Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»

ОТЧЁТ

По лабораторной работе

По курсу “Тестирование и отладка ПО”

Исполнители — Студенты: Андосов А.И.
Меркулов Д.В.
Таразанов А.М.
Группа: ИУ7-71
Принял — Преподаватель: Рогозин О.В.

Москва 2016

Цель лабораторной работы

Написать спецификацию программного продукта и протестировать его с помощью модульных, интеграционных, функциональных и системных тестов.

Задача

Реализовать программу для анализа космической обстановки с использованием модели прогноза SGP4/SDP4. Программа должна предоставлять графический интерфейс для выполнения типовых задач.

Спецификация программного продукта

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

- загрузка TLE вручную оператором из файла в форматах 2 строки с названием, либо без названия. Реализовать автоматическое определение формата.
- список космических объектов. Поля: номер по спутниковому каталогу, международный номер, название. Фильтры по номерам, названию;
- просмотр TLE для одного или нескольких объектов на выбранную дату. Предусмотреть возможность сохранения в файл. Выгрузка должна быть в формате 3 строк;
- отображение информации на карте.

Доступные функции:

- загрузка TLE вручную;
- просмотр TLE с возможностью выгрузки в файл;
- установка текущего времени;
- выделение объекта в списке. Если объект выбран для отображения, необходимо выделять его на карте.

Политика выгрузки в файл: выгружать только те записи, которые имеются на выбранную дату.

Политика загрузки из файла: загружать все нормальные записи. Уведомлять пользователя о наличии некорректных записей или неверном формате файла.

На главной форме приложения должны отображаться следующие элементы:

- список объектов с фильтрами с возможностью выбора объектов для отображения;

- карта, на которую наносятся текущий и следующий витки;
- текущие дата/время;

Под текущим временем здесь будет пониматься время, указанное в элементе управления.

При запуске программы устанавливать текущее время на системное время.

Пользователь может указать любое другое текущее время. В любом случае при расчетах необходимо использовать TLE на ближайшую дату к текущему времени.

Отказы программы вследствие действий пользователя недопустимы.

Использованные технологии

- библиотека **SGP4**, используемая для расчёта координат спутников по TLE;
- библиотека **sqlite3** для работы с базой данных;
- библиотека **PyQt** для рисования графического интерфейса;
- средства **unittest** и **unittest.mock** стандартной библиотеки **Python** для модульного тестирования; **PQAut** для системного тестирования;
- coverage.py для определения степени покрытия ПО тестами.

Модульные тесты

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

С помощью модульных тестов мы протестировали основные компоненты нашего приложения.

Модуль Renderer

Класс Renderer

Метод render(self, t1, t2, odata_list)

Классы эквивалентности:

1. t1 >= t2;
2. odata_list пустой;
3. t1 < t2 и odata_list имеет элементы.

Граничные условия:

1. odata_list = [];
2. t1 = t2.

Тесты:

1	Входные данные	`odata = OrbitData()`
		`t1 = 1479416101.167`
		`t2 = 1479408901.167`
		`odata_list = [odata]`

	Ожидаемый результат	`[]`
	Покрытые классы	`1`

2	Входные данные	`t1 = 1479402901.233`
		`t2 = 1479412901.233`
		`odata_list = []`

	Ожидаемый результат	`[]`
	Покрытые классы	`2`

3	Входные данные	`odata = OrbitData()`
		`t1 = 1479408901.167`
		`t2 = 1479412901.233`
		`odata_list = [odata]`

	Ожидаемый результат	`непустой массив объектов Line`
	Покрытые классы	`3`

Класс Line

Метод clamp(self)

Классы эквивалентности:

1. x1, y1, x2, y2 в диапазоне [-1; 1] (отрезок полностью в видимой области);
2. одна из x координат вне необходимого диапазона;
3. одна из y координат вне необходимого диапазона;
4. отрезок не пересекает видимую область;
5. отрезок горизонтальный;
6. отрезок вертикальный.

Граничные условия:

1. x1, y1, x2, y2 = -1, 1;
2. x1 = x2;
3. y1 = y2.

