной последовательности байт. Сериализация и десериализация — удачные механизмы для передачи объектов.

Рассмотрим, как работает механизм сериализации. Пусть разработан некоторый класс `Students` (код этого класса приведен на рис. 2.23). Ключевое слово `Serializable` показывает, что класс `Students` — сериализуемый. Если есть поля, которые по какой-то причине нужно исключить из сериализации, их необходимо пометить ключевым словом `NonSerialized`.

Для наглядности создадим список элементов сериализованного класса `Students`, как показано на рис. 2.24.



```
[Serializable]
ссылка 3
class Students
{
    /**
     * Конструктор, создание экземпляра класс, с входными параметрами
     * */
    ссылка 0
    public Students(String lastname, String firstname, String middlename, DateTime birthday)
    {
        this.lastname = lastname;
        this.firstname = firstname;
        this.middlename = middlename;
        this.birthday = birthday;
    }

    /**
     * описание полей
     * */
    // Фамилия
    ссылка 1
    public String lastname { get; set; }
    // Имя
    ссылка 1
    public String firstname { get; set; }
    // Отчество
    ссылка 1
    public String middlename { get; set; }
    // День рождения
    ссылка 1
    public DateTime birthday { get; set; }
}
```

Рис. 2.23. Пример сериализуемого класса

```
/*Пример описание статичного списка студентов*/
0 references
public static List<Students> getList()
{
    List<Students> list_students = new List<Students>();
    list_students.Add(new Students("Акимов", "Аким", "Акимов", new DateTime(1997, 01, 10)));
    list_students.Add(new Students("Иванов", "Иван", "Иванович", new DateTime(1997, 11, 05)));
    list_students.Add(new Students("Лаптев", "Иван", "Васильевич", new DateTime(1997, 07, 07)));
    list_students.Add(new Students("Мировнов", "Мирон", "Петрович", new DateTime(1997, 03, 15)));
    list_students.Add(new Students("Петров", "Петр", "Петрович", new DateTime(1997, 07, 25)));
    list_students.Add(new Students("Сидоров", "Сидор", "Сидорович", new DateTime(1997, 09, 19)));
    list_students.Add(new Students("Титов", "Тит", "Иванович", new DateTime(1997, 04, 05)));
    list_students.Add(new Students("Яковлев", "Яков", "Петрович", new DateTime(1997, 07, 205)));
    return list_students;
}
```

Рис. 2.24. Пример создание набора данных студентов для дальнейшего сохранения в файл

Для сохранения сериализованного объекта в файл используем метод Save (), код которого приведен на рис. 2.25.

```

/*Метод сохранения сериализованных объектов в отдельный файл*/
1 reference
public static void Save(string fileName, Object objToSerialize)
{
    FileStream fstream = File.Open(fileName, FileMode.Create);
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    binaryFormatter.Serialize(fstream, objToSerialize);
    fstream.Close();
}

/*Метод, который получает набор данных и передает их методу Save()*/
0 references
public static void SaveData()
{
    List<Students> s = Students.getList();
    Save("studentsSerialize", s);
}

```

Рис. 2.25. Пример сохранения сериализованных объектов в файл

После выполнения метода `SaveData ()` в том же каталоге, что и запускаемое приложение, будет создан файл `studentsSerialize` с сохраненным сериализованным объектом.

Для десериализации объекта из файла используем метод `ReadObjectFromFile ()`, код которого приведен на рис. 2.26.

В методе `getStudents` добавлена проверка типа объекта, который считывается из файла, чтобы избежать возникновения ошибки несоответствия типов. Ниже (рис. 2.26) показан десериализованный из файла объект.

Теперь рассмотрим вопрос шифрования паролей. Из соображений безопасности не стоит хранить пароли в открытом виде. Один из простых способов шифрования паролей — это применение хеш-функции MD5.

Хеш-функция — это математическое преобразование входного массива данных в строку определенной длины. Для MD5 длина получившейся строки равна 128 битам.

В C# разработан абстрактный класс MD5 (пространство имен `System.Security.Cryptography`), который реализует алгоритм MD5. Перечислим его основные методы:

- `Create ()` — создает экземпляр реализации по умолчанию MD5 хеш-алгоритма;
- `ComputeHash (Byte [])` — вычисляет хеш-значение для заданного массива байтов;
- `Equals (Object)` — определяет, равен ли заданный объект текущему объекту.

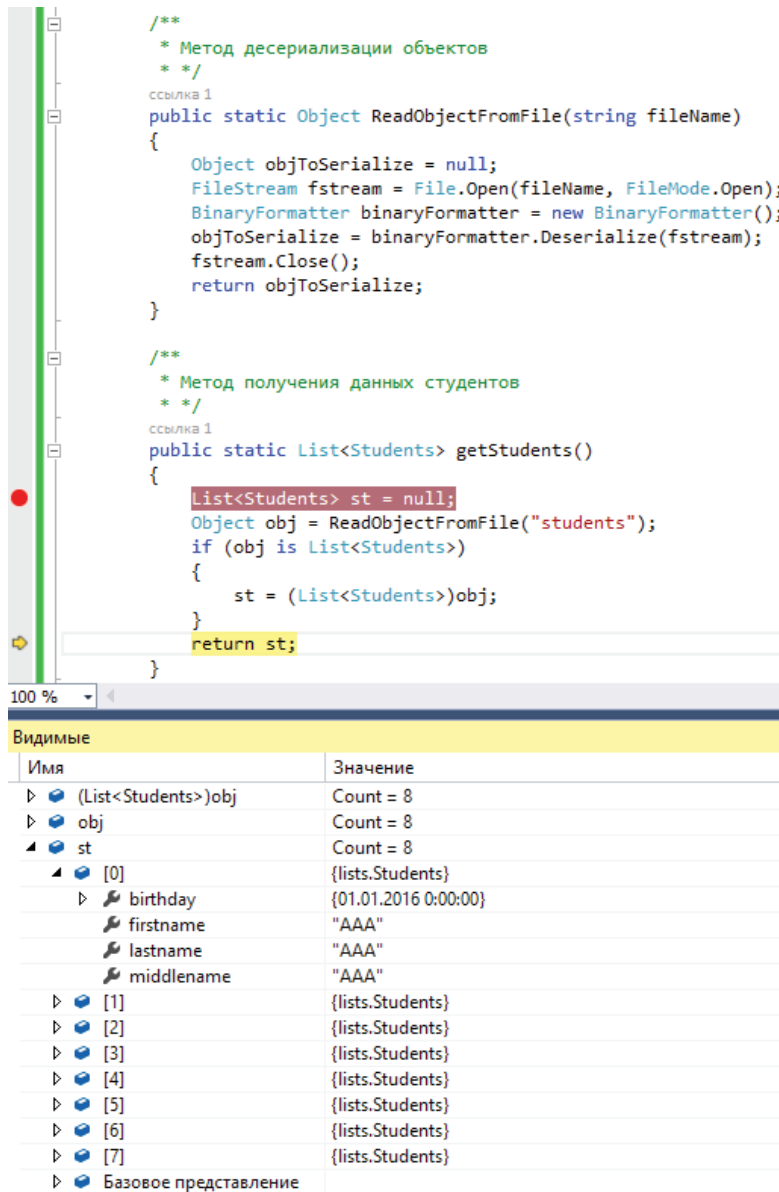


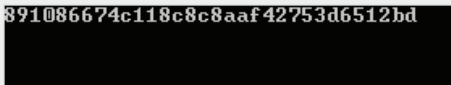
Рис. 2.26. Десериализация объекта из файла

Следует отметить, что не существует обратного преобразования получившейся строки в исходный массив данных. Для проверки пароля пользователя необходимо приводить вводимый пароль к хеш-значению и сравнивать его с сохраненным в настройках.

Пример реализации метода шифрования и получившаяся строка приведены на рис. 2.27.

```
/**
 * Метод шифрования текста (пароля) с помощью метода MD5
 * */
ссылка 1
public static String encrypt(String input)
{
    // создаем объект этого класса. Отмечу, что он создается не через new, а вызовом метода Create
    MD5 md5Hasher = MD5.Create();
    // Преобразуем входную строку в массив байт и вычисляем хэш
    byte[] data = md5Hasher.ComputeHash(Encoding.Default.GetBytes(input));
    // Создаем новый StringBuilder (Изменяемую строку) для набора байт
    StringBuilder sBuilder = new StringBuilder();

    // Преобразуем каждый байт хэша в шестнадцатеричную строку
    for (int i = 0; i < data.Length; i++)
    {
        //указывает, что нужно преобразовать элемент в шестнадцатеричную строку длиной в два символа
        sBuilder.Append(data[i].ToString("x2"));
    }
    return sBuilder.ToString();
}
ссылка 1
public static void example()
{
    Console.WriteLine(encrypt("dedaMoroz"));
    Console.ReadKey();
}
```



891086674c118c8c8aaf42753d6512bd

Рис. 2.27. Пример реализации функции шифрования по алгоритму MD5

Для анализа скорости работы пользователя в разрабатываемой системе можно использовать закон Фитса, который позволяет вычислить время, необходимое для того, чтобы достичь объекта. Закон Фитса может быть записан в виде следующей формулы:

$$T = \log_2 \left( \frac{D}{W} + 1 \right),$$

где  $T$  — время достижения цели;  $D$  — расстояние до цели;  $W$  — размер цели вдоль линии перемещения курсора.

