Министерство образования и науки Российской Федерации Федеральное
государственное бюджетное образовательное учреждение высшего образования
Петрозаводский государственный университет
Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

Отчет по дисциплине «Верификация ПО»

Модуль диспетчеризации для мобильного сервиса поддержки оказания скорой и первой доврачебной медицинской помощи.

Выполнил:
студент группы 22608, Лебедев Н.О.

Преподаватель:
к.ф-м.н., доцент К. А. Кулаков

Петрозаводск, 2016

Содержание

Содержание **2**

Объект тестирования **4**

Функциональные требования к объекту тестирования **6**

Описание модулей **6**

Модуль интерфейса пользователя (CLI) 6

Модуль инициализации (Initializer) 7

Модуль получения сигнала тревоги (Alarm Consumer) 7

Класс Alarm (AlarmClass) 7

Модуль онтологии (Ontology) 8

Модуль обработки сигнала тревоги (Alarm Processor) 8

Модуль алгоритма поиска свободного волонтера (Searching Algorithm) 9

Модуль работы с интеллектуальным пространством (SmartSpace Helpers) 9

План тестирования. **10**

Стратегия тестирования 10

Общая информация 10

Стратегия блочного тестирования 10

Стратегия интеграционного тестирования 10

Схема интеграции 10

Этапы интеграции 11

Стратегия аттестационного тестирования 11

Стратегия нагрузочного тестирования 12

Критерий прохождения тестов 13

Критерий приостановки тестов 13

Критерий возобновления тестирования 13

Оборудование для проведения тестирования **13**

Описание тестов **13**

Типы тестов: 13

Блочные тесты 14

Модуль интерфейса пользователя 14

Функция void print_message(char* message) 14

Функция char* get_user_answer() 14

Модуль инициализации (Initializer) 14

Функция void ctrl_c_handler(int dummy) 14

Модуль получения сигнала тревоги (Alarm Consumer) 14

Функция void sbcr_thread() 14

Функция int subscribe_to_alarm() 15

Функция void new_alarm_handler(sslog_subscription_t *subscription) 15
Класс Alarm (AlarmClass) 15
Метод Alarm::Alarm(void) 15
Метод Alarm::Alarm(URI uri) 15
Метод Alarm::Alarm(char* uri) 16
Метод Alarm::~Alarm() 16
Метод URI* Alarm::getUri() 16
Методы std::chrono::time_point<std::chrono::steady_clock> Alarm::getTime() и void Alarm::setTime(std::chrono::time_point<std::chrono::steady_clock> _time 16
Модуль обработки сигнала тревоги (Alarm Processor) 17
Функция void alarm_thread() 17
Функция void single_alarm_worker(Alarm* alarm_object) 17
Функция void ask_help_of_volunteer(sslog_individual_t* volunteer, sslog_individual_t* alarm) 18
Функция void help_response_handler(sslog_subscription_t* subscription) 18
Модуль алгоритма поиска свободного волонтера (Searching Algorithm) 19
Функция double calculate_distance(sslog_individual_t*, sslog_individual_t*) 19
Модуль работы с интеллектуальным пространством (SmartSpace Helpers) 20
Функция int connect_to_smartspace() 20
Функция char* generate_uri(sslog_class_t* _class) 21
Функция sslog_triple_t* retrieve_first_triple(const char* subject, const char* predicate, const char* object) 22
Функция char* retrieve_object_by_property(sslog_property_t* property, const char* subject) 23
Функция char* retrieve_subject_by_property(sslog_property_t* property, const char* object) 23
Функция sslog_individual_t* retrieve_range_individual(sslog_property_t* property, sslog_individual_t* domain) 24
Функция sslog_individual_t* retrieve_domain_individual(sslog_property_t* property, sslog_individual_t* range) 25
Интеграционное тестирование 25
Тесты 25
Аттестационное тестирование 28
Тесты 28
Нагрузочное тестирование 30

Тестовое покрытие 30

Примеры реализации тестов 31

Отчет о выполнении тестов 33
Результаты блочного тестирования 33
Результаты интеграционного тестирования 35
Результаты аттестационного тестирования 35
Результаты нагрузочного тестирования 35

Отчеты об ошибках 37
Отчет №1 37
Отчет №2 37
Отчет №3 37
Отчет №4 38
Отчет №5 38

Результаты **38**

Объект тестирования

В современном мире наблюдается растущий интерес к непрерывному мониторинга состояния здоровья человека. Это может быть необходимо как для людей, имеющих определенного рода нарушения здоровья, так и для тех, кто хочет контролировать свои жизненные показатели с целью обеспечить своевременное обнаружение аномалий.

В случае возникновения критических ситуаций со здоровьем, может случиться так, что человек будет не в состоянии самостоятельно вызвать врача, а рядом не будет людей, готовых моментально ему помочь. Также, в случае, если пациенту удалось вызвать скорую помощь, она может задержаться в пути в силу различных обстоятельств, таких как загруженность городского трафика, подбор неоптимального маршрута или нехватка свободных автомобилей.
Данную проблему можно решить с использованием сервиса поддержки оказания скорой и первой доврачебной медицинской помощи. Данный сервис позволяет отслеживать координаты пациента, отправлять сигнал вручную или автоматически при неотложных ситуаций, подбирать врача или волонтера для оказания первой или скорой помощи. Данный сервис построен с использованием парадигмы интеллектуальных пространств и разработан на основе платформы Smart-M3. Архитектура сервиса показана на рисунке 1. Сервис состоит из нескольких модулей-агентов (процессоров знаний). Агенты волонтера и пациента реализуют функционал, необходимый для отправки и получения сигнала тревоги соответствующими лицами. Агенты Скорой помощи, Врача предоставляют возможности взаимодействия с сервисом сотрудникам скорой медицинской помощи. Агент “Медицинская информационная система” предоставляет доступ к записям в соответствующей системе.
Важной составляющей данного сервиса является распределение пациентов, которым необходима помощь, среди волонтеров или карет скорой помощи, которые находятся на оптимальном расстоянии от пациента. Данную задачу решает модуль диспетчеризации, который является объектом тестирования в данном отчете.

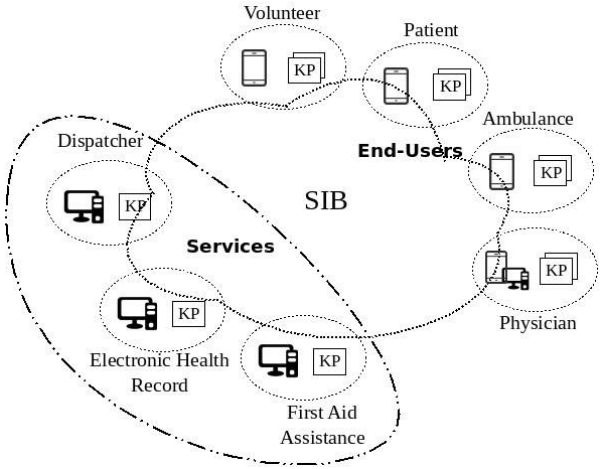


Рисунок 1 “Архитектура сервиса поддержки оказания первой и скорой медицинской помощи”

