Петрозаводский государственный университет Институт математики и информационных технологий Кафедра информатики и математического обеспечения

Отчет по учебному курсу “Методы тестирования программного обеспечения” Тестирование проекта “Организация сбора данных с датчиков в программно-аппаратном комплексе диагностики и мониторинга оборудования”

Выполнила:
студентка 4 курса
группы 22407
Д.В. Горбунова

Петрозаводск - 2020

Объект тестирования

Введение

Разрабатываемый модуль предназначен для сбора данных с сервисов датчиков температуры, акселерометров и токовых клещей с учетом данных сервиса тахометров. Входные данные – данные с с сервисов датчиков температуры, тахометров, акселерометров и токовых клещей. Выходные данные – файлы с данными в организованной структуре, которые будут использоваться для пост-анализа.

Аппаратные аспекты решения

Датчик тахометра используется для определения количества оборотов, которое показывает начало и окончание работы тестируемого оборудования. Датчики температуры используются для получения “сырых” данных в моменты начала и окончания работы. Датчики акселерометров и токовых клещей используются для получения “сырых” данных во время мониторинга оборудования.
Для разрабатываемого модуля требуется установка от одного тахометра, акслелерометра, токовых клещей и термометра на оборудование, к которому подключается устройство сбора данных, передающее показания датчиков модулю.

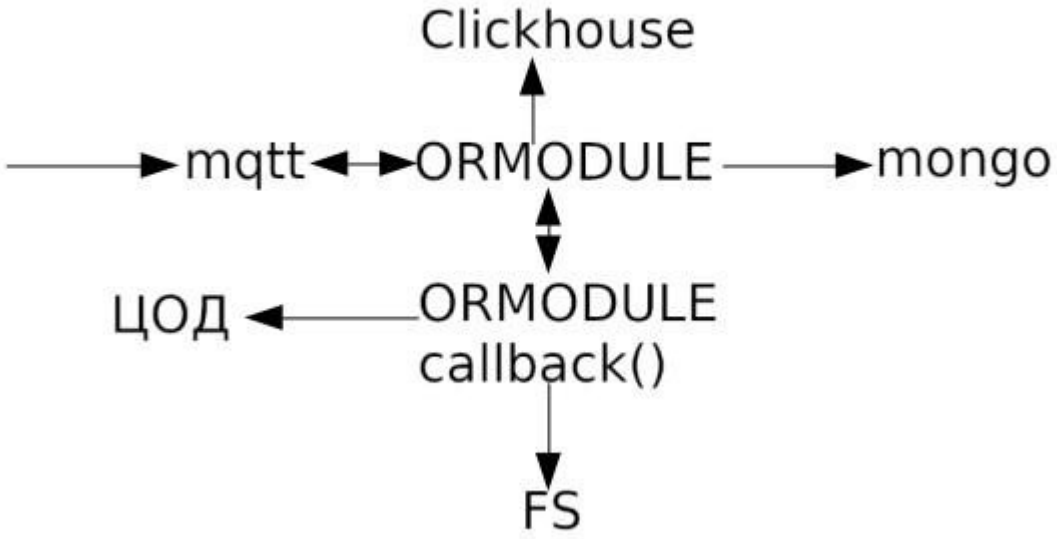
Требования и ограничения

1. Модуль должен производить сбор показаний тахометра.
2. Модуль должен производить сбор показаний акселерометра.
3. Модуль должен производить сбор показаний датчика токовых клещей.
4. Модуль должен производить сбор показаний датчика температуры.
5. Модуль должен записывать файлы с “сырыми” данными акселерометров.
6. Модуль должен записывать файлы с “сырыми” данными токовых клещей.
7. Модуль должен записывать файлы с “сырыми” данными датчиков температуры.
8. Модуль должен работать со всеми комбинациями датчиков температуры, акселерометров, токовых клещей и тахометров.

