



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

---

**Институт информационных технологий (ИИТ)**

**Кафедра математического обеспечения и стандартизации  
информационных технологий (МОСИТ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1**

по дисциплине «Тестирование и верификация программного обеспечения»

**Тема:** Тестирование программного продукта методом «черного ящика»

**Группа:** ИКБО-66-23

**Состав команды:** Спесивцев Ю. А.  
Анисимов Д. Д.  
Корявцев А. Д.  
Хомченко В. А.

**Дата выполнения:** 16 сентября 2025 г.

Москва 2025

## ***Часть 1. Разработка технического задания и программного продукта***

### **1. Техническое задание (ТЗ) собственного ПП**

#### **1.1 Введение**

Настоящее техническое задание определяет цели, требования и этапы разработки десктопного приложения с графическим интерфейсом "ToDoMaster" (далее — "Программа"). Проект реализуется на языке Python с использованием библиотеки Tkinter и предназначен для удобного управления списком персональных задач.

#### **1.2 Основания для разработки**

Разработка проводится в рамках самостоятельного обучения разработчика.

Основанием является личная инициатива с целью:

- Практического применения знаний Python.
- Создания законченного рабочего проекта для портфолио.
- Изучения принципов структурирования кода и разработки приложений.

#### **1.3 Назначение работника**

Программа предназначена для пользователей, которым требуется простой и быстрый инструмент для ведения списка текущих дел в командной строке.

#### **1.4 Требование к программе**

- Добавление новой задачи в список.
- Просмотр списка всех задач в виде карточек.
- Отметка задачи как выполненной (чекбокс).
- Удаление задачи из списка.
- Фильтрация задач: "Все", "Активные", "Выполненные".
- Подсветка статуса задач: выполненные — серым, просроченные — красным, задачи на сегодня — зеленым.

- Отображение статистики: общее количество задач и количество невыполненных.
- Сохранение списка задач в файл tasks.json для восстановления между сеансам

#### **1.4.3 Требования к надежности**

- Программа не должна завершаться с ошибкой при вводе некорректных данных.
- Данные должны сохраняться корректно, без потерь. Должна быть реализована обработка ошибок при работе с файлом.

#### **1.4.4 Требования к эксплуатации**

- ОС: Windows, Linux, macOS.
- Необходим интерпретатор Python версии 3.6 или выше.
- Необходимы библиотеки: tkinter (входит в стандартную поставку Python).

#### **1.4.4 Требования к совместимости**

Приложение должно быть кроссплатформенным и кроссбраузерным.

Совместимость с операционными системами: • Приложение должно работать под управлением любых операционных систем, на которых могут быть запущены современные веб-браузеры (включая Windows, macOS, Linux). Это обеспечивается тем, что исполняемой средой является браузер, а не ОС.

#### **1.5 Требования к интерфейсу**

- Графический интерфейс (GUI), реализованный на Tkinter.
- Поле ввода для новой задачи.
- Область для отображения списка задач в виде карточек.
- Чекбокс для отметки выполнения.
- Кнопка "Удалить" для каждой задачи.

- Панель фильтров ("Все", "Активные", "Выполненные").
- Строка статуса с статистикой.

## 1.6 Требования приемки

Проект считается завершенным и успешным, если:

1. Все функции, перечисленные в п. 1.4, реализованы и работают корректно.
2. Программа работает стабильно, без критических ошибок.
3. Интерфейс интуитивно понятен и соответствует требованиям.
4. Код читаем, хорошо структурирован и содержит комментарии.

## 1.7 Порядок контроля приемки

Процедура контроля и приемки выполняется на основании формализованного процесса верификации, описанного в данном ТЗ.

1. **Метод тестирования:** Тестирование проводится методом черного ящика (Black Box Testing). Тестировщик не имеет доступа к исходному коду и руководствуется только данным ТЗ и внешним поведением приложения, доступным через пользовательский интерфейс.
2. **Цель тестирования:** Проверить соответствие готового продукта всем функциональным и нефункциональным требованиям, изложенным в разделах 1.4 и 1.5 настоящего ТЗ.
3. **Процедура приемки:**
  - **Шаг 1:** Предоставление готового программного продукта и инструкции по запуску.
  - **Шаг 2:** Проведение приемочного тестирования по следующим критериям:
    - Соответствие функциональным требованиям (п. 1.4).
    - Соответствие требованиям к интерфейсу (п. 1.4).
    - Отсутствие критических ошибок (завершение работы, потеря данных).

- **Шаг 3:** Фиксация результатов тестирования в виде отчета о тестировании.