Таким образом, чем дальше находится объект от текущей позиции курсора или чем меньше размеры этого объекта, тем больше времени потребуется пользователю для перемещения к нему курсора.

Отметим, что не стоит всегда слепо использовать закон Фитса. Существуют ограничения, при которых он неприменим, к примеру, пользователь не всегда перемещает курсор мыши к цели

уверенно и по прямой траектории. Однако база, лежащая в основе этого закона, остается.

В разделе 2.2 был показан процесс разработки формы для аутентификации пользователей приложения и работы администратора системы (просмотр, изменение и удаление информации о пользователях). Для закрепления знаний, полученных в разделе 2.4, студентам предлагается завершить разработку этих функций в своих приложениях.

Разработанная система должна сохранять все данные о пользователях в файле в виде сериализованных объектов (списков объектов). Также у администратора должна быть возможность удалять элементы из сохраненных данных, например, администратор может удалять любого пользователя и всю информацию, связанную с ним.

Пароли должны храниться в зашифрованном виде; для этого предлагается использовать хеш-функцию MD5.

По окончании разработки приложения студенты могут:

- выбрать уже реализованную функцию приложения, например, просмотр данных;
- оценить время выполнения пользователем этой функции, используя закон Фитса;
- провести анализ полученных результатов;
- предложить изменения ПИ, которые позволили бы сократить время работы пользователя (например, заменить кнопки на контекстное меню или предложить выполнение функции другим способом).

## **2.5. Документирование**

Для получения практических навыков работы по использованию возможностей приложений Microsoft Word и Excel для отображения информации из разрабатываемой информационной системы необходимо решить следующие задачи:

1. Ознакомиться с возможностями создания документов Microsoft Word и Excel с данными из приложения, разработанного в ходе изучения предыдущих разделов.

2. Разработать функции формирования отчетов в приложениях Microsoft Word и Excel.

3. Провести эвристическую оценку разработанного приложения на основе эвристик Якоба Нильсена.

Рассмотрим, какие возможности для документов Microsoft Word и Excel предоставляет нам среда разработки Visual Studio C#.

### **Общие сведения об объектной модели Excel.**

Для разработки решений, использующих Microsoft Office Excel, необходимо взаимодействие с объектами, предоставляемыми объектной моделью Excel:

- Microsoft.Office.Interop.Excel.Application;
- Microsoft.Office.Interop.Excel.Workbook;
- Microsoft.Office.Interop.Excel.Worksheet;
- Microsoft.Office.Interop.Excel.Range.

Объектная модель Excel точно соответствует пользовательскому интерфейсу. Объект Application представляет приложение в целом, а каждый из объектов Workbook содержит коллекцию объектов Worksheet. Объект Range позволяет работать с отдельными ячейками или группой ячеек.

Для работы с этими объектами необходимо в раздел проекта References добавить сборку Microsoft.Office.Interop.Excel (см. рис. 2.28).

Для запуска Excel необходимо выполнить следующие действия:

- подключить с псевдонимом пространство имен:  
`using Excel = Microsoft.Office.Interop.Excel;`
- объявить переменную для работы с приложением:  
`private Excel.Application exApp;`
- загрузить в память новый экземпляр приложения:  
`exApp = new Excel.Application ();`  
`exApp.Visible = true;`

Для работы с книгой Excel необходимо выполнить следующие действия:

1) создать новую книгу, например с одним листом:

```
exApp.SheetsInNewWorkbook = 1;  
exApp.Workbooks.Add ();
```

Свойство `SheetsInNewWorkbook` определяет количество листов, которое будет создано в новой рабочей книге Excel.

Методу Add в качестве необязательного параметра можно передать имя шаблона рабочей книги;

2) записать значение в текущую ячейку:

```
exApp.ActiveCell.Value = 1;
```

3) объявить переменную для работы с листом:

```
private Excel.Worksheet exWrkSht;
```

4) получить ссылку на лист 1:

```
exWrkSht = exApp.Workbooks [1].Worksheets.get_Item (1);
```

5) запись значения в конкретную ячейку:

```
exWrkSht.get_Range ("B1", "B1").Value = 1;
```

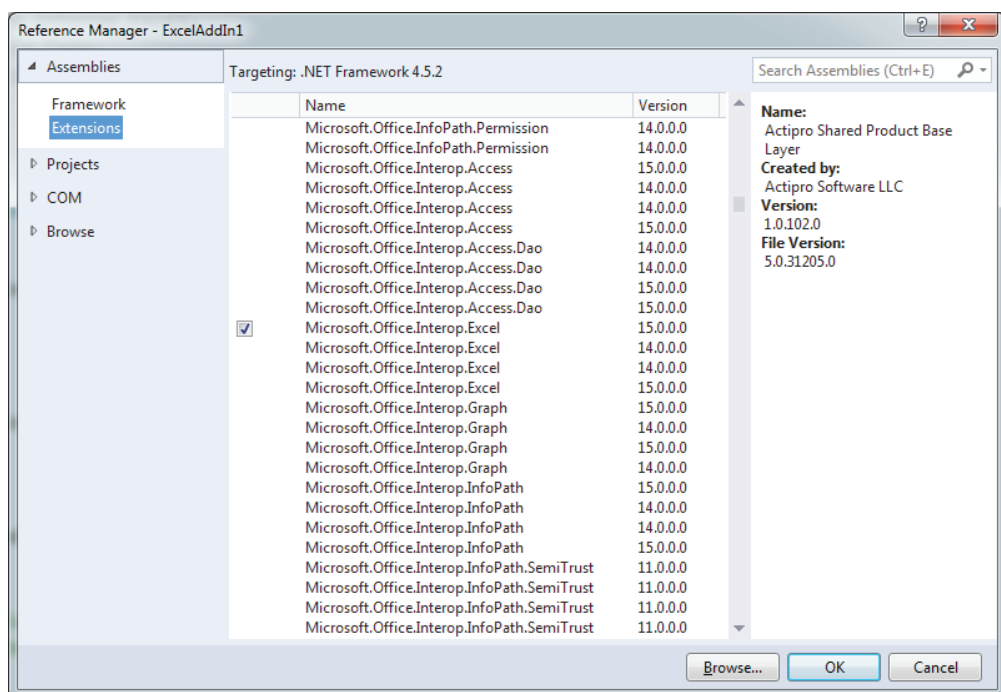


Рис. 2.28. Добавление сборки Microsoft.Office.Interop.Excel в проект

Метод Save () сохраняет книгу под именем, присваиваемым по умолчанию, в папке «Мои документы». Метод SaveAs () позволяет сохранить ее под указанным именем в нужной папке и в нужном формате.

Для сохранения книги можно использовать следующий код:

```
exApp.Workbooks [1].Save ();
```

```
exApp.Workbooks [1].SaveAs ("c:\\Users\\1.xlsx");
```

Для открытия существующего файла есть метод `Open ()`:

```
exApp.Workbooks [1].Open ("c:\\Users\\1.xlsx");
```

На рис. 2.29 показан пример кода для вывода информации в таблицу и вставка формулы «Сумма по первому столбцу».

```
for (var m = 1; m < 5; m++)
{
    for (var n = 1; n < 5; n++)
    {
        exWrkSht.Cells[m, n].Value = m;
    }
}
Excel.Range rg = exWrkSht.get_Range("A1", "D4");
rg.Borders.LineStyle = Excel.XlLineStyle.xlContinuous;
exWrkSht.get_Range("A5", "A5").Formula = "=SUM(A1:A4)";
```

Рис. 2.29. Пример кода для работы с Excel

### Общие сведения об объектной модели Word

Для разработки решений, использующих Microsoft Office Word, необходимо взаимодействие с объектами, предоставляемыми объектной моделью Word:

- Microsoft.Office.Interop.Word.Application
- Microsoft.Office.Interop.Word.Document,
- Microsoft.Office.Interop.Word.Selection,
- Microsoft.Office.Interop.Word.Bookmark,
- Microsoft.Office.Interop.Word.Range.