9. Модуль должен иметь возможность настроек времени записи файлов для каждого датчика(для температуры отдельно для начала работы, конца работы оборудования), а так же для соответствующего ему тахометра.
10. Модуль должен иметь возможность настроек промежутка времени, раз в который записываются файлы при мониторинге
11. Входные данные обновляются автоматически при изменении
12. Модуль должен отправлять записанные данные в ЦОД

Стратегия тестирования

Архитектура

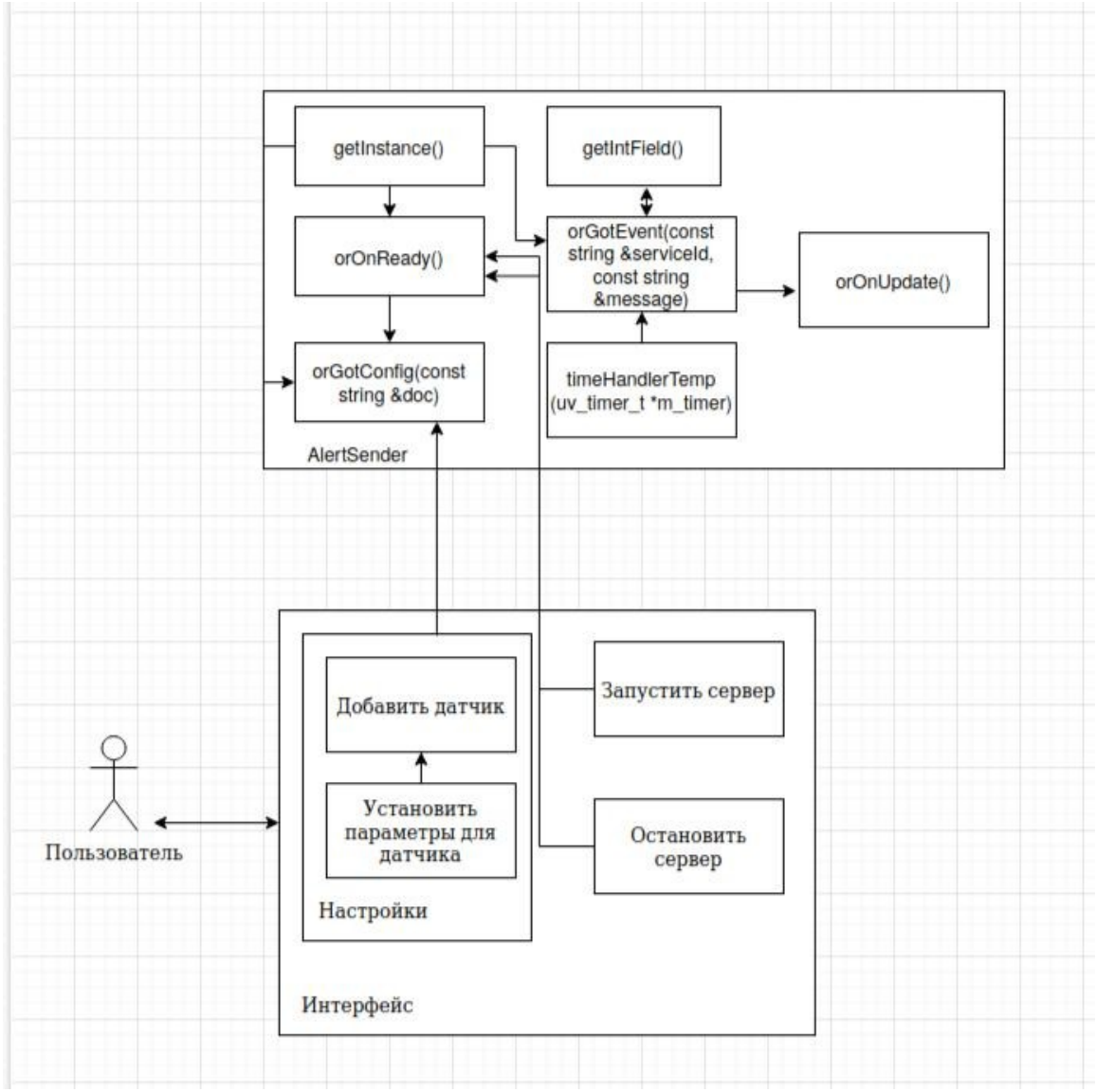


Входные данные передаются в mosquitto. С помощью библиотеки ORMODULE входные данные используются для работы разрабатываемого сервиса, а так же события при работе сохраняются в mongo и clickhouse для дальнейшей разработки интерфейса. Выходные данные отправляются в ЦОД и сохраняются в файловой системе.

