ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ
ПИ (филиал) ДГТУ в г. Таганроге

ЦМК «Прикладная информатика»

ПРАКТИКУМ

по дисциплине

«Управление проектами»

Авторы
Андриян О.В.,
Андриян И.В.

Ростов-на-Дону, 2025

Аннотация

Практикум по выполнению практической работы по дисциплине «Управление проектами». ПИ (филиал) ДГТУ в г. Таганроге, 2025 г.

В практикуме кратко изложены теоретические вопросы, необходимые для успешного выполнения практической работы, рабочее задание и контрольные вопросы для самопроверки.

Предназначено для обучающихся по специальности 09.02.07 Информационные системы и программирование.

Авторы

Председатель ЦМК «Прикладная информатика»
Андриян О.В.

Преподаватель «Прикладная информатика»
Андриян И.В.

Оглавление

Введение

В учебно-методическом пособии к практикуму по курсу «Управление проектами» изложены сведения, необходимые для успешного выполнения практических занятий по данному курсу. Описан процесс работы с инструментарием, применяемым на практических занятиях, представлен ряд типичных задач и подходы к их решению. Практические занятия посвящены углубленному знакомству обучающихся с текстовыми редакторами, графическими редакторами, электронными таблицами, базами данных, архиваторами, виртуальными средами, операционными системами, способами безопасности, основы работы с реестром и облачными технологиями. Цель настоящего пособия – помочь обучающимся при выполнении практических работ, выполняемых для закрепления знаний по теоретическим основам и получения практических навыков работы на компьютерах.

Обучающийся должен знать: задачи планирования и контроля развития проекта; современные стандарты качества программного продукта и процессов его обеспечения; стандартные метрики по прогнозированию затрат, сроков и качества; способы измерения характеристик программного проекта; способы оптимизации программного кода с использованием специализированных программных средств.

Общие положения. Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний. Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет). Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию. Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующего задания.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Оформление согласно следующим требованиям:

– расстояние от верхней и нижней строки текста до верхней и нижней рамки должно быть не менее 10 мм;

– гарнитура шрифта – Times New Roman;

– размер шрифта для основного текста – 14;

– междустрочный интервал – 1,5
– размер шрифта для примечаний, ссылок – 12;
– абзацный отступ –1,25 мм;
– выравнивание основного текста – по ширине страницы.
Для заполнения ячеек основной надписи:
– гарнитура шрифта Arial;
– курсив;
– для обозначения работы размер – 20.

Практическая работа № 1. Анализ программных средств для управления проектами

Цель работы: изучить базовые отличия программ и выделить их отличия.

Теоретическая часть

1. Объективные критерии: Перед началом сравнительного анализа необходимо определиться с объективными критериями, по которым будет оцениваться каждое программное средство. Критерии должны быть конкретными, измеримыми, достижимыми, актуальными и ограниченными по времени.

Пример объективных критериев

Стоимость: стоимость лицензии, ежемесячная абонентская плата, наличие бесплатных версий.

Функциональность: возможности планирования, управления задачами, отслеживания прогресса, ведения документации, коммуникации, отчетности.

Интеграция: возможность интеграции с другими сервисами (например, почта, календарь, CRM).

Удобство использования: интуитивность интерфейса, простота освоения, доступность обучающих материалов.

Поддержка: качество и скорость технической поддержки, доступность документации, активность сообщества пользователей.

2. Выбор программных средств. Для сравнительного анализа необходимо выбрать 5 программных средств для управления проектами, которые отвечают требованиям конкретного проекта или организации. При выборе следует учитывать:

– размер и сложность проекта: небольшие проекты могут не требовать сложного программного обеспечения, а для крупных проектов необходимы мощные инструменты;

– бюджет проекта: стоимость программного обеспечения может значительно отличаться;

– требования к функциональности: необходимо учесть, какие функции необходимы для успешного ведения проекта;

– опыт работы с программным обеспечением: если у команды есть опыт работы с каким-то программным средством, это может быть преимуществом;

– отзывы пользователей: изучение отзывов других пользователей может помочь в выборе подходящего программного средства.

3. Анализ по критериям: После выбора программных средств необходимо оценить их по выбранным объективным критериям. Для этого можно использовать таблицу сравнения, где в строках будут указаны программы, а в столбцах – критерии. В ячейках таблицы необходимо указать информацию по каждому критерию для каждой программы.

4. Заключение: Проведение сравнительного анализа программных средств для управления проектами – важный этап в выборе подходящего инструмента для успешного ведения проекта. Отчет, составленный на основе анализа, поможет выбрать программное средство, которое максимально соответствует требованиям проекта и обеспечит эффективную работу команды.

Рабочее задание: необходимо провести сравнительный анализ 5 программных средств для управления проектами, по 5 выбранным объективным критериям. Результаты представить в виде отчета.

Время работы: 2 часа.

Практическая работа №2, ч. 1. Управление проектом в ProjectLibre

Цель работы: изучить основы управления проектом в ProjectLibre

Теоретическая часть

Согласно национальному российскому стандарту в области управления проектами «Проект (Project) — целенаправленная деятельность временного характера, предназначенная для создания уникального продукта или услуги, ограниченная во времени и связанная с потреблением ресурсов». В рамках проекта обязательно должны быть четко обозначены:

- цель (цели) и запланированный результат (результаты);
- качество;
- этапы, сроки выполнения работ;
- бюджет (стоимость).

При планировании проекта обычно выделяют три основных ограничения: по бюджету, по времени, по ресурсам.

Управление проектом (Project Management) — это процесс планирования, организации и контроля за состоянием задач и ресурсов проекта, направленный на своевременное достижение цели проекта в рамках заданного бюджета и сроков. Процесс управления проектом должен обеспечить решение следующих задач:

• соблюдение сроков проекта;

• правильное распределение материальных ресурсов и рабочего времени исполнителей между задачами проекта и во времени;

• коррекция исходного плана при необходимости, если реальное положением дел не соответствует прогнозу

Microsoft Project [5] — наиболее распространенный во всем мире программный продукт, предназначенный для управления проектами. Приложение Microsoft Projec имеет интуитивно-понятный интерфейс (родственный с Microsoft Office) и все необходимые компоненты, необходимые менеджеру проекта для управления планом и его ресурсами.

ProjectLibre [1] — аналог Microsoft Project. В отличие от Microsoft Project является бесплатным приложением. Аналогично с Microsoft Project, разработана для управления проектами. Программа является кроссплатформенной и совместима со следующими операционными системами: Microsoft Windows,

Linux, Mac OS X. Приложение поддерживает основные европейские языки. Кроме того, включена поддержка русского языка. К основным функциональным возможностям можно отнести: поддержку тех форматов файлов, которые используются в Microsoft Project 2010; Ribbon интерфейс; метод освоенного объема (Earned Value costing); использование диаграмм Ганта.

Рассмотрим пример создания и управления проектом в программе ProjectLibre.

Пусть у нас имеется краткое ТЗ по созданию ПС по автоматизации учета медикаментов в ФАП (Фельдшерско-акушерском пункте), в котором указана информация по входным данным, требуемом функционале, выходных данных.

Входные данные:

• Накладная с указанным номером и датой. Накладная содержит список медикаментов с указанием названия, цены и количества упаковок. Список может содержать как лекарственные препараты, так и другие медицинские средства (шприцы, бинты и т.д.)

• Данные о расходе. В конце каждого рабочего дня сотрудник ФАП фиксирует данные об израсходованных медикаментах: из какой партии (номер накладной) израсходованы медикаменты, сколько полных упаковок израсходовано и сколько единиц в открытых упаковках израсходовано.

Должен быть реализован следующий функционал:

• просмотр информации об имеющейся номенклатуре;

• ввод данных о новом приходе;

• ввод данных о расходе;

• составление отчета об израсходованных медикаментах за указанный период и остатках на конец отчетного периода.

Выходные данные:

Отчет об израсходованных медикаментах за указанный период и остатках на конец отчетного периода. Отчет должен быть в виде таблицы. Колонки таблицы: Название, Номер накладной, Цена, Расход полных упаковок, Расход в открытых упаковках, Стоимость расхода, Остаток полных упаковок, Остаток в открытых упаковках, Остаточная стоимость. Строки таблицы: Номенклатура, сгруппированная по стандартному наименованию.

Рассмотрим по шагам пример создания и управления проектом в программе ProjectLibre.

Создание проекта

При запуске программы предлагается два варианта действия: <Создать проект и <Открыть проект>. Выбираем <Создать проект>. В открывшемся окне указываем название проекта, менеджера (можно указать фамилию и инициалы) и дату окончания проекта (рис. 1). Если не известна дата окончания проекта, то необходимо установить галочку <Планирование вперед> и указать дату начала проекта. При необходимости можно указать какие-либо замечания (комментарии) по проекту. После заполнения всех необходимых данных нажимаем на кнопку <ОК>.

После нажатия на кнопку <ОК> откроется главное окно программы, которое состоит из меню и области данных проекта (рис. 2).

Чтобы не потерять данные, сразу сохраняем проект на диске. Для этого нажимаем кнопку <Сохранить проект> и в открывшемся окне указываем путь и имя файла (рис. 3).

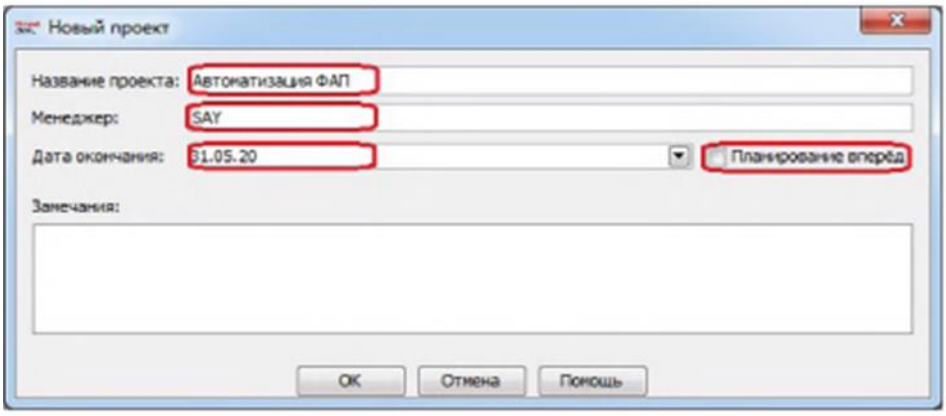


Рисунок 1 – Создание проекта

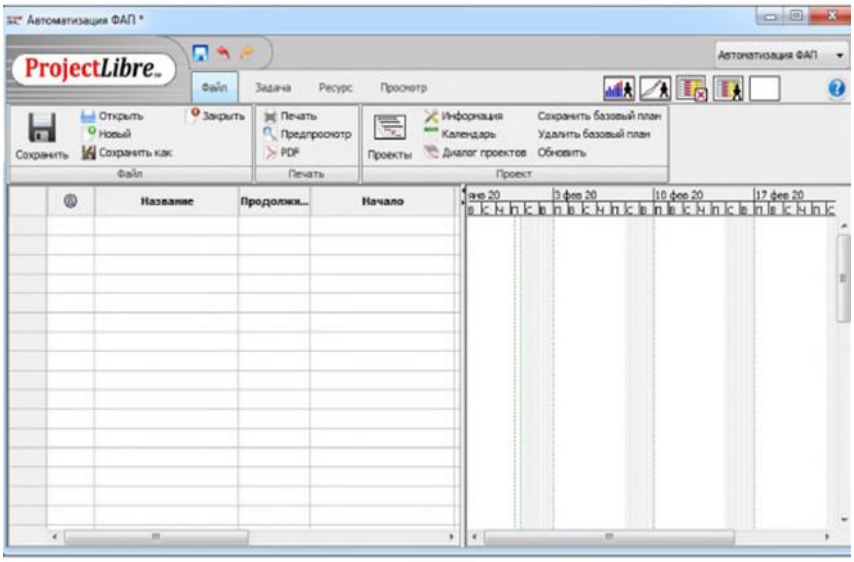


Рисунок 2 – Главное окно программы

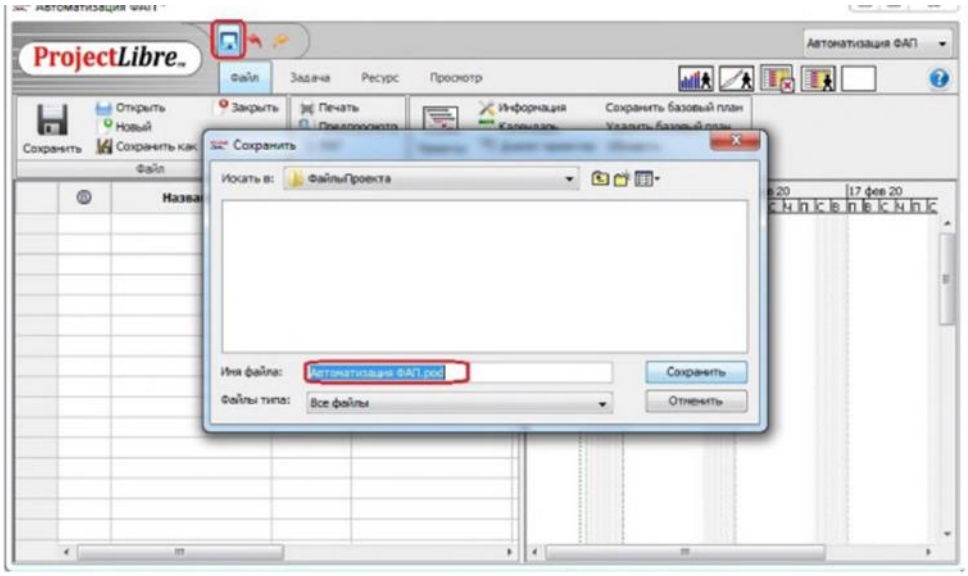


Рисунок 3 - Сохранение проекта

Добавление задач

Основная работа в программе ProjectLibre заключается в составлении списка задач. Добавим задачу «Формулировка требований». Для этого в колонке «Название» таблицы задач вводим нужный текст (рис. 4). После этого двойным нажатием <ЛКМ> переходим на редактирование свойств задачи (рис. 5). В окне редактирования свойств задачи указываем продолжительность и время начала (или время окончания). Нажимаем на кнопку <Закрыть>.

На главном окне слева отображается добавленная задача со всеми заполненными колонками, а справа отображаются данные о продолжительности задач в виде диаграммы Ганта (рис. 6). Добавим остальные задачи. В нашем случае укрупненно это будут:

• разработка функционала;

• тестирование разработанного функционала;
• составление руководства пользователя;
• внедрение НС в ФАП;

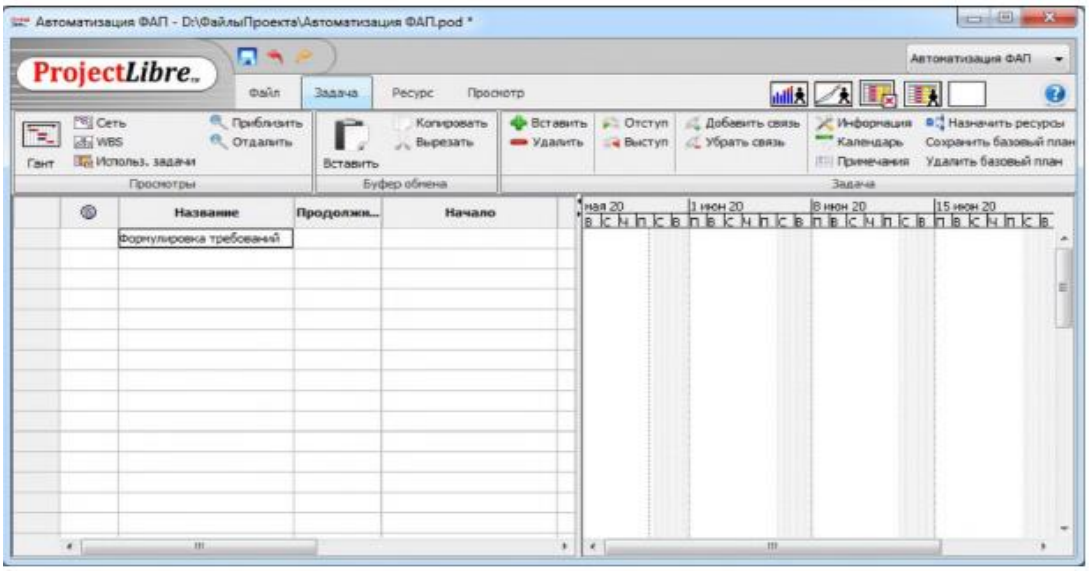


Рисунок 4 - Добавление задачи

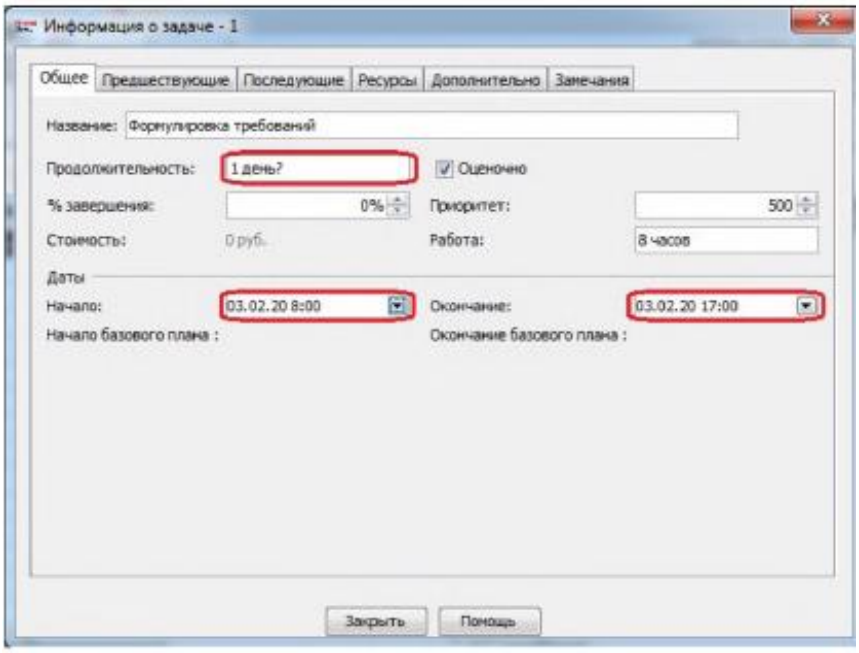


Рисунок 5 – Редактирование свойств задачи

• обучение пользователей;
• поддержка пользователей.