Объект `Application` представляет текущий экземпляр Word. Объект `Application` содержит объекты `Document` («Документ»), `Selection` («Текущая выбранная область»), `Bookmark` («Закладка») и `Range` («Диапазон текста»). Каждый из этих объектов содержит множество методов и свойств, к которым можно обращаться для работы с объектом и взаимодействия с ним.

Иерархия этих объектов показана на рис. 2.30.

Для работы с этими объектами в раздел проекта `References` необходимо добавить сборку `Microsoft.Office.Interop.Word`.

Для запуска Word необходимо выполнить следующие действия:

1) подключить с псевдонимом пространство имен:

```
using Word = Microsoft.Office.Interop.Word;
```

2) объявить переменную для работы с приложением:

```
private Word.Application wdApp;
```



3) загрузить в память новый экземпляр приложения:

```
wdApp = new Word.Application ();  
wdApp.Visible = true;
```

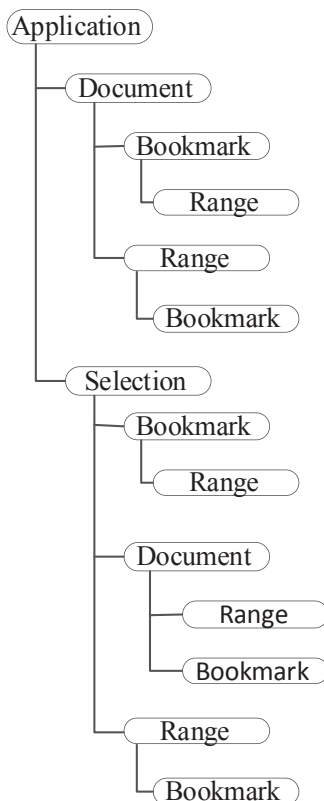


Рис. 2.30. Объектная модель Word

Для работы с документом Word необходимо выполнить следующие действия:

1) создать новый документ:

```
wdApp.Documents.Add ();
```

Методу Add в качестве необязательного параметра можно передать имя шаблона для создания нового документа;

2) объявить переменную для работы с документом:

```
private Word.Document docum;
```

3) получить ссылку на созданный документ можно по номеру или имени документа:

```
docum = wdApp.Documents.get_Item (1);
```

4) сохранить текущий документ.

Для сохранения документа можно использовать два метода — `Save ()` и `SaveAs ()`. Метод `Save ()` не имеет параметров и приводит к открытию стандартного диалога сохранения документа Word, который позволяет ввести название документа. В методе `SaveAs ()` в качестве параметра необходимо указать имя файла и формат. Например:

```
docum.SaveAs ("c:\\users\\doc2.docx");
```

Для открытия существующего документа используется метод `Open ()`.

Для вставки информации в документ можно использовать объект `Selection`, который всегда присутствует в документе:

```
wdApp.Selection.Text = "Hello Word";
```

Пример вставки таблицы в документ Word приведен на рис. 2.31.

```
var wdRange = docum.Range(0, 0);
var wdTable = docum.Tables.Add(wdRange, 3, 2);
wdTable.Cell(1, 1).Range.Text = "ФИО";
wdTable.Cell(1, 2).Range.Text = "Баллы";
wdTable.set_Style("Table Grid");
```

Рис. 2.31. Пример вставки таблицы в документ Word

Для получения практических навыков в соответствии с выбранной ранее тематикой разрабатываемого приложения необходимо реализовать функцию формирования отчетов в файлах Word и Excel (например, для темы «Электронный деканат» в Word можно формировать «хвостовки», а в Excel — отчет, содержащий оценки студента). Дать возможность авторизованному пользователю из меню и из некоторого окна формировать отчеты.

Эвристическая оценка приложения позволяет выявить основные проблемы, с которыми, скорее всего, столкнется пользователь. При этом интерфейс оценивают по некоторому набору критериев. В данной работе предлагается использовать эвристики Якоба Нильсена [10]. Кратко рассмотрим их:

- во всех ли случаях пользователю понятно, что происходит в приложении;
- понятна ли пользователю терминология, которая используется в приложении (названия пунктов меню, формулировки сообщений и т. п.);

- может ли пользователь отменить свои действия;
- придерживались ли единообразия и стандартов при разработке ПИ;
- предотвращает ли ПИ ошибки пользователей;
- понятны ли пользователю его действия при работе с системой (подсказки, продуманность элементов ПИ);
- присутствуют ли гибкость и эффективность реализации функций приложения для пользователей с разной подготовкой;
- придерживались ли эстетичного и минималистического дизайна;
- достаточное ли количество сообщений об ошибках и предупреждений, понятны ли их сообщения;
- есть ли справочные материалы и документация по приложению?

Для практического знакомства с эвристической оценкой приложения в качестве экспертов выберите двух студентов из вашей группы — они должны будут изучить разработанный интерфейс; при этом им не разрешается общаться и советоваться. Отчет эксперта должен содержать информацию о том, все ли критерии выполнены, и список выявленных ошибок и замечаний. Также эксперты могут добавить оценку каких-то критериев, не входящих в список эвристик. После этого необходимо проанализировать полученные результаты на предмет выявления основных юзабилити-проблем, которые необходимо устранить и привести рекомендации по исправлению самых существенных юзабилити-ошибок.

## **2.6. Применение технологии WPF для разработки интерфейса пользователя**

---

Для изучения основных особенностей технологии WPF следует:

- 1) ознакомиться с особенностями компоновки WPF.
- 2) попробовать самостоятельно разработать формы с элементами управления.

WPF (Windows Presentation Foundation) — API-интерфейс, основанный на DirectX. Платформа WPF включает знакомые стандартные элементы управления, но она рисует каждый текст, контур и фон самостоятельно. В результате WPF может предоставить намного более мощные средства, которые позволяют изменить визуализацию любой части экранного содержимого [11].

Для создания простейшего WPF-приложения необходимо выбрать соответствующий шаблон (см. рис. 2.32).

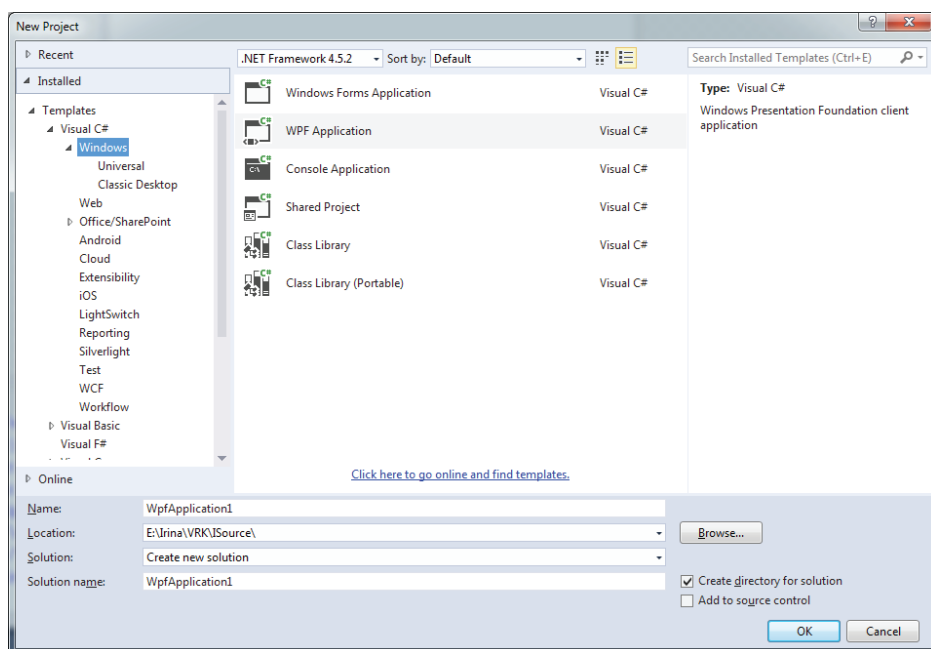


Рис. 2.32. Выбор шаблона для WPF-приложения

В результате будет создано приложение, представляющее пустую форму. Рассмотрим более подробно получившееся приложение.

Файл App.xaml определяет приложение WPF и все его ресурсы. В этом файле также указывается начальный пользовательский интерфейс, который автоматически открывается при запуске приложения. В данном случае — это MainWindow.xaml (рис. 2.33).

