**Аннотация теста:**

**Название проекта**: RepairServiceApp

**Рабочая версия**: 1.0

**Имя тестирующего**: Иванов И.И.

**Дата(ы) теста**: 20.09.2024  
  
  
  
**Тестовый пример #**: TC\_FUNC\_1

**Приоритет тестирования**: Высокий

**Заголовок/название теста**: Проверка корректного расчёта стоимости ремонта

**Краткое изложение теста**: Тест проверяет правильность расчёта стоимости ремонта при заданных значениях стоимости деталей, работы и дополнительных сборов.

**Этапы теста**:

1. Импортировать класс RepairCalculator из repair\_calculator.py.
2. Создать экземпляр класса RepairCalculator.
3. Вызвать метод calculate\_repair\_cost с параметрами (500, 300, 50).
4. Проверить, что результат равен 850.

**Тестовые данные**: Стоимость деталей — 500, работа — 300, дополнительные сборы — 50.

**Ожидаемый результат**: Стоимость ремонта должна быть 850.

**Фактический результат**: (проверяется во время выполнения теста).

**Статус**: (определяется во время выполнения теста).

**Предварительное условие**: Должна быть доступна библиотека repair\_calculator.py.

**Постусловие**: Система должна корректно вернуть рассчитанную стоимость.

**Тестовый пример #2:**

**Тестовый пример #**: TC\_FUNC\_2

**Приоритет тестирования**: Средний

**Заголовок/название теста**: Проверка корректного расчёта времени выполнения заказа

**Краткое изложение теста**: Тест проверяет правильность вычисления времени выполнения заказа.

**Этапы теста**:

1. Импортировать класс RepairCalculator из repair\_calculator.py.
2. Создать экземпляр класса RepairCalculator.
3. Вызвать метод calculate\_repair\_time с параметрами (2, 30).
4. Проверить, что результат равен (2, 30).

**Тестовые данные**: Время выполнения — 2 часа и 30 минут.

**Ожидаемый результат**: Время должно быть (2, 30).

**Фактический результат**: (проверяется во время выполнения теста).

**Статус**: (определяется во время выполнения теста).

**Предварительное условие**: Должна быть доступна библиотека repair\_calculator.py.

**Постусловие**: Система должна вернуть корректное время.

**Тестовый пример #3:**

**Тестовый пример #**: TC\_FUNC\_3

**Приоритет тестирования**: Средний

**Заголовок/название теста**: Проверка корректного расчёта скидки

**Краткое изложение теста**: Тест проверяет правильность вычисления итоговой стоимости с учётом скидки.

**Этапы теста**:

1. Импортировать класс RepairCalculator из repair\_calculator.py.
2. Создать экземпляр класса RepairCalculator.
3. Вызвать метод calculate\_discounted\_cost с параметрами (1000, 20).
4. Проверить, что результат равен 800.

**Тестовые данные**: Общая стоимость — 1000, скидка — 20%.

**Ожидаемый результат**: Итоговая стоимость должна быть 800.

**Фактический результат**: (проверяется во время выполнения теста).

**Статус**: (определяется во время выполнения теста).

**Предварительное условие**: Должна быть доступна библиотека repair\_calculator.py.

**Постусловие**: Система должна вернуть итоговую стоимость.

**Тестовый пример #4:**

**Тестовый пример #**: TC\_FUNC\_4

**Приоритет тестирования**: Низкий

**Заголовок/название теста**: Проверка корректного расчёта стоимости ремонта с нулевыми значениями

**Краткое изложение теста**: Тест проверяет обработку нулевых значений для расчёта стоимости.

**Этапы теста**:

1. Импортировать класс RepairCalculator из repair\_calculator.py.
2. Создать экземпляр класса RepairCalculator.
3. Вызвать метод calculate\_repair\_cost с параметрами (0, 0, 0).
4. Проверить, что результат равен 0.

**Тестовые данные**: Все значения равны 0.

**Ожидаемый результат**: Стоимость ремонта должна быть 0.

**Фактический результат**: (проверяется во время выполнения теста).

**Статус**: (определяется во время выполнения теста).

**Предварительное условие**: Должна быть доступна библиотека repair\_calculator.py.

**Постусловие**: Система должна вернуть 0.

**Тестовый пример #5:**

**Тестовый пример #**: TC\_FUNC\_5

**Приоритет тестирования**: Средний

**Заголовок/название теста**: Проверка корректного расчёта времени с неверными значениями

**Краткое изложение теста**: Тест проверяет, как система обрабатывает неверные значения для расчёта времени.

**Этапы теста**:

1. Импортировать класс RepairCalculator из repair\_calculator.py.
2. Создать экземпляр класса RepairCalculator.
3. Вызвать метод calculate\_repair\_time с параметрами (-1, 30).
4. Проверить, что возникает исключение или ошибка.

**Тестовые данные**: Неверные значения времени (-1, 30).

**Ожидаемый результат**: Должно возникнуть исключение.

**Фактический результат**: (проверяется во время выполнения теста).

**Статус**: (определяется во время выполнения теста).

**Предварительное условие**: Должна быть доступна библиотека repair\_calculator.py.

**Постусловие**: Система должна обработать ошибку.