Даты начала и завершения распределим так, чтобы задачи не пересекались.

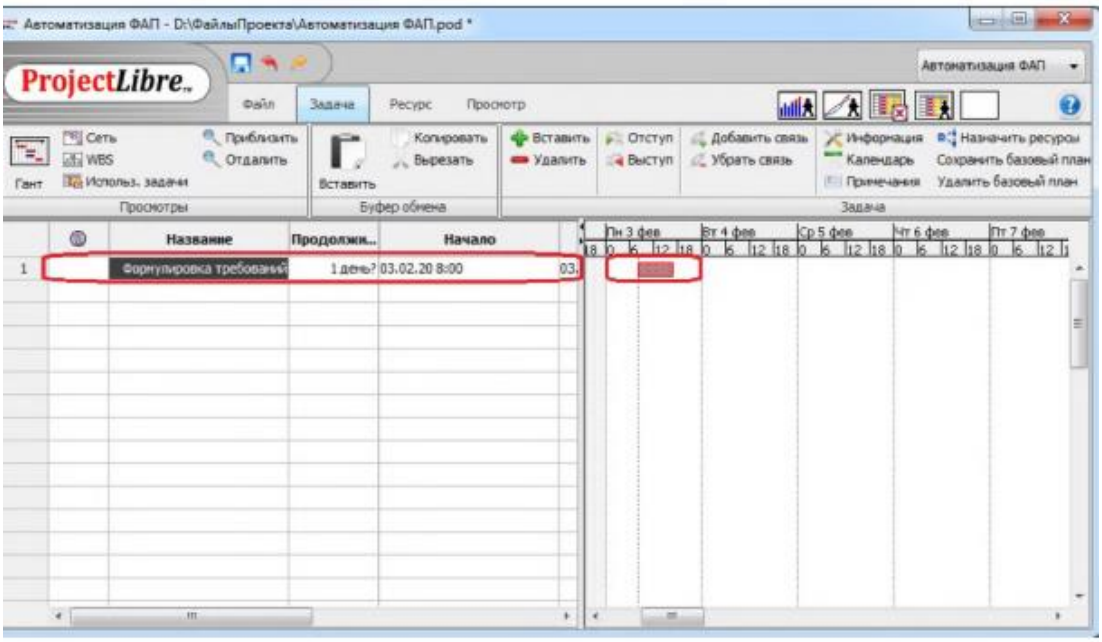


Рисунок 6 – Добавленная задача на главном окне

По диаграмме Ганта видно, что каждая последующая задача начинается только после завершения предыдущей задачи (рис. 7).

Добавление вложенных задач

Распишем задачу «разработка функционала» более подробно. Для этого выбираем строку, следующую за данной и нажимаем <Вставить> (рис. 8). В добавленной строке вводим название подзадачи и нажимаем на кнопку <Отступ> (рис. 9). Аналогично добавляем остальные подзадачи и указываем их продолжительность.

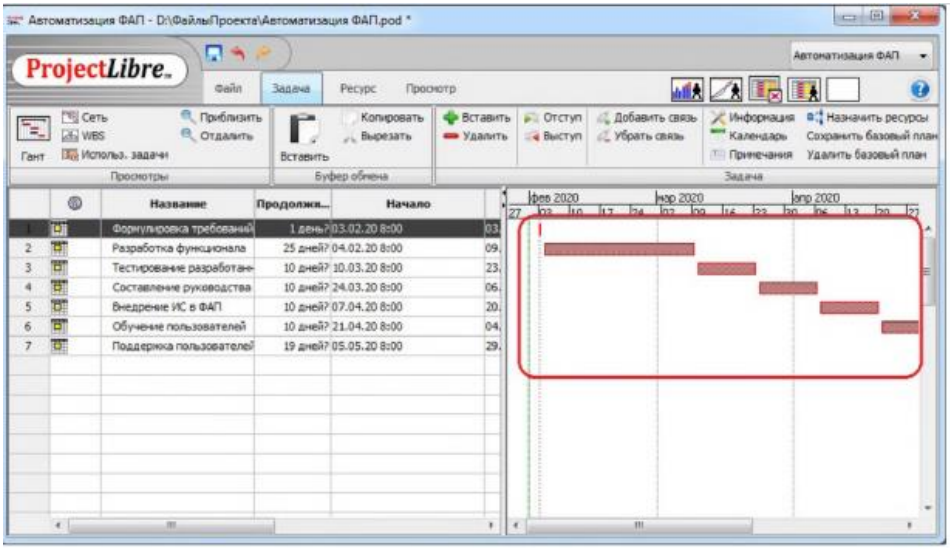


Рисунок 7– Добавление остальных задач

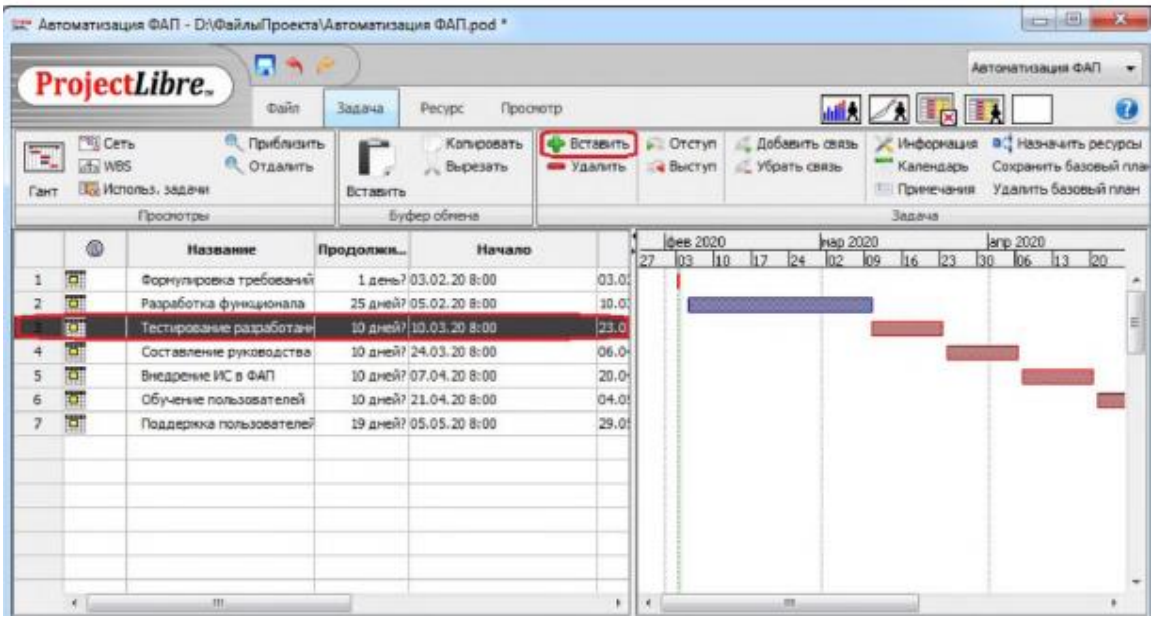


Рисунок 8 – Добавление вложенной задачи

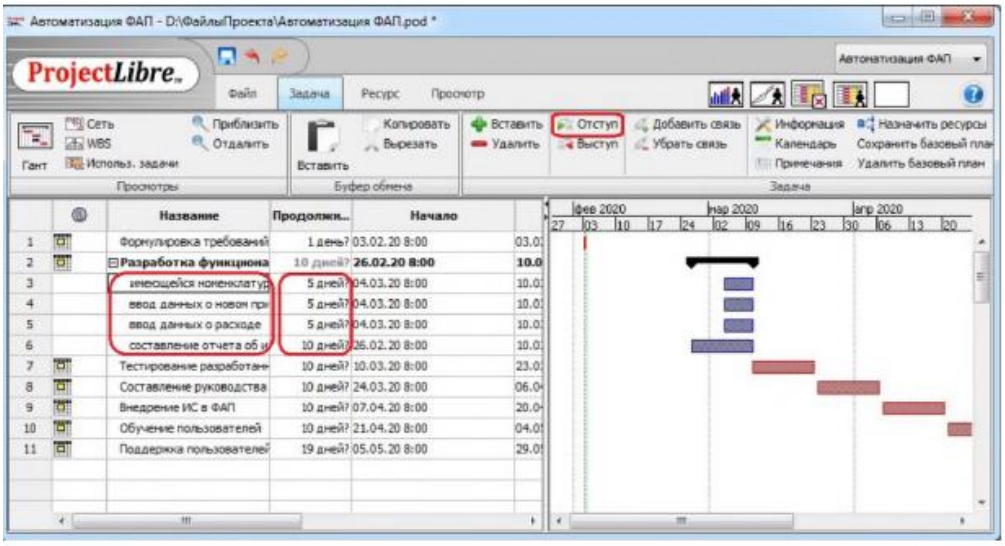


Рисунок 9 - Добавление вложенных задач

Установка связей

Очевидно, что не все задачи могут быть выполнены параллельно, и не всегда требуется ждать завершения одной задачи, чтобы приступить к пополнению другой. Для этого в ProjectLibre используются связи. Выделяем две задачи и нажимаем кнопку <Добавить связь> (рис. 10). Допустимы следующие типы связей:

- FS — предшественник заканчивается, и начинается следующая задача;
- SF — начало предшественника определяет окончание следующей задачи;
- FF — обе задачи заканчиваются одновременно;
- SS — задача начинается одновременно с предшественником.

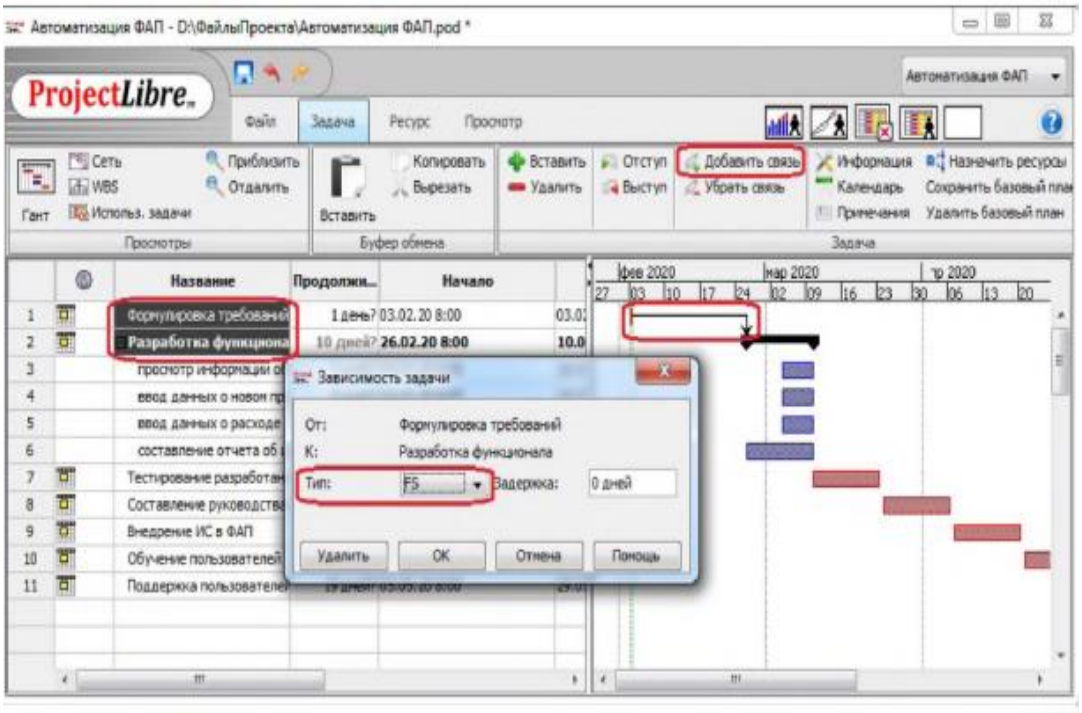


Рисунок 10 – Добавление связи между задачами

По умолчанию связь имеет тип FS. Если необходимо выбрать другой тип, то нажимаем <ЛКМ> на появившейся стрелке на диаграмме Ганта, в открывшемся окне выбираем тип связи и нажимаем кнопку <ОК>.

Аналогично добавим остальные связи между задачами (рис. 11). В некоторых случаях задачи могут зависеть от различных факторов, вызывающих задержку либо необходимость одновременного выполнения, из-за чего приходится указывать задержки или раннее начало в описываемой связи. Задержку или раннее начало можно указывать как в единицах времени, так и в процентах от длительности предшествующей задачи. Чтобы указать задержку, необходимо открыть форму редактирования свойств связи (<ЛКМ> на связи) и ввести величину задержки в днях.

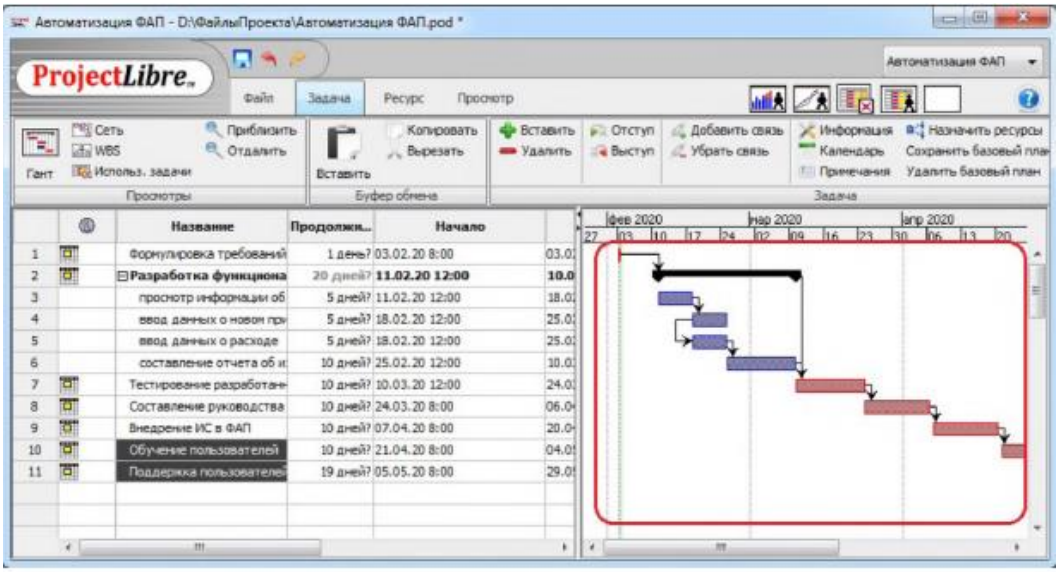


Рисунок 11 - Добавление всех связей

Для того, чтобы разделить задачу на несколько частей так, чтобы части можно было выполнять отдельно, необходимо нажать <ПКМ> на задаче в диаграмме Ганта и в всплывающем меню выбрать пункт <Разделить>.

Установка ограничений

Может быть так, что некоторые задачи нужно завершить к определенной дате. Кроме того, может возникнуть необходимость указания промежуточных вех. Можно учитывать данные особенности при планировании путем назначения задачам так называемых ограничений. Существует 8 типов ограничений (рис. 12), разделенных на две категории — гибкое ограничение и негибкое ограничение. Для указания ограничения необходимо открыть форму редактирования свойств задачи, на вкладке <Дополнительно> выбрать нужное ограничение и указать дату.

Негибкие ограничения значительно ограничивают возможность планирования, а гибкие ограничения позволяют ProjectLibre рассчитать расписание и выполнить соответствующие изменения в зависимости от указанного ограничения. Негибкие ограничения могут вызвать конфликты между парами последовательных задач, из-за чего может потребоваться исключение таких ограничений.

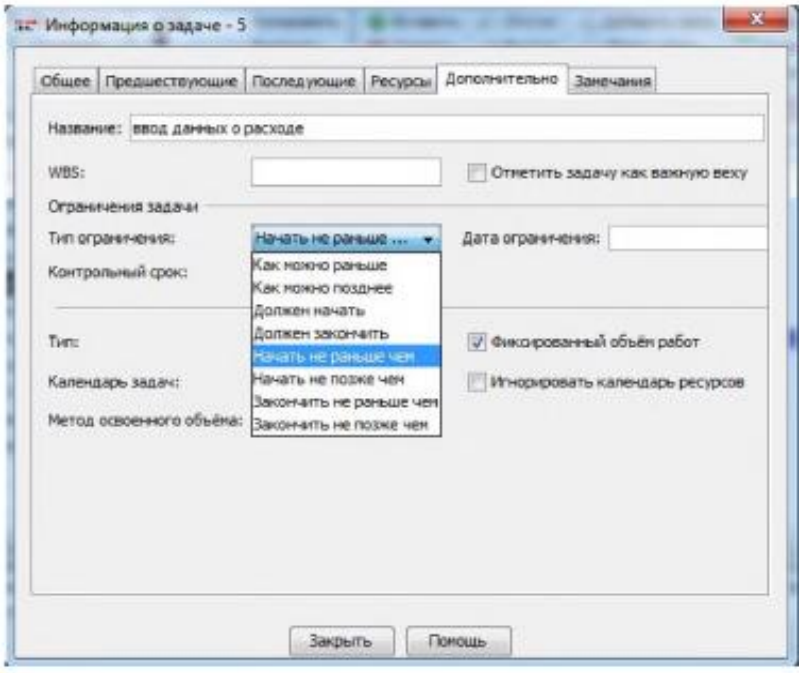


Рисунок 12 – Ограничения

В таких случаях рекомендуется установка даты контрольного срока вместо негибкого ограничения, что никак не влияет на расписание задачи.

Если задача не выполнена в срок, то ProjectLibre укажет об этом крестиком в красном кружке в столбце Индикатор диаграммы Ганта.

Рабочее задание:

1.Создать проект по выбранной теме (либо по указанной преподавателем теме). Требования к проекту: •дата завершения — конец учебного семестра; • основной календарь — пятидневка.

2. Сохранить файл проекта.

3. Добавить задачи в проект. Требования к задачам: • число задач должно быть не меньше 10; • обязательно наличие как минимум двух уровней вложенности.