После получения информации какие датчики используются, мы создаем папку для каждого датчика. Для данных с сервисов акселерометров и токовых клещей записываем файлы каждый заданный промежуток

времени в течение заданных в настройках времен, называем файлы по времени начала записи. После записи создаем в их папках файлы, куда записывается информация о соответсвующем датчику тахометре и его показаниях, файлы называются по времени записи. Для данных с сервисов температуры мы отслеживаем показания тахометра и при изменении от 0 до >0 и от >0 до 0 записываем файлы при запуске и остановке соответсвенно. Если станок включился или выключился до конца записи файла, файл не сохраняется.

Архитектура ORMODULE callback()



Пользователь взаимодействует с веб интерфейсом:

1) Модуль настроек содержит сводную таблицу, которая позволяет добавлять новые датчики в работу программы,а так же устанавливать параметры для каждого датчика(время записи, выбор тахометра) и для всех датчиков (промежуток между записью файлов). Настройки сервиса отправляются в конфиг и читаются функцией orGotConfig.

2) Интрефейс так же содержит кнопку запуска/ остановки сервера. Запускается функция orOnReady.

Внутри программы AlertSender, которая отвечает за чтение конфигурационных файлов, событий и запись файлов:

1) getInstance() - получение экземпляра класса AlertSender, который необходим для раоты всего модуля

2) getIntField(const json& field) – возращает целое число из json, string, char

3) orOnReady() - запуск сервиса, начало чтение конфигурационного файла

4) orOnUpdate() - проверка приходят ли события с датчиков

5) orGotConfig(const string &doc) – чтение конфигурационного файла

6) timeHandlerTemp(uv_timer_t *m_timer) – отправляет событие о записи файла датчику

Стратегия модульного тестирования

Для модульных тестов используется Google testing framework (gtest) и Cmake. Большинство тестов, проверяющие работу отдельных функций, будут интеграционными, поскольку большинство из них зависит от функции получения экземпляра класса AlertSender, однако мы можем протестировать одну функцию исключительно модульным тестированием:

1) getIntField(const json& field) возращает целое число из json, string, char

Для данной функции продевется проверка правильности работы с корректными данными и результат работы с некорректными данными(json, string, char, содержащие символы помимо чисел).

Стратегия интеграционного тестирования

Для интеграционного тестирования используется Google testing framework (gtest) и Cmake. В результате данной группы тестирования, нужно проверить работу программных функций в группе:

1) getInstance() и orOnReady(), getInstance() получает экземпляра класса AlertSender, который необходим для работы orOnReady(). OrOnReady() void-функция, отвечающая за запуск сервиса,мы будем проверять ее успешное выполнение без ошибок.

2) getInstance() и orGotConfig()

getInstance() получает экземпляра класса AlertSender, который необходим для работы orGotConfig(). OrGotConfig() void-функция, отвечающая за чтение конфигурационного файла,мы будем проверять ее успешное выполнение без ошибок с правильным конфигурационным файлом и ее реакцию на неполный конфигурационный файл.

3)getInstance() и orOnUpdate()

getInstance() получает экземпляра класса AlertSender, который необходим для работы orOnUpdate(). orOnUpdate() void-функция, отвечающая за проверку приходят ли события с датчиков раз в определенный промежуток времени

4) getInstance() и timeHandlerTemp(uv_timer_t *m_timer)

getInstance() получает экземпляра класса AlertSender, который необходим для работы timeHandlerTemp(uv_timer_t *m_timer). timeHandlerTemp(uv_timer_t *m_timer) void-функция, отвечающая за отправку событий на запись файла сервисам, отвечающим за датчики раз в определенный промежуток времени, определение таймера находятся в getInstance()

Стратегия аттестационного тестирования

В ходе проведение аттестационного тестирования, будет проверяться соответствие сервиса по функциональным требованиям. Для тестирования были выбраны следующие функции

1) Настройки
Здесь будут тестироваться возможность добавления нового датчика, установка параметров настроек программы

2) Выключение

Возможность прерывать в любой момент работу программы без сохранения незаписанных файлов

3) Включение

Отключить сервисы датчиков и проверить реакцию сервиса

Стратегия специального тестирования

Будет проводиться нагрузочное тестирование, то есть подвид тестирования производительности, сбор показателей и определение производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе.
Будет проверена функция добавления новых датчиков, а именно – корректная работа программы при добавлении различного количества датчиков
Осуществляется 2 теста в каждом из которых постепенно увеличивается количестово добавляемых датчиков с 3 до максимального доступного сейчас – 4.