Архитектура модуля диспетчеризации показана на рисунке 2

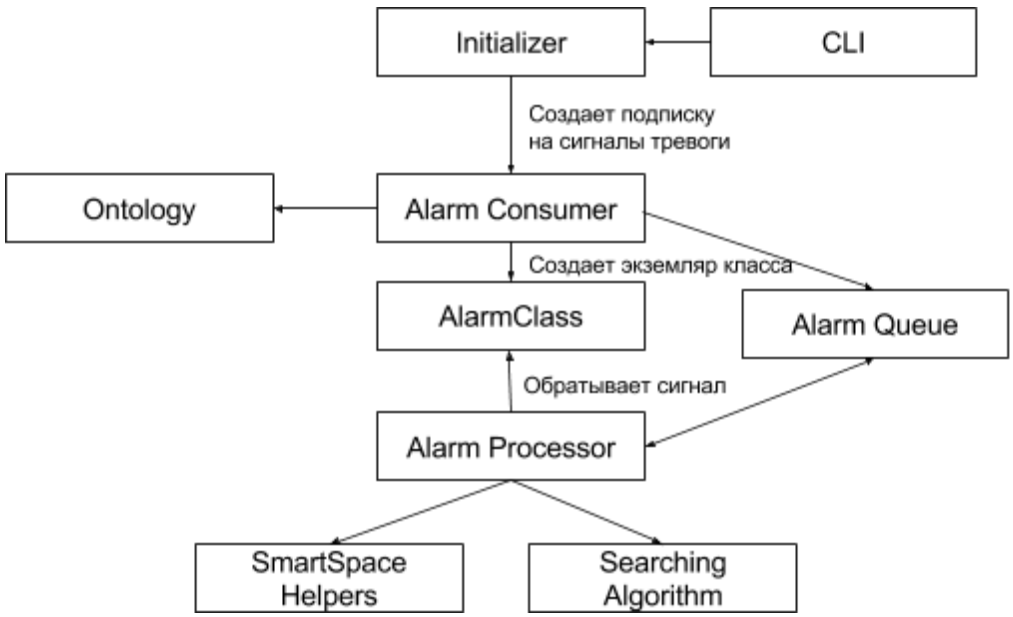


Рисунок 2 “Архитектура модуля диспетчеризации”

Схема взаимодействия модуля с приложениями волонтера и пациента показана на рисунке 2

В следующих разделах будут подробнее рассмотрены составляющие части модуля диспетчеризации, а также план и стратегия тестирования модуля.

Рисунок 3 “Жизненный цикл сигнала тревоги и роль модуля диспетчеризации в этом процессе”

Функциональные требования к объекту тестирования

Тестируемый объект должен реализовывать следующий набор функциональных требований:

1. Получение сигналов тревоги
2. Подбор ближайшего свободного волонтера
3. Поиск следующего волонтера в случае получения отказа
4. Отправка уведомления выбранному волонтеру
5. Обработка нескольких последовательных сигналов тревоги (проверка очереди)
6. Наличие информационных сообщений во время выполнения программы
7. Восстановление работы в случае прерывания интернет-соединения

Описание модулей

Модуль интерфейса пользователя (CLI)

Назначение:

Отображение интерфейса командной строки, для взаимодействия с конечным пользователем. Вывод информационных сообщений, сообщений отладки, сообщений об ошибках приложения.

Файлы:

- dispatcher.cpp
- dispatcher.hpp

Участие в блочном тестировании: все функции

Функции и/или методы:

1. void print_message(char* message)
2. char* get_user_answer()

Модуль инициализации (Initializer)

Назначение:

Обеспечивает инициализацию остальных модулей, производит запуск приложение и назначает обработчик сигнала завершения приложения.

Файлы:

- dispatcher.cpp
- dispatcher.hpp

Участие в блочном тестировании: ctrl_c_handler, функция main не учавствует в тестировании, т.к. является точкой запуска остальных частей приложения, ее работоспособность проверяется на этапах интеграционного тестирования.

Функции и/или методы:

1. int main(int argc, char** argv) - функция необходимая для запуска приложения в целом, стандартная для C/C++ приложений
2. void ctrl_c_handler(int dummy) - завершает приложение по сигналу Ctrl-C

Модуль получения сигнала тревоги (Alarm Consumer)

Назначение:

Организует подписку на появление нового сигнала тревоги в интеллектуальном пространстве. Помещает полученные сигналы в очередь на обработку.

Файлы:

- alram_subscription.cpp
- alram_subscription.hpp

Участие в блочном тестировании: все функции

Функции и/или методы:

1. void sbcr_thread() - инициализирует поток подписки на сигналы тревоги
2. int subscribe_to_alarm() - обертка над несколькими вызовами функций библиотеки SmartSlog для оформления подписки на сигналы тревоги
3. void new_alarm_handler(sslog_subscription_t *subscription) - обрабатывает новый сигнал тревоги пришедший по подписке, помещает его в очередь

Класс Alarm (AlarmClass)

Назначение:

Содержит поля и методы для работы с сигналом тревоги. Предоставляет методы по преобразованию сущности онтологии Alarm в объект данного класса и обратно.

Файлы:

- AlarmClass.cpp
- AlarmClass.hpp

Участие в блочном тестировании: все методы, кроме ~Alarm. Деструктор объявляется неявно, за очистку памяти несет ответственность компилятор C++ и платформа
Функции и/или методы:

1. Alarm::Alarm(void) - пустой конструктор, создает пустой Alarm объект
2. Alarm::Alarm(URI uri) - создает Alarm объект с URI, заданном в виде структуры URI (обертка над string)
3. Alarm::Alarm(char* uri) - создает Alarm объект из URI заданном в char*, этот тип возвращают функции библиотеки SmartSlog
4. Alarm::~Alarm() - деструктор, уничтожает объект, освобождает память
5. URI* Alarm::getUri() - метод для получения URI
6. std::chrono::time_point<std::chrono::steady_clock> Alarm::getTime() - метод получения времени установленного в Alarm (необходима для отладочных нужд)
7. void Alarm::setTime(std::chrono::time_point<std::chrono::steady_clock> _time - метод для установки времени в объекте класса Alarm (необходима для отладочных нужд)

Модуль онтологии (Ontology)

Назначение:

Представляет набор классов и свойств, используемых данным приложением в интеллектуальном пространстве.

Файлы:

- ontology/smartcare.c
- ontology/smartcare.h

Участие в блочном тестировании: не участвует, т.к. автоматически генерируется сторонним приложением, которое не является объектом данного тестирования.
Функции и/или методы: не содержит

Модуль обработки сигнала тревоги (Alarm Processor)

Назначение:

Забирает сигналы тревоги из очереди. Назначает свободного волонтера.

Файлы:

- alram_processing.cpp
- alram_processing.hpp

Участие в блочном тестировании: все функции, кроме single_alarm_worker, т.к. в текущей версии не реализована возможность параллельной обработки нескольких сигналов тревоги и эта функция заменена заглушкой.
Функции и/или методы:

1. void alarm_thread() - инициализиурет поток обработки сигналов тревоги, забирает сигналы из очереди
2. void process_alarm(Alarm* alarm) - обраатывает единственный сигнал тревоги, передает его в single_alarm_worker, создавая отдельный поток (нереализовано)
3. void wait(int milliseconds) - приостановка программы между потоками, обертка над sleep/функциями Boost::chrono

4. void single_alarm_worker(Alarm* alarm_object) - запускает цикл вызова функций поиска свободного волнтера
5. void ask_help_of_volunteer(sslog_individual_t* volunteer, sslog_individual_t* alarm) - ищет свободного волонтера, отправляет запрос помощи, оформляет подписку на ответ волонтера
6. void help_response_handler(sslog_subscription_t* subscription) - обработчик ответа волонтера

Модуль алгоритма поиска свободного волонтера (Searching Algorithm)

Назначение:

Реализует набор алгоритмов поиска свободного волонтера.

Файлы:

- volunteer_algorithm.cpp
- volunteer_algorithm.hpp

Участие в блочном тестировании: все функции

Функции и/или методы:

1. double calculate_distance(sslog_individual_t*, sslog_individual_t*) - вычисляет расстояние на основании координат двух индивидов

Модуль работы с интеллектуальным пространством (SmartSpace Helpers)

Назначение:

Реализует функции взаимодействия с интеллектуальным пространством, не представленные библиотекой SmartSlog

Файлы:

- smartspace_utils.cpp
- smartspace_utils.hpp

Участие в блочном тестировании: все функции

Функции и/или методы:

1. int connect_to_smartspace() - подключение к интеллектуальному пространству
2. char* generate_uri(sslog_class_t* _class) - декоратор над функцией generate_uri библиотеки SmartSlog, дополняет полученный от SmartSlog URI индивида доп. символами для обеспечения уникальности значения
3. sslog_triple_t* retrieve_first_triple(const char* subject, const char* predicate, const char* object) - забирает первую тройку в выдаче из интеллектуального пространства по шаблону
4. char* retrieve_object_by_property(sslog_property_t* property, const char* subject) - получает object-атрибут тройки по атрибутам property и subject
5. char* retrieve_subject_by_property(sslog_property_t* property, const char* object) - получает subjectt-атрибут тройки по атрибутам property и object
6. sslog_individual_t* retrieve_range_individual(sslog_property_t* property, subjectsslog_individual_t* domain) - получает индивида заданного свойством property индивида domain

7. sslog_individual_t* retrieve_domain_individual(sslog_property_t* property, sslog_individual_t* range) - получает родительского индивида, свойства property которого имеет значения range

План тестирования.