**4. Критерий успешного прохождения приемки:** Успешное выполнение всех тестовых сценариев, составленных на основе требований данного ТЗ, и отсутствие критических дефектов.

функциональным и нефункциональным требованиям, изложенным в разделах 1.4 и 1.5 настоящего ТЗ.

Процедура приемки:

1. Предоставление ПП и инструкции по запуску
2. Проведение приемочного тестирования
3. Фиксация результатов.

### 1.8 Этапы и сроки разработки

№	Этап разработки	Срок исполнения	Примечание
1	Проектирование интерфейса	1 день	
2	Реализация базовой логики	1 день	
3	Реализация работы с файлами	2 дня	
4	Тестирование и отладка	1 день	
5	Тестирование и отладка	1 день	
6	Рефакторинг кода, комментирование	1 день	
7	Написание документации	1 день	
	Итого:	7 дней	<i>Сроки ориентировочные</i>

## 2. Дополнительная документация на собственный ПП

### 2.1 Инструкция по запуску

1. Убедитесь, что установлен Python 3.6+.
2. Сохраните код программы в файл todo.py.
3. Запустите в терминале: python todo.py.

### 2.2 Инструкции по управлению

- **Добавление задачи:** Введите текст в верхнее поле ввода и нажмите Enter.
- **Отметка выполнения:** Установите или снимите флажок (чекбокс) рядом с задачей.
- **Удаление задачи:** Нажмите кнопку "Удалить" на карточке задачи.
- **Фильтрация:** Используйте кнопки "Все", "Активные", "Выполненные" в нижней части окна.
- **Статистика:** Количество задач отображается в строке статуса внизу окна.

## 3. Описание ошибок внесенных в собственное ПО

В целях выполнения практической работы №1 в ПО были внесены следующие ошибки:

1. Ошибка надежности: Нет обработки исключений для файла (например, если он поврежден или имеет неверный формат JSON).

*Листинг 1. Описание добавленной ошибки*

```
def load_tasks(self):
    if os.path.exists('tasks.json'):
        with open('tasks.json', 'r', encoding='utf-8')
as f:
            self.tasks = json.load(f) # Нет блока try-
except для обработки ошибок JSON
        self.render_tasks()
```

2. Ошибка интерфейса: Отсутствует кнопка "Добавить" для новой задачи, добавление привязано только к клавише Enter.

*Листинг 2. Описание добавленной ошибки*

```
# add_btn = ttk.Button(top_frame, text="Добавить",
command=self.add_task) # ЗАКОММЕНТИРОВАННАЯ КНОПКА
# add_btn.grid(row=0, column=1)
```

3. Ошибка логики: Статус бар (отображение статистики) не обновляется при добавлении, удалении или изменении статуса задачи. Метод `update_status()` нигде не вызывается.

*Листинг 3. Описание добавленной ошибки*

```
def add_task(self, event=None):  
    # ... код добавления задачи ...  
    # self.update_status() # Ошибка логики: вызов метода  
    обновления статуса закомментирован
```

4. Ошибка функционала: Поле `due_date` (срок выполнения) объявлено в структуре задачи, но полностью отсутствует функционал для его установки, редактирования и проверки.

*Листинг 4. Описание добавленной ошибки*

```
new_task = {  
    "id": len(self.tasks),  
    "text": task_text,  
    "completed": False,  
    "due_date": None # Ошибка функционала: Поле есть, но  
функционала нет  
}
```

5. Ошибка логики: Удаление задачи происходит по `id`, но `id` назначаются по порядковому номеру (индексу) в списке. После удаления одной задачи `id` остальных задач перестают соответствовать их позиции в списке, что может привести к ошибкам.

*Листинг 5. Описание добавленной ошибки*

```
def delete_task(self, task_id):  
    # Ошибка логики №5: Удаление по id, но id назначаются  
по индексу.  
    self.tasks = [task for task in self.tasks if  
task["id"] != task_id]  
    self.render_tasks()  
    self.save_tasks()
```

6. Ошибка интерфейса: Для карточек задач используется `tk.Frame` со стилем `relief='solid'`. Данный стиль не работает с `tk.Frame`, предназначен для `tk.Frame`. Границы карточек отображаться не будут.

*Листинг 6. Описание добавленной ошибки*

```
task_card = ttk.Frame(self.task_frame, relief='solid',  
borderwidth=1)  
# Ошибка интерфейса №6: Стилль 'solid' не работает с  
ttk.Frame
```

7. Ошибка логики: При отрисовке задач есть проверка на просроченность (`is_overdue`), которая опирается на поле `due_date`. Однако это поле всегда равно `None`, т.к. функционал его заполнения отсутствует. Условие всегда ложно, и пометка "ПРОСРОЧЕНО" никогда не появляется.