4. Оценить примерную продолжительность каждой задачи.

5. Распределить задачи по времени с учетом того, что исполнитель будет один.

6. Сохранить проект.

Время работы: 2 часа.

Практическая работа №2, ч. 2. Управление проектом в ProjectLibre

Цель работы: изучить управление ресурсами проекта в программе ProjectLibre

Теоретическая часть

Ресурсы

Ресурсы бывают двух типов:

• Работа — выполняют задачи путем затраты некоторого количества времени. Обычно это люди (рабочие), назначенные выполнять работу в рамках проекта. Также в качестве рабочего ресурса может быть использовано некоторое оборудование.

• Материал — ресурсы, которые представлены в виде расходных материалов и компонентов, используемых при выполнении задач. Материалы можно отслеживать и назначать на задачи.

Чтобы добавить ресурсы нажимаем на меню <Ресурс —> Ресурсы> (рис. 13) и в таблице вводим необходимые данные. Для всех ресурсов указываем название и тип. Для материальных ресурсов указываем единицы измерения.

В столбце «Максимальные единицы» указывается максимально возможная загрузка, которая допустима для ресурса во время выполнения задач в определенный период времени. По умолчанию значения указываются в процентах. Например, если указано 100%, то этот ресурс будет работать восемь часов в случае восьмичасового рабочего дня. Если указано 50%, то ресурс будет работать четыре часа в день. В случае набора ресурсов, состоящего из 2 работников, максимальных единиц будет 200%.

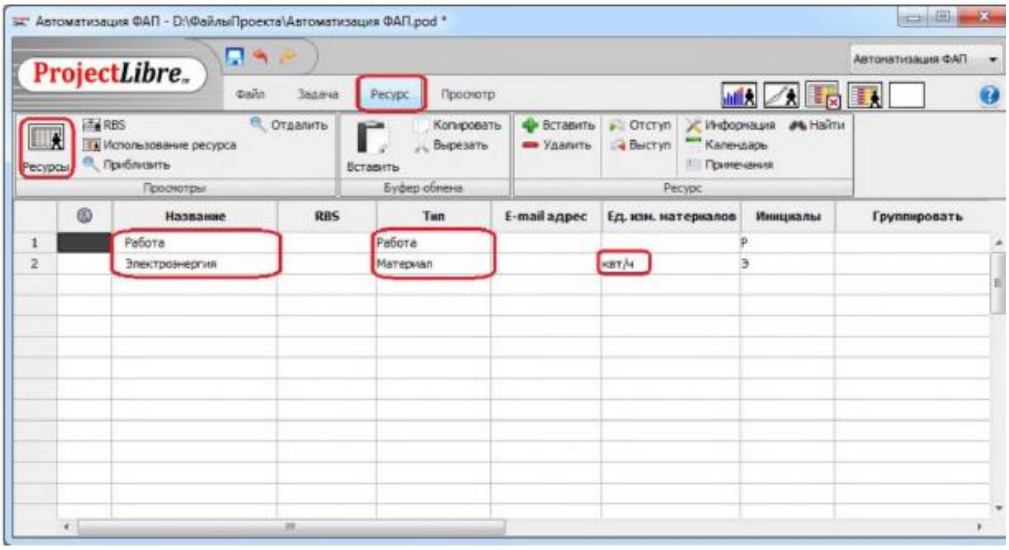


Рисунок 13 – Добавление ресурсов

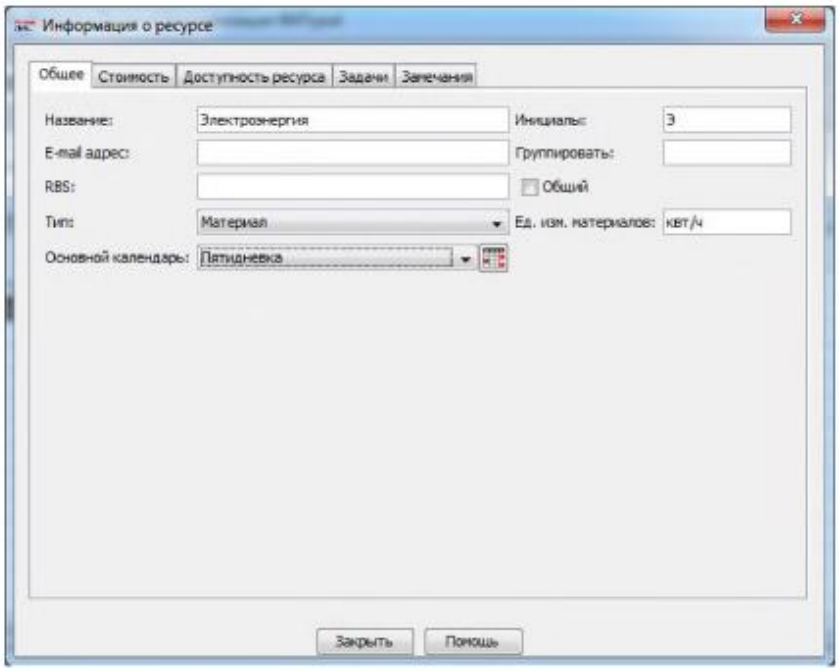


Рисунок 14 - Редактирование информации о ресурсе

В отличие от работы, материал не содержит величины «Максимальные единицы». Такие ресурсы содержат стоимость потребления, которая может быть, как фиксированной, так и переменной.

Для просмотра или редактирования информации по одному ресурсу нужно открыть форму «Информация о ресурсе». Для этого выделяем нужный ресурс и нажимаем на кнопку <Информация> (либо открываем форму двойным нажатием <ЛКМ> на нужном ресурсе) (рис. 14).

Назначение ресурсов на задачи

Откроем форму назначения ресурса. Для этого переходим в меню <Задача —> Назначить ресурсы>. В открывшемся окне выбираем нужный ресурс и нажимаем <Назначить> (рис. 15).

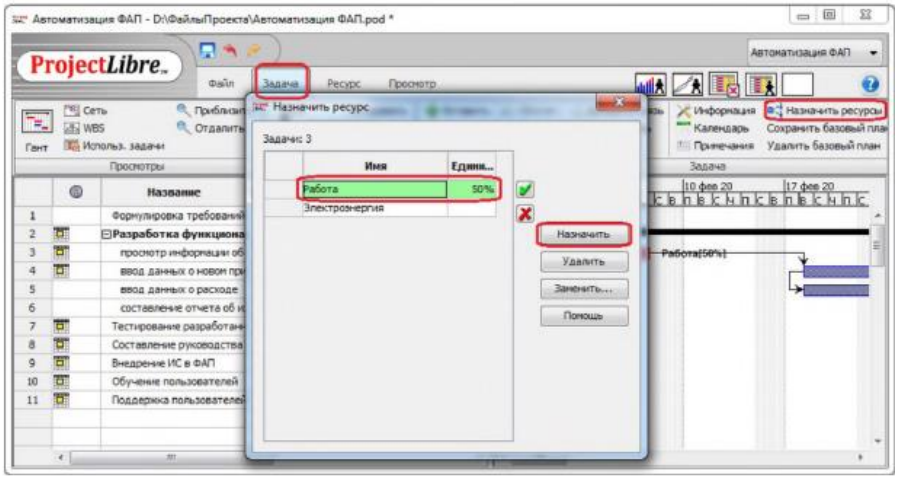


Рисунок 15 – Назначение ресурсов

Типы задач

В ProjectLibre используются три типа задач, позволяющие пересчитывать значения переменных, когда на эти задачи назначены ресурсы:

- Фиксированные единицы;
- Фиксированная работа;
- Фиксированная продолжительность.

Каждый тип определения основывается на том, что одна из переменных фиксируется.

Для изменения типа задачи открываем окно Информация о задаче, на вкладке Дополнительно выбираем нужный тип задачи и нажимаем кнопку <Закрыть> (рис. 16). Аналогично устанавливаем типы и назначаем ресурсы остальным задачам (рис. 17).

Внесение информации о стоимости

Чтобы узнать общую стоимость работ, необходимо установить стоимость ресурсов. Для этого в меню выбираем <Ресурс —> Ресурсы>, выбираем нужный ресурс и нажимаем кнопку <Информация>.

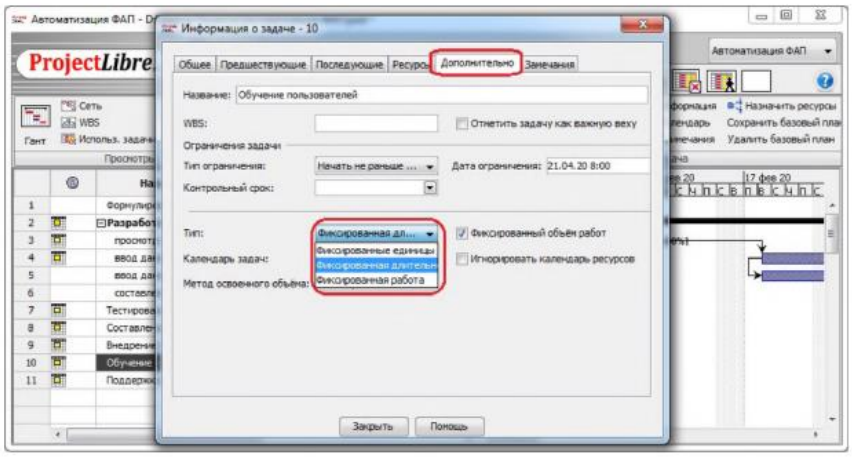


Рисунок 16 – Изменение типа задачи

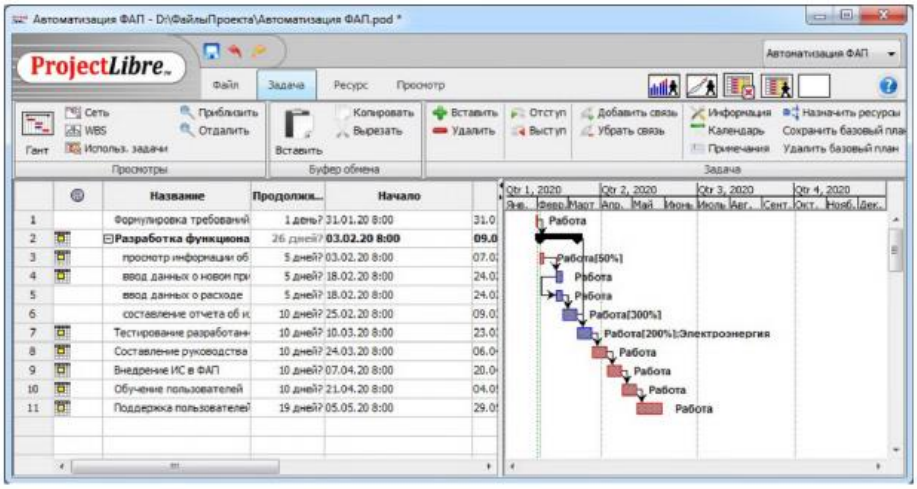


Рисунок 17 – Назначение всех ресурсов

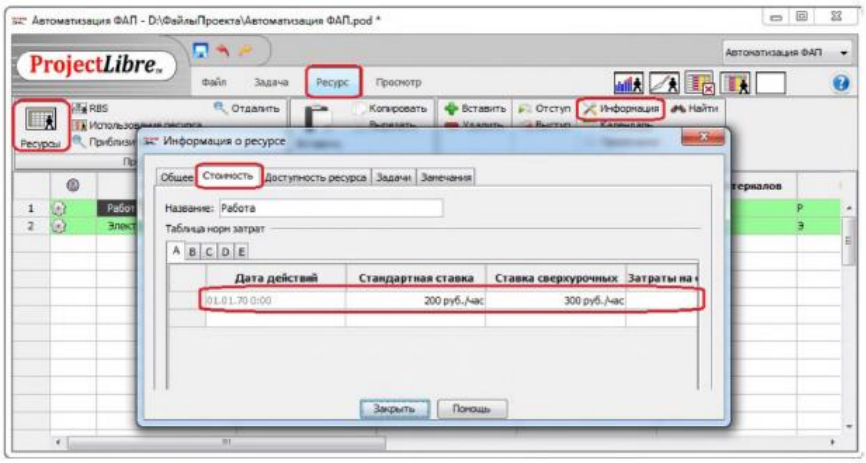


Рисунок 18 - Установка стоимости

В открывшемся окне на вкладке Стоимость указываем значения (рис. 18). Если стоимость со временем меняется, то новые значения задаются на следующих строках таблицы.

Рабочее задание:

1. Открыть проект, созданный в практической работе №1.

2. Установить связи между задачами. Требования к связям: • связи должны быть указаны для всех задач; • должны быть использованы не менее двух типов связей.

3. Установить ограничения на задачи. Требования к ограничениям: • должны быть использованы не менее двух типов ограничений.

4. Сохранить проект.

Время работы: 2 часа.

Практическая работа №2, ч. 3. Управление проектом в ProjectLibre

Цель работы: изучить управление продолжительностью проекта в программе ProjectLibre

Теоретическая часть

Контуры

Ресурсы — это обычно рабочие, и они могут вносить непредсказуемость (например, они могут уволиться, выйти на больничный, не справиться с задачей и т.д.). В назначениях и распределении работ необходимо учитывать такие риски.

ProjectLibre позволяет учитывать ресурсы для различных дат начала или окончания задачи, делать перерывы в назначениях. Это позволяет ресурсам работать над другими задачами при необходимости, а также назначать сверхурочные работы — иными словами, возможно изменение контуров назначений.

Контур в ProjectLibre — это форма распределения работ внутри назначения. По умолчанию, если ресурсу назначают 100% использование, то задача будет начата немедленно, как только станет доступной. Это так называемый плоский контур, где работа каждого подразделения равномерно распределяется по всей длительности задачи.

Иногда может потребоваться изменить распределение работы над заданием, применяя заданный контур. ProjectLibre содержит 8 готовых контуров:

• Плоский — контур по умолчанию с равномерным распределением работы;

• Загрузка в конце — пик активности возникает в конце проекта;

• Загрузка в начале — пик активности возникает в начале проекта;

• Двойной пик — в проекте существуют два пика активности;

• Ранний пик — такой же, как на начальном этапе, но с рампой до пика активности;

• Поздний пик — пик активности в конце проекта, но с рампой;

• «Колокол» — одиночный пик активности в середине проекта;

• «Черепаха» — плоская загрузка без выраженных пиков, с рампами в начале и в конце.

Для указания контура выбираем в меню <Ресурс —> Использование ресурса>(или <Задача —> Использование задачи>) и в колонке «Профиль загрузки» открывшейся таблицы назначений выбираем нужный контур (рис. 19).

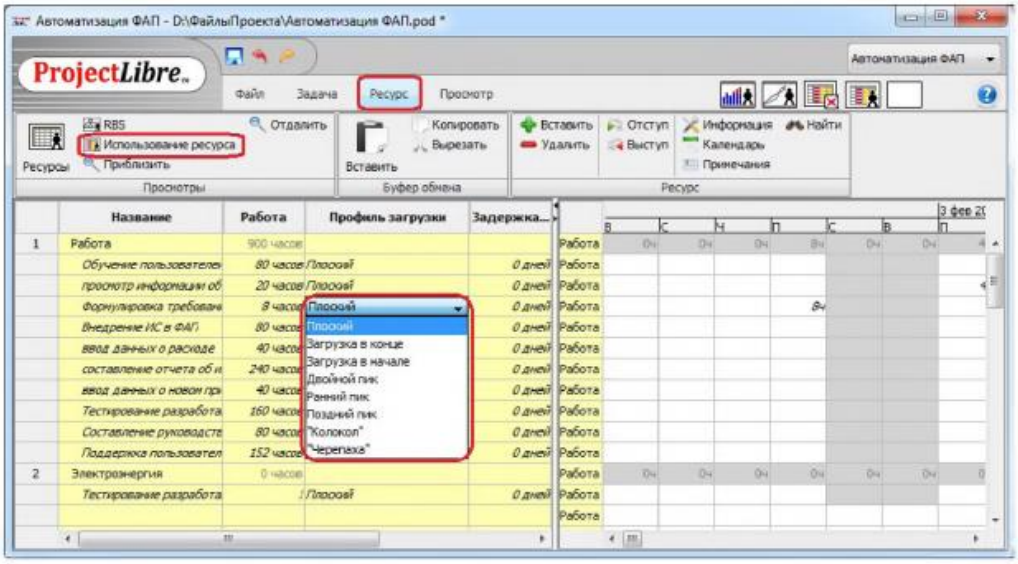


Рисунок 19 – Выбор контура

Перегруженные ресурсы

Ресурс перегружен, когда общая сумма в какой-либо отрезок времени работ превышает доступное количество единиц ресурса. Это может быть в тех случаях, когда один и тот же ресурс запланирован в разных задачах либо в проекте могут оказаться случайные превышения.

Превышения ресурсов можно найти на гистограмме загрузки ресурсов. Для открытия гистограммы выбираем в меню <Просмотр —> Гистограмма> (рис. 20).

При обнаружении перегруженных ресурсов для их устранения можно:

• задержать выполнение задачи;

• разделить задачу;

• назначить дополнительные ресурсы или назначить другие ресурсы;

• проверить значение переменной Оставшееся наличие, чтобы узнать доступные интервалы времени для ресурса.

ProjectLibre имеет функцию выравнивания, которая может разрешить проблемы с превышением ресурсов путем разделения или задержки задач. Выравнивание выполняется с использованием следующих данных:

• доступное время;

• приоритеты, зависимости и ограничения задач;

• идентификаторы задач;

• даты расписания.