Стратегия тестирования

Общая информация

Для тестирования будет использоваться инструмент CUnit, т.к. он является наиболее часто используемым при разработке программ на C, а также Boost Unit Test Framework, т.к. код написан с применением C++ библиотеки Boost.

Стратегия блочного тестирования

Для модульного тестирования функций, взаимодействующих с SmartSpace, предполагается использование mock-объектов, имитирующих работу SIB.

Стратегия интеграционного тестирования

Схема интеграции

Интеграционное тестирование будет проводиться по схеме “сверху-вниз”. Т.е. вначале будут протестированы модули верхнего уровня, а их зависимости будут заменены mock-объектами. Далее постепенно mock-объекты будут заменяться на реальные модули. Данная схема выбрана по причине того, что предполагается расширение объекта тестирования и не все спроектированные интерфейсы имеют конкретную реализацию. Графически схема интеграции показана на рисунке 4. Зеленым цветом отмечены реальные модули, желтым - заглушки.

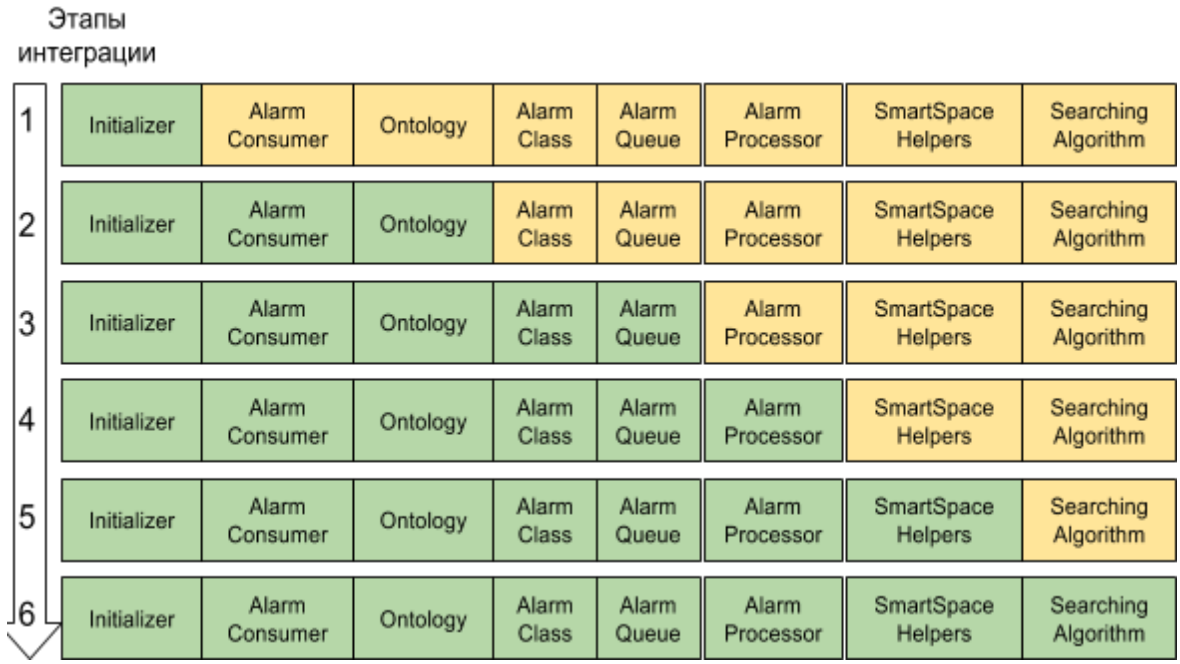


Рисунок 4 “Схема интеграции”

Этапы интеграции

1. Модуль Initializer
2. Модули Initializer, Alarm Consumer, Ontology
3. Модули Initializer, Alarm Consumer, Ontology, AlarmClass, Alarm Queue
4. Модули Initializer, Alarm Consumer, Ontology, AlarmClass, Alarm Queue, Alarm Processor
5. Модули Initializer, Alarm Consumer, Ontology, AlarmClass, Alarm Queue, Alarm Processor, SmartSpace Helpers
6. Модули Initializer, Alarm Consumer, Ontology, AlarmClass, Alarm Queue, Alarm Processor, SmartSpace Helpers, Searching Algorithm

При тестировании предполагается использование фикстур (заранее подготовленных набор данных), содержащих данные местоположения фиктивных волонтеров и пациентов.

Стратегия аттестационного тестирования

Аттестационные тесты покрывают набор функциональных требований 1-7. Для требования 6 не предусмотрено отдельного теста, т.к. его выполнение проверяется при проверке всех других требований (информационные сообщения должны выводиться при любых операциях с тестируемым объектом)

Стратегия нагрузочного тестирования

Сервис поддержки оказания первой и скорой медицинской помощи предполагается к использованию большим количеством людей (не только пациентов лечебных учреждений, но и людей, желающих находиться под контролем врача или волонтера непрерывно). Отправков сигналов тревоги, в свою очередь, может быть совершена в автоматическом режиме на основании показателей измерений с биомедицинских сенсоров, данных опросов и т.п. Также в системе одновременно может находиться несколько волонтеров и пациентов, точное число которых определить невозможно заранее.

В этом случае тестируемый объект предполагает обработку больших потоков данных, например получение сигналов тревоги от пациентов, значит целесообразно провести специальное нагрузочное тестирование с целью выявить точку отказа.

Для нагрузочного тестирования необходимо реализовать программно эмуляторы приложений для волонтера и пациента, а также реализовать скрипты запуска данных эмуляторов в необходимом количестве.

В рамках нагрузочного тестирования проводится серия испытаний с тестируемым объектом с разными значениями параметров: интенсивностью входящего потока сигналов тревоги(λ), количеством свободных активных волонтеров (N_v), вероятностью отказа принимать сигнал неотложного вызова волонтером(P_r).

Входящий поток сигналов тревоги является случайным пуассоновским процессом. Интенсивность входящего потока λ была выбрана равной 0.8, т.е. в среднем 0.8 сигналов тревоги в секунду. Таким образом время между поступающими в агент диспетчера сигналами тревоги моделируется в соответствии с формулой $-\ln(x) / \lambda$, где x — случайная величина, распределенная равномерно на отрезке [0,1].

Количество свободных волонтеров должно быть одинаковым для каждого сигна ла тревоги. Таким образом для проведения испытаний были выбраны варианты, когда число свободных волонтеров изменяется на отрезке [10,50] с шагом 10. Для поддержания одинакового числа волонтеров на протяжении всего испытания, при принятии или отклонении волонтером сигнала тревоги, запускался новый «свободный» агент волонтера. События отказа или согласия волонтера принимать сигнал тревоги распределены по равномерному закону, для проведения испытаний были выбраны вероятности отказа на отрезке [0.1,0.8] с шагом 0.1. Вероятности 0.9 и 1.0 не интересны с точки зрения проведения испытаний, т.к. в этих случаях волонтеры отказываются от приема практически любых сигналов тревоги и время ожидания в очереди (T_q) стремится к бесконечности. Варьируя переменные P_r и N_v проводится 40 испытаний, в каждом из которых были обработаны 50 сигналов тревоги. Общее время нахождения сигнала тревоги в тестируемом объекте является суммой $T, = T_q + T_p$.

Для нагрузочного тестирования будет использоваться язык bash. С помощью него реализуются скрипты, имитирующие реальных пользователей, которые создают нагрузку на объект тестирования.

Критерий прохождения тестов

Тест считается успешно пройденным, если ожидаемый и фактический результаты совпадают. Если тест завершается неудачей, то перед принятием решения целесообразно проверить правильность самого теста. Если тест завершился неудачей и тест реализован правильно, то производится заключение о найденной ошибке.
Тестирование считается пройденным, если во время его прохождения не выявлено критических ошибок, а процент непройденных тестов меньше 1% от общего количества.