Тесты:

4	Входные данные	`x1 = 0.1`
		`y1 = 0.3`
		`x2 = 0.2`
		`y2 = -0.6`

	Ожидаемый результат	`Line(0.1, 0.3, 0.2, -0.6)`
	Покрытые классы	`1`

5	Входные данные	x1 = -2.0 y1 = 0.0 x2 = 0.0 y2 = 1.0
	Ожидаемый результат	Line(-1.0, 0.5, 0.0, 1.0)
	Покрытые классы	2

6	Входные данные	x1 = 0.1 y1 = 2.0 x2 = 0.8 y2 = 0.0
	Ожидаемый результат	Line(0.45, 1.0, 0.8, 0.0)
	Покрытые классы	3

7	Входные данные	x1 = 1.5 y1 = 1.5 x2 = -1.2 y2 = -1.2
	Ожидаемый результат	Line(1.0, 1.0, -1.0, -1.0)
	Покрытые классы	2, 3

8	Входные данные	x1 = 2.1 y1 = 1.0 x2 = 3.8 y2 = 5.0
	Ожидаемый результат	None
	Покрытые классы	2, 4

9	Входные данные	x1 = 0.2 y1 = 3.0 x2 = 0.86 y2 = 4.0
	Ожидаемый результат	None
	Покрытые классы	3, 4

10	Входные данные	x1 = -2.0 y1 = 0.2 x2 = 1.0 y2 = 0.2
	Ожидаемый результат	Line(-1.0, 0.2, 1.0, 0.2)
	Покрытые классы	2, 5

11	Входные данные	x1 = 0.0 y1 = 3.0 x2 = 0.0 y2 = -0.5
	Ожидаемый результат	Line(0.0, 1.0, 0.0, -0.5)
	Покрытые классы	3, 6

Модуль TLEDecoder

Класс TLEListDecoder

Метод decode(self, str)

Классы эквивалентности:

1. нормальный TLE;
2. пустой TLE;
3. неполный TLE;
4. TLE без названия;
5. несовпадение контрольной суммы;
6. некорректные данные;
7. дано несколько TLE (полных).

Граничные условия:

1. Пустой TLE.

Тесты:

12	Входные данные	str = ""
	Ожидаемый результат	Ошибка
	Покрытые классы	2

13	Входные данные	str = "123"
	Ожидаемый результат	Ошибка
	Покрытые классы	3

14	Входные данные	str = "1 41480U 98067IA 16139.46459569 -.00000000 -00000-0 12670-2 0 19"
	Ожидаемый результат	Ошибка
	Покрытые классы	3, 4

15	Входные данные	str = "1 4327U 70009A 16138.91781019 -.00000000 -00000-0 24138-3 0 14 2 4327 99.3001 247.1187 0004353 340.4561 127.3943 13.58430395 19"
	Ожидаемый результат	Список из одного спутника с ID 4327U
	Покрытые классы	1, 4

16	Входные данные	str = "SERT 2 1 4327U 70009A 16138.91781019 -.00000000 -00000-0 24138-3 0 14 2 4327 99.3001 247.1187 0004353 340.4561 127.3943 13.58430395 19"
	Ожидаемый результат	Список из спутника “SERT 2” и ID 4327U
	Покрытые классы	1

17	Входные данные	str = "1 4327U 70009A 16138.91000019 -.00000000 -00000-0 24138-3 0 14 2 4327 99.3001 247.1187 0000000 310.4561 127.3943 13.58430395 19"
	Ожидаемый результат	Ошибка
	Покрытые классы	4, 5

18	Входные данные	str = "SERT 2 1 4327U 70009A 16138.91781019 -.00000000 -00000-0 24138-3 0 11 2 4327 99.3001 247.1187 0004353 340.4561 127.3943 13.58430395 11"
	Ожидаемый результат	Ошибка
	Покрытые классы	5

19	Входные данные	str = "1 4327U 70009A 16138.91781019 -.00000000 -00000-0 24138-3 0 14 2 4327 99.3001 247.1187 0004353 340.4561 127.3943 13.58430395 19"
	Ожидаемый результат	Ошибка
	Покрытые классы	4, 6

20	Входные данные	str = "1998-067IA 1 4327U 70009A 16138.91781019 -.00000000 -00000-0 24138-3 14 2 427 99.3001 247.1187 0353 340.4561 127.3943 13.58430395 19"
	Ожидаемый результат	Ошибка
	Покрытые классы	6

21	Входные данные	str = "SERT 2
		1 4327U 70009A 16138.91781019 -.00000000 -00000-0
		      24138-3 0 14
		2 4327 99.3001 247.1187 0004353 340.4561 127.3943
		      13.58430395 19
		SERT 2
		1 4327U 70009A 16138.91781019 -.00000000 -00000-0
		      24138-3 0 14
		2 4327 99.3001 247.1187 0004353 340.4561 127.3943
		      13.58430395 19"

	Ожидаемый результат	Список с двумя спутниками
	Покрытые классы	1, 7

22	Входные данные	str = "SERT 2
		1 4327U 70009A 16138.91781019 -.00000000 -00000-0
		      24138-3 0 14
		2 4327 99.3001 247.1187 0004353 340.4561 127.3943
		      13.58430395 19
		1998-067IA
		1 4327U 70009A 16138.91781019 -.00000000 -00000-0
		      24138-3 14
		2 427 99.3001 247.1187 0353 340.4561 127.3943
		      13.58430395 19"