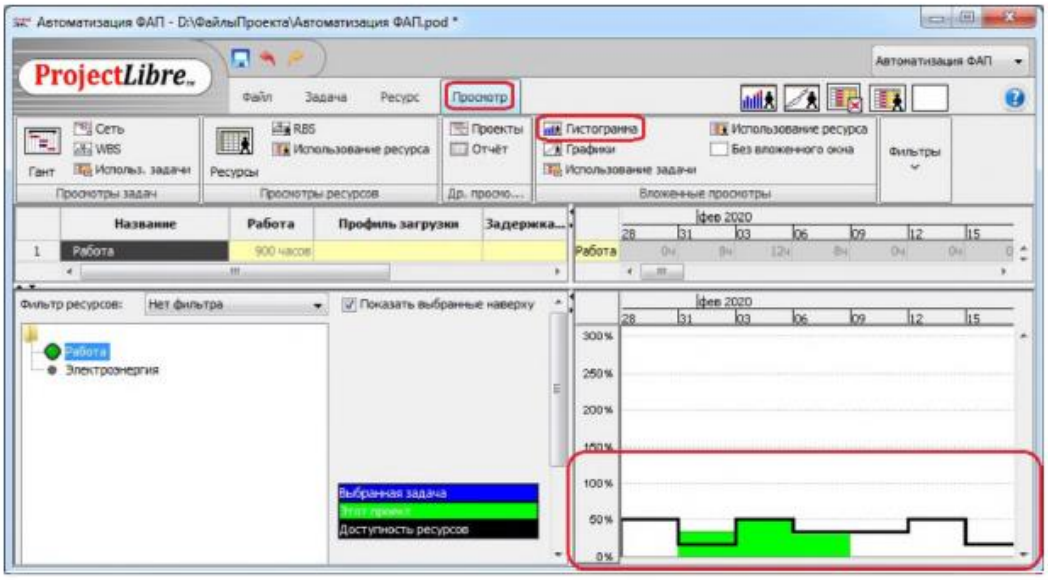


Рисунок 20 – Просмотр гистограммы загрузки ресурсов

Рабочее задание:

1.Создать проект по выбранной теме (либо по указанной преподавателем теме). Требования к проекту: • дата завершения — конец учебного семестра; • основной календарь — пятидневка.

2. Сохранить файл проекта.

3. Добавить задачи в проект. Требования к задачам: • число задач должно быть не меньше 10; • обязательно наличие как минимум двух уровней вложенности.

4. Оценить примерную продолжительность каждой задачи.

5. Распределить задачи по времени с учетом того, что исполнитель будет один.

6. Сохранить проект.

Время работы: 2 часа.

Практическая работа №2, ч. 4. Управление проектом в ProjectLibre

Цель работы: изучить возможности программы ProjectLibre для анализа проекта

Теоретическая часть

Отслеживание работы

Кроме качественного управления проектом, хороший руководитель должен гарантировать, что все задачи проекта будут решены в срок и в рамках бюджета. Требуется упорядоченный подход к отслеживанию хода выполнения задач, позволяющий предвидеть последствия, вызываемые отставанием от графика, пересмотром планов, перераспределением ресурсов.

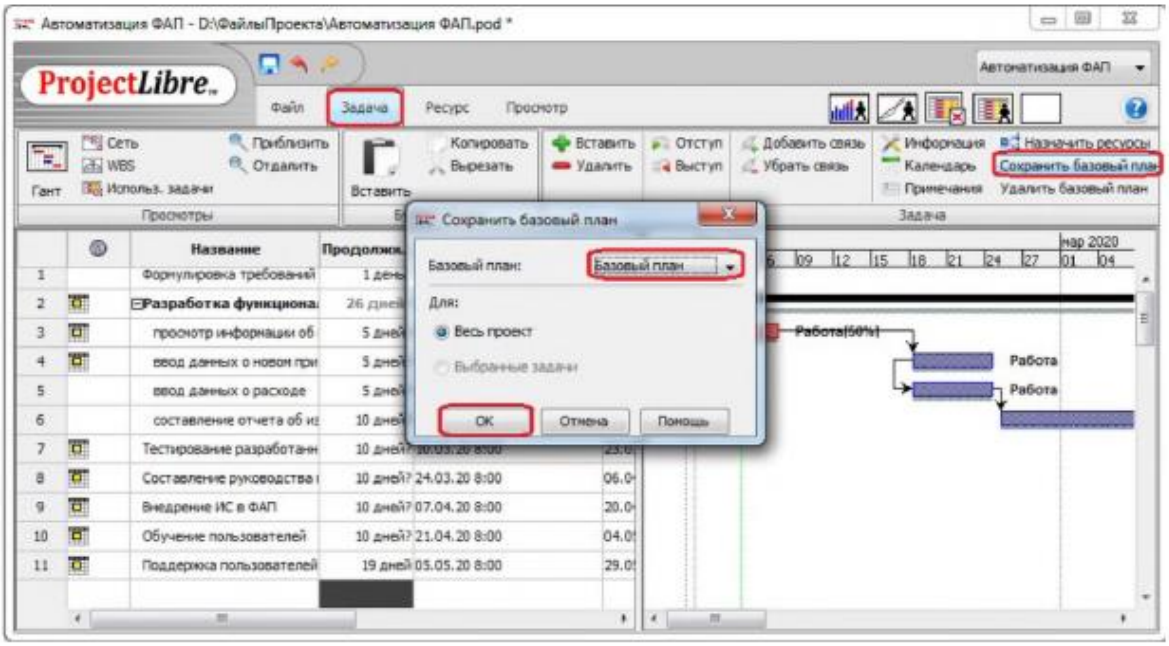


Рисунок 21 – Сохранение базового плана

Подход заключается в сохранении начального проекта в качестве базового с дальнейшим сравнением его с фактическим графиком работ. Дата, продолжительность, стоимость работ — это поля в текущем расписании, в которые вводятся фактические даты и затраты, показывающие фактический ход проекта.

Сохраним базовый план. Для этого в меню выбираем <Задача —> Сохранить базовый план>, в открывшемся окне выбираем имя базового плана и нажимаем кнопку <ОК> (рис. 21).

Для задач и для назначений существуют базовые и фактические поля. Расхождение полей показывают разницу между текущим и базовым значением для каждой задачи. Положительное отклонение означает, что есть отставание от плана.

На диаграмме Ганта красным либо синим цветом отображаются фактические данные, а серым — базовый план. Внесем изменение в план, а именно увеличим продолжительность 11-й задачи на 10 часов. На рисунке 22 стрелкой обозначено то место, в котором можно увидеть расхождение от базового плана.

С помощью меню (<ПКМ> на диаграмме Ганта) можно по своему усмотрению настраивать отображаемые данные (рис. 22).

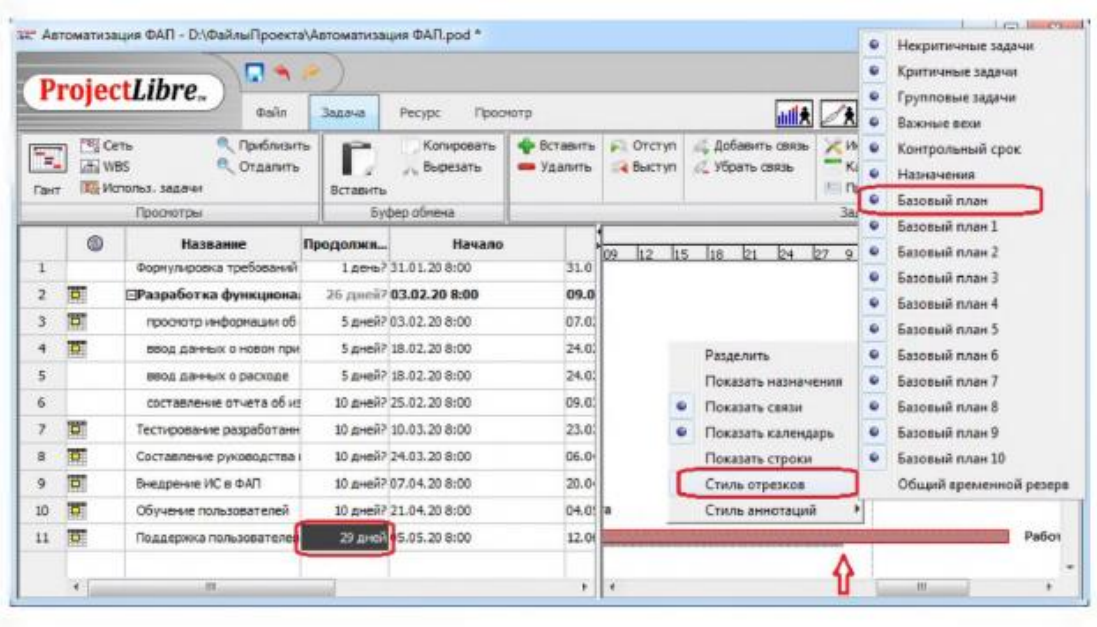


Рисунок 22 – Отслеживание работ

Для того, чтобы принять изменения по текущим выполненным работам, необходимо обновить проект. Для этого в меню нажимаем <Файл —> Обновить^ в открывшемся окне выбираем дату, на которую введены фактические данные, и нажимаем кнопку <ОК> (рис. 23).

Построим отчет по задачам по отклонению фактических данных от базового плана. Для этого в меню нажимаем <Просмотр —> Отчет>, выбираем отчет «Информация по задачам» («Task Information») и колонки «Отклонение расписания» (рис. 24). В правом нижнем углу видим искомое отклонение. Там же можно посмотреть все интересующие отчеты.

Анализ эффективности

Анализ плана и фактического исполнения работ представляет собой набор простых вычислений. Наиболее важными параметрами для расчета отклонений и коэффициентов, позволяющими оценить производительность и управление, являются:

• Базовая стоимость запланированных работ, или повременные затраты по базовому плану на дату отчета.

B C W S = <базовая стоимость часа> • <количество базовых часов>

• Базовая стоимость выполненных работ или базовая стоимость фактически затраченных часов.

B C W P = <базовая стоимость часа> • <фактическое количество часов>

• Фактическая стоимость выполненных работ или фактические затраты, понесенные для задач.

A C W P = <фактическая стоимость часа> ^фактическое количество часов>

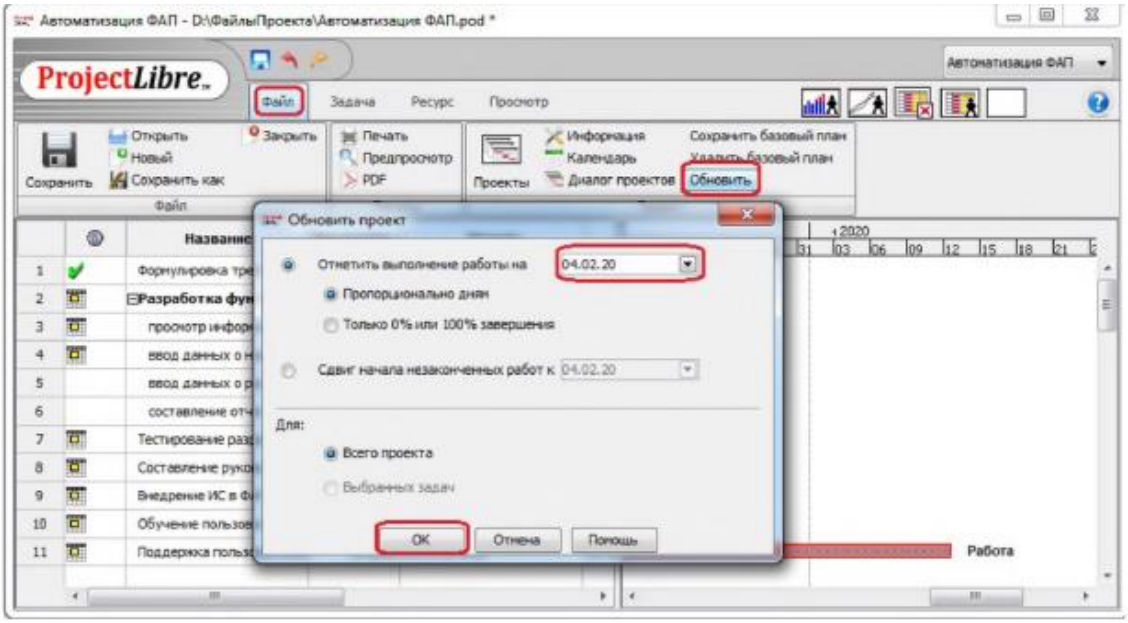


Рисунок 23 - Обновление проекта

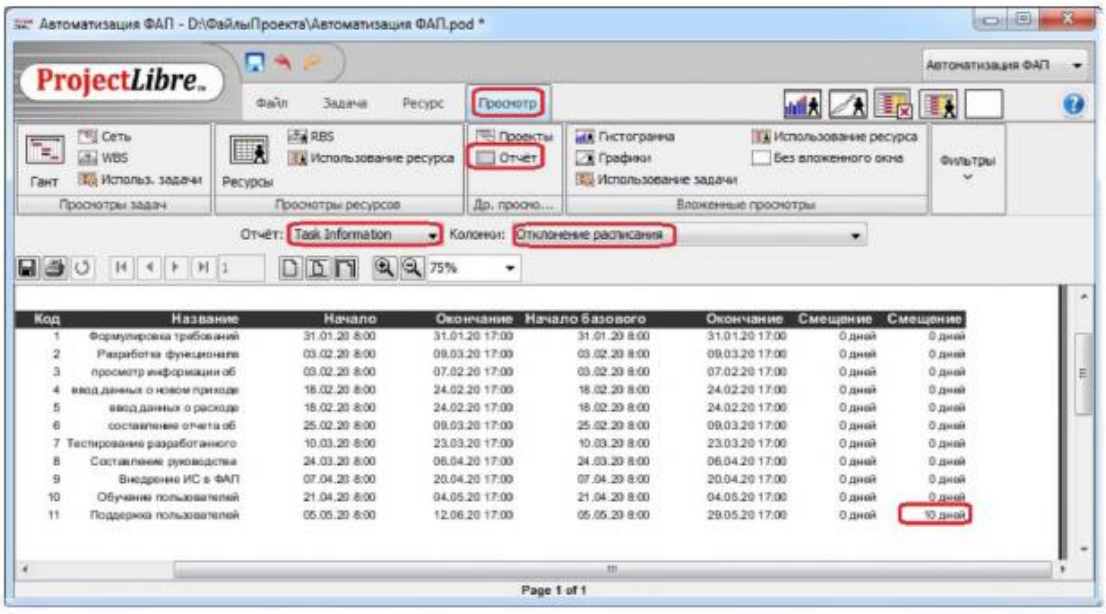


Рисунок 24 – Построение отчета

Эти три параметра используются для расчета следующих двух показателей:

Фактически выполненная работа • Индекс эффективности расписания = Запланированная работа

Планируемые затраты • Индекс эффективности затрат = Фактические затраты

Чем больше величина этих показателей, тем больше разница между фактической производительностью и планами. Основную часть информации можно посмотреть в отчетах по освоенным объемам (Индикаторы календарного плана освоенного объема, Освоенный объем, Показатели затрат для освоенного объема).

Рабочее задание 1:

1. Открыть проект из лабораторной работы №4.
2. Сохранить базовый план.
3. Выполнить все задачи и зафиксировать затраченные ресурсы.
4. Ввести в проект данные о фактических затратах.
5. Построить отчет по отклонение расписания.
6. Построить отчеты по освоенным объемам.
7. Проанализировать эффективность работы.
8. Сохранить проект.
9. Построить отчет по проделанной работе.

Рабочее задание 2:

1. Открыть проект из части 3.
2. Указать контуры в назначениях ресурсов. Требования к назначениям:

• необходимо использовать не менее трех различных контуров.

3. Открыть гистограмму загрузки ресурсов. Выяснить, есть ли перегруженные ресурсы.
4. Если нет перегруженных ресурсов, то искусственно создать случай перегруженных ресурсов, уменьшив время доступности одного из них.
5. Исключить перегруженные ресурсы путем разделения задач и перераспределения ресурсов.
6. Выполнить все задачи и зафиксировать затраченные ресурсы.
7. Ввести в проект данные о фактических затратах.
8. Построить отчет по отклонение расписания.
9. Построить отчеты по освоенным объемам.
10. Проанализировать эффективность работы.
11. Сохранить проект.
12. Построить отчет по проделанной работе.

Время работы: 2 часа.

Практическая работа 3. Использование метрик программного продукта

Цель работы: изучение требований к создаваемому программному продукту, разработка технического задания.

Теоретическая часть:

Техническое задание представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемно-сдаточных испытаний. В разработке технического задания участвуют как представители заказчика, так и представители исполнителя.

В основе этого документа лежат исходные требования заказчика, анализ передовых достижений техники, результаты выполнения научно-исследовательских работ, предпроектных исследований, научного прогнозирования. Основные факторы, определяющие характеристики разрабатываемого программного обеспечения:

- исходные данные и требуемые результаты, которые определяют функции программы или системы;
- среда функционирования (программная и аппаратная);
- возможное взаимодействие с другим программным обеспечением или специальными техническими средствами;
- также может быть определено, а может выбираться исходя из набора выполняемых функций.

Разработка технического задания выполняется в следующей последовательности. Прежде всего, устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных.

Затем определяют перечень результатов, их характеристики и способы представления. Далее уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В случаях, когда разрабатываемое программное обеспечение собирает и хранит некоторую информацию или включается в управление каким-либо техническим процессом, необходимо также четко регламентировать действия программы в случае сбоев оборудования и энергоснабжения.

На техническое задание существует стандарт ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению».

В соответствии с этим стандартом техническое задание должно содержать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе или программному изделию;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки.

При необходимости допускается в техническое задание включать приложения. Рассмотрим более подробно содержание каждого раздела. Введение должно

включать наименование и краткую характеристику области применения программы или программного продукта, а также объекта (например, системы) в котором предполагается их использовать. Назначение введения - продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

Раздел «Основания для разработки» должен содержать наименование документа, на основании которого ведется разработка, организации, утвердившей данный документ, и наименование или условное обозначение темы разработки. Таким документом может служить план, приказ, договор и другие.

Раздел «Назначение разработки» должен содержать описание функционального и эксплуатационного назначения программного продукта с указанием категорий пользователей.

Раздел «Требования к программе или программному изделию» должен включать следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к транспортированию и хранению;
- специальные требования.

Наиболее важным из перечисленных выше является подраздел «Требования к функциональным характеристикам». В этом разделе должны быть перечислены выполняемые функции и описаны состав, характеристики и формы представления исходных данных и результатов. В этом же разделе при необходимости указывают критерии эффективности: максимально допустимое время ответа системы, максимальный объем используемой оперативной и/или внешней памяти.