*Листинг 7. Описание добавленной ошибки*

```
is_overdue = task["due_date"] and  
datetime.now().date() >  
datetime.strptime(task["due_date"], "%Y-%m-%d").date()  
# Ошибка логики №7: ... Условие всегда ложно.
```

8. Ошибка интерфейса: Попытка изменить цвет текста метки (`foreground='gray'`) для выполненных задач прямым способом (`lbl.config`). Для виджетов `ttk` необходимо использовать стили (`ttk.Style`), прямое изменение атрибутов может не работать на всех платформах.

*Листинг 8. Описание добавленной ошибки*

```
lbl = ttk.Label(task_card, text=lbl_text, anchor='w')  
if task["completed"]:  
    lbl.config(foreground='gray') # Ошибка интерфейса  
№8: Для ttk.Label нужно использовать style
```



## ***Часть II. Тестирование ПП***

### **1. Техническое задание и документация ПП другой команды.**

#### **2.1.1 Введение**

Настоящее техническое задание определяет цели, требования и этапы разработки десктопного приложения с графическим интерфейсом "Калькулятор" (далее — "Программа"). Проект реализуется на языке Golang с использованием библиотек `bufio`, `fmt`, `os`, `strcow`, `strings` и предназначен для удобного управления списком персональных задач.

#### **2.1.2 Основания для разработки**

Разработка проводится в рамках самостоятельного обучения разработчика. Основанием является личная инициатива с целью:

- Практического применения знаний Golang.
- Создания законченного рабочего проекта для портфолио.
- Изучения принципов структурирования кода и разработки приложений.

#### **2.1.3 Назначения работника**

Программа предназначена для пользователей, которым требуется простой и быстрый инструмент для математических вычислений.

#### **2.1.4 Требование к программе**

- Операция сложения.
- Операция вычитания.
- Операция деления.
- Операция умножения.

#### **2.1.5 Требования к надежности**

- Программа не должна завершаться с ошибкой при вводе некорректных данных.

#### **2.1.6 Требования к эксплуатации**

- ОС: Windows, Linux, macOS.
- Необходим интерпретатор Golang версии 1.19.0 или выше.
- Необходимы библиотеки: `bufio`, `fmt`, `os`, `strcow`, `string`.

### **2.1.7 Требования к совместимости**

Приложение должно быть кроссплатформенным. Совместимость с операционными системами:

- Приложение должно работать под управлением любых операционных систем, на которых могут быть запущены современные веб-браузеры (включая Windows, macOS, Linux). Это обеспечивается тем, что исполняемой средой является браузер, а не ОС.

### **2.1.8 Требования к интерфейсу**

- Графический интерфейс (CLI), реализованный на Golang.
- Поле ввода для новой задачи.
- Поле вывода результата.
- Инструкция пользования приложения
- Строка статуса с статистикой.

### **2.1.9 Требования приемки**

Проект считается завершенным и успешным, если:

1. Все функции, перечисленные в п. 1.4, реализованы и работают корректно.
2. Программа работает стабильно, без критических ошибок.
3. Интерфейс интуитивно понятен и соответствует требованиям.
4. Код читаем, хорошо структурирован и содержит комментарии.

### **2.1.10 Порядок контроля приемки**

Процедура контроля и приемки выполняется на основании формализованного процесса верификации, описанного в данном ТЗ.

1. Метод тестирования: Тестирование калькулятора проводится методом черного ящика. Фокус направлен на проверку корректности вычислений и обработки ошибок через пользовательский интерфейс, без доступа к исходному коду.
2. Цель тестирования: Проверить соответствие готового продукта всем функциональным и нефункциональным требованиям, изложенным в разделах 1.4 и 1.5 настоящего ТЗ.

### 3. Процедура приемки:

- Шаг 1: Предоставление готового программного продукта и инструкции по запуску.
- Шаг 2: Проведение приемочного тестирования по следующим критериям:
  - Соответствие функциональным требованиям (п. 1.4).
  - Соответствие требованиям к интерфейсу (п. 1.4).
  - Отсутствие критических ошибок (завершение работы, потеря данных).
- Шаг 3: Фиксация результатов тестирования в виде отчета о тестировании.

4. Критерий успешного прохождения приемки: Успешное выполнение всех тестовых сценариев, составленных на основе требований данного ТЗ, и отсутствие критических дефектов.