Файл MainWindow.xaml представляет главное окно приложения. Класс Window определяет свойства окна (заголовок, размер и значок) и обрабатывает события (открытие и закрытие окна).

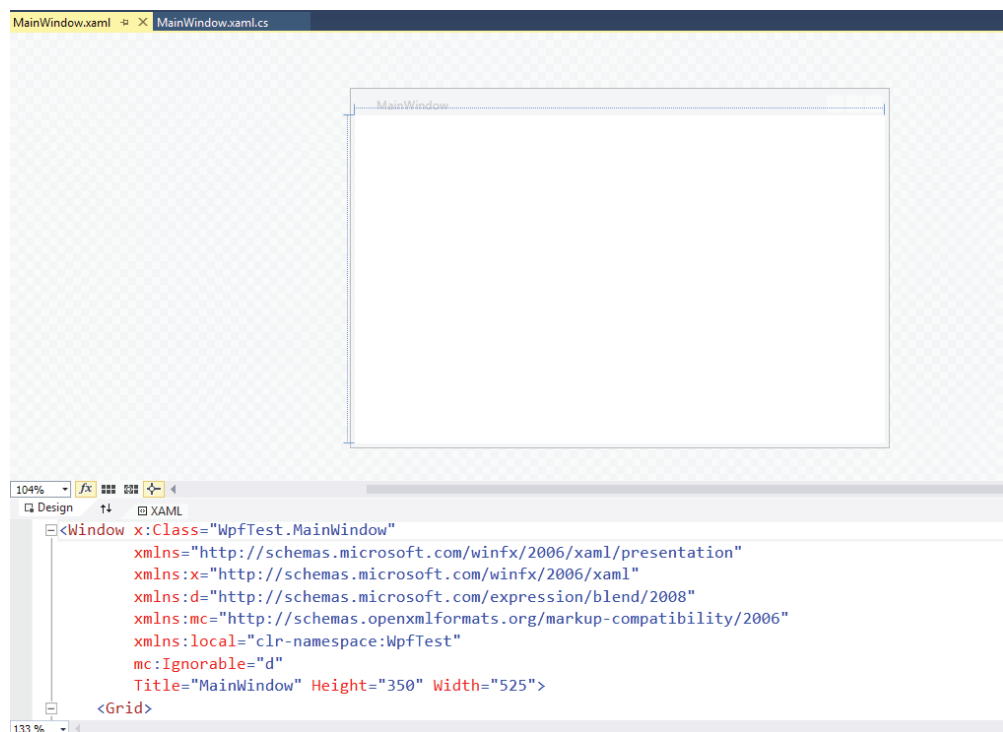


Рис. 2.33. Файл MainWindow.xaml

При построении пользовательских интерфейсов для WPF-приложений используется язык расширенной разметки приложений XAML (Extensible Application Markup Language). XAML-документ содержит разметку, описывающую внешний вид и поведение окна или страницы приложения, а связанные с ним файлы кода C# — логику приложения. Язык XAML обеспечивает разделение процесса дизайна приложения (графической части) и разработки бизнес-логики (программного кода) между дизайнерами и разработчиками. WPF XAML является подмножеством языка XML и позволяет описывать WPF-содержимое таких элементов, как векторная графика, элементы управления и документы.

Каждый элемент XAML-документа отображается на некоторый экземпляр класса .NET. Имя такого элемента в точности соответствует имени класса. Например, элемент Button служит для WPF инструкцией для построения объекта класса Button;

Элементы XAML можно вкладывать друг в друга. Вложение элементов разметки обычно отображает вложенность элементов интерфейса.

Свойства класса определяются с помощью атрибутов или с помощью вложенных дескрипторов со специальным синтаксисом. Атрибуты элементов используются для задания свойств (Name, Height, Width и т. п.) и событий (Click, Load и т. д.) соответствующих объектов.

Для программного управления элементами управления, описанными в XAML-документе, необходимо для элемента управления задать XAML атрибут Name. Так, для задания имени элементу Button необходимо записать следующую разметку:

```
<ButtonName=>grid>
```

```
</Button>
```

Простые свойства задаются в XAML-документе в соответствии со следующим синтаксисом:

```
ИмяСвойства =«значение»
```

```
Например, Name = «Button1»
```

При необходимости надо задать свойство, которое является полноценным объектом; в коде используются сложные свойства в соответствии с синтаксисом «свойство-элемент»:

```
Родитель.ИмяСвойства
```

## Компоновка

При объявлении элемента управления непосредственно внутри окна, не имеющего панелей, он размещается в центре окна. Для организации содержимого окна в WPF используются разнообразные контейнеры. Все контейнеры компоновки WPF являются панелями, которые унаследованы от абстрактного класса System.Windows.Controls.Panel. Пространство имен System.Windows.Controls предлагает многочисленные панели. Перечислим основные:

- StackPanel — размещает элементы в горизонтальном или вертикальном стеке. Этот контейнер компоновки обычно используется в небольших разделах крупного и более сложного окна;
- WrapPanel — размещает элементы в последовательностях строк с переносом. В горизонтальной ориентации панель WrapPanel располагает элементы в строке слева направо, за-

тем переходит к следующей строке. В вертикальной ориентации панель `WrapPanel` располагает элементы сверху вниз, используя дополнительные колонки для дополнения оставшихся элементов;

- `DockPanel` — выравнивает элементы по краю контейнера;
- `Grid` — выстраивает элементы в строки и колонки невидимой таблицы. Это один из наиболее гибких и широко используемых контейнеров компоновки;
- `UniformGrid` — помещает элементы в невидимую таблицу, устанавливая одинаковый размер для всех ячеек. Данный контейнер компоновки используется нечасто;
- `Canvas` — позволяет элементам позиционироваться абсолютно — по фиксированным координатам. Этот контейнер компоновки более всего похож на традиционный компоновщик `Windows Forms`, но не предусматривает средств привязки и стыковки. В результате это неподходящий выбор для окон переменного размера.

Компоновка определяется контейнером, но дочерние элементы тоже могут влиять на расположение на форме. Панели компоновки взаимодействуют со своими дочерними элементами через набор свойств компоновки:

- `HorizontalAlignment` — определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по горизонтали. Доступные значения: `Center`, `Left`, `Right` или `Stretch`;
- `VerticalAlignment` — определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по вертикали. Доступные значения: `Center`, `Top`, `Bottom` или `Stretch`;
- `Margin` — добавляет некоторое пространство вокруг элемента. Свойство `Margin` — это экземпляр структуры `System.Windows.Thickness` с отдельными компонентами для верхней, нижней, левой и правой граней;
- `MinWidth`, `MinHeight` — устанавливают минимальные размеры элемента. Если элемент слишком велик, чтобы поместиться в его контейнер компоновки, он будет усечен;
- `MaxWidth`, `MaxHeight` — устанавливают максимальные размеры элемента. Если контейнер имеет свободное простран-

ство, элемент не будет увеличен сверх указанных пределов, даже если свойства `HorizontalAlignment` и `VerticalAlignment` установлены в значение `Stretch`;

- `Width`, `Height` — явно устанавливают размеры элемента. Эта установка переопределяет значение `Stretch` для свойств `HorizontalAlignment` и `VerticalAlignment`. Однако данный размер не будет установлен, если выходит за пределы, заданные в свойствах `MinWidth`, `MinHeight`, `MaxWidth` и `MaxHeight`.

Все эти свойства унаследованы от базового класса «`Framework-Element`», поэтому поддерживаются всеми графическими элементами, которые можно использовать в окне WPF.

### Элементы управления

Элементы управления содержимым (`Label`, `Button`, `CheckBox` и `RadioButton`) являются специализированным типом элементов управления, которые могут хранить некоторое содержимое — один или несколько элементов. Все элементы управления содержимым являются наследниками класса `ContentControl`.

Класс `ContentControl` наследуется от класса `System.Windows.Control`, который наделяет его и все дочерние классы базовыми характеристиками, которые:

- позволяют определять содержимое внутри элемента управления;
- позволяют определять порядок перехода с использованием клавиш табуляции;
- поддерживают рисование фона, переднего плана и рамки;
- поддерживают форматирование размера и шрифта текстового содержания.

Метка `Label` представляет собой самый простой элемент управления содержимым. Особенностью этого элемента является поддержка мнемонических команд. Все мнемонические команды работают при одновременном нажатии клавиши «`Alt`» и заданной клавиши быстрого доступа. При этом фокус передается элементу управления, связанному с меткой. Для поддержки этой функции у метки в свойстве `Target` необходимо указать выражение привязки. В нем прописывается другой элемент управления, на который будет переходить фокус при нажатии клавиши быстрого доступа:



```
<Label Target="{Binding ElementName = txtA}">Выбор _A</Label>
<TextBox Name="txtA">Выбор текста</TextBox>
```

Символ подчеркивания в тексте метки указывает на клавишу быстрого доступа. В приведенном выше коде при нажатии комбинации «Alt + A» фокус перейдет на элемент управления TextBox с именем txtA.