Примечание. Если разработанное программное обеспечение не будет выполнять указанных в техническом задании функций, то оно считается не соответствующим техническому заданию, т. е. неправильным с точки зрения критериев качества.

Универсальность будущего продукта также обычно специально не оговаривается, но подразумевается.

В подразделе «Требования к надежности» указывают уровень надежности, который должен быть обеспечен разрабатываемой системой и время восстановления системы после сбоя. Для систем с обычными требованиями к надежности в этом разделе иногда регламентируют действия разрабатываемого продукта по

увеличению надежности результатов (контроль входной и выходной информации, создание резервных копий промежуточных результатов).

В подразделе «Условия эксплуатации», указывают особые требования к условиям эксплуатации: температуре окружающей среды, относительной влажности воздуха. Как правило, подобные требования формулируют, если разрабатываемая система будет эксплуатироваться в нестандартных условиях или использует специальные внешние устройства, например, для хранения информации. Здесь же указывают вид обслуживания, необходимое количество и квалификация персонала. В противном случае допускается указывать, что требования не предъявляются.

В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их основных технических характеристик: тип микропроцессора, объем памяти, наличие внешних устройств. При этом часто указывают два варианта конфигурации: минимальный и рекомендуемый.

В подразделе «Требования к информационной и программной совместимости» при необходимости можно задать методы решения, определить язык или среду программирования для разработки, а также используемую операционную систему и другие системные и пользовательские программные средства, с которым должно взаимодействовать разрабатываемое программное обеспечение. В этом же разделе при необходимости указывают, какую степень защиты информации необходимо предусмотреть.

В раз деле «Требования к программной документации» указывают необходимость наличия руководства программиста, руководства пользователя, руководства системного программиста, пояснительной записки. На все эти типы документов также существуют ГОСТы.

В разделе «Технико-экономические показатели» рекомендуется указывать ориентировочную экономическую эффективность, предполагаемую годовую потребность и экономические преимущества по сравнению с существующими аналогами.

В разделе «Стадии и этапы разработки» указывают стадии разработки, этапы и содержание работ с указанием сроков разработки и исполнителей.

В разделе Порядок контроля и приемки указывают виды испытаний и общие требования к приемке работы. В приложениях при необходимости приводят: перечень научно-исследовательских работ, обосновывающих разработку; схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые следует использовать при разработке. В зависимости от особенностей разрабатываемого продукта разрешается уточнять содержание разделов, т. е. использо-

вать подразделы, вводить новые разделы или объединять их. В случаях, если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются». Разработка технического задания - процесс трудоемкий, требующий определенных навыков. Наиболее сложным, как правило, является четкое формулирование основных разделов: введения, назначения и требований к программному продукту.

Рабочее задание:

1 Изучить нормативные документы по разработке технического задания на разработку программного продукта.

2. В соответствии с предложенным вариантом составить ТЗ.

2 Ответить на контрольные вопросы.

Варианты заданий:

1. Программа планирования дел «Ежедневник».
2. Информационная система учета услуг в автомастерской.
3. Программа информационной поддержки спортивных соревнований.
4. Информационно-справочная система для продажи билетов в кинотеатре.
5. Программа учета и анализа продаж в продовольственном магазине.
6. Информационная система факультета «Абитуриент».
7. Программа информационного обеспечения фестиваля художественной самодеятельности студентов.
8. Программа информационной поддержки спартакиады университета.
9. Программа учета и анализа доходов и расходов семьи.
10. Программа формирования счетов-квитанций для жильцов ТСЖ.
11. Программа обработки данных аттестации студентов.
12. Программа управления очередностью обслуживания клиентов в поликлинике.
13. Информационная система учета посетителей гостиницы
14. Информационная система учета недвижимости
15. Информационная система библиотечного учета
16. Информационная система для автоматизации складского учета
17. Информационная система для автоматизации учета посетителей санатория
18. Информационная система для автоматизации работы страховой компании
19. Информационная система поддержки транспортных перевозок организации
20. ИС для автоматизации учета и обработки данных в сети АЗС
21. ИС управления общественным транспортом
22. ИС для автоматизации деятельности аудиторской компании
23. ИС учета компьютерной техники организации
24. ИС для автоматизации деятельности приюта бездомных животных

25. ИС учета заявок телекоммуникационной компании
26. ИС для строительной компании
27. ИС для компании по организации праздников

Контрольные вопросы:

1. Что понимают под технологичностью программного обеспечения?
2. Какие типы программных продуктов можно выделить?
3. Назовите основные эксплуатационные требования к ПП.
4. В каких ситуациях необходимы предпроектные исследования?
5. Какой раздел технического задания можно считать основным и почему?

Время работы: 2 часа.

Практическая работа №4. Выполнение измерений характеристик кода в среде VisualStudio

Цель работы: изучить как выполняется измерение характеристик кода в среде VisualStudio. Оценка объектно-ориентированных программных систем путем расчета проектных метрик, ориентированных на классы.

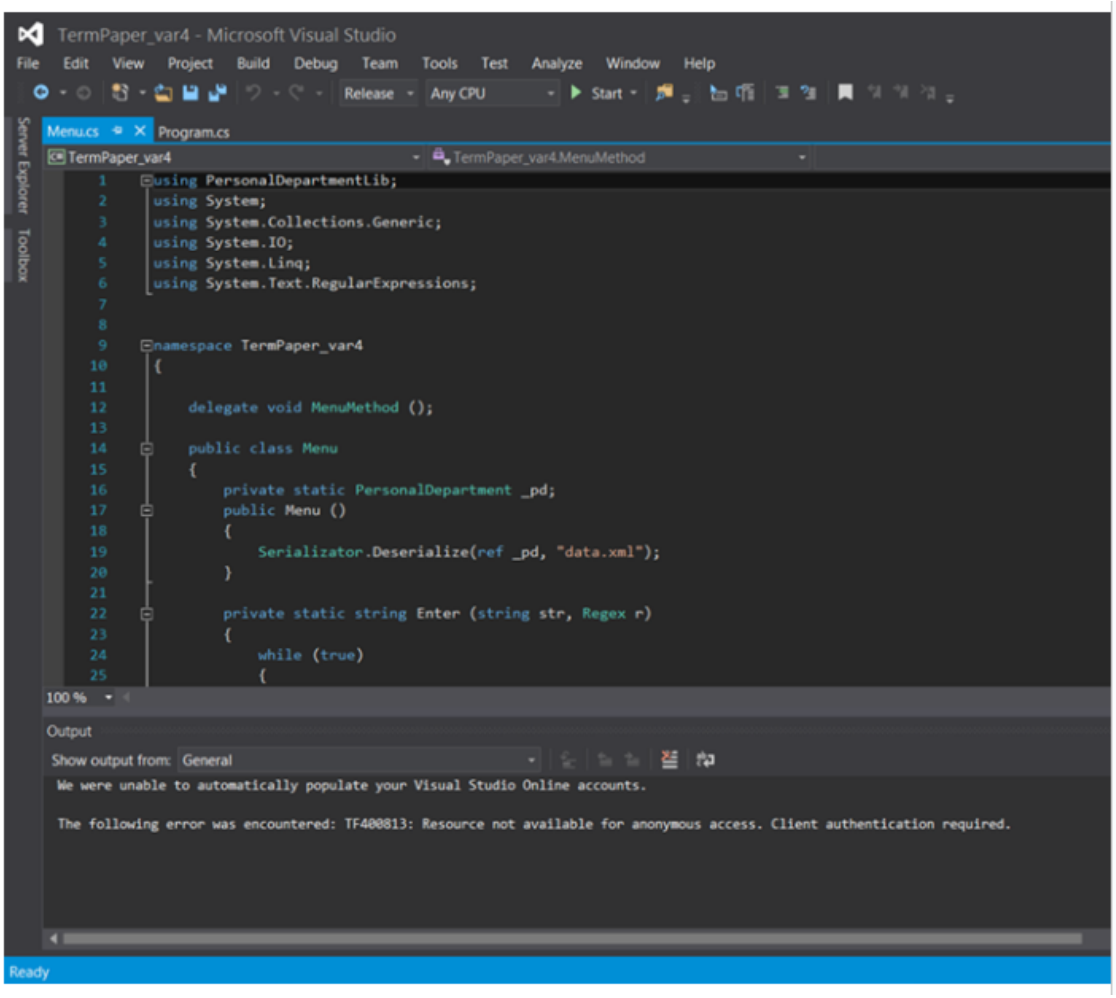
Теоретические сведения

Метрика программного обеспечения (англ. Software metric) – это некая мера определенного свойства программного обеспечения или же его спецификаций. Как известно, мера – это средство измерения. Важно понять, что мера - это числовое значение. Таким образом, метрика программного обеспечения будет показывать некое числовое значение определенного свойства ПО.

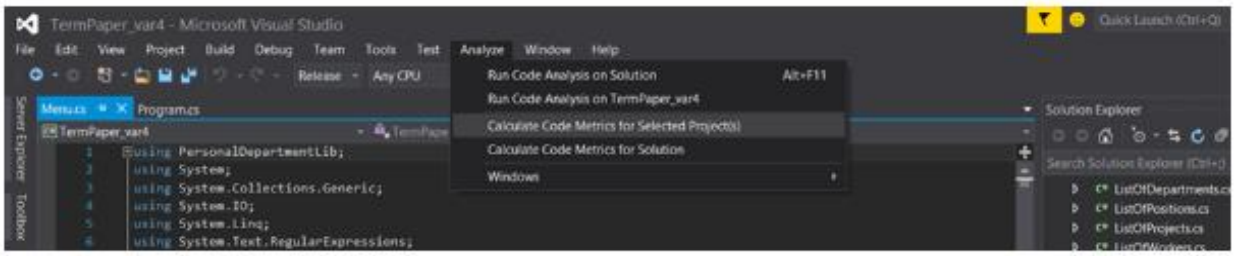
Метрики в Visual Studio

Стоит заметить сразу, что метрики подвергаются критике. Это, как минимум, поверхностно и неточно.

Откроем проект для изучения.

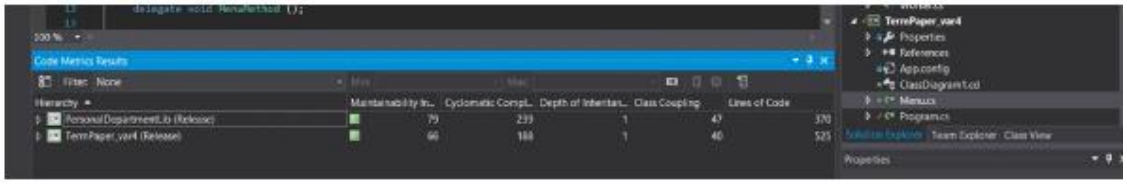


Далее, рассмотрим вкладку Analyze

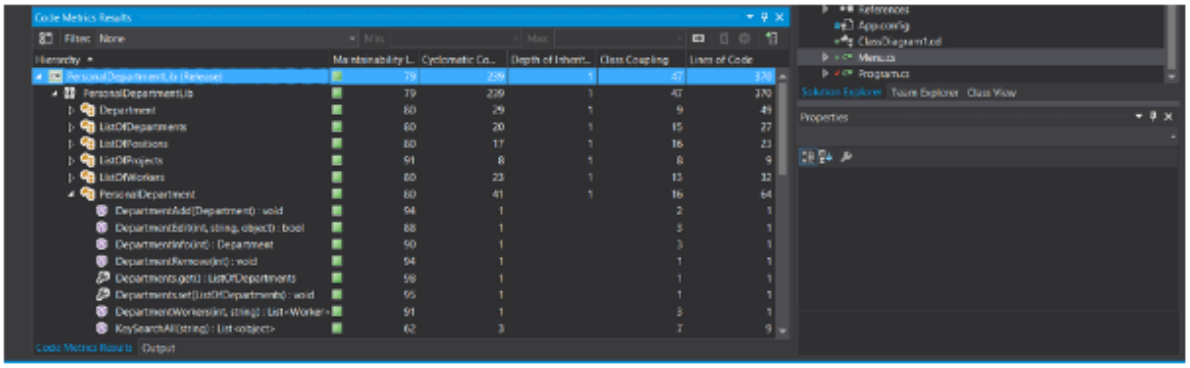


В этой вкладке необходимо выбрать Calculate Code Metrics for ...

Разница лишь в том, что будет анализироваться. Или же выбранные проекты в Solution Explorer, или же сразу весь Solution. После нажатия придется немного подождать. Время зависит от конфигурации Вашего компьютера. Когда анализ будет завершен, Вы увидите внизу окно



Здесь будет видна иерархия всего Solution. В данном случае это отдельная dll библиотека и проект. Если развернуть библиотеку, мы увидим следующий уровень иерархии, и так далее



Теперь разберемся со столбцами дальше.

1. Maintainability Index – это комплексный показатель качества кода. Эта метрика рассчитывается по следующей формуле:

MI = MAX(0, (171 — 5.2 * ln(HV) — 0.23 * CC — 16.2 * ln(LoC)) * 100 / 171)

HV – Halstead Volume, вычислительная сложность. Чем больше операторов, тем больше значение этой метрики;

CC – Cyclomatic Complexity (Эта метрика описана ниже);

LoC – количество строк кода (Эта метрика описана ниже).

2. Cyclomatic Complexity – показывает структурную сложность кода. Иными словами, количество различных ветвей кода. Считается на основе операторов в Вашем коде, строя графы переходов от одного оператора к другому. К примеру, оператор if-else увеличит эту метрику, потому что здесь будут разные ветви выполнения.

3. Depth of Inheritance – глубина наследования. Для каждого класса эта метрика показывает, насколько глубоко он в цепочке наследования.

4. Class Coupling – указывает на зависимость классов друг от друга. Проект с множеством зависимостей очень трудно и дорого поддерживать.

5. Lines of Code – количество строк кода. Напрямую используется редко. В наши дни, с множеством разнообразных как подходов к программированию, так и языков, эта метрика дает нам мало полезной информации. Если брать во внимание отдельный метод, то можно разбить его на несколько методов поменьше.

Использования метрик

Изначально стоит обращать внимание на Maintainability Index. Старайтесь придерживать его около 70-90. Это значительно облегчит сопровождения кода как Вами, так и другими программистами. Иногда стоит оставить его на уровне 50-60, так как переписать некоторые участки кода бывает очень затратным. Оценивайте здраво как код, так и Ваши возможности с затратами.

Стоит также уделить много внимания Class Coupling. Эта метрика должна быть как можно меньшей. Ведь она так же способствует поддержке кода. Для оптимизации возможно придется пересматривать дизайн проекта и некоторые архитектурные решения.

Теперь стоит уделить внимание Cyclomatic Complexity. Эта метрика показывает сложность кода, а это так же влияет на поддержку кода в будущем. Иногда приходится переписывать куски кода, которые писали до Вас другие люди, так как Вы просто не можете понять, что, как и зачем в этом методе. Конечно, этому еще способствует стиль кода и идея, но не забывайте о Cyclomatic Complexity при рефакторинге.

Вывод

С таким инструментом в руках Вы можете быстро и относительно легко сделать review проекта и найти его уязвимые места. Также можно постоянно мониторить метрики и делать даже некие выводы об усталости работника или его отношении к работе. Более того, можно увидеть динамику роста качества кода каждого программиста. Но здесь стоит отчетливо понимать все детали так, как мы говорили об этом в критике.

1. Табулирование функций

С помощью делегата Function строим таблицы значений двух различных функций одним и тем же кодом.

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {
 class Program {
  delegate double Function (double x);
  static void Table (double x1, double dx, double x2, Function f) {
   double y;
   string OutputFormat = "{0:F4}  {1:F4}";
   Console.WriteLine (OutputFormat, "x", "y");
   for (double x = x1; x <= x2; x += dx) {
         y = f (x);
         Console.WriteLine (OutputFormat,x,y);
   }
  }
  static void Main (string[] args) {
   double sin_ (double x) { return Math.Sin (x); }
   double cos_ (double x) { return Math.Cos (x); }
```

```
   Console.Clear ();
   Function function = sin_;
   Table (0, Math.PI / 10, Math.PI, function);
   function = cos_;
   Table (0, Math.PI / 10, Math.PI, function);

   Console.ReadKey();
  }
 }
}
```

2. Второй класс в проекте, генерирующий случайное число

Для добавления нового класса достаточно обратиться к меню Проект -> Добавить класс и назначить новому классу имя.

Если классы располагаются в одном пространстве имён, для ссылки на метод объекта второго класса главной программе достаточно выполнить что-то вроде

```
var obj = new ProjectNamespace.SecondClass();
obj.Method();
//---------------------- файл Program.cs

//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {
 class Program {
  static void Main (string[] args) {
   var gen = new ConsoleApp1.RandomNumber();
   Console.WriteLine ("{0:F6}",gen.get());
   Console.ReadKey();
  }
 }
}

//---------------------- файл RandomNumber.cs

//меню Проект - Добавить класс
using System;
```

```
namespace ConsoleApp1 {
 class RandomNumber {
  private double min, max;
  private Random rand;
  public RandomNumber (double min = 0, double max = 1) {
   this.rand = new Random ();
   this.min = min;
   this.max = max;
  }
  public double get () {
   return rand.NextDouble() * ( this.max - this.min ) + this.min;
  }
 }
}
```

3. Два класса в одном файле

Достаточно, чтобы каждый класс располагался в своих операторных скобках внутри общего namespace

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {
 class A {
  private string str;
  private double val;
  public A(string str="", double val = 0) { this.str = str;  this.val = val; }
  public string view() {
   return str + ", " + val.ToString();
  }
 }
 class Program {
  static void Main (string[] args) {
   string name = "Паоло Гудини";
   A a = new A(name,1);
   Console.WriteLine (a.view());
   Console.ReadKey();
  }
 }
}
```

4. Матрицы обычная и ступенчатая

По сути, в C# есть "паскалеподобная" матрица с квадратными скобками вида [i,j], для которой количество элементов в каждой строке одинаково, и "си-подобная" ("ступенчатая") со скобками вида [i][j] и возможностью рассматривать матрицу как вектор векторов, соответственно, имея в различных строках различное количество элементов.

Показаны выделение памяти, заполнение и построчный вывод элементов.