Функциональным и нефункциональным требованиям, изложенным в разделах

1.4 и 1.5 настоящего ТЗ.

Процедура приемки:

1. Предоставление ПП и инструкции по запуску
2. Проведение приемочного тестирования
3. Фиксация результатов.

## **2. Ошибки в продукте и документации другой команды.**

### **Ошибка 1:**

Требования к совместимости: Полное несоответствие.

- Ошибка: Утверждение о том, что приложение работает в браузере ("исполняемой средой является браузер"), полностью противоречит всему остальному ТЗ. В 2.1.1 указано, что это десктопное приложение на Go, использующее пакеты для работы с консолью (bufio, fmt, os), а в 2.1.8 указан CLI (Command Line Interface), то есть интерфейс командной строки, а не веб-браузер.
- Исправление: Раздел 2.1.7 необходимо переписать. Правильное требование: «Приложение должно быть кроссплатформенным консольным (CLI) приложением, совместимым с операционными системами Windows, Linux, macOS».

### **Ошибка 2: Противоречие в требованиях к запуску.**

- Ошибка: В 2.1.6 указано, что для работы нужен интерпретатор Go, а в 2.1.7 — что исполняемой средой является браузер. Это взаимоисключающие параграфы.
- Исправление: Привести к единому требованию. Для CLI-приложения на Go корректно требование из п. 2.1.6 (наличие исполняемого файла или интерпретатора Go), а п. 2.1.7 должен быть исправлен, как указано выше.

**Описание бага №1:** При операции сложения выполняется операция вычитания.

```
Консольный калькулятор  
  
Результат:  
  
Для выхода введите 'exit'  
Для ввода нажмите 'Enter'  
Пример ввода: '2 + 2'  
> 10 + 25  
Консольный калькулятор  
  
Результат: -15  
  
Для выхода введите 'exit'  
Для ввода нажмите 'Enter'  
Пример ввода: '2 + 2'  
> |
```

Рисунок 1 – Демонстрация бага №1.

**Описание бага №2:** При операции вычитания выполняется операция сложения.

```
Для выхода введите 'exit'  
Для ввода нажмите 'Enter'  
Пример ввода: '2 + 2'  
> 11 - 33  
Консольный калькулятор  
  
Результат: 44
```

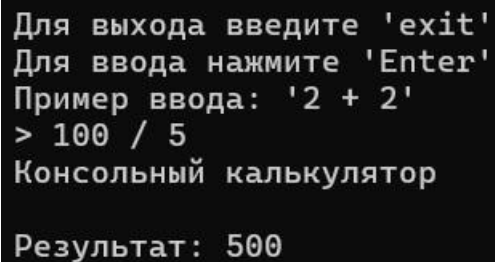
Рисунок 2 – Демонстрация бага №2.

**Описание бага №3:** При операции умножения выполняется операция деления.

```
Для выхода введите 'exit'  
Для ввода нажмите 'Enter'  
Пример ввода: '2 + 2'  
> 30 * 3  
Консольный калькулятор  
  
Результат: 10
```

Рисунок 3 – Демонстрация бага №3.

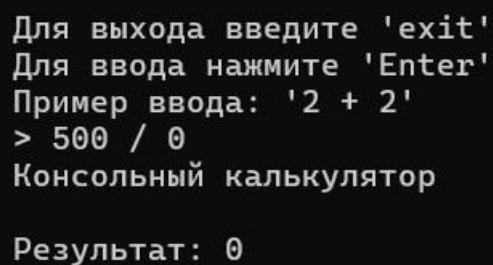
**Описание бага №4:** При операции деления выполняется операция умножения.



```
Для выхода введите 'exit'  
Для ввода нажмите 'Enter'  
Пример ввода: '2 + 2'  
> 100 / 5  
Консольный калькулятор  
Результат: 500
```

Рисунок 4 – Демонстрация бага 4.

**Описание бага №5:** Отсутствие проверки деления на 0.



```
Для выхода введите 'exit'  
Для ввода нажмите 'Enter'  
Пример ввода: '2 + 2'  
> 500 / 0  
Консольный калькулятор  
Результат: 0
```

Рисунок 5 – Демонстрация бага 5.

## **Заключение**

Изучение технического задания и сопроводительной документации позволило выявить ряд замечаний и потенциальных проблем, связанных с неполным описанием функционала, недостаточной проработкой механизмов защиты данных и возможными сценариями возникновения ошибок. Для минимизации рисков на этапе разработки и эксплуатации необходимо доработать ТЗ, детализировав функциональные требования, тестовые сценарии и меры по обеспечению безопасности приложения.