Для классов Button, CheckBox и RadioButton определено событие Click.

Класс Button добавляет два свойства, доступные для записи: IsCancel и IsDefault.

При значении свойства IsCancel, равном true, кнопка будет работать как кнопка отмены окна, и если нажать кнопку «Esc», когда текущее окно находится в фокусе, то кнопка сработает. Если значение свойства IsDefault равно true, то кнопка считается кнопкой по умолчанию.

Объекты CheckBox и RadioButton имеют два состояния: выбраны или не выбраны. В классе имеются события Checked, Unchecked и Intermediate, которые генерируются при включении, выключении или переходе кнопки в неопределенное состояние.

Для кнопки CheckBox включение элемента управления означает отметку в нем флажка. Свойство IsChecked, наследуемое от класса ToggleButton, может принимать три значения: true (включено), false (выключено), null (неопределено, которое отображается в виде затененного окна и используется для промежуточного состояния). Пример XAML-описания трех кнопок CheckBox:

```
<CheckBoxHeight=>16» Name=>checkBox1»
Width=>120» IsChecked=>False»
ClickMode="Release">Выбор A</CheckBox>
<CheckBox Height="16" Name="checkBox2"
Width="120" IsChecked="True"
ClickMode="Press">Выбор B</CheckBox>
<CheckBox Height="16" Name="checkBox3"
Width="120" IsChecked="{x: Null}"
ClickMode="Hover">Выбор B</CheckBox>
```

Для кнопки RadioButton добавлено свойство GroupName, которое позволяет управлять расположением переключателей в группе. Из группы можно выбрать только один переключатель.

```

<GroupBoxHeader=»Группа радиокнопок« Height=»100«
Name="groupBox1" Width="200">
<StackPanel>
<RadioButton Height="16" Name="radioButton2"
Width="120">Выбор Г</RadioButton>
<RadioButton Height="16" Name="radioButton1"
Width="120">Выбор Д</RadioButton>
<RadioButton Height="16" Name="radioButton3"
Width="120">Выбор Е</RadioButton>
</StackPanel>
</GroupBox>

```

Для вывода контекстного окна для элемента управления существует свойство `ToolTip`, например для кнопки:

```
<ButtonToolTip=»Подсказка для кнопки А«></Button>
```

Элемент управления `TextBox`, как правило хранит одну строку текста:

```
<TextBoxName=»txtA«> Выбор текста</TextBox>
```

Если необходимо создать многострочное представление, то свойству `TextWrapping` необходимо присвоить значение `Wrap`. Для многострочного элемента `TextBox` можно задать минимальное и максимальное количество строк, используя свойства `MinLines` и `MaxLines`.

Элементы управления `ListBox` и `ComboBox` являются элементами управления списками.

Для добавления элементов в `ListBox` можно вложить элементы `ListBoxItem` в элемент управления `ListBox`, как это показано для составления списка цветов (зеленый, голубой, желтый, красный):

```

<ListBox>
<ListBoxItem>Зеленый</ListBoxItem>
<ListBoxItem>Голубой</ListBoxItem>
<ListBoxItem>Желтый</ListBoxItem>
<ListBoxItem>Красный</ListBoxItem>
</ListBox>

```

Элемент управления `ListBox` хранит каждый вложенный объект в своей коллекции, при этом `ListBoxItem` может хранить не только строки, но и любой произвольный элемент.

Элемент управления `ComboBox` подобен элементу `ListBox`.

## Стили

Стили — это коллекция значений свойств, которые могут быть применены к элементу. Они позволяют определить общий набор характеристик форматирования и применять их по всему приложению для обеспечения согласованности. Стили в WPF могут устанавливать любое свойство зависимостей. Их можно применять для стандартизации визуального поведения каких-либо элементов. Стили WPF поддерживают триггеры, которые позволяют изменять стиль элемента при изменении других свойств. Стили позволяют использовать шаблоны для переопределения стандартного визуального представления элементов управления.

Стиль создается на базе класса `Style` со следующими свойствами:

- `BasedOn` — возвращает или задает определенный стиль, являющийся основой текущего стиля;
- `Dispatcher` — возвращает объект `Dispatcher`, с которым связан этот объект `DispatcherObject`;
- `IsSealed` — возвращает значение, указывающее, доступен ли стиль только для чтения;
- `Resources` — возвращает или задает коллекцию ресурсов, которые могут использоваться в области видимости данного стиля;
- `Setters` — возвращает коллекцию объектов `Setter` и `EventSetter`;
- `TargetType` — возвращает или задает тип, для которого предназначен данный стиль;
- `Triggers` — возвращает коллекцию объектов `TriggerBase`, изменяющих значения свойств на основе заданных условий.

Например, в приложении необходимо многократно использовать кнопки, которые имеют светло-голубой фон и голубую рамку:

```
<Style x: Key="ButtonStyle" TargetType="Button">
  <Setter Property="Background" Value="LightBlue"/>
  <Setter Property="BorderBrush" Value="Blue"/>
</Style>
```

Имя стилю задается атрибутом `x: Key` (значение `ButtonStyle`). Атрибут `TargetType` определяет тип, для которого будет применяться стиль (значение `Button`). Каждый объект `Setter` устанавливает в элементе одно свойство, которое обязательно должно

быть свойством зависимостей. Установка свойств производится с помощью атрибутов `Property`, который определяет имя свойства, и `Value`, который задает значение свойства.

В стиль можно добавить объект `EventSetters`, в котором события привязываются к определенному обработчику.

Триггеры позволяют вносить изменения в стиль при выполнении определенных условий. Добавим в стиль для кнопки `ButtonStyle` триггер, который будет формировать красный фон при наведении указателя мыши на кнопку:

```
<Style x: Key="ButtonStyle" TargetType="Button">
  <Setter Property="Background" Value="LightBlue"/>
  <Setter Property="BorderBrush" Value="Blue"/>
  <Style.Triggers>
    <Trigger Property="IsMouseOver" Value="true">
      <Setter Property="Background" Value="Red"/>
    </Trigger>
  </Style.Triggers>
</Style>
```

В триггере должно быть указано идентифицирующее свойство (в показанном выше примере — `IsMouseOver`), за которым должно вестись наблюдение, и значение, которого следует ожидать (в показанном выше примере — `true`). Когда появляется необходимое значение, устанавливается свойство, которое определено объектом `Setter`. В приведенном примере свойство `IsMouseOver` имеет значение `true`, то есть при наведении указателя мыши на кнопку свойству `Background` присваивается значение `Red`, следовательно, фон кнопки становится красным. Когда указатель мыши покидает кнопку, условия срабатывания триггера нарушаются, и фон кнопки устанавливается в прежнее состояние.

Кроме триггера, ожидающего изменение свойства, существуют триггеры события (`EventTrigger`), которые ожидают возникновения определенного события.

Пример создания главного меню:

```
<Menu>
  <MenuItem Header="Файл" >
    <MenuItem Header="Создать" ></MenuItem>
    <MenuItem Header="Редактировать" ></MenuItem>
    <MenuItem Header="Сохранить" ></MenuItem>
```

```
<MenuItem Header="Найти"/>
<Separator></Separator>
<MenuItem Header="Удалить" ></MenuItem>
</MenuItem>
<MenuItem Header="Выход"></MenuItem>
</Menu>
```

Для получения практических навыков студентам предлагается разработать интерфейс формы приложения на базе технологии WPF с использованием элементов пользовательского интерфейса: `ListBox`, `ListView`, `TextBox`, `ComboBox`, `Menu`.

## **2.7. Разработка WPF-приложения для работы с данными**

---

Для разработки WPF-приложений, взаимодействующих с данными, необходимо решить следующие задачи:

- 1) ознакомиться со способом представления и взаимодействия с данными (привязкой данных в WPF-приложениях);
- 2) изучить интерфейс `INotifyPropertyChanged` и конвертеры значений;
- 3) разработать WPF-приложения, взаимодействующие с данными.

Привязка данных представляет собой взаимодействие двух объектов: источника и приемника. Объект-приемник создает привязку к определенному свойству объекта-источника. При изменении объекта-источника объект-приемник также будет изменяться [11].