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {
 class Program {
  static void PrintMatrix(int [,] a) {
   for (int i = 0; i < a.GetLength(0); i++) {
         for (int j = 0; j < a.GetLength(1); j++) {
         Console.Write ("{0:F4} ",a[i,j]);
         }
         Console.WriteLine ();
   }
  }
  static void PrintJaggedMatrix (int [][] a) {
   for (int i = 0; i < a.Length; i++) {
         for (int j = 0; j < a[i].Length; j++) {
         Console.Write ("{0:F4} ", a[i][j]);
         }
         Console.WriteLine ();
   }
  }
  static void Main (string[] args) {

   //"Паскалеподобная" матрица, количество элементов в каждой строке одинаково
   int [,] matrix = {
         {1,2,3 },
         {4,5,6 }
   };
   PrintMatrix (matrix);
```

```
//"Си-подобная" матрица, количество элементов в каждой строке может быть различным
const int jaggedMarixRows = 3;
int [] [] jaggedMarix = new int [jaggedMarixRows] [];
for (int i = 0, k = 1; i < jaggedMarix.Length; i++) {
try {
jaggedMarix [i] = new int [i + 1];
}
catch (OutOfMemoryException e) {
Console.WriteLine ("Memory allocation failed \"{0:C}\" in string {0:D}\n",e,i);
break;
}
for (int j = 0; j < jaggedMarix[i].Length; j++) jaggedMarix [i] [j] = k++;
}
PrintJaggedMatrix (jaggedMarix);

Console.ReadKey ();
}
}
}
```

5. Шаблон класса стека

Показаны стек целых и стек вещественных чисел, использующие один и тот же шаблон класса.

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {

public class Stack <T> where T : new() {
//у типа данных стека должен быть конструктор по умолчанию
T [] stck; // массив, содержащий стек
int tos; // индекс вершины стека
public Stack (int size) {
stck = new T [size]; // распределить память для стека
tos = 0;
}
```

```
public int Push (T ch) { // Поместить элементы в стек
 if (tos == stck.Length) { //стек заполнен
       return -1;
 }
 stck [tos] = ch;
 tos++;
 return tos;
}
public T Pop () { // Извлечь элемент из стека
 if (tos == 0) { //стек пуст
       return default (T);
 }
 tos--;
 return stck [tos];
}
public bool IsFull () { // Возвратить значение true, если стек заполнен
 return tos == stck.Length;
}
public bool IsEmpty () { // Возвратить значение true, если стек пуст
 return tos == 0;
}
public int Capacity () { // Возвратить общую емкость стека
 return stck.Length;
}
public int GetNum () { // Возвратить количество объектов, находящихся в
данный момент в стеке
 return tos;
}
}

class Program {
 static void Main (string[] args) {
  const int Stack1Size = 10;
  Stack <int> stk1 = new Stack <int> (Stack1Size);
  for (int i = 0; !stk1.IsFull (); i++) stk1.Push (i+1);
  if (stk1.IsFull ()) Console.WriteLine ("Стек stk1 заполнен.");
  // Вывести содержимое стека stk1.
  Console.Write ("Содержимое стека stk1: ");
  while (!stk1.IsEmpty ()) {
```

```
        int i = stk1.Pop ();
        Console.Write ("{0} ", i);
    }
    Console.WriteLine ();
    if (stk1.IsEmpty ()) Console.WriteLine ("Стек stk1 пуст.\n");
    // Поместить дополнительные символы в стек stk1.
    Console.WriteLine ("Вновь поместить элементы в стек stk1.");
    for (int i = 0; !stk1.IsFull (); i++) stk1.Push (Stack1Size - i );
    Console.WriteLine ("А теперь извлечь символы из стека stk1 " +
        "и поместить их в стек stk2, добавив дробную часть");
    Stack <double> stk2 = new Stack <double> (Stack1Size);
    while (!stk1.IsEmpty ()) {
        int i = stk1.Pop ();
        stk2.Push ((double)i + 0.5);
    }
    Console.Write ("Содержимое стека stk2: ");
    while (!stk2.IsEmpty ()) {
        double d = stk2.Pop ();
        Console.Write ("{0:F1} ", d);
    }
    Console.WriteLine ();
    Console.WriteLine ("Емкость стека stk2: " + stk2.Capacity ());
    Console.WriteLine ("Количество объектов в стеке stk2: " + stk2.GetNum ());

    Console.ReadKey ();
   }
  }
 }
```

6. Шаблон функции и аргументы по ссылке

Передача и возврат аргументов по ссылке, функция с переменным числом аргументов.

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {

 class Program {
  static void Swap <T> (ref T lhs, ref T rhs) { //Аргументы получены по ссылке
```

```
        T temp = lhs;   lhs = rhs;   rhs = temp;
       }

       static void GetNumberParts (double n, out double whole, out double frac) {
        //Второй и третий аргументы будут возвращены по ссылке
        whole = Math.Floor(n);
        frac = n - whole;
       }

       static double Summa (params double [] nums) { //Функция может иметь переменное число аргументов
        double sum = 0;
        for (int i = 0; i < nums.Length; i++) sum += nums [i];
        return sum;
       }

       static void Main (string[] args) {

        double a = 1.5;
        double b = 2.7;
        Swap (ref a, ref b); //Аргументы переданы по ссылке
        Console.WriteLine (a + " " + b);

        double d, f;
        GetNumberParts (a, out d, out f); //Второй и третий аргументы будут возвращены по ссылке
        Console.WriteLine ("Целая часть числа равна " + d);
        Console.WriteLine ("Дробная часть числа равна " + f);

        double s = Summa (1, 2, 3, 4);
        Console.WriteLine ("1+2+3+4= " + s);

        Console.ReadKey ();
       }
      }
     }
```

7. Фабрика объектов

Так называют статический метод в классе, возвращающий новый объект этого же класса. Имеет смысл, если по каким-то причинам не хотим делать конструктор класса публичным методом.

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {
 class Factory {
  private int val;
  private Factory (int val = 0) { //Конструктор класса приватен
   this.val = ++val;
  }
  public static Factory makeFactory (int val = 0) { //Но есть фабрика объектов
   Factory factory = new Factory (val);
   return factory;
  }
  public override string ToString () { //переписываем встроенный метод
   return val.ToString ();
  }
 }

 class Program {
  static void Main (string[] args) {
   const int Size = 10;
   Factory [] Objects = new Factory [Size];
   //Объекты массива Factory пока пустые ссылки (null), то есть, конструктор по умолчанию
   //всё равно доступен. А вот new Factory (0) [Size] не сработало бы
   for (int i = 0; i < Size; i++) {
        Objects [i] = Factory.makeFactory (i);
        Console.WriteLine ("Объект номер {0}: {1}",i, Objects [i]);
   }
   Console.ReadKey ();
  }
 }
}
```

8. Статические члены класса и оценивание арифметических выражений

Описываем в классе статический счётчик созданных объектов и оцениваем арифметические выражения одной строчкой кода с проверкой корректности (метод Exec).

```
//C# -> Консольное приложение (.NET Framework)
using System;
using System.Data;

namespace ConsoleApp1 {
 class Program {

  class Compute {
   private static int Count = 0; //Счётчик созданных объектов класса
   public Compute () {
        Count++; //Увеличить счётчик на 1 при создании объекта
   }
   public double Exec (string expr) {
        return Convert.ToDouble (new DataTable ().Compute (expr, ""));
   }
   public static int GetCount () { return Count; } //Узнать значение счётчика
  }
  static void Main () {
   string [] Expressions = {
        "(5-2)%2 + 5./4",
        "-1+2/3",
        "1*2*3*error"
   };
   for (int i = 0; i < Expressions.Length; i++) {
        Compute Expr = new Compute (); //На самом деле, хватило бы статического метода в Compute
        try {
        double d = Expr.Exec (Expressions [i]);
        Console.WriteLine ("{0} = {1:F4}", Expressions [i], d);
        }
        catch (Exception e) {
        Console.WriteLine ("Ошибочное выражение {0}: \"{1}\"", Expressions [i], e.Message);
        }
```

```
        }

        Console.WriteLine ("Всего обработано выражений: {0}", Com-
pute.GetCount ());
        Console.ReadKey ();
       }
      }
     }
```

9. Работаем с объектом "Таблица данных"

Программно создаём таблицу с заданными характеристиками, добавляем туда данные строк и вычисляемый столбец, считаем по формулам с помощью агрегатных выражений.

```
//C# -> Консольное приложение (.NET Framework)
using System;
using System.Data;

namespace ConsoleApp1 {

 class MyDataTable {
  DataTable dt;
  DataRow dr;
  DataColumn dc;
  public MyDataTable (string TableName, string [] ColumnNames, Type []
ColumnTypes) {
   dt = new DataTable ();
   dt.TableName = TableName;
   for (int i = 0; i < ColumnNames.Length; i++) dt.Columns.Add (ColumnNames
[i], ColumnTypes [i]);
  }
  public DataRow CreateRow () { //Создать строку
   dr = dt.NewRow ();
   return dr;
  }
  public void Add (DataRow dr) { //Добавить строку в базу
   dt.Rows.Add (dr);
  }
  public DataColumn CreateColumn (string Name, Type type, string expr) {
//Создать вычисляемый столбец
```

```
 dc = new DataColumn ();
 dc.ColumnName = Name;
 dc.DataType = type;
 dc.Expression = expr;
 dt.Columns.Add (dc);
 return dc;
 }
 public void AddRow (DataRow dr) { //Добавить строку в базу
 dt.Rows.Add (dr);
 }

 public string Exec (string expr) {
 return Convert.ToString (dt.Compute (expr, ""));
 }
 }

 class Program {
 static void Main () {
 string [] ColumnNames = { "Имя", "Оплата", "Комиссия"};
 Type [] ColumnTypes = { typeof (string), typeof (int), typeof (double) };
 MyDataTable Formula = new MyDataTable ("Таблица", ColumnNames,
ColumnTypes);
 //Создали таблицу
 DataRow dr = Formula.CreateRow ();
 dr ["Имя"] = "Вася";
 dr ["Оплата"] = 30000;
 dr ["Комиссия"] = 0.15;
 Formula.AddRow (dr);
 dr = Formula.CreateRow ();
 dr ["Имя"] = "Петя";
 dr ["Оплата"] = 40000;
 dr ["Комиссия"] = 0.20;
 Formula.AddRow (dr);
 //Добавили данные в строки
 Console.WriteLine ("Avg(Оплата) = " + Formula.Exec ("Avg(Оплата)"));
 //Посчитали по формуле
 DataColumn dc = Formula.CreateColumn ("Итого", typeof (double), "Оплата
- Оплата * Комиссия");
 Console.WriteLine ("Sum(Итого) = " + Formula.Exec ("Sum(Итого)"));
```

```
        //Посчитали выражение над добавленным столбцом
       //(30000-30000*0.15)+(40000-40000*0.20)

        Console.ReadKey ();
       }
      }
     }
```

10. Запускаем десять потоков и выводим их состояние

Управляем временем выполнения потоков, а также позицией курсора в консоли (для вывода сообщений о состоянии потоков).

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApp1 {
 internal class Program {
  private static void Main (string [] args) {

   var Tasks = new List<Task> ();
   for (var i = 0; i < 10; i++) Tasks.Add (new Task (Method, TaskCreation-
Options.LongRunning));

   Tasks.ForEach (t => t.Start ());

   var startY = Console.CursorTop;

   do {
       PrintStatus (Tasks);
       Console.CursorTop = startY;
   } while (!Task.WaitAll (Tasks.ToArray (), TimeSpan.FromSeconds (1)));

   PrintStatus (Tasks);

   Console.Write ("Выполнено");
   Console.ReadKey ();
  }
```

```
 private static void PrintStatus (IEnumerable <Task> Tasks) {
  foreach (var task in Tasks)
        Console.WriteLine ($"Состояние задачи #{task.Id}: {task.Status}");
 }

 private static void Method () {
  Thread.Sleep (TimeSpan.FromSeconds (2 * Task.CurrentId ?? 1));
 }
 }
}
```

11. Перегрузка операторов

Показаны все основные виды переопределения операторов в классе C# (кроме некоторых редко используемых нюансов).

```
using System;

namespace ConsoleApp1 {
 class Point3D {
  private double x, y, z; //Трёхмерные координаты
  public Point3D () { x = y = z = 0; }
  public Point3D (double x, double y, double z = 0) { this.x = x; this.y = y; this.z = z; }
  public static Point3D operator + (Point3D op1, Point3D op2) { //Перегрузить бинарный оператор +
   Point3D result = new Point3D ();
   result.x = op1.x + op2.x;
   result.y = op1.y + op2.y;
   result.z = op1.z + op2.z;
   return result;
  }
  public static Point3D operator + (Point3D op1, double op2) {
   //Перегрузить бинарный + для сложения с числом (вторым операндом)
   Point3D result = new Point3D ();
   result.x = op1.x + op2;
   result.y = op1.y + op2;
   result.z = op1.z + op2;
   return result;
  }
  public static Point3D operator + (double op1, Point3D op2) {
```

```
//Перегрузить бинарный + для сложения с числом, если число является первым операндом
Point3D result = new Point3D ();
result.x = op2.x + op1;
result.y = op2.y + op1;
result.z = op2.z + op1;
return result;
}
public static Point3D operator - (Point3D op) { // Перегрузить унарный оператор -
Point3D result = new Point3D ();
result.x = -op.x;
result.y = -op.y;
result.z = -op.z;
return result;
}
public static bool operator < (Point3D op1, Point3D op2) { // Оператор сравнения
return (
Math.Sqrt (op1.x * op1.x + op1.y * op1.y + op1.z * op1.z) <
Math.Sqrt (op2.x * op2.x + op2.y * op2.y + op2.z * op2.z) ? true : false );
}
public static bool operator > (Point3D op1, Point3D op2) { // "Меньше" и "больше" работают только вместе
return (
Math.Sqrt (op1.x * op1.x + op1.y * op1.y + op1.z * op1.z) >
Math.Sqrt (op2.x * op2.x + op2.y * op2.y + op2.z * op2.z) ? true : false );
}
public static bool operator true (Point3D op) {
// Перегрузка true, истинна, если хотя бы 1 координата не равна 0
return op.x != 0 || op.y != 0 || op.z != 0 ? true : false;
}
public static bool operator false (Point3D op) { // true и false работают только вместе
return op.x == 0 && op.y == 0 && op.z == 0 ? true : false; //все координаты равны 0
}
public static bool operator | (Point3D op1, Point3D op2) {
```

```
        // Перегрузка логического "или", истинна, если хотя бы одна коррдината ненулевая
        return op1.x * op2.x != 0 || op1.y * op2.y != 0 || op1.z * op2.z != 0 ? true : false;
       }
       public static bool operator & (Point3D op1, Point3D op2) {
        // Перегрузка логического "и", истинна, если все коррдинаты ненулевые
        return op1.x * op2.x != 0 & op1.y * op2.y != 0 & op1.z * op2.z != 0 ? true : false;
       }
       public static bool operator ! (Point3D op) {
        // Перегрузка true, ложна, если хотя бы 1 координата не равна 0
        return op.x != 0 || op.y != 0 || op.z != 0 ? false : true;
       }
       public static implicit operator double (Point3D op1) {
        //Неявное преобразование типа, выполняется автоматически, когда объект используется
        //в выражении вместе со значением целевого типа
        return op1.x * op1.y * op1.z;
       }
       public static explicit operator float (Point3D op1) {
        //Явное преобразование типа
        return (float)op1.x * (float) op1.y * (float) op1.z;
       }

       public override string ToString () {
        // Вернуть координаты в виде строки, перегрузив метод ToString по умолчанию
        return this.x + ", " + this.y + ", " + this.z;
       }
      }
      internal class Program {
       private static void Main () {
        Point3D a = new Point3D (1, 2, 3);
        Point3D b = new Point3D (1, 1, 1);
        Point3D c = new Point3D ();
        Console.WriteLine ("Координаты точки a: {0}", a.ToString ());
        Console.WriteLine ("Координаты точки b: {0}", b.ToString ());
        Console.WriteLine ("Координаты точки c: {0}", c.ToString ());
        Point3D d = a + b + c;
```

```
            Console.WriteLine ("A + B + C = {0}", d.ToString ());

            d = -a;
            Console.WriteLine ("D = -A = {0}", d.ToString ());
            d += 1; //не нужно отдельно перегружать оператор "+="!
            Console.WriteLine ("D = -A + 1 = {0}", d.ToString ());
            d = 1 + d; //а здесь вызовется второй оператор для сложения с числом
            Console.WriteLine ("D = 1 + D = {0}", d.ToString ());

            bool cond = a < b;
            Console.WriteLine ("A < B = {0}", cond);
            cond = a > b;
            Console.WriteLine ("A > B = {0}", cond);

            if (c) Console.WriteLine ("Точка C истинна");
            else Console.WriteLine ("Точка C ложна");
            if (d) Console.WriteLine ("Точка D истинна");
            else Console.WriteLine ("Точка D ложна");

            if (a & d) Console.WriteLine ("a & d истинно");
            else Console.WriteLine ("a & d ложно");
            if (a | d) Console.WriteLine ("a | d истинно");
            else Console.WriteLine ("a | d ложно");
                  //с "укороченными" формами &&, || такие перегрузки работать не бу-
дут
            if (!c) Console.WriteLine ("Точка !C истинна");
            else Console.WriteLine ("Точка !C ложна");

            double val  = d * 2 + b; // преобразовать в тип double неявно
            Console.WriteLine ("d * 2 + b = " + val);
            float fval = (float)b * (float)Math.PI; // преобразовать в тип float явно
            Console.WriteLine ("b * PI = " + fval);

            Console.ReadKey ();
           }

          }
         }
```

12. Преобразовать строку в число

Основные способы и простейшая обработка исключений при преобразовании.

```
using System;
using System.Globalization;

namespace ConsoleApp1 {
internal class Program {
 private static void Main () {

  //Способ 1: Parse
  Int32 val1 = Int32.Parse ("100"); //100
      //простое преобразование
  Int32 val2 = Int32.Parse ("(200)", NumberStyles.AllowParentheses); //-200
      //перегрузка с указанием стиля
  int val3 = int.Parse ("30,000", NumberStyles.AllowThousands, new CultureInfo ("en-au")); //30000
      //перегрузка с указанием стиля и культуры
  Console.WriteLine ($"{val1} {val2} {val3}");

  //Обработка исключений при преобразованиях 1 и 2:
  string str1 = "100f";
  try {
      val1 = Int32.Parse (str1);
      Console.WriteLine (val1);
  }
  catch (Exception e) {
      Console.WriteLine ("Неверный формат Parse: " + str1);
  }
  try {
      val1 = Convert.ToInt32 (str1);
      Console.WriteLine (val1);
  }
  catch (Exception e) {
      Console.WriteLine ("Неверный формат ToInt32: " + str1);
  }

  Console.ReadKey ();
 }
```

```
 }
}
```

13. Преобразовать строку в число

Основные способы и простейшая обработка исключений при преобразовании.