Условия начала, окончания и перехода между этапами тестирования

1. Этап тестирования считается завершенным, если пройдены все заявленные тесты в данном этапе и получены результаты в формате “Пройден”, “Не пройден”.
2. Следующий этап тестирования можно начать после завершения предыдущего
3. Тест пройден успешно, если полученный результат совпадает с ожидаемым результатом.

Условия возобновления и приостановки выполнения тестов

1. Тестирование может быть приостановлено только в случае ошибки, мешающей дальнейшему выполнению тестирования.
2. Возобновление процедуры тестирования происходит после устранения ошибки, мешающей дальнейшему выполнению тестирования.

План тестирования

Модульное тестирование

1. М-1 Цель теста – проверка правильности перевода из json в int Тип теста – позитивный

Объект тестирования – функция getIntField(const json&

field) Входные данные – 0.0 Ожидаемый результат – 0

2. М-2 Цель теста – проверка правильности перевода из str в int Тип теста – позитивный

Объект тестирования – функция getIntField(const json&

field) Входные данные – “13.2” Ожидаемый результат –

13

3. М-3 Цель теста – проверка правильности перевода из int в int Тип теста – позитивный

Объект тестирования – функция getIntField(const json&

field) Входные данные – 0 Ожидаемый результат – 0

4. М-4 Цель теста – проверка правильности перевода из str из символов в int Тип теста – негативный

Объект тестирования – функция getIntField(const json&

field) Входные данные – “why”

Ожидаемый результат - -1

5. М-5 Цель теста – проверка правильности перевода из str из символов и чисел в int

Тип теста – негативный

Объект тестирования – функция getIntField(const json& field)

Входные данные – "lf1"

Ожидаемый результат - -1

5. М-6 Цель теста – проверка правильности перевода из str из символов и чисел, где число идет первым, в int

Тип теста – негативный

Объект тестирования – функция getIntField(const json& field)

Входные данные – ""3.y"

Ожидаемый результат - -1

Интеграционное тестирование

1.И-1Цель теста – проверка вызова модуля orOnReady() с помощью полученного экземпляра класса AlertSender из getInstance()

Тип теста – общий.

Объект тестирования – модули

orOnReady(),getInstance() Входные данные –

получаем instance из getInstance(), функция

orOnReady() вызывается из полученного в

getInstance() экземпляра класса(instance)

Ожидаемый результат – успешный запуск

2.И-2Цель теста – проверка вызова модуля orOnUpdate() с помощью полученного экземпляра класса AlertSender из getInstance()

Тип теста – общий.

Объект тестирования – модули

orOnUpdate(),getInstance() Входные данные –

получаем instance из getInstance(), функция

orOnUpdate() вызывается из полученного в

getInstance() экземпляра класса(instance)

Ожидаемый результат – успешный запуск

3.И-3 Цель теста – проверка вызова модуля orGotConfig() с помощью полученного экземпляра класса AlertSender из getInstance() при правильной конфигурации

Тип теста – Позитивный

Объект тестирования – модули orGotConfig() ,getInstance()

Входные данные – получаем instance из getInstance(), const char config[] =
R"({"_id":{"$oid":"5f5f24f9012b1a44800d27c2"},"adcAccelerometerModuleId":
[{"id":"5ee1f8cb012b1a35af7bbc62"}],"adcEnergyModuleId":
[{"id":"5e8c5e74012b1a22797165a2"}],"adcTemp":
[{"id":"5e8c5e74012b1a22797165a2"}],"cnt":"1","counters":
[{"acc":"5e8c5eab012b1a7cd61c1c93","counter":"5f870fb51fd2e8324c6a3b72","pauseti
me":"3"}],"data_type":null,"description":"Тестовый","energym":
[{"counter":"5f870fb51fd2e8324c6a3b72","ener":"5e8c5dd4012b1a30c06c7fd2","pause
time":"3"}],"interval":"1","name":"Отправка данных в
ЦОД","rms_lvl2":"500","rms_lvl3":"1000","rpm":"1","service_type":
{"$oid":"5f5f23c713fea75d38d0f27d"},"settings":null,"start_on_restart":"","startupnam
e":"cloud-
collector5f5f24f9012b1a44800d27c2","status":"Configuring","systemctl_status":"failed
","tags":"","tempr":
[{"counter":"5f870fb51fd2e8324c6a3b72","offtime":"3","ontime":"3","pausetime":"3","
taho":"5e8c5dee012b1a05da5bcf93"}],"time_accel":"10","time_energy":"3","time_temp
":"3","time_temp1":"600","updated_at":{"$date":1608080321000}})"