	Ожидаемый результат	Список с одним спутником
	Покрытые классы	5, 7

23	Входные данные	str = "SERT 2
		1 4327U 70009A 16138.91781019 -.00000000 -00000-0
		      24138-3 0 14
		2 4327 99.3001 247.1187 0004353 340.4561 127.3943
		      13.58430395 19
		SERT 2
		1 4327U 70009A 16138.91781019 -.00000000 -00000-0
		      24138-3 0 11
		2 4327 99.3001 247.1187 0004353 340.4561 127.3943
		      13.58430395 11"

	Ожидаемый результат	Список с одним спутником
	Покрытые классы	6, 7

24	Входные данные	`str = "1998-067IA`
		`1  4327U 70009A   16138.91781019 -.00000000 -00000-0`
		`      24138-3     14`
		`2  427  99.3001 247.1187 0353 340.4561 127.3943`
		`      13.58430395    19`
		`SERT 2`
		`1  4327U 70009A   16138.91781019 -.00000000 -00000-0`
		`      24138-3 0    11`
		`2  4327  99.3001 247.1187 0004353 340.4561 127.3943`
		`      13.58430395    11"`

	Ожидаемый результат	`Ошибка`
	Покрытые классы	`5, 6, 7`

Модуль SatView

Класс SatView

Метод sat_by_norad(self)

Тесты:

25	Условия	`Попытка выбора спутника из пустого списка по NORAD ID`
	Ожидаемый результат	`Возврат None (вызов sv.sat_by_norad)`

26	Условия	`Выбор спутника по NORAD ID из списка, где есть такой спутник`
	Ожидаемый результат	`Возврат спутника (вызов sv.sat_by_norad)`

Метод save_tle(self)

Тесты:

27	Условия	`Попытка сохранить список TLE в файл с некорректным названием/путём`
	Ожидаемый результат	`Вывод ошибки (вызов gui.show_error)`

Метод load_tle(self)

Тесты:

28	Условия	`Попытка загрузить список TLE из файла с некорректным названием/путём`
	Ожидаемый результат	`Вывод ошибки (вызов gui.show_error)`

Метод render(self)

Тесты:

29	Условия	`Попытка отрисовки пустого списка спутников`
	Ожидаемый результат	`Интерфейс ничего не отрисовывает (вызов gui.render_paths с аргументом [])`

30	Условия	`Попытка отрисовки непустого списка спутников`
	Ожидаемый результат	`Интерфейс отрисовывает траектории спутников из списка (вызов gui.render_paths с непустым аргументом)`

Метод frame(self)

Тесты:

31	Условия	`В очередь событий от интерфейса поступило несколько событий`
	Ожидаемый результат	`Обработка каждого из событий и обновление графического интерфейса (вызов gui.poll_events, gui.update и EVENT.process)`

Анализ покрытия кода модульными тестами

Тестовое покрытие — метрика оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода. Существуют следующие подходы к оценке и измерению тестового покрытия:

1. покрытие требований;
2. покрытие кода (операторов);
3. покрытие на базе анализа потока управления.

Покрытие требований

$$\mathbf{Tcov = (Lcov/Ltotal) * 100\%} \qquad (1)$$

где:

Tcov *- тестовое покрытие*

Lcov *- количество требований, проверяемых тест кейсами*

Ltotal *- общее количество требований*

Для измерения покрытия требований необходимо определить множество требований к продукту (по пунктам). Каждый пункт связывается с тест кейсами, проверяющими его. Тогда покрытие будет вычислено как отношение количества требований, проверенных тестами, к общему числу требований.

Покрытие кода (операторов)

$$Tcov = (Ltc/Lcode) * 100\% \quad (2)$$

где:

Tcov *- тестовое покрытие*

Ltc *- кол-ва строк кода, покрытых тестами*

Lcode *- общее кол-во строк кода.*

Для вычисления покрытия кода необходимо найти отношение всех строк программы, в которые были вхождения во время тестирования, к общему числу строк программы.

Покрытие на базе анализа потоков управления

Тестирование потоков управления — это одна из техник тестирования белого ящика, основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

Для тестирования потоков управления определены разные **уровни тестового покрытия**:

	Название	**Краткое описание**
0	--	“Test whatever you test, users will test the rest”
1	Покрытие операторов	Каждый оператор должен быть выполнен как минимум один раз.
2	Покрытие альтернатив / ветвей	Каждая ветвь (альтернатива) каждого узла с ветвлением выполнена как минимум один раз.
3	Покрытие условий	Каждое условие, имеющее TRUE и FALSE на выходе, выполнено как минимум один раз.
4	Покрытие условий альтернатив	Тестовые случаи создаются для каждого условия и альтернативы
5	Покрытие множественных условий	Достигается покрытие альтернатив, условий и условий альтернатив (уровни 2, 3 и 4)
6	“Покрытие бесконечного числа путей”	Если, в случае зацикливания, количество путей становится бесконечным, допускается существенное их сокращение, ограничивая количество циклов выполнения, для уменьшения числа тестовых случаев.
7	Покрытие путей	Все пути должны быть проверены