```
using System;
using System.Globalization;

namespace ConsoleApp1 {
internal class Program {
  private static void Main () {

   //Способ 2: Convert
   val1 = Convert.ToInt32 ("123456"); //123456
   val2 = Convert.ToInt32 ("10000",2); //16
   val3 = Convert.ToInt32 ("100", 16); //256
   Console.WriteLine ($"{val1} {val2} {val3}");

   //Обработка исключений при преобразованиях 1 и 2:
   string str1 = "100f";
   try {
         val1 = Int32.Parse (str1);
         Console.WriteLine (val1);
   }
   catch (Exception e) {
         Console.WriteLine ("Неверный формат Parse: " + str1);
   }
   try {
         val1 = Convert.ToInt32 (str1);
         Console.WriteLine (val1);
   }
   catch (Exception e) {
         Console.WriteLine ("Неверный формат ToInt32: " + str1);
   }

   Console.ReadKey ();
  }
```

```
 }
}
```

14. Преобразовать строку в число

Основные способы и простейшая обработка исключений при преобразовании.

```
using System;
using System.Globalization;

namespace ConsoleApp1 {
internal class Program {
  private static void Main () {

    //Способ 3: TryParse
    string numberStr = "123456";
    int number;
    bool isParsable = Int32.TryParse (numberStr, out number); //123456
    if (isParsable) Console.WriteLine (number);
    else Console.WriteLine ("Неверный формат: "+ numberStr);
    numberStr = "123,45";
    double val;
    isParsable = double.TryParse (numberStr, NumberStyles.Float, NumberFormatInfo.CurrentInfo, out val);
    //val будет равно 123,45 , если локаль русская
    if (isParsable) Console.WriteLine (val);
    else Console.WriteLine ("Неверный формат: "+ numberStr);

    //Обработка исключений при преобразованиях 1 и 2:
    string str1 = "100f";
    try {
        val1 = Int32.Parse (str1);
        Console.WriteLine (val1);
    }
    catch (Exception e) {
     Console.WriteLine ("Неверный формат Parse: " + str1);
    }
    try {
        val1 = Convert.ToInt32 (str1);
```

```
        Console.WriteLine (val1);
      }
      catch (Exception e) {
        Console.WriteLine ("Неверный формат ToInt32: " + str1);
      }

      Console.ReadKey ();
     }

    }
   }
```

15. Преобразовать число в строку

Способы как с добавлением, так и без добавления дополнительного текстового содержимого к полученной строке.

```
using System;
using System.Text;

namespace ConsoleApp1 {
 class Program {
  private static void Main () {

   //Способ 1: ToString
   int val = 0xff;
   Console.WriteLine (val.ToString ()); //255
   val = (int) 1e5;
   Console.WriteLine (val.ToString ()); //100000

   Console.ReadKey ();
  }

 }
}
```

16. Преобразовать число в строку

Способы как с добавлением, так и без добавления дополнительного текстового содержимого к полученной строке.

```
using System;
using System.Text;
```

```
namespace ConsoleApp1 {
 class Program {
  private static void Main () {

   //Способ 2: "+" со строкой
   val = (int) Math.Floor(Math.PI);
   string str = "" + val;
   Console.WriteLine (str); //3

   Console.ReadKey ();
  }

 }
}
```

17. Преобразовать число в строку

Способы как с добавлением, так и без добавления дополнительного текстового содержимого к полученной строке.

```
using System;
using System.Text;

namespace ConsoleApp1 {
 class Program {
  private static void Main () {

   //Способ 3: StringBuilder
   var builder = new StringBuilder (); //System.Text
   builder.Append ("There are ");
   builder.Append (val).ToString ();
   builder.Append (" wolfs in our forest");
   Console.WriteLine (builder); //There are 3 wolfs in our forest

   Console.ReadKey ();
  }

 }
}
```

18. Преобразовать число в строку

Способы как с добавлением, так и без добавления дополнительного текстового содержимого к полученной строке.

```
using System;
using System.Text;

namespace ConsoleApp1 {
 class Program {
  private static void Main () {

   //Способ 4: Convert
   string msg = "There are " + Convert.ToString (val) + " wolfs in our forest";
   Console.WriteLine (msg); //There are 3 wolfs in our forest

   Console.ReadKey ();
  }

 }
}
```

19. Преобразовать число в строку

Способы как с добавлением, так и без добавления дополнительного текстового содержимого к полученной строке.

```
using System;
using System.Text;

namespace ConsoleApp1 {
 class Program {
  private static void Main () {

   //Способ 5: Format
   string msg2 = string.Format ("There are {0} wolfs in our forest", val);
   Console.WriteLine (msg2); //There are 3 wolfs in our forest

   Console.ReadKey ();
  }

 }
}
```

20. Преобразовать число в строку

Способы как с добавлением, так и без добавления дополнительного текстового содержимого к полученной строке.

```
using System;
using System.Text;

namespace ConsoleApp1 {
 class Program {
  private static void Main () {

   //Способ 6: $
   string msg3 = $"There are {val} wolfs in our forest";
   Console.WriteLine (msg3); //There are 3 wolfs in our forest

   Console.ReadKey ();
  }

 }
}
```

21. Индексаторы, свойства и автоматически реализуемые свойства

Пример на индексаторы и свойства C#.

Ограничения индексаторов таковы: значение, отдаваемое индексатором, нельзя передавать методу в качестве параметра ref или out, поскольку в индексаторе не определено место в памяти для его хранения. Индексатор должен быть членом своего класса и поэтому не может быть объявлен как static.

Свойство сочетает в себе поле с методами доступа к нему и состоит из имени и аксессоров get и set. Аксессоры служат для получения и установки значения переменной. Имя свойства может быть использовано в выражениях и операторах присваивания аналогично имени обычной переменной, но в действительности при обращении к свойству по имени автоматически вызываются его аксессоры get и set.

Свойства не определяют место в памяти для хранения полей, а лишь управляют доступом к полям. Это означает, что само свойство не предоставляет поле, поэтому поле должно быть определено независимо от свойства. Свойство также не должно изменять состояние базовой переменной при вызове аксессора get. Исключение из этого правила составляет автоматически реализуемое свойство.

Автоматически реализуемое свойство не может быть доступно только для чтения или только для записи. При его объявлении нужно указывать оба аксессора

— get и set, хотя любой из них можно сделать приватным, доступным только методам своего класса.

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {
 public class Array <T> where T : new() { //"Отказоустойчивый" вектор с индексаторами
  T [] a; //Ссылка на базовый массив
  public int Length { get; set; } //Длина массива сделана автосвойством
  public bool ErrFlag { get; private set; }
  //Результат последней операции - автосвойство "только для чтения" извне
  public Array (int size) { //Конструктор
   if (size > 0) {
        a = new T [size];
        Length = size;
   }
   else Length = 0;
  }

  public T this [int index] { //Индексатор
   get { //Аксессор get
        if (IndexIsValid (index)) {
        ErrFlag = false;
        return a [index];
        }
        else {
        ErrFlag = true;
        return default (T);
        }
   }
   set { //Аксессор set; получает неявный параметр value!
        if (IndexIsValid (index)) {
        a [index] = value;
        ErrFlag = false;
        }
        else ErrFlag = true;
   }
  }
```

```
public T this [double index] { //Ещё один индексатор, для индекса типа dou-
ble
 get {
      int intIndex = (int) Math.Round (index);
      return this [intIndex];
 }
 set {
      int intIndex = (int) Math.Round (index);
      this [intIndex] = value;
 }
}
public bool this [bool index] {
 //Индексатор без set, только для чтения, возвращает состояние this.ErrFlag
 get {
      return ErrFlag;
 }
 //Аксессор set отсутствует
}
private bool IndexIsValid (int index) {
 if (Length < 1) return false;
 return (index >= 0 & index < Length ? true : false);
}
}

public class Array2D <T> where T : new() { //"Отказоустойчивая" матрица с
индексаторами
 T [,] a; //Ссылка на базовый массив
 int rows, cols; //Количество строк и столбцов, приватные
 int len; //Длина массива, на этот раз приватная
 public int Length { //Получим её свойством "только для чтения"
 get { return len; }
 }
 bool err; //Результат последней операции, приватный
 public bool ErrFlag { //Получим его свойством "только для чтения"
 get { return err; }
 }
 public Array2D (int r, int c) {
 if (r > 0 & c > 0) {
      rows = r; cols = c;
```

```
        a = new T [rows, cols];
        len = rows * cols;
    }
    else len = 0;
}
public void Print (String hdr = "") { //Вывод в консоль
    Console.WriteLine ();
    Console.WriteLine (hdr);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
        Console.Write ("{0} ", a [i, j]);
        Console.WriteLine ();
    }
}
public T this [int row, int col] { //Двумерный индексатор
    get {
        if (IndexIsValid (row, col)) {
        err = false;
        return a [row, col];
        }
        else {
        err = true;
        return default (T);
        }
    }
    set {
        if (IndexIsValid (row, col)) {
        a [row, col] = value;
        err = false;
        }
        else err = true;
    }
}
public int Rows { //свойство Rows для изменения приватного количества
строк
    get { return rows; }
    set { if (value > -1 & value < rows) rows = value; }
}
```

```
        public int Cols { //свойство Rows для изменения приватного количества столбцов
         get { return cols; }
         set { if (value > -1 & value < cols) cols = value; }
        }
        private bool IndexIsValid (int r, int c) {
         if (len < 1) return false;
         return ( r > -1 & r < rows & c > -1 & c < cols ? true : false );
        }
       }

       class Program {
        static void Main (string [] args) {
         Array <int> arr = new Array <int> (5);
         for (int i = 0; i < arr.Length + 1; i++) { //Берём 1 лишний элемент!
              arr [i] = i + 1;
              Console.WriteLine (arr [i] + "  (ErrFlag = "+ arr.ErrFlag + ")");
         }
         Array <double> darr = new Array <double> (5);
         for (double x = 0.1; x < arr.Length ; x += 0.9) { //Берём 1 лишний элемент, 2-й индексатор
              darr [x] = x + 1;
              Console.WriteLine (arr [x] + "  (ErrFlag = " + arr.ErrFlag + ")");
         }
         Console.WriteLine ("arr [true]   = " + arr [true]); //Третий индексатор

         Array2D <int> matr = new Array2D<int> (2, 2);
         for (int i = 0; i < 3; i++) { //Лишняя строка
              Console.WriteLine();
              for (int j = 0; j < 2; j++) {
              matr [i, j] = i + j;
              Console.Write ("{0:D} ({1}) ", matr [i, j], matr.ErrFlag);
              }
         }
         matr.Rows = 2; //Меняем количество строк с помощью свойства
         matr.Print ("Матрица после изменения количества строк");

        Console.ReadKey ();
        }
```

```
 }

}
```

22. Наследование

Довольно развёрнутый пример, показывающий особенности реализации наследования в классах C#.

Абстрактный, базовый и производный классы, приватные данные и публичные свойства-"обёртки" над ними, абстрактные и виртуальные методы, приватные, публичные и защищённые члены класса, неуниверсальный статический класс с расширениями, класс, запрещённый к наследованию, массив из объектов базового класса.

```
//C# -> Консольное приложение (.NET Framework)
using System;
using System.Collections.Generic;

namespace ConsoleApp1 {
 abstract class Shape { //Абстрактный класс "фигура"
  public abstract string GetTypeString (); //Абстрактный метод, должен быть определён у потомков
 }

 class Shape2D : Shape { //Базовый класс - рамка фигуры
  double width, height; //Приватные ширина и высота
  int frac = 2; //Количество знаков в дробной части при выводе
  public Shape2D (double width = 0, double height = 0, int frac = 2) {
   //Конструктор базового класса-"рамки"
   Width = width;
   Height = height;
   this.frac = frac;
  }
  public double Width { // Публичные свойства ширины и высоты двумерного объекта
   get { return width; }
   set { width = value < 0 ? -value : value; }
  }
  public double Height {
   get { return height; }
   set { height = value < 0 ? -value : value; }
```

```
        }
        public int Frac { //Публичное свойство "количество знаков в дробной части"
         get { return frac; }
         set { width = frac < 1 ? 0 : ( frac > 14 ? 14 : value ); }
        }
        public double RoundTo (double d) { //Округлить до нужного количества зна-
ков
         return Math.Round (d, Frac);
        }
        public string GetSizeString () { //Строка с габаритами
         return RoundTo (Width) + " x " + RoundTo (Height);
        }
        public override string GetTypeString () { //Реализация абстрактного метода
         return "рамка фигуры";
        }
        public virtual double Area () { //Виртуальный метод "площадь рамки"
         return RoundTo (Width * Height);
        }
       }

       class Triangle : Shape2D { //Производный класс - треугольник
        protected string Style; //Тип треугольника, защищённое свойство
        protected double A, B, C; //Стороны треугольника, защищённые свойства
        public Triangle (double A = 3, double B = 4, double C = 5) {
         if (A + B > C & A + C > B & B + C > A) {
              double [] Temp = new double [] { A , B , C };
              Array.Sort (Temp);
              this.C = Width = Temp [2];
              this.B = Temp [1];
              this.A = Temp [0];
              Height = GetHeight (Temp [1]);
              var Temp2 = new List <double> () { Temp [0] * Temp [0], Temp [1] *
Temp [1], Temp [2] * Temp [2] };
              this.Style = Temp2 [2] < Temp2 [0] + Temp2 [1] ?
              (Temp2 [0] == Temp2 [1] ? "равносторонний" : "остроугольный") :
              (Temp2 [2] > Temp2 [0] + Temp2 [1] ? "тупоугольный" : "прямоуголь-
ный");
         }
         else {
```

```
            this.A = this.B = this.C = Width = Height = 0;
            this.Style = "не существует";
        }
        }
        private double GetHeight (double a) {//высота, опущенная на сторону a (большую из всех)
        double P = ( A + B + C ) / 2;
        return RoundTo (2 * Math.Sqrt (P * ( P - A ) * ( P - B ) * ( P - C )) / A);
        }
        public override double Area () { //Площадь треугольника
        double P = ( A + B + C ) / 2;
        return RoundTo (Math.Sqrt(P * (P - A) * (P - B) * (P - C)));
        }
        public override string GetTypeString () { //Переопределение виртуального метода
        return Style;
        }
        }
        public static class Measures { //Неуниверсальный статический класс с расширениями
        public static double ToRadians (this double angleInDegree) { //Угол в градусах - в радианы
        return ( angleInDegree * Math.PI ) / 180.0;
        }
        }

        sealed class Triangle2C : Triangle {
        //Производный от треугольника - треугольник, заданный двумя сторонами и углом между ними
        //Этот класс наследовать уже нельзя (sealed)
        public Triangle2C (double A, double B, double angleC) :
        base (A, B,
            Math.Sqrt(Math.Pow(A,2) + Math.Pow(B,2) - 2*A*B*Math.Cos(Measures.ToRadians(angleC)))) {
        //использует конструктор базового класса, вычислив по теореме косинусов третью сторону
        }
        }
```

```
class Program { //Главный класс, демонстрация
 static void Main (string [] args) {
  Triangle t1 = new Triangle ();
  Console.WriteLine ("Сведения об объекте t1: тип {0}, габариты {1}",
       t1.GetTypeString (), t1.GetSizeString ());
  Console.WriteLine ("Площадь равна " + t1.Area () + System.Environment.NewLine);

  Triangle t2 = new Triangle (4,4,4);
  Console.WriteLine ("Сведения об объекте t2: тип {0}", t2.GetTypeString ());
  Console.WriteLine ("Габариты равны {0}x{1}", t2.Width, t2.Height);
  Console.WriteLine ("Площадь равна " + t2.Area ());
  Console.WriteLine ();

  Triangle2C t3 = new Triangle2C (4,4,60); //Совпадёт со 2-м, но задан по-другому
  Console.WriteLine ("Сведения об объекте t3: тип {0}, габариты {1}",
       t3.GetTypeString (), t3.GetSizeString ());
  //тип изменился с "равносторонний" на "остроугольный" из-за погрешностей при расчёте C!
  Console.WriteLine ("Площадь равна " + t3.Area ());
  Console.WriteLine ();

  Shape2D [] Shapes = new Shape2D [4]; //Массив объектов базового класса
  Shapes [0] = t1;
  Shapes [1] = t2;
  Shapes [2] = t3;
  Shapes [3] = new Shape2D(3,-4,1); //Свойство Height превратит "-4" в "4"
  foreach (var Shape in Shapes)
       Console.WriteLine ($"Тип объекта = {Shape.GetTypeString()}");
  Console.WriteLine ();

  Console.ReadKey ();
 }
}

} //namespace
```

23. Упаковка и распаковка

Все типы в C#, включая простые типы значений, являются производными от класса object. Следовательно, ссылкой типа object можно воспользоваться для обращения к любому другому типу, в том числе и к типам значений.

Когда ссылка на объект класса object используется для обращения к типу значения, такой процесс называется упаковкой. Упаковка приводит к тому, что значение простого типа сохраняется в экземпляре объекта, т.е. "упаковывается" в объекте, который затем используется, как и любой другой объект. Но в любом случае упаковка происходит автоматически. Для этого достаточно присвоить значение переменной ссылочного типа object, а об остальном позаботится компилятор.

Распаковка представляет собой процесс извлечения упакованного значения из объекта. Это делается с помощью явного приведения типа ссылки на объект класса object к соответствующему типу значения. Попытка распаковать объект в другой тип может привести к ошибке времени выполнения.

В примере значение типа int передаётся в качестве аргумента методу Sqr(), который, в свою очередь, принимает параметр типа object.