файл конфигурации подается на вход в функцию orGotConfig(),

функция orGotConfig() вызывается из полученного в getInstance()

экземпляра класса (instance)

Ожидаемый результат – успешный запуск

4.И-4 Цель теста – проверка вызова модуля timeHandlerTemp(uv_timer_t *m_timer)
с помощью полученного экземпляра класса AlertSender из getInstance()

Тип теста – общий.

Объект тестирования – модули timeHandlerTemp(uv_timer_t

*m_timer),getInstance()

Входные данные – получаем instance из getInstance(), функция

timeHandlerTemp(uv_timer_t *m_timer) вызывается из полученного в

getInstance() экземпляра класса(instance)

Ожидаемый результат – успешный запуск

5.И-5 Цель теста – проверка вызова модуля orGotConfig() с помощью полученного экземпляра класса AlertSender из getInstance() с неполной конфигурацией(без тахометра)

Тип теста – негативный

Объект тестирования – модули orGotConfig() ,getInstance()

Входные данные – получаем instance из getInstance(), {"_id":
{"$oid":"5f5f24f9012b1a44800d27c2"},"adcAccelerometerModuleId":
[{"id":"5ee1f8cb012b1a35af7bbc62"}],"adcEnergyModuleId":
[{"id":"5e8c5e74012b1a22797165a2"}],"adcTemp":
[{"id":"5e8c5e74012b1a22797165a2"}],"cnt":"1","counters":
[{"acc":"5e8c5eab012b1a7cd61c1c93","counter":"5f870fb51fd2e8324c6a3b72","pauseti
me":"3"}],"data_type":null,"description":"Тестовый","energym":"interval":"1","name":
"Отправка данных в
ЦОД","rms_lvl2":"500","rms_lvl3":"1000","rpm":"1","service_type":

{"$oid":"5f5f23c713fea75d38d0f27d"},"settings":null,"start_on_restart":"","startupnam
e":"cloud-
collector5f5f24f9012b1a44800d27c2","status":"Configuring","systemctl_status":"failed
","tags":"","tempr":
[{"counter":"5f870fb51fd2e8324c6a3b72","offtime":"3","ontime":"3","pausetime":"3","
taho":"5e8c5dee012b1a05da5bcf93"}],"time_accel":"10","time_energy":"3","time_temp
":"3","time_temp1":"600","updated_at":{"$date":1608080321000}}

файл конфигурации подается на вход в функцию orGotConfig(),

функция orGotConfig() вызывается из полученного в getInstance()

икземпляра класса(instance)

Ожидаемый результат – service_status_event (2020-12-

16T21:33:07+03:00):

{"_id":"fda52e35801407d1d14aa54","code":1,"ldescription":"Ошибка

разбора основных настроек модуля: Can not find a tahometer

field","timestamp"l1608143587,""type"l:"service_status_event"}

Аттестационное тестирование

1.А-1 Цель теста – проверка возможности отключить сервис во время записи файлов

Тип теста – общий.

Объект тестирования – интерфейс

Входные данные - нажимается кнопка “Остановить”.

Ожидаемый результат – файл удаляется

2. А-2 Цель теста – проверка возможности включить сервис с выключенными датчиками

Тип теста – общий.

Объект тестирования – интерфейс

Входные данные - нажимается кнопка “Запустить”.

Ожидаемый результат – на экране появляется Событие: service_status_event (2020-12-16T21:33:07+03:00): {""_id":"5fda52e35801407d1d14aa54","code":1,"description":"Ошибка

разбора основных настроек модуля: …

3. А-3 Цель теста – проверка возможности добавить датчик Тип теста –

общий.