Для определения привязки в `xaml`-файле используется следующее выражение:

```
{Binding ElementName=<Имя объекта-источника>, Path = <Свойство объекта-источника>}
```

Ниже приведен пример привязки. Компонент `TextBlox` с названием `txt1TextBox` будет источником, а компонент `TextBlox` с названием `txt2TextBox` — приемником:

```
<TextBox x: Name="txt1TextBox">
  <TextBox x: Name=" txt2TextBox " Text="{Binding ElementName=
txt2TextBox, Path=Text}"/>
```

Привязка может быть установлена также в коде `C#`:

```

public MainWindow ()
{
    InitializeComponent ();
    Binding binding = new Binding ();
    binding.ElementName = "txt1TextBox";//элемент-источник
    binding.Path = new PropertyPath ("Text");//свойство элемен-
та-источника
    txt2TextBox.SetBinding (TextBox.TextProperty, bind-
ing);//установка привязки для элемента-приемника
}

```

Для удаления привязки можно воспользоваться классом `BindingOperations` и его методами:

- `ClearAllBindings (DependencyObject)` — удаляет все привязки для указанного объекта `DependencyObject`;
- `ClearBinding (DependencyObject, DependencyProperty)` — удаляет привязку из свойства `DependencyProperty` объекта `DependencyObject`, если она существует.

К примеру:

```

BindingOperations.ClearBinding (txt2TextBox, TextBlox.Text-
Property);

```

или

```

BindingOperations.ClearAllBindings (txt2TextBox);

```

Ниже приводятся некоторые свойства класса `Binding`:

- `ElementName` — определяет имя элемента, который будет использоваться в качестве источника привязки;
- `IsAsync` — определяет значение, которое указывает на использование асинхронного режима получения данных из объекта. По умолчанию равно `False`;
- `Mode` — определяет режим привязки;
- `Path` — определяет свойство объекта, к которому идет привязка;
- `TargetNullValue` — определяет значение по умолчанию, которое используется в случае, если значение привязанного свойства источника привязки равно `null`;
- `RelativeSource` — определяет источник привязки относительно положения текущего объекта;
- `Source` — определяет объект-источник привязки, если он не является элементом управления;

- XPath — определяет путь к xml-данным или XPath запрос, используется вместо свойства Path.

Ниже эти свойства рассматриваются подробнее.

Режимы привязки определяют возможность изменения данных в источнике и приемнике. Для этого используется свойство Mode класса Binding. Оно может принимать одно из значений BindingMode (перечислимый тип):

- Default — по умолчанию (если меняется свойство TextBox.Text, то значение TwoWay, в остальных случаях — OneWay);
- OneWay — свойство объекта-приемника изменяется в случае изменения свойства объекта-источника, целесообразно использовать в том случае, если объект-приемник доступен только для чтения;
- OneTime — свойство объекта-приемника устанавливается по свойству объекта-источника только один раз, например, при запуске приложения. Используется, когда источник является статичным, то есть не изменяется при работе приложения;
- TwoWay — оба объекта, приемник и источник, могут изменять привязанные свойства друг друга; используется для просмотра и редактирования данных на форме;
- OneWayToSource — объект-приемник изменяет соответствующее свойство объекта-источника.

При использовании односторонней привязки изменения источника сразу же отображаются в соответствующем свойстве приемника. При использовании двусторонней привязки при изменении приемника свойство источника не изменяется мгновенно. Свойство UpdateSourceTrigger класса Binding определяет, как будет происходить обновление. Это свойство может принимать одно из следующих значений:

- Default — значение по умолчанию. Для большинства свойств разных компонент это значение PropertyChanged, для свойства Text элемента TextBox это значение LostFocus;
- Explicit — источник обновляется только при вызове метода BindingExpression.UpdateSource ();
- LostFocus — источник привязки обновляется каждый раз при потере фокуса приемником;
- PropertyChanged — источник привязки обновляется сразу же при изменении свойства в приемнике.

Если необходимо установить привязку к объекту, который не является элементом управления WPF, то для этого используется свойство «Source» класса Binding.

Свойство TargetNullValue класса Binding позволяет задать некоторое значение по умолчанию, если свойство в источнике привязки имеет значение null, то есть оно не определено

Свойство RelativeSource класса Binding используется для установки привязки относительно элемента-источника, который связан какими-нибудь отношениями с элементом-приемником. Например, элемент-источник может быть одним из внешних контейнеров для элемента-приемника или источник и приемник представляет собой один и тот же элемент.

Для установки этого свойства используется объект RelativeSource, который имеет свойство Mode. Это свойство задает способ привязки. Оно может принимать одно из значений перечисления RelativeSourceMode:

- Self — привязка осуществляется к свойству этого же элемента, то есть источник привязки также является и приемником привязки;
- FindAncestor — привязка осуществляется к свойству элемента-предка в последовательности родительских элементов элемента с привязкой.

Выше обсуждалось свойство Source для указания источника привязки. С помощью свойства DataContext можно установить область видимости, внутри которой у всех свойств с привязкой будет один источник привязки. Это свойство есть у всех элементов, которые наследуются от объекта FrameworkElement. Следует отметить, что источник, указанный через свойство Source, имеет приоритет над наследуемым контекстом данных.

Обсуждаемые выше режимы привязки не помогают решить проблему отображения изменений в объекте, который через Binding привязан к свойству визуального компонента. Для этого необходимо реализовать в его классе интерфейс INotifyPropertyChanged. Ниже приведен пример описания класса.

```
class Client: INotifyPropertyChanged
{
    private string fullName;
    private string shortName;
```



```

public event PropertyChangedEventHandler PropertyChanged;

public void RaisePropertyChanged (string propertyName)
{
    if (PropertyChanged != null)
        PropertyChanged (this, new
            PropertyChangedEventArgs (propertyName));
}

public string FullName
{
    get {return fullName;}
    set
    {
        fullName = value;
        RaisePropertyChanged ("FullName");
    }
}

public string ShortName
{
    get{return shortName;}
    set
    {
        shortName = value;
        RaisePropertyChanged («ShortName»);
    }
}
}

```

Ниже показан пример кода для отображения свойств класса на форме.

```

<Label Content="Полное наименование"/>
<TextBox x: Name="login" Grid.Column="1" Margin="0,5,0,5"
MaxWidth="250" MinWidth="150" Text="{Binding Path=FullName,
Mode=TwoWay}"/>
<Label Grid.Row="1" Grid.Column="0" Content="Краткое наиме-
нование"/>

```

```
<TextBox x: Name="paswd" Grid.Row="1" Grid.Column="1" Margin="0,5,0,5" Padding="5" MaxWidth="250" MinWidth="150" Text="{Binding Path=ShortName, Mode=TwoWay}"/>
```

В некоторых случаях недостаточно просто использовать привязку, необходимо преобразовывать данные. К примеру, свойство класса может храниться в базе данных в виде кода, но на экране нужно показать понятное для пользователя значение, или же источник и приемник привязки имеют соответствующие свойства, несовместимые по типу. В последнем случае необходимо использовать форматирование значений.

В зависимости от типа визуального компонента для форматирования значений используются различные свойства:

- `StringFormat` — для класса `Binding`;
- `ContentStringFormat` — для классов `ContentControl`, `ContentPresenter`, `TabControl`;
- `ItemStringFormat` — для класса `ItemsControl`;
- `HeaderStringFormat` — для класса `HeaderContentControl`;
- `ColumnHeaderStringFormat` — для классов `GridView`, `GridViewHeaderRowPresenter`;
- `SelectionBoxItemStringFormat` — для классов `ComboBox`, `RibbonComboBox`.

Чтобы форматирование значений заработало, необходимо реализовать в конвертере значений интерфейс `System.Windows.Data.IValueConverter`. Этот интерфейс определяет два метода: `Convert()`, который преобразует пришедшее от привязки значение в тип, который понимается приемником привязки, и `ConvertBack()`, который выполняет обратное преобразование.

Эти методы имеют четыре параметра:

- `object value` — значение, которое надо преобразовать;
- `Type targetType` — тип, к которому надо преобразовать значение `value`;
- `object parameter` — вспомогательный параметр;
- `CultureInfo culture` — текущая культура приложения.