Критерий приостановки тестов

Тестирование должно быть приостановлено, если количество непройденных тестов превысит 10% от их общего количества. Тестирование должно быть приостановлено при обнаружении критических ошибок.

Критерий возобновления тестирования

Тестирование возобновляется после исправления ошибок, выявленных при предыдущем тестировании.

Оборудование для проведения тестирования

Для проведения тестирования используется настольный компьютер с установленной операционной системой Ubuntu 15.04.

Описание тестов

Типы тестов:

- П – простой
- О – общий
- С – специальный
- Н – негативный
- К – краевой

Блочные тесты

Модуль интерфейса пользователя

Функция void print_message(char* message)

Результат: отображение сообщения в командной строке.
Тесты:
Тест 1
Тип: П
Входные данные: сообщение для вывода
Алгоритм:

- Подготовить сообщение

Цель: Проверить эквивалентность ожидаемого и фактического вывода сообщения на экран.
Ожидаемый результат: отображение заданного сообщения в командной строке.

Функция char* get_user_answer()

Результат: чтение введенных пользовательских данных
Тесты:
Тест 2
Тип: П
Входные данные: сообщение для ввода
Алгоритм:

- Ввести сообщение в терминал

Цель: Проверить эквивалентность ожидаемого и фактического вывода чтения сообщения
Ожидаемый результат: Переменная содержит введенное пользователем сообщение

Модуль инициализации (Initializer)

Функция void ctrl_c_handler(int dummy)

Тест 3
Тип: П
Входные данные: нет
Алгоритм:

- Ожидание ввода Ctrl-C

Цель: Проверить завершение работы приложения по сигналу Ctrl-C
Ожидаемый результат: При нажатии Ctrl-C приложение завершается.

Модуль получения сигнала тревоги (Alarm Consumer)

Функция void sbcr_thread()

Тест 4

Тип: O
Входные данные: нет
Косвенные входные данные: Очередь сигналов тревоги (Alarm Queue)
Алгоритм:

- Создать индивидов Alarm и поместить в очередь
- Вызвать функцию

Цель: Проверить добавление Alarm в очереди
Ожидаемый результат: Размер очереди увеличился на 1

Функция int subscribe_to_alarm()

Тест 5
Тип: O
Входные данные: нет
Косвенные входные данные: подключение к SmartSpace
Алгоритм:

- Вызвать функцию

Цель: Проверить создание подписки
Ожидаемый результат: Создается подписка на класс Alarm

Функция void new_alarm_handler(sslog_subscription_t *subscription)

Тест 6
Тип: H, K
Входные данные: NULL параметр
Косвенные входные данные: нет
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “NULL в качестве параметра”

Класс Alarm (AlarmClass)

Метод Alarm::Alarm(void)

Тест 7
Тип: O
Входные данные: нет
Косвенные входные данные: нет
Алгоритм:

- Вызвать конструктор

Цель: Проверить создание объекта
Ожидаемый результат: Создан объект с пустым URI

Метод Alarm::Alarm(URI uri)

Тест 8
Тип: O

Входные данные: uri
Косвенные входные данные: нет
Алгоритм:

- Создать объект структуры URI
- Вызвать конструктор с параметром
- Вызвать метод getUri

Цель: Проверить создание объекта
Ожидаемый результат: Объект создан с заданным URI

Метод Alarm::Alarm(char* uri)

Тест 9
Тип: O
Входные данные: uri
Косвенные входные данные: нет
Алгоритм:

- Вызвать конструктор с параметром
- Вызвать метод getUri

Цель: Проверить создание объекта
Ожидаемый результат: Объект создан с заданным URI

Метод Alarm::~Alarm()

Тест 10
Тип: O
Входные данные: нет
Косвенные входные данные: нет
Алгоритм:

- Вызвать конструктор
- Вызвать деструктор

Цель: Проверить удаление объекта
Ожидаемый результат: Объект удален

Метод URI* Alarm::getUri()

Тест 11
Тип: O
Входные данные: uri
Косвенные входные данные: нет
Алгоритм:

- Вызвать конструктор с параметром
- Вызвать метод getUri

Цель: Проверить создание объекта
Ожидаемый результат: метод возвращает URI объекта в виде структуры

Методы std::chrono::time_point<std::chrono::steady_clock> Alarm::getTime() и void Alarm::setTime(std::chrono::time_point<std::chrono::steady_clock> _time

Тест 12

Тип: O
Входные данные: uri
Косвенные входные данные: нет
Алгоритм:

- Вызвать конструктор
- Вызвать метод setTime
- Вызвать метод getTime

Цель: Проверить заполнение поля и получение значения этого поля
Ожидаемый результат: Объект создан с заданным URI

Модуль обработки сигнала тревоги (Alarm Processor)

Функция void alarm_thread()

Тест 13
Тип: O
Входные данные: нет
Косвенные входные данные: Очередь сигналов тревоги (Alarm Queue)
Алгоритм:

- Создать индивидов Alarm и поместить в очередь
- Вызвать функцию

Цель: Проверить удаление Alarm из очереди
Ожидаемый результат: Размер очереди уменьшился на 1

Функция void single_alarm_worker(Alarm* alarm_object)

Тест 14
Тип: H, K
Входные данные: Alarm
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Опубликовать Alarm без связи к Patient
- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “No Patient”
Тест
Тип: H, K
Входные данные: Alarm с несуществующим URI
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “No Alarm”

Тест 15
Тип: H, K
Входные данные: Alarm

Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Опубликовать Alarm для Patient без связей к Location
- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “No Location”

Функция void ask_help_of_volunteer(sslog_individual_t* volunteer, sslog_individual_t* alarm)

Тест 16
Тип: Н, К
Входные данные: volunteer = NULL, alarm != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр volunteer”
Тест 17
Тип: Н, К
Входные данные: volunteer != NULL, alarm = NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр alarm”
Тест 18
Тип: О
Входные данные: volunteer != NULL, alarm != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Создать индивидов
- Вызвать функцию

Цель: Проверить создание связи и подписки
Ожидаемый результат: Создается связь волонтер-аларм в интеллектуальном пространстве и подписка на ответ волонтера

Функция void help_response_handler(sslog_subscription_t* subscription)

Тест 19
Тип: Н, К
Входные данные: NULL параметр
Косвенные входные данные: нет
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “NULL в качестве параметра”

Модуль алгоритма поиска свободного волонтера (Searching Algorithm)

Функция double calculate_distance(sslog_individual_t*, sslog_individual_t*)

Тест 20
Тип: H
Входные данные: первый параметр = individual, второй параметр = NULL
Алгоритм:
- Создать индивид
- Вызвать функцию

Цель: Проверить вывод ошибки
Ожидаемый результат: Ошибка “NULL параметер”

Тест 21
Тип: H
Входные данные: первый параметр = NULL, второй параметр = individual
Алгоритм:
- Создать индивид
- Вызвать функцию

Цель: Проверить вывод ошибки
Ожидаемый результат: Ошибка “NULL параметер”

Тест 22
Тип: H
Входные данные: первый параметр = NULL, второй параметр = NULL
Алгоритм:
- Вызвать функцию с NULL в качестве параметров

Цель: Проверить вывод ошибки
Ожидаемый результат: Ошибка “NULL параметер”

Тест 23
Тип: H
Входные данные: первый или второй индивид, не содержащий свойств местоположения
Алгоритм:
- Создать индивид
- Вызвать функцию

Цель: Проверить вывод ошибки
Ожидаемый результат: Ошибка “Не задано или задано некорректно свойство Location”

Тест 24
Тип: O
Входные данные: Первый и второй параметры - полностью заданные индивиды

Алгоритм:

- Создать индивидов
- Вызвать функцию

Цель: Проверить работу функции при правильных параметрах
Ожидаемый результат: Расстояние между координатами первого и второго индивидов

Тест 25
Тип: Н, К
Входные данные: Отрицательные координаты одного из индивидов
Алгоритм:

- Создать индивидов
- Вызвать функцию

Цель: Проверить работу функции при неверных параметрах
Ожидаемый результат: Ошибка “Не задано или задано некорректно свойство Location”

Тест 26
Тип: О, К
Входные данные: Одинаковые координаты у каждого индивида
Алгоритм:

- Создать индивидов
- Вызвать функцию

Цель: Проверить работу функции при одинаковых параметрах
Ожидаемый результат: 0 - возвращаемое значение

Модуль работы с интеллектуальным пространством (SmartSpace Helpers)

Функция int connect_to_smartspace()

Тест 27
Тип: О
Входные данные: нет
Косвенные входные данные: IP адрес, порт, имя интеллектуального пространства
Алгоритм:

- Вызвать функцию

Цель: Проверить подключение
Ожидаемый результат: Инициализация глобальной переменной - точки входа в интеллектуальное пространство

Тест 28
Тип: Н
Входные данные: нет

Косвенные входные данные: неверный/недоступный IP адрес, порт, имя интеллектуального пространства
Алгоритм:

- Вызвать функцию

Цель: Проверить подключение
Ожидаемый результат: Ошибка “Не удалось подключиься к SS”

Функция char* generate_uri(sslog_class_t* _class)

Тест 29
Тип: Н
Входные данные: NULL - параметр
Косвенные входные данные: Онтология
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр”

Тест 30
Тип: Н, К
Входные данные: Несуществующий онтологический класс
Косвенные входные данные: Онтология
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр”

Тест 31
Тип: О
Входные данные: Существующий онтологический класс
Косвенные входные данные: Онтология
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат URI
Ожидаемый результат: URI заданного класса

Тест 32
Тип: О,С
Входные данные: Существующий онтологический класс
Косвенные входные данные: Онтология
Алгоритм:

- Вызвать функцию несколько раз подряд с небольшими временными интервалами (0,1 сек)

Цель: Проверить уникльность значений
Ожидаемый результат: Отсутствие повторяющихся URI

Функция sslog_triple_t* retrieve_first_triple(const char* subject, const char* predicate, const char* object)

Тест 33
Тип: Н, К
Входные данные: subject = NULL, predicate != NULL, object != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:
- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр subject”

Тест 34
Тип: Н, К
Входные данные: subject != NULL, predicate = NULL, object != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:
- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр predicate”

Тест 35
Тип: Н, К
Входные данные: subject != NULL, predicate != NULL, object = NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:
- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр object”

Тест 36
Тип: О
Входные данные: subject != NULL, predicate != NULL, object != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:
- Поместить в SS несколько заданных одинаковых троек
- Вызвать функцию

Цель: Проверить возврат единственной тройки
Ожидаемый результат: Объект sslog_triple_t с заданными во входных данных субъектом, предикатом, объектом

Тест 37
Тип: О, К
Входные данные: subject != NULL, predicate != NULL, object != NULL
Косвенные входные данные: Подключение к Smart Space

Алгоритм:

- Очистить Smart Space
- Вызвать функцию

Цель: Проверить возврат NULL
Ожидаемый результат: NULL

Функция char* retrieve_object_by_property(sslog_property_t* property, const char* subject)

Тест 38
Тип: Н, К
Входные данные: property = NULL, subject != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр property”

Тест 39
Тип: Н, К
Входные данные: property != NULL, subject = NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр subject”

Тест 40
Тип: О
Входные данные: property != NULL, subject != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Поместить тройку в SmartSpace
- Вызвать функцию

Цель: Проверить возврат object
Ожидаемый результат: object тройки

Функция char* retrieve_subject_by_property(sslog_property_t* property, const char* object)

Тест 41
Тип: Н, К
Входные данные: property = NULL, object != NULL

Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр property”
Тест
Тип: Н, К
Входные данные: property != NULL, object = NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр object”

Тест 42
Тип: О
Входные данные: property != NULL, object != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Поместить тройку в SmartSpace
- Вызвать функцию

Цель: Проверить возврат subject
Ожидаемый результат: subject тройки

Функция sslog_individual_t* retrieve_range_individual(sslog_property_t* property, sslog_individual_t* domain)

Тест 43
Тип: Н, К
Входные данные: property = NULL, domain != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр property”

Тест 44
Тип: Н, К
Входные данные: property != NULL, domain = NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр domain”

Тест 45

Тип: О
Входные данные: property != NULL, domain != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Поместить два индвида связанных свойством property в SmartSpace
- Вызвать функцию

Цель: Проверить возврат range индвида
Ожидаемый результат: Индивид

Функция sslog_individual_t* retrieve_domain_individual(sslog_property_t* property, sslog_individual_t* range)

Тест 46
Тип: Н, К
Входные данные: property = NULL, range != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр property”

Тест 47
Тип: Н, К
Входные данные: property != NULL, range = NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Вызвать функцию

Цель: Проверить возврат ошибки
Ожидаемый результат: Ошибка “Неверный параметр range”

Тест 48
Тип: О
Входные данные: property != NULL, range != NULL
Косвенные входные данные: Подключение к Smart Space
Алгоритм:

- Поместить два индвида связанных свойством property в SmartSpace
- Вызвать функцию

Цель: Проверить возврат domain индвида
Ожидаемый результат: Индивид

Интеграционное тестирование

Тесты

Тест И1
Описание: Проверка сборки и запуска приложения без ошибок

Тип: О
Входные данные: нет
Алгоритм:
- Собрать приложение с заглушками вместо всех модулей кроме Initializer.
- Запустить приложение
- Выйти из приложения

Цель: Проверить вывод информационных сообщений и отсутствие ошибок при запуске.
Ожидаемый результат: Приложение запускается. По сигналу Ctrl-C приложение закрывается.

Тест И2
Описание: Проверка интеграции с модулем получения новых сигналов тревоги на основе заданной Онтологии
Тип: О
Входные данные: индивид класса Alarm
Алгоритм:
- Создать экземпляр Alarm Consumer.
- Запустить функцию sbcr_thread (инициализировать поток)
- Получить alarm

Цель: Проверка создания потока с подпиской на сигнал тревоги
Ожидаемый результат: Создание подписки, получение сигналов тревоги.

Тест И3
Описание: Проверка конвертации сигнала тревоги из Smart Space в экземпляр AlarmClass, проверка очереди.
Тип: О
Входные данные: индивид Alarm
Алгоритм:
- Запустить sbcr_thread()
- Эмулировать получение сигнала тревоги

Цель: проверить помещение экземпляра AlarmClass в очередь
Ожидаемый результат: Очередь сигналов тревоги содержит экземпляр AlarmClass. Метод AlarmClass::getURI() возвращает uri индивида Alarm (из входных данных)

Тест И4
Описание: Проверка интеграции с обработчиком сигнала тревоги
Тип: О
Входные данные: индивид Alarm, индивид Patient, индивид Volunteer
Алгоритм:
- Запустить sbcr_thread()
- Эмулировать получение сигнала тревоги
- Запустить alarm_thread()

Цель: Проверить изъятие сигнала тревоги из очереди для обработки.
Ожидаемый результат: Непустая очередь после добавления сигнала тревоги. Пустая очередь после запуска alarm_thread.

Тест И5
Описание: Проверка взаимодействия с интеллектуальным пространством (Smart Space)

Тип: О
Входные данные: индивид Alarm, индивид Patient, индивид Volunteer, объект ss_node (точка подключения к Smart Space), IP адрес, порт, имя Smart Space
Алгоритм:

- Подключиться к SmartSpace
- Опубликовать индивиды Alarm, Patient, Volunteer
- Запустить sbcr_thread()
- Эмулировать получение сигнала тревоги
- Запустить alarm_thread()

Цель: Проверить получение всех трех индивидов тестируемым объектом в процессе выполнения программы
Ожидаемый результат: Подключение к Smart Space. Получение сигнала тревоги, Публикация тройки связи единственного индивида Volunteer с единственным индивидом Alarm

Тест И6
Описание: Проверка взаимодействия с интеллектуальным пространством (Smart Space). Получение неполных данных.
Тип: О
Входные данные: индивид Alarm, индивид Patient, индивид Volunteer, объект ss_node (точка подключения к Smart Space), IP адрес, порт, имя Smart Space
Алгоритм:

- Подключиться к SmartSpace
- Опубликовать индивиды Alarm, Patient, Volunteer
- Запустить sbcr_thread()
- Эмулировать получение сигнала тревоги без установленной связи с пациентом
- Запустить alarm_thread()

Цель: Проверить получение всех трех индивидов тестируемым объектом в процессе выполнения программы и возврата ошибки при неустановленных связях между ними.
Ожидаемый результат: Подключение к Smart Space. Получение сигнала тревоги, Возврат ошибки при неустановленной связи сигнала тревоги и пациента.