Объект тестирования – интерфейс
Входные данные - нажимается кнока “Настройки”, “+” в таблице, заполняются поля, нажать “Сохранить”.
Ожидаемый результат – в таблицу добавляется датчик

4. А-4 Цель теста – проверка возможности менять настройки Тип теста –

общий.

Объект тестирования – интерфейс

Входные данные - нажимается кнопка “Настройки”, в таблице заполняются поля, нажать “Сохранить”.

Ожидаемый результат – в таблице меняются

данные

Специальные тесты

1.С-1 Цель теста – проверка

производительности приложения при загрузке

большого количества данных.

Тип теста – общий.

Объект тестирования – функция добавления датчика

Входные данные – в настройках добавляется 3 новых датчика (акселерометр X, температура, токовые клещи) с заполненными данными

Ожидаемый результат - сервис продолжает нормально работать работать.

2.С-2 Цель теста – проверка производительности приложения при загрузке большого количества данных.

Тип теста – общий.

Объект тестирования – функция добавления датчика

Входные данные – в настройках добавляется 1 новый датчик – акселерометрZ (всего датчиков 4) с заполненными данными

Ожидаемый результат - сервис продолжает нормально работать работать.

Пример реализации теста

В качестве примера реализации тестов представлены тесты, проверяющие правильность работы функции getIntField(const json& field)

```
#include <gtest/gtest.h>
#include "../app/AlertSender.h"

TEST(AlertSender, liteTest) {
    ASSERT_EQ(0, AlertSender::getIntField(0.0));
    ASSERT_EQ(0, AlertSender::getIntField(0));
    ASSERT_EQ(13, AlertSender::getIntField("13.2"));
}
TEST(AlertSender, liteTest2) {
     ASSERT_EQ(-1, AlertSender::getIntField("why"));
     ASSERT_EQ(-1, AlertSender::getIntField("lf1"));
     ASSERT_EQ(-1, AlertSender::getIntField("3.y"));
}
```

В первом тесте

– проверка json 0.0, должен вернуться 0

– проверка int 0, должен вернуться 0

- проверка строки “13.2”, должно вернуться 13

Во втором тесте

- проверка строки “why”, должно вернуться -1

- проверка строки “lf1”, должно вернуться -1

- проверка строки “3.y”, должно вернуться -1

Выполнение:

```
[----------] 2 tests from AlertSender
[ RUN      ] AlertSender.liteTest
[       OK ] AlertSender.liteTest (1 ms)
[ RUN      ] AlertSender.liteTest2
[       OK ] AlertSender.liteTest2 (1 ms)
[----------] 2 tests from AlertSender (2 ms total)
```

Оценка покрытия тестирования

Процент покрытия тестирования был выбран как рассчитываемое по следующей формуле:

Lcov = (Ltc/Lcode) * 100%,

Lcov - тестовое покрытие

Ltc - количество строк кода, покрытых

тестами Lcode - общее количество строк

кода

Получаем

(416 / 609) * 100 % = 68 %

Журнал тестирования

Тест	Результат тестирования	Номер ошибки
М-1	Пройден	
М-2	Пройден	
М-3	Пройден	
М-4	Пройден	
М-5	Пройден	№1
М-6	Пройден	
И-1	Пройден	
И-2	Пройден	

И-3	Пройден	
И-4	Пройден	
И-5	Пройден	
А-1	Пройден	№2
А-2	Пройден	
А-3	Пройден	
А-4 С-1 С-2	Пройден Пройден Пройден	

Журнал найденных ошибок

1.Ошибка No1.

Тест: М-5

Входные данные: “lf1”

Ожидаемый результат - -1

Фактический результат: 1

Состояние: исправлено.

2.Ошибка No2.

Тест: А-2.

Входные данные: нажимается кнопка “Остановить”.

Ожидаемый результат: файл удаляется

Фактический результат: незаконченный файл остался

Состояние: исправлено.

Результаты

Во время выполнения тестирования было найдено две ошибоки, которые были исправлены. Полноценное специальное тестирование на данный момент невозможно провести из-за отстуствия некоторых датчиков. Приложение было протестировано не полностью, многие взаимодействия и связи нужно будет тестировать в будущем. Тестирование можно считать пройденным успешно.