**Пример 1.** Добавим в класс `Client` еще одно свойство, значение которого будем отображать с помощью `ComboBox`:

```
class Client: INotifyPropertyChanged
{
```

```

private string fullName;
private string shortName;
private string indOrJur;

public string IndOrJur
{
    get {return indOrJur;}
    set
    {
        indOrJur = value;
        RaisePropertyChanged ("IndOrJur");
    }
}

```

**Определим конвертер:**

```

public class IndOrJurConvert: IValueConverter
{
    public object Convert (object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        if (value.ToString () == "Ф")
            return 1;
        else
            return 0;
    }

    public object ConvertBack (object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        if (value.ToString () == "0")
            return "Ю";
        else return «Ф»;
    }
}

```

Укажем его в ресурсах окна:

```
<Window.Resources>
    <local: IndOrJurConvert x: Key="IndOrJurConverter"/>
</Window.Resources>
Определим ComboBox:
<ComboBox x: Name="comboBox" HorizontalAlignment="Left"
Margin="9,0,0,0"
    Grid.Row="2" VerticalAlignment="Top" Width="120"
    SelectedIndex="{Binding Path=IndOrJur,
    Converter={StaticResource IndOrJurConverter},
    Mode=TwoWay}"/>
    <TextBlock>юр. лицо</TextBlock>
    <TextBlock>физ. лицо</TextBlock>
</ComboBox>
```

**Пример 2.** Отобразим свойство IndOrJur с помощью RadioButtons:

```
<RadioButton x: Name="radioButton" GroupName="typClient"
Content="юр. лицо" Grid.Column="1" HorizontalAlignment="Left"
Margin="13,3,0,0" Grid.Row="2" VerticalAlignment="Top"
IsChecked="{Binding Path=IndOrJur,
Converter={StaticResource IndOrJurConverterBool},
ConverterParameter=0,
Mode=TwoWay}"/>
<RadioButton x: Name="radioButton1" GroupName="typClient"
Content="физ. лицо" Grid.Column="1" HorizontalAlignment="Left"
Margin="117,3,0,0" Grid.Row="2" VerticalAlignment="Top"
IsChecked="{Binding Path=IndOrJur,
Converter={StaticResource IndOrJurConverterBool},
ConverterParameter=1,
Mode=TwoWay}"/>
```

Конвертер будет выглядеть следующим образом:

```
public class IndOrJurConvertBool: IValueConverter
{
    public object Convert (object value, Type targetType,
    object parameter, CultureInfo culture)
    {
```

```
        if (parameter.ToString () == "1")
            return (value.ToString () == "Ф");
        else
            return! (value.ToString () == "Ф");
    }

    public object ConvertBack (object value, Type
targetType, object parameter, CultureInfo culture)
    {
        if (parameter.ToString () == "1")
            return ((bool)value? "Ф": "Ю");
        else
            return ((bool)value? "Ю": "Ф");
    }
}
```

Для закрепления полученной информации студентам предлагается разработать приложение в технологии WPF, которое считывает информацию об объекте (например, студент) из структурированного файла (например, xml) и отображает его на форме с использованием не менее трех типов элементов управления. Изменения, внесенные пользователем с помощью элементов управления, сохраняются обратно в файл.

## **2.8. Оценка эффективности пользовательского интерфейса по критерию скорости на основе модели GOMS**

---

Для изучения возможности оценки эффективности пользовательского интерфейса по критерию скорости на основе модели GOMS необходимо решить следующие задачи:

- 1) ознакомиться с моделью GOMS;
- 2) оценить скорость работы с разработанной ранее интерфейсной формой на основе модели GOMS.

GOMS — это английская аббревиатура: Goals, Operators, Methods and Selection Rules (цели, операторы, методы и правила выбора). Такой способ моделирования был предложен Кар-

дом, Мораном и Ньюэллом в 1983 г. — оно позволяет оценить время выполнения конкретной операции при использовании данной модели ПИ [6].

При работе с моделью GOMS предполагается, что время, необходимое пользователю для выполнения определенной задачи с помощью ПИ, является суммой всех временных интервалов, за которые выполняются элементарные жесты, составляющие данную задачу. Для разных пользователей время выполнения каждого жеста может отличаться, но для сравнительного анализа различных интерфейсов по скорости их использования допустимо использовать средние значения, которые получены исследователями (см. табл. 2.6).

Таблица 2.6

**Средние временные интервалы,  
требуемые для выполнения элементарных жестов**

Операция, интервал	Название	Описание
$K = 0,2 \text{ с}$	Нажатие клавиши	Время, необходимое для нажатия клавиши
$B = 0,1 \text{ с}$	Нажатие кнопки	Время, необходимое для нажатия на кнопку мыши
$P = 1,1 \text{ с}$	Указание	Время, необходимое для указания на какую-то позицию на экране монитора
$H = 0,4 \text{ с}$	Перемещение	Время, необходимое для перемещения руки с клавиатуры на мышь или с мыши на клавиатуру
$M = 1,35 \text{ с}$	Ментальная подготовка	Время, необходимое для умственной подготовки к следующему шагу, то есть время принятия решения о следующем действии
$R = 0,1 \text{ с}$	Ответ	Время ожидания ответа компьютера. Для базовых операций, таких, как работа с меню, это время можно не учитывать, то есть считать равным нулю

Для вычисления времени, затрачиваемого на выполнение операции с помощью модели GOMS, сначала необходимо перечислить и записать последовательность производимых пользователем элементарных действий из списка жестов модели GOMS (см. табл. 2.6). Затем в эту последовательность нужно добавить ментальные операции, т.е. умственные операции по подготовке

к следующему действию. На следующем этапе следует исключить ментальные операции там, где последовательные действия не требуют времени на размышление. В табл. 2.7 предложены некоторые правила расстановки.

Таблица 2.7

**Правила расстановки ментальных операций**

Правило 0: начальная расстановка операторов <i>M</i>	<p>Операторы <i>M</i> необходимо устанавливать:</p> <ul style="list-style-type: none"> <li>• перед всеми операторами <i>K</i>;</li> <li>• перед всеми операторами <i>P</i>, предназначенными для выбора команд.</li> </ul> <p>Перед операторами <i>P</i>, предназначенными для указания на аргументы выбранных команд, ставить оператор <i>M</i> не надо</p>
Правило 1: удаление ожидаемых операторов <i>M</i>	<p>Если оператор, следующий за оператором <i>M</i>, является полностью ожидаемым с точки зрения предыдущего оператора, то этот оператор <i>M</i> может быть удален.</p> <p>Например, если курсор мыши перемещается к нужному объекту, чтобы его выбрать, то есть пользователь нажмет на кнопку мыши, когда курсор достигнет объекта. Эта последовательность разделяется ментальной операцией, поэтому, согласно правилу 0, эта ментальная операция убирается</p>
Правило 2: удаление операторов <i>M</i> внутри когнитивных единиц	<p>Когнитивной единицей является непрерывная последовательность вводимых символов, которые образуют название команды или аргумент (например, имена файлов при сохранении, значения каких-то параметров для выполнения команды).</p> <p>Если строка вида <i>MKMKMK</i> ... принадлежит когнитивной единице, то необходимо удалить все операторы <i>M</i>, кроме первого.</p>
Правило 3: удаление операторов <i>M</i> перед последовательными разделителями	<p>Разделитель — это символ, которым обозначено начало или конец значимого фрагмента текста. Например, пробелы являются разделителями для большинства слов.</p> <p>Если для выполнения команды требуется дополнительная информация (к примеру, если для перевода значения из метров в километры пользователю требуется указать это значение), эта информация называется аргументом данной команды.</p> <p>Если оператор <i>K</i> означает лишний разделитель, стоящий в конце когнитивной единицы, то следует удалить находящийся перед ним оператор <i>M</i></p>

Окончание табл. 2.7

Правило 4: удаление операторов $M$ , которые являются прерывателями команд	Постоянная строка — это последовательность символов, которая каждый раз вводится одинаково. Если оператор $K$ является разделителем, стоящим после постоянной строки, то стоящий перед ним оператор $M$ удаляется. Если оператор $K$ является разделителем для строки аргументов или любой другой изменяемой строки, то оператор $M$ перед ним сохраняется
Правило 5: удаление перекрывающих операторов $M$	Не следует учитывать любую часть оператора $M$ , перекрывающую оператор $R$ , который является задержкой из-за ожидания ответа системы

В конце, согласно полученной схеме, подсчитывается суммарное время выполнения пользователем операции.

Приведем пример использования модели GOMS.

На рис. 2.34 представлена интерфейсная форма для входа в систему.