Также показана работа с массивом из разнотипных элементов, точнее, из элементов базового класса object.

```
//C# -> Консольное приложение (.NET Framework)
using System;

namespace ConsoleApp1 {

 class BoxingDemo {
  int x;
  BoxingDemo (int x = 0) { this.x = x;  } //Конструктор
  public override string ToString () { return x.ToString (); }
  //Переопределение метода ToString класса object
  public static int Sqr (object o) { //Метод для возведения в квадрат с параметром типа object
   return (int) o * (int) o;
  }
 }

 class Program { //Главный класс Program
  static int Main () {
   int x = 10;
   Console.WriteLine ("x = " + x);
```

```
        x = BoxingDemo.Sqr (x);
            // значение x автоматически упаковывается, когда оно передается методу Sqr()
        Console.WriteLine ("x^2 = " + x);

        object obj = x; // упаковать значение переменной x в объект
        int y = (int) obj;
            // распаковать значение из объекта, доступного по ссылке obj, в переменную типа int
        Console.WriteLine ("y = " + y);

        object [] arr = new object [3]; //массив из разнотипных значений
        arr [0] = x;
        arr [1] = (double) x + 0.5;
        arr [2] = "Привет";
        for (int i = 0; i < arr.Length; i++) {
            var item = arr [i];
            var type = item.GetType ();
            Console.WriteLine ($"arr[{i}] = {item} ({type})");
        }

        Console.ReadKey ();
        return 0;
       }
      } //Program

     } //namespace
```

Практическая работа №5. Метрики потока данных компьютерных программ

Цель работы: получить навыки в расчете метрик потока данных компьютерных программ.

Теоретическая часть

При оценке сложности программ, как правило, выделяют три основные группы метрик: – метрики размера программ; – метрики сложности потока управления программ; – метрики сложности потока данных программ.

Оценки первой группы наиболее просты и поэтому получили широкое распространение. Традиционной характеристикой размера программ является количество строк исходного текста. Под строкой понимается любой оператор программы. Непосредственное измерение размера программы, несмотря на свою простоту, дает хорошие результаты. Оценка размера программы недостаточна для принятия решения о ее сложности.

Таким образом, оценка размера программы есть оценка по номинальной шкале, на основе которой определяются только категории программ без уточнения оценки для каждой категории. К группе оценок размера программ можно отнести также метрику Холстеда. За базу принят подсчет количества операторов и операндов, используемых в программе, т.е. определение размера программы.

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

– h1 - число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

– h2 – число уникальных операндов программы (словарь операндов);

– N1 – общее число операторов в программе;

– N2 – общее число операндов в программе.

Опираясь на эти характеристики, получаемые непосредственно при анализе исходных текстов программ, Холстед вводит следующие оценки:

Словарь программы $h=h1 + h2$;

Длину программы $N=N1+N2$;

Объем программы $V=N \log_2 h$.

Смысл оценок h и N достаточно очевиден, поэтому подробно рассмотрим только характеристику V. Количество символов, используемых при реализации некоторого алгоритма, определяется в числе прочих параметров и словарей программы h, представляющим собой минимально необходимое число символов, обеспечивающих реализацию алгоритма.

Далее Холстед вводит h* - теоретический словарь программы, т.е. словарный запас, необходимый для написания программы с учетом того, что необходимая функция уже реализована в данном языке и, следовательно, программа сводится к вызову этой функции.

Следующей метрикой сложности потока данных программ является метрика Чепина.

Существует несколько ее модификаций. Рассмотрим более простой, а с точки зрения практического использования – достаточно эффективный вариант этой метрики. Суть метода состоит в оценке информационной прочности отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода.

Все множество переменных, составляющих список ввода вывода, разбивается на четыре функциональные группы P – вводимые переменные для расчетов и для обеспечения вывода.

Примером может служить используемая в программах лексического анализатора переменная, содержащая строку исходного текста программы, то есть сама переменная не модифицируется, а только содержит исходную информацию. M – модифицируемые или создаваемые внутри программы переменные. C – переменные, участвующие в управлении работой программного модуля (управляющие переменные). Далее вводится значение метрики Чепина:

Q = a1P + a2M + a3C + a4T, где a1, a2, a3, a4 – весовые коэффициенты.

Весовые коэффициенты использованы для отражения различного влияния на сложность программы каждой функциональной группы. По мнению автора метрики, наибольший вес, равный трем, имеет функциональная группа C, так как она влияет на поток управления программы. Весовые коэффициенты остальных групп распределяются следующим образом: a1=1; a2=2; a4=0.5. Весовой коэффициент группы T не равен нулю, поскольку «паразитные» переменные не увеличивают сложности потока данных программы, но иногда затрудняют ее понимание.

С учетом весовых коэффициентов выражение примет вид:

Q = P + 2M + 3C + 0.5T

Следует отметить, что рассмотренные метрики сложности программы основаны на анализе исходных текстов программ, что обеспечивает единый подход к автоматизации их расчета.

Рабочее задание: На основании выданного преподавателем задания (см. согласно варианту в практических работе 4) написать программу на заданном языке программирования. В ходе написания программы реализовать вывод всех входных и выходных данных.

Используя исходный текст программы, необходимо рассчитать 3 базовых метрики Холстеда и метрику Чепина. Ответить на контрольные вопросы.

Контрольные вопросы:

Какие метрики оценки сложности программ существуют?

Какие характеристики составляют метрику Холстеда?

Что такое метрика сложности потока данных?

Как рассчитывается метрика Чепина?

Время работы: 2 часа.

Практическая работа № 6. Тестирование белым ящиком

Цель работы: научиться проводить тестирование программы методом белого ящика

Теоретическая часть

Динамическое тестирование предполагает запуск программы, выполнение всех ее функциональных модулей и сравнение фактического ее поведения с ожидаемым.

Статическое тестирование позволяет обнаружить дефекты, которые являются результатом ошибки и привести к сбоям в программном обеспечении. Динамическое тестирование позволяет продемонстрировать непосредственно сбои в программном обеспечении.

Существует несколько признаков, по которым принято производить классификацию видов тестирования. По знанию системы выделяют:

– тестирование «черного ящика» (black box testing);

– тестирование «белого ящика» (white boxogting);

– тестирование «серого ящика» (grey box testing).

Метод белого ящика (white box testing, open box testing, clear box testing, glass box testing) - у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.

Разработка тестов методом белого ящика (white-box test design technique) предполагает процедуру разработки или выбора тестовых сценариев на основании анализа внутренней структуры компонента или системы.

Техники, основанные на структуре, или методе белого ящика

– тестирование операторов;

– тестирование альтернатив.

Альтернатива (decision): Точка программы, в которой управление имеет два или более альтернативных путей. Узел с двумя или более связями для разделения ветвей.

Тестирование условий альтернатив (decision condition testing): Разработка тестов методом белого ящика, при котором тестовые сценарии проектируются для исходов условий и результатов альтернатив.

Покрытие (coverage): Уровень, выражаемый в процентах, на который определенный элемент покрытия был проверен набором тестов.

Покрытие альтернатив (decision coverage): Процент результатов альтернативы, который был проверен набором тестов. Стопроцентное покрытие решений подразумевает стопроцентное покрытие ветвей и стопроцентное покрытие операторов.

Покрытие кода (code coverage): Метод анализа, определяющий, какие части программного обеспечения были проверены (покрыты) набором тестов, а какие

нет, например, покрытие операторов, покрытие альтернатив или покрытие условий. Еще выделяют серый Ящик

Рабочие задания

Задание 1. Разработать программу на Python.

Даны длины сторон треугольника, определить вид треугольника и его площадь. Выполнить контроль вводимых чисел.

1. Разносторонний треугольник
2. Равносторонний треугольник
3. Равнобедренный треугольник

Ограничения:

- три числа не могут быть определены как стороны треугольника, если хотя бы одно из них меньше или равно 0;
- сумма двух из них меньше третьего.

Задание 2. Подготовить набор тестовых вариантов для обнаружения ошибок и программе.

Результат оформить в следующем виде:

A	B	C	Ожидаемый результат	Объект проверки
Значение	Значение	Значение	Что должно получиться	Значения вводимых данных, либо ожидаемый результат
..	…..	..	…..	…..

Задание 3. Разработать программу на Python.

Даны длины сторон треугольника, определить вид треугольника и его площадь. Выполнить контроль вводимых чисел.

1. Остроугольный треугольник
2. Тупоугольный треугольник
3. Прямоугольный треугольник

Ограничения:

- три числа не могут быть определены как стороны треугольника, если хотя бы одно из них меньше или равно 0;
- сумма двух из них меньше третьего.

Подготовить набор тестовых вариантов для обнаружения ошибок в программе и оформить результат.

Задание 4. На основании проведенных тестов составьте рекомендации по исправлению ошибок, выявленных в ходе тестирования в виде отчета.

Пример:

В ходе проведения первого теста было обнаружено, что при в ведении некорректных данных площадь все равно высчитывается.

Рекомендуется: в случае, если пользователь введет не корректные данные, следует выводить сообщение с просьбой исправить введенные значения. Добавить в программу проверку введенных значений на соответствие ограничения.

Время работы: 2 часа.

Практическая работа №7. Тестирование методом черного ящика

Цель работы: изучить метод тестирования «Черным ящиком»

Теоретическая часть

Сегодня тестирование — это обязательная часть процесса разработки программного обеспечения (далее ПО). Это связано с жесткими правилами конкуренции для компаний, производящих программные продукты (ПП).

Раньше таких компаний на рынке было мало, и пользователи программных продуктов были продвинутыми и заменяли тестеров. Если в программе обнаруживались баги, то пользователь звонил или отправлял письмо в компанию, где ошибку исправляли и по почте отправляли дискетку со свежим релизом. Но начиная с 1990 года, согласно статистики продажи персональных компьютеров с каждым годом удваивались. И появилась армия пользователей, которая не готова была что-то тестировать. Если что-то не устроило было проще обменять на другой софт, т.к. число компаний, производящих ПО, тоже росло ежегодно. И у пользователей появился выбор что покупать и чем пользоваться.

Таким образом, тестирование ушло внутрь компаний, и появилась профессия тестировщика.

Рассмотрим определение, которое записано в SWEBOK.

Тестирование ПО – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом. [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].

Все виды тестирования можно условно разделить на две большие группы: статическое тестирование (static testing), динамическое тестирование (dynamic testing).

Статическое тестирование – это процесс анализа самой разработки программного обеспечения, т. е. тестирование без запуска программы. Данный вид

тестирования осуществляется в основном программистами. Проводят тестирование артефактов разработки программного обеспечения, таких как требования, дизайн или программный код, проводимое без исполнения этих артефактов. Например, с помощью рецензирования или статического анализа. Статический анализ кода (static code analysis) это анализ исходного кода, производимый без его исполнения.

Динамическое тестирование – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта.

Динамическое тестирование предполагает запуск программы, выполнение всех её функциональных модулей и сравнение фактического ее поведения с ожидаемым.

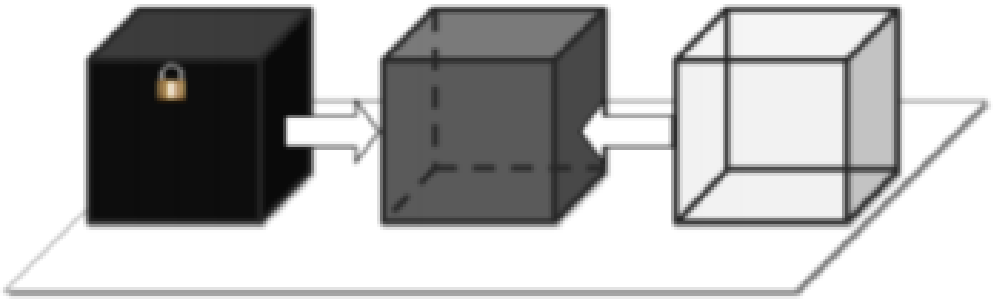
Статическое тестирование позволяет обнаружить дефекты, которые являются результатом ошибки и привести к сбоям в программном обеспечении. Динамическое тестирование позволяет продемонстрировать непосредственно сбои в программном обеспечении

Существует несколько признаков, по которым принято производить классификацию видов тестирования. По знанию системы выделяют:

– тестирование «черного ящика» (black box testing);

– тестирование «белого ящика» (white box testing); тестирование «серого ящика» (grey box testing).

Метод чёрного ящика (black box testing, closed box testing) у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования.



Процедура создания и/или выбора тестовых сценариев, основанная на анализе функциональной или нефункциональной спецификация компонента или системы без знания внутренней структуры. Техники разработки тестов на основе спецификаций, или методе черного ящика:

– эквивалентное разбиение;

– анализ граничных значений;

– тестирование таблицы решений;

Эквивалентное разбиение (equivalence partitioning) - разработка тестов методом черного ящика, в которой тестовые сценарии создаются для

проверки элементов эквивалентной области. Как правило, тестовые сценарии разрабатываются для покрытия каждой области как минимум один раз.

Входные данные для программного обеспечения или системы, от которых ожидается сходное поведение системы, разбиваются на группы, то есть они должны обрабатываться аналогичным образом. Эквивалентные области (или классы) могут быть определены как для валидных, так и для невалидных данных, то есть тех значений, которые должны отвергаться. Эквивалентное разбиение применимо на всех уровнях тестирования, может быть использовано с целью покрытия входных и выходных данных. Оно может применяться при ручном вводе данных, при передаче данных через интерфейсы в систему, или при проверке параметров интерфейсов в интеграционном тестировании.

Анализ граничных значений (boundary value analysis) – разработка тестов методом черного ящика, при котором тестовые сценарии проектируются на основании граничных значений.

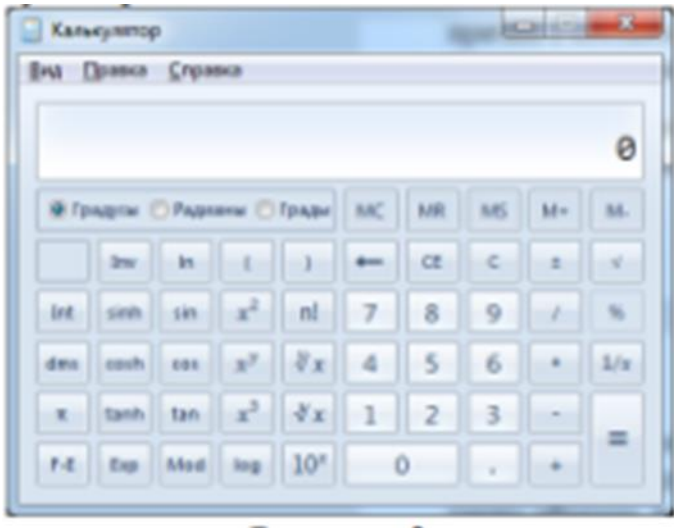
Граничное значение (boundary value): Входное значение или выходные данные, которые находятся на грани эквивалентной области или на наименьшем расстоянии от обеих сторон грани, например, минимальное или максимальное значение области. Анализ граничных значений может применяться на всех уровнях тестирования.

Таблица решений (decision table): Таблица, отражающая комбинации входных данных и/или причин с соответствующими выходными данными и/или действиям (следствиям), которая может быть использована для проектирования тестовых сценариев. Таблицы решений хороший метод для сбора системных требований, содержащих логические условия и документирования внутреннего дизайна системы. Они могут использоваться для записи сложных бизнес-правил, которые должна реализовывать система. Анализируются спецификации и определяются условия и действия системы. Входные условия и действия чаще всего формулируются таким образом, чтобы они могли принимать логические значения «истина» или «ложь».

Сильной стороной тестирования таблицы решений является то, что она создает комбинации условий, которые могли бы быть не проверены в ходе тестирования иным способом. Этот метод может быть применен ко всем ситуациям, в которых действие программного продукта зависит от нескольких логических альтернатив.

Рабочие задания

Задание 1. Написать калькулятор с небольшими багами.



Задание 2. Обменяться программой с другими студентами. Провести тестирование и написать отчет.

Название теста	Описание сценария	Входные данные	Выходные данные	Удачное или неудачное тестирование	Предложения по исправлению найденных ошибок.	Название теста
Функция сумму	Сложение двух положительных чисел, проверка результата	Первая переменная = 3 Вторая переменная =8	Результат =11	неудачное	-	Функция сумму

Время работы: 2 часа

Перечень использованных информационных источников

1. Староверова, Н.А. Операционные системы [Электронный ресурс]: учебное пособие / Н.А. Староверова, Э.П. Ибрагимова. — Электрон. дан. — Казань: КНИТУ, 2016. — 312 с. — Режим доступа: https://e.lanbook.com/book/101906.

2. Назаров, С.В. Современные операционные системы [Электронный ресурс]: учебное пособие / С.В. Назаров, А.И. Широков. — Электрон. дан. — Москва: 2016. — 351 с. — Режим доступа: https://e.lanbook.com/book/100498.

3. Мясников, В.И. Операционные системы реального времени: лабораторный практикум [Электронный ресурс]: учебное пособие / В.И. Мясников. — Электрон. дан. — Йошкар-Ола: ПГТУ, 2016. — 140 с. — Режим доступа: https://e.lanbook.com/book/92562.

4. Шубина, М.А. Операционные системы [Электронный ресурс]: учебное пособие / М.А. Шубина. — Электрон. дан. — Санкт-Петербург: СПбГЛТУ, 2015. — 132 с. — Режим доступа: https://e.lanbook.com/book/71880.

5. Куль, Т.П. Операционные системы: учебное пособие / Т.П. Куль. - Минск: РИПО, 2015. - 312 с.: ил. - Библиогр. в кн. - ISBN 978-985-503-460-6; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=463629.

6. Проскурин, В.Г. Защита в операционных системах [Электронный ресурс]: учебное пособие / В.Г. Проскурин. — Электрон. дан. — Москва: Горячая линия-Телеком, 2016. — 192 с. — Режим доступа: https://e.lanbook.com/book/111091.

7. Курячий, Г.В. Операционная система Linux [Электронный ресурс]: учебник / Г.В. Курячий, К.А. Маслинский. — Электрон. дан. — Москва: 2016. — 450 с. — Режим доступа: https://e.lanbook.com/book/100278.

8. Операционные системы [Электронный ресурс] / Д.В. Груздев. — Воронеж: Издательский дом ВГУ, 2017. — 42 с. — Режим доступа: https://lib.rucont.ru/efd/670095.