Тест И7
Описание: Проверка взаимодействия с интеллектуальным пространством (Smart Space). Отправка запроса помощи
Тип: О
Входные данные: индивид Alarm, индивид Patient, индивид Volunteer, объект ss_node (точка подключения к Smart Space), IP адрес, порт, имя Smart Space
Алгоритм:

- Подключиться к SmartSpace
- Опубликовать индивиды Alarm, Patient, Volunteer
- Запустить sbcr_thread()
- Эмулировать получение сигнала тревоги
- Запустить alarm_thread()
- Отправить запрос о помощи волонтеру

Цель: Проверить создание в интеллектуальном пространстве свойства о помощи
Ожидаемый результат: Публикация тройки связи единственного индивида Volunteer с единственным индивидом Alarm

Тест И8
Описание: Проверка использования алгоритма поиска. Проверка системы в целом.

Тип: О
Входные данные: индивид Alarm, индивид Patient, 2 индивида Volunteer, объект ss_node (точка подключения к Smart Space), IP адрес, порт, имя Smart Space
Алгоритм:

- Подключиться к SmartSpace
- Опубликовать индивиды Alarm, Patient, Volunteer (x2), Location для пациента и волонтеров
- Запустить sbcr_thread()
- Эмулировать получение сигнала тревоги
- Запустить alarm_thread()

Цель: Проверка использования алгоритма поиска для поиска ближайшего волонтера.
Ожидаемый результат: Публикация тройки связи индивида Volunteer, который наиболее “близок”, основываясь на заданных координатах с единственным индивидом Alarm.

Тест И9
Описание: Проверка ошибки при несуществующих координатах Волонтера или пациента
Тип: О
Входные данные: индивид Alarm, индивид Patient, 2 индивида Volunteer, объект ss_node (точка подключения к Smart Space), IP адрес, порт, имя Smart Space
Алгоритм:

- Подключиться к SmartSpace
- Опубликовать индивиды Alarm, Patient, Volunteer (x2), Location для пациента
- Запустить sbcr_thread()
- Эмулировать получение сигнала тревоги
- Запустить alarm_thread()

Цель: Проверка возврата ошибки при неверных/несуществующих координата пациента и/или волонтера
Ожидаемый результат: Сообщение об ошибке (“Несуществующие координаты волонтера. Невозможно рассчитать дистанцию”). Возвращение объекта Alarm в очередь для повторной обработки в дальнейшем

Аттестационное тестирование

Тесты

Тест А1
Описание: Проверка получения сигналов тревоги тестируемым объектом
Тип: О
Входные данные: сигнал тревоги
Алгоритм:

- Запустить приложение пациента
- Запустить тестируемый объект
- Отправить сигнал тревоги из приложения пациента

Цель: Проверить получения одного/нескольких сигналов тревоги, помещаемых в интеллектуальное пространство

Ожидаемый результат: Отображение сообщения о получении сигнала тревоги и постановке в очередь. Отображение идентификатора полученного сигнала тревоги.

Тест А2

Описание: Проверка поиска ближайшего к пациенту волонтера и отправка уведомления этому волонтеру.

Тип: О

Входные данные: Сигнал тревоги, Данные местоположения пациента, Данные местоположения волонтеров.

Алгоритм:

- Запустить приложение пациента
- Запустить несколько приложений волонтера (с различными координатами местоположения)
- Запустить тестируемый объект
- Отправить сигнал тревоги из приложения пациента

Цель: Проверить, что найденный волонтер является ближайшим к пациенту из всех имеющихся волонтеров.

Ожидаемый результат: Отображение сообщения об успешном поиске ближайшего волонтеру. Публикация тройки с идентификатором связи волонтера и сигнала тревоги. (Отправка уведомления о необходимости помощи ближайшему волонтеру.)

Тест А3

Описание: Проверка поиска следующего волонтера в случае получения отказа от предыдущего

Тип: О

Входные данные: Сигнал тревоги. Данные местоположения пациентов, Данные местоположения волонтеров

Алгоритм:

- Запустить приложение пациента
- Запустить несколько приложений волонтера (с различными координатами местоположения)
- Запустить тестируемый объект
- Отправить сигнал тревоги из приложения пациента
- Отклонить запрос о помощи в приложении волонтера, в которое пришло уведомление.

Цель: Проверить получение сигнала тревоги следующим по расстоянию волонтером.

Ожидаемый результат: Отображение сообщения об успешном поиске ближайшего волонтеру. Публикация тройки с идентификатором связи волонтера и сигнала тревоги. Повторный поиск. Отображение повторного сообщения об успешном поиске. Повторная публикация связи нового волонтера и сигнала тревоги.

Тест А4

Описание: Проверка получения нескольких последовательных сигналов тревоги

Тип: О

Входные данные: Несколько сигналов тревоги

Алгоритм:

- Запустить несколько приложений пациента
- Запустить тестируемый объект
- Отправить несколько последовательных сигналов тревоги из разных приложений

Цель: Проверить, что сигналы тревоги добавляются в очередь и последовательно обрабатываются.
Ожидаемый результат: Отображение сообщений о получении сигналов тревоги. Отображение сообщений о попытках обработки разных сигналов тревоги.

Тест А5
Описание: Проверка восстановления работы после обрыва интернет-соединения
Тип: О
Входные данные: подключение к сети интернет
Алгоритм:

- Запустить тестируемый объект
- Запустить приложения пациента
- Отправить сигнал тревоги
- Отключить интернет-соединение
- Включить интернет-соединение

Цель: Проверить, что тестируемый объект продолжает обработку сигналов тревоги после прерывания интернет-соединения.
Ожидаемый результат: Отображение сообщения о потере соединения. Отображение сообщения о восстановлении соединения. Продолжение обработки сигналов тревоги, находящихся в очереди.

Нагрузочное тестирование

Тип теста: С
Алгоритм:

- Запустить тестируемый объект
- Запустить скрипт генерации волонтеров
- Запустить скрипт генерации пациентов

Цель: Выявить точку отказа тестируемого объекта, проверить время обработки сигнала тревоги в условиях загруженности.
Ожидаемый результат: Константное время обработки сигнала тревоги или линейная зависимость сигнала тревоги от вероятности отказа/количества волонтеров. Время обработки сигнала тревоги не превышает 30 сек. при поиске среди 50 волонтеров с вероятностью отказа 0.5.

Тестовое покрытие

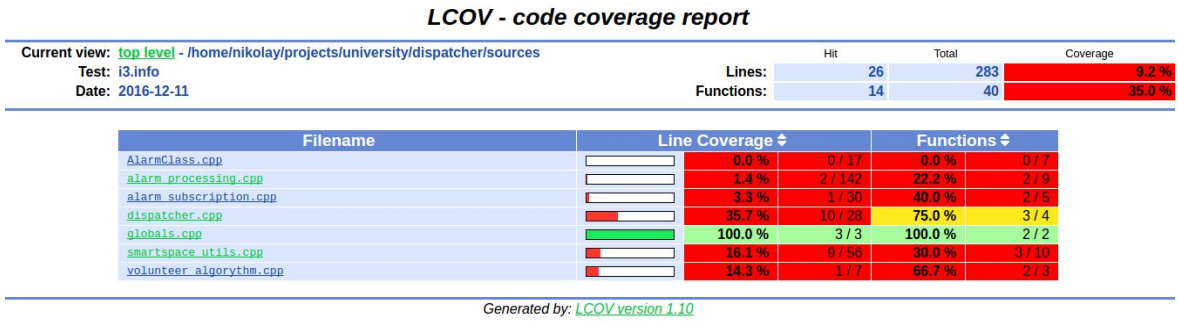
Сложность современного программного обеспечения и инфраструктуры сделало невыполнимой задачу проведения тестирования со 100% тестовым покрытием. Расчет тестового покрытия относительно исполняемого кода программного обеспечения проводится по формуле:

$$T_{cov} = \frac{L_{tc}}{L_{code}} * 100\%,$$ где:

- T_{cov} - тестовое покрытие
- L_{tc} - количество строк кода, покрытых тестами
- L_{code} - общее количество строк кода

На данный момент, без учета файлов онтологии и заголовочных файлов, количество строк кода проекта составляет 353 строки. Из них покрыто тестами: 313 строк. Таким образом тестовое покрытие составляет $\frac{313}{353} * 100\% = 88\%$

Утилита gcov и ее версия с графическим интерфейсом lcov показывают иной результат. Отчет утилиты показан на рисунке 5



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - /home/nikolay/projects/university/dispatcher/sources	Lines:	26	283	9.2 %
Test:	i3.info	Functions:	14	40	35.0 %
Date:	2016-12-11				

Filename	Line Coverage		Functions	
AlarmClass.cpp	0.0 %	0 / 17	0.0 %	0 / 7
alarm_processing.cpp	1.4 %	2 / 142	22.2 %	2 / 9
alarm_subscription.cpp	3.3 %	1 / 30	40.0 %	2 / 5
dispatcher.cpp	35.7 %	10 / 28	75.0 %	3 / 4
globals.cpp	100.0 %	3 / 3	100.0 %	2 / 2
smartspace_utils.cpp	16.1 %	9 / 56	30.0 %	3 / 10
volunteer_algorythm.cpp	14.3 %	1 / 7	66.7 %	2 / 3

Generated by: LCOV version 1.10

Рисунок 5 “Отчет о тестовом покрытии, полученный с помощью утилиты LCOV”

Как видно из отчета, тестовое покрытие составляет 35% функций. Скорее всего это связано с тем, что утилита основывается на фактических вызовах функций и методов тестируемого объекта, которые в рамках тестирования заменялись копиями или подключались из дополнительных файлов, созданных специально для тестирования. Таким образом, реальные функции были задействованы не всегда и не вошли в отчет, сгенерированный утилитой.

Примеры реализации тестов

Пример реализации тесткейса для функции connect_to_smartspace() модуля SmartSpace Helpers.

```
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MAIN
#include <boost/test/included/unit_test.hpp>
#include "../sources/smartspace_utilds.hpp"

BOOST_AUTO_TEST_CASE( correctConnection ) {
    int ret;

    ret = connect_to_smartspace();
    BOOST_CHECK_EQUALS(ret,0);
}

BOOST_AUTO_TEST_CASE( incorrectConnection ) {
    int ret;
```

```
    ret = connect_to_smartspace();
    BOOST_CHECK_EQUALS(ret,-1);
}
```

Пример реализации скрипта запуска эмулятора пациента при проведении нагрузочного тестирования

```
#!/bin/bash
RANDOM=10
JOBS_COUNTER=0
MAX_CHILDREN=101
MY_PID=$$
LIMIT=100
sleep_time=1000
LAMBDA=0.8
for i in {0..100}
do
   JOBS_COUNTER=$((`ps ax -Ao ppid | grep $MY_PID | wc -l`))
   while [ $JOBS_COUNTER -ge $MAX_CHILDREN ]
   do
      JOBS_COUNTER=$((`ps ax -Ao ppid | grep $MY_PID | wc -l`))
   done
      exec "./test-kps/patient/patient" &
      echo "$i"
         while [ $(echo "$sleep_time > $LIMIT" | bc -l) -ge 1 ]
         do
                  rnd=$(perl -e 'printf("%.2f\n", rand()/2)')
         sleep_time=$(echo "(-1/$LAMBDA)*l($rnd)" | bc -l)
                  echo $sleep_time
         done
         echo "$sleep_time"
      sleep $sleep_time
         sleep_time=1000
done
# wait for children here
while [ $JOBS_COUNTER -gt 1 ]
do
   JOBS_COUNTER=$((`ps ax -Ao ppid | grep $MY_PID | wc -l`))
done
```

Отчет о выполнении тестов

Результаты блочного тестирования

Номера тестов	Модуль	Количество запусков	Отчеты об ошибках	Дата проведения
1	CLI	10	-	07.12.2016
2	CLI	10	-	07.12.2016
3	I	10	-	07.12.2016
4	AC	10	-	07.12.2016
5	AC	10	-	07.12.2016
6	AC	10	-	07.12.2016
7	AL	10	-	07.12.2016
8	AL	10	-	07.12.2016
9	AL	10	-	07.12.2016
10	AL	10	-	07.12.2016
11	AL	10	-	07.12.2016
12	AL	10	-	07.12.2016
13	AP	10	-	07.12.2016
14	AP	10	-	07.12.2016
15	AP	10	-	07.12.2016
16	AP	10	-	07.12.2016
17	AP	10	-	07.12.2016
18	AP	10	-	07.12.2016
19	AP	10	-	07.12.2016
20	SA	10	-	07.12.2016
21	SA	10	-	07.12.2016
22	SA	10	-	07.12.2016

23	SA	10	-	07.12.2016
24	SA	10	-	07.12.2016
25	SA	10	Отчет №1	07.12.2016
26	SA	10	-	07.12.2016
27	SS	10	-	07.12.2016
28	SS	10	-	07.12.2016
29	SS	10	-	07.12.2016
30	SS	10	-	07.12.2016
31	SS	10	-	07.12.2016
32	SS	10	-	07.12.2016
33	SS	10	-	07.12.2016
34	SS	10	-	07.12.2016
35	SS	10	Отчет №2	07.12.2016
36	SS	10	-	07.12.2016
37	SS	10	-	07.12.2016
38	SS	10	-	07.12.2016
39	SS	10	-	07.12.2016
40	SS	10	-	07.12.2016
41	SS	10	-	07.12.2016
42	SS	10	-	07.12.2016
43	SS	10	-	07.12.2016
44	SS	10	Отчет №3	07.12.2016
45	SS	10	-	07.12.2016
46	SS	10	-	07.12.2016
47	SS	10	-	07.12.2016
48	SS	10	-	07.12.2016

Результаты интеграционного тестирования

Номер теста	Этап интеграции	Количество запусков	Отчет об ошибках	Дата проведения
И1	1	10	-	07.12.2016
И2	2	10	-	07.12.2016
И3	3	10	-	07.12.2016
И4	4	10	-	07.12.2016
И5	5	10	-	07.12.2016
И6	5	10	-	07.12.2016
И7	5	10	-	07.12.2016
И8	6	10	-	07.12.2016
И9	6	10	-	07.12.2016

Результаты аттестационного тестирования

Номер теста	Функциональное требование	Количество запусков	Отчет об ошибках	Дата проведения
А1	1,6	10	-	07.12.2016
А2	2,4,6	10	-	07.12.2016
А3	3,4,6	10	-	07.12.2016
А4	5,6	10	-	07.12.2016
А5	6,7	10	Отчет №4	07.12.2016

Результаты нагрузочного тестирования

Результаты нагрузочного тестирования представлены в виде графиков на рисунках 6 и 7. На графиках видно, что имеет место экспоненциальная зависимость между временем ожидания сигнала неотложного вызова в очереди перед обработкой и количеством одновременно свободных волонтеров. Также в ходе испытаний было

выяснено, что имеет место экспоненциальная зависимость между временем ожидания сигнала неотложного вызова в очереди и вероятностью отказа волонтера от помощи.
Дата проведения: 07.12.2016
Общий результат: тест не пройден (см. отчет об ошибках №5).