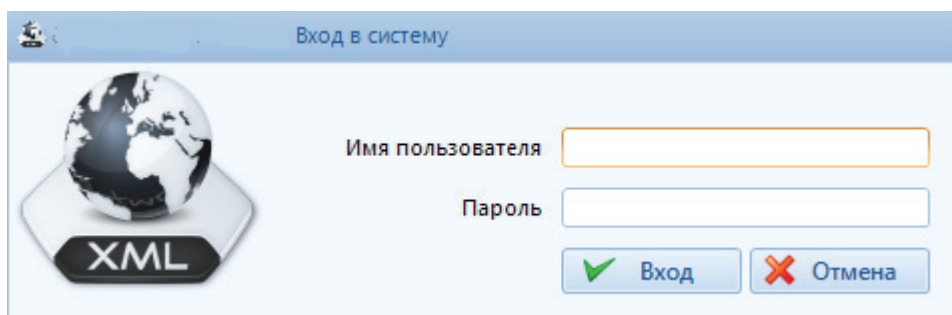


Рис. 2.34. Интерфейсная форма для входа в систему

Оценим эффективность работы с моделью GOMS при использовании мыши для перемещения между элементами ПИ (действия пользователя показаны в табл. 2.8).

Таблица 2.8

#### Действия пользователя (использование мыши)

Очередное действие пользователя	Формируемая последовательность операций
Ввод имени пользователя (8 символов)	KKKKKKKK
Перемещение руки на мышь	KKKKKKKKH



Окончание табл. 2.8

Очередное действие пользователя	Формируемая последовательность операций
Указание на поле ввода пароля	<i>KKKKKKKKHP</i>
Ввод пароля (8 символов)	<i>KKKKKKKKKKHPKKKKKKKK</i>
Перемещение руки на мышь	<i>KKKKKKKKKKHPKKKKKKKKH</i>
Указание на кнопку «Вход»	<i>KKKKKKKKKKHPKKKKKKKKHP</i>
Нажатие кнопки «Вход»	<i>KKKKKKKKKKHPKKKKKKKKHPB</i>

Расставим ментальные операции (табл. 2.9).

Таблица 2.9

#### Расстановка ментальных операций при использовании мыши

Описание	Формируемая последовательность операций
Начальная расстановка операторов <i>M</i> согласно правилу 0 (табл. 2.7). Имя пользователя и пароль — это аргументы команды. Внутри когнитивных единиц операторы <i>M</i> , согласно правилу 2, сразу не ставятся	<i><u>M</u>KKKKKKKKKKHP<u>M</u>KKKKKKKKKKHPB</i>
Удаление ожидаемых операторов <i>M</i> по правилу 1 (табл. 2.7)	<i>MKKKKKKKKKKHP<u>K</u>KKKKKKKKKKHPB</i>
Замена символов операторов соответствующими интервалами согласно табл. 2.6 и подсчет общего времени работы	$1,35 + 0,2 \cdot 8 + 0,4 + 1,1 + 0,2 \cdot 8 + 0,4 + 1,1 + 0,1 = 7,65 \text{ с}$

Оценим эффективность работы с моделью GOMS при использовании кнопки «Tab» на клавиатуре для перемещения между элементами ПИ (действия пользователя показаны в табл. 2.10).

Таблица 2.10

#### Действия пользователя (использование только клавиатуры)

Очередное действие пользователя	Формируемая последовательность операций
Ввод имени пользователя (8 символов)	<i>KKKKKKKKKK</i>
Нажатие на клавишу «Tab» (переход к следующему полю)	<i>KKKKKKKKKK</i>

Окончание табл. 2.10

Очередное действие пользователя	Формируемая последовательность операций
Ввод пароля (8 символов)	<i>KKKKKKKKKKKKKKKKKK</i>
Нажатие на клавишу «Tab» (переход к кнопке «Вход»)	<i>KKKKKKKKKKKKKKKKKKKK</i>
Нажатие кнопки «Вход»	<i>KKKKKKKKKKKKKKKKKKKB</i>

Расставим ментальные операции (табл. 2.11).

Таблица 2.11

**Расстановка ментальных операций при использовании клавиатуры**

Описание	Формируемая последовательность операций
Начальная расстановка операторов <i>M</i> согласно правилу 0 (табл. 2.7). Имя пользователя и пароль — это аргументы команды. Внутри когнитивных единиц операторы <i>M</i> , согласно правилу 2, сразу не ставятся	<i><u>M</u>KKKKKKKK<u>M</u>K<u>M</u>KKKKKKKK<u>M</u>K<u>B</u></i>
Удаление ожидаемых операторов <i>M</i> по правилу 1 (табл. 2.7)	<i><u>M</u>KKKKKKKKKK<u>M</u>KKKKKKKKK<u>B</u></i>
Замена символов операторов соответствующими интервалами согласно табл. 2.6 и подсчет общего времени работы	$1,35 + 0,2 \cdot 9 + 1,35 + 0,2 \cdot 9 + 0,1 = 6,4 \text{ с}$

Анализ двух вариантов работы показывает, что работа с простой для ввода интерфейсной формой только с клавиатуры является более эффективной. Это следует учитывать при разработке ПИ.

Для закрепления полученных теоретических знаний студентам предлагается сравнить по модели GOMS эффективность интерфейса одной из разработанных форм системы при работе с помощью мыши и клавиатуры.

---

## Заключение

---

В учебном пособии были рассмотрены современные принципы проектирования пользовательского интерфейса информационных систем и различные концепции качества пользовательского интерфейса, которые позволяют ответить на вопрос, что такое хороший интерфейс. Кроме того, авторы представили вниманию читателей технологию проектирования пользовательского интерфейса при использовании CASE-средства BPsim.SD.

Для того чтобы закрепить полученные теоретические знания по проектированию, в издании были приведены некоторые практические задачи для разработки интерфейса программного обеспечения; каждая задача была подробно объяснена.

Данное учебное пособие будет полезно студентам при изучении вопросов разработки и проектирования визуального интерфейса пользователя информационных систем.

---

## Список библиографических ссылок

---

1. The Role of BAs in User Interface Design [Electronic resource] // Business analyst learnings [site]. URL: <https://businessanalystlearnings.com/blog/2014/7/29/the-role-of-bas-in-user-interface-design> (accessed: 22.04.2017).
2. Леоненков А. Самоучитель UML 2. СПб. : БХВ-Петербург, 2007. 558 с.
3. Вендров А. М. Проектирование программного обеспечения экономических информационных систем. М. : Финансы и статистика, 2005. 544 с.
4. Алан Купер об интерфейсе. Основы проектирования взаимодействия / пер. с англ. СПб. : Символ-Плюс, 2009. 688 с.
5. Андреев В. О чем надо помнить при разработке пользовательского интерфейса [Электронный ресурс] // Usability в России [сайт]. URL: <http://www.usability.ru/Articles/instruction.htm> (дата обращения: 22.04.2017).
6. Раскин Дж. Интерфейс: новые направления в проектировании компьютерных систем. М. : Символ-Плюс, 2005. 160 с.
7. Андрейчиков А. В., Андрейчикова О. Н. Интеллектуальные информационные системы. М. : Финансы и статистика, 2004. 422 с.
8. Головач В. Дизайн пользовательского интерфейса. Искусство мыть слона [Электронный ресурс] // Юзетикс [сайт]. URL: <http://www.uzethics.ru/lib> (дата обращения: 28.04.2017).
9. Торес Р. Дж. Практическое руководство по проектированию и разработке пользовательского интерфейса / пер. с англ. М. : Вильямс, 2002. 400 с.
10. Жарков С. Shareware: профессиональная разработка и продвижение программ. СПб. : БХВ-Петербург, 2002. 320 с.
11. Руководство по WPF [Электронный ресурс] // Сайт о программировании [сайт]. URL: <https://metanit.com/sharp/wpf> (дата обращения: 28.04.2017).

*Учебное издание*

**Спицина** Ирина Александровна  
**Аксёнов** Константин Александрович

**ПРИМЕНЕНИЕ СИСТЕМНОГО АНАЛИЗА  
ПРИ РАЗРАБОТКЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА  
ИНФОРМАЦИОННЫХ СИСТЕМ**

Редактор М. А. Терновая  
Верстка О. П. Игнатьевой

Подписано в печать 18.10.2017. Формат 70×100/16.  
Бумага офсетная. Цифровая печать. Усл. печ. л. 8,1.  
Уч.-изд. л. 5,3. Тираж 50 экз. Заказ 9

Издательство Уральского университета  
Редакционно-издательский отдел ИПЦ УрФУ  
620049, Екатеринбург, ул. С. Ковалевской, 5  
Тел.: +7 (343) 375-48-25, 375-46-85, 374-19-41  
E-mail: [rio@urfu.ru](mailto:rio@urfu.ru)

Отпечатано в Издательско-полиграфическом центре УрФУ  
620083, Екатеринбург, ул. Тургенева, 4  
Тел.: +7 (343) 358-93-06, 350-58-20, 350-90-13  
Факс: +7 (343) 358-93-06  
<http://print.urfu.ru>