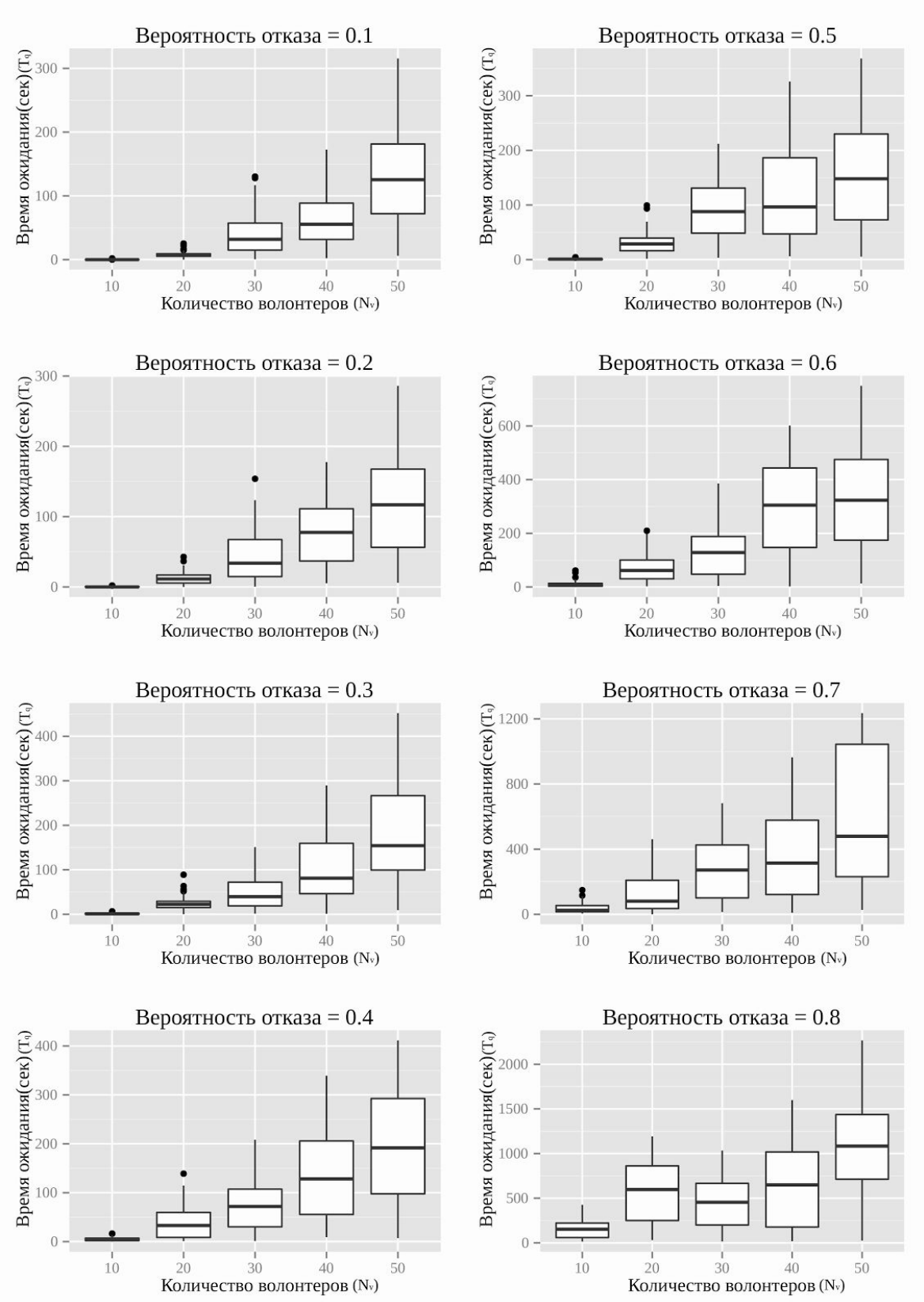


Рисунок 6 “Время обработки сигнала тревоги в зависимости от количества волонтеров с группировкой по вероятности отказа”

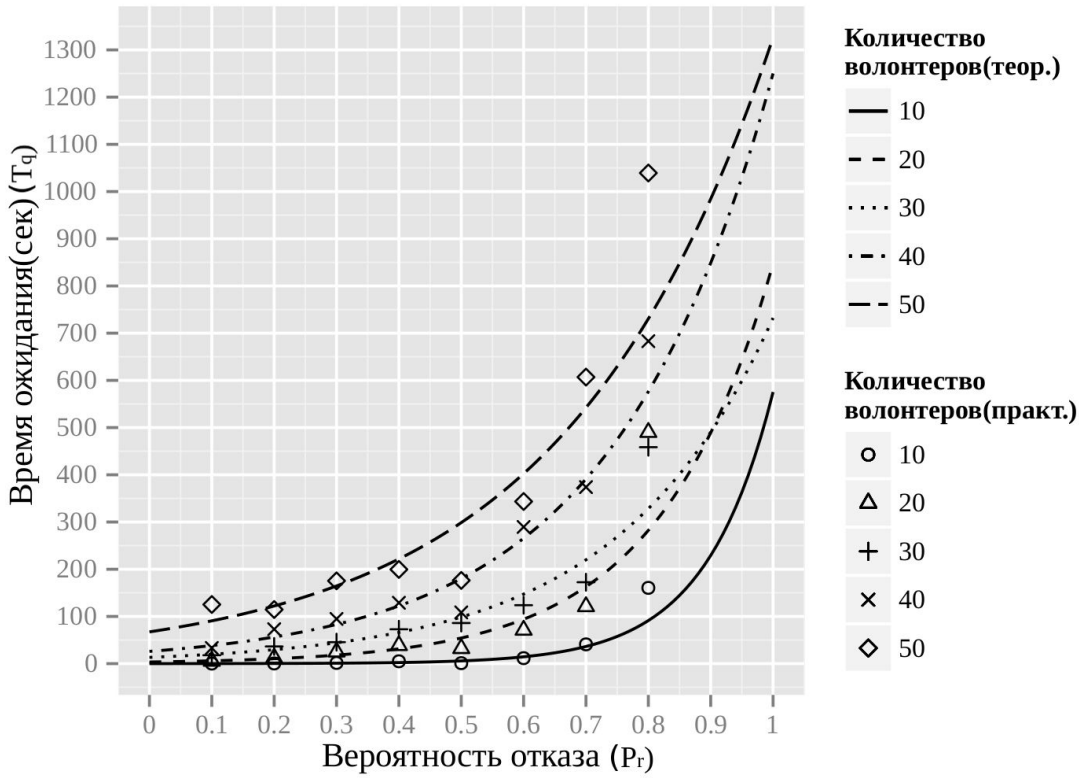


Рисунок 7 “Время обработки сигнала тревоги в зависимости от вероятности отказа с группировкой по количеству волонтеров”

Отчеты об ошибках

Отчет №1

Краткое описание: Ошибка при выполнении теста 25. Проверка выбрасывания ошибки при отрицательных значениях координат местоположения индивидов
Ожидаемый результат: Возврат ошибки выполнения функции.
Фактический результат: Функция считает расстояние, основываясь на отрицательных (недопустимых) значениях координат местоположения пациента и/или волонтера. Ошибка не возвращается.

Отчет №2

Краткое описание: Ошибка при выполнении теста 35. Проверка выбрасывания ошибки при NULL значении object
Ожидаемый результат: Возврат ошибки выполнения функции.
Фактический результат: Функция не выполняется, но сообщений об ошибке нет.

Отчет №3

Краткое описание: Ошибка при выполнении теста 44. Проверка выбрасывания ошибки при NULL значении domain
Ожидаемый результат: Возврат ошибки выполнения функции.
Фактический результат: Функция не выполняется, но сообщений об ошибке нет.

Отчет №4

Краткое описание: Ошибка при выполнении теста A5. Тест проверяет восстановление работы при обрыве интернет-соединения.
Ожидаемый результат: Продолжение работы после восстановления соединения
Фактический результат: После обрыва и восстановления интернет-соединения, тестируемый объект не восстанавливает соединение с интеллектуальным пространством, в следствие чего нарушается процесс получения новых сигналов тревоги и отправки уведомления свободным волонтерам.

Отчет №5

Краткое описание: Ошибка при выполнении нагрузочного тестирования. Тест проверяет работу объекта при высокой интенсивности поступающих сигналов тревоги.
Ожидаемый результат: Линейное время обработки сигнала, не превышающее 30 секунд
Фактический результат: Время обработки сигнала тревоги увеличивается экспоненциально. Время обработки сигнала тревоги превышает 30 секунд.

Результаты

В ходе разработки и выполнения тестов было выявлено несколько ошибок, которые представлены в отчетах 1-5.
По результатам тестирования, можно сказать, что объект готов к работе после некоторых доработок. Критическими ошибками, которые необходимо исправлять в первую очередь являются: невозможность продолжения работы после прерывания интернет-соединения и невозможность справиться с высокими нагрузками на сервис. Остальные найденные ошибки критическими не являются.