Coverage.py

Для анализа покрытия нами использовалась утилита **coverage.py**. Данный пакет может работать в двух режимах: режиме анализа покрытия операторов и режиме анализа покрытия ветвей. Утилита coverage.py использует встроенную в интерпретаторы Python возможность задать т.н. trace-функцию, которая выполняется при каждом переходе на новую строку кода, при вызове функции, возникновении исключения и т.д. В режиме анализа покрытия операторов coverage.py с помощью trace-функции определяет номера строк, выполнившихся в ходе работы тестов, а затем, используя дебаг-информацию в файлах байткода Python (**.pyc*), которая содержит также и номера строк, соответствующие операторам, считает покрытие по формуле (2), или, в терминах таблицы ниже

$$\textbf{coverage} = (\textbf{statements} - \textbf{missing}) / \textbf{statements} * 100\% \quad (3)$$

Покрытие считается как для всего проекта, так и для каждого модуля в отдельности. На выходе при работе в данном режиме получаем отчет вида:

Coverage report: 70%

Module ↓	*statements*	*missing*	*excluded*	*coverage*
modules\coordconv.py	29	0	0	100%
modules\dbcontroller.py	88	66	0	25%
modules\decoder.py	19	3	0	84%
modules\events.py	17	6	0	65%
modules\gui.py	23	11	0	52%
modules\gui_qt.py	144	112	0	22%
modules\gui_qt_utils.py	72	55	0	24%
modules\renderer.py	136	40	0	71%
modules\satellite.py	66	3	0	95%
modules\satview.py	105	25	0	76%
modules\tledecoder.py	110	1	0	99%
modules\utils.py	17	1	0	94%
test_misc.py	20	0	0	100%
test_renderer.py	37	0	0	100%
test_satview.py	143	0	0	100%
test_tledecoder.py	62	0	0	100%
Total	**1088**	**323**	**0**	**70%**

В режиме покрытия ветвей (coverage.py run --branch) coverage.py статически анализирует байткод, определяя номера строк, содержащих условные переходы, таким образом определяя множество возможных точек перехода и возможных назначений этих

переходов. Во время работы trace-функция собирает пары номеров строк (какая строка выполнилась предыдущей и какая строка выполнится сейчас), сравнивая их с полученным заранее множеством переходов, таким образом определяя, какие ветви переходов были выполнены. В этом случае покрытие кода будет измеряться по формуле

$$\mathbf{Tcov = (Btc/Bcode) * 100\%} \quad (4)$$

где:

Tcov *- тестовое покрытие*

Btc *- кол-во ветвей (назначений условных переходов), покрытых тестами*

Bcode *- общее кол-во ветвей в коде.*

В branch-режиме coverage.py также считает и покрытие операторов, поэтому реальная формула будет выглядеть так:

$$\mathbf{Tcov = (Ltc + Btc) / (Lcode + Bcode) * 100\%} \quad (5)$$

А отчет так:

Coverage report: 67%

Module ↓	statements	missing	excluded	branches	partial	coverage
modules\coordconv.py	29	0	0	4	0	100%
modules\dbcontroller.py	89	67	0	18	0	21%
modules\decoder.py	19	3	0	2	1	81%
modules\events.py	17	6	0	0	0	65%
modules\gui.py	23	11	0	2	0	48%
modules\gui_qt.py	144	112	0	28	0	19%
modules\gui_qt_utils.py	72	55	0	8	0	21%
modules\renderer.py	136	40	0	58	8	66%
modules\satellite.py	66	3	0	4	1	94%
modules\satview.py	105	25	0	20	2	72%
modules\tledecoder.py	110	1	0	20	1	98%
modules\utils.py	17	1	0	8	1	92%
test_misc.py	20	0	0	0	0	100%
test_renderer.py	37	0	0	6	0	100%
test_satview.py	143	0	0	0	0	100%
test_tledecoder.py	62	0	0	0	0	100%
Total	**1089**	**324**	**0**	**178**	**14**	**67%**

Для вычисления покрытия в дальнейшем будем использовать стандартный режим.

Интеграционные тесты

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности. Тестирование этих проектируемых единиц — объединения, множества или группы модулей — выполняется через их интерфейс, с использованием тестирования «чёрного ящика».

Нами было произведено интеграционное тестирование следующих модулей:

- SatView (Application);
- DBController.

Тестирование взаимодействия DBController и SatView

Информация о спутниках в программе и базе данных представлена двумя сущностями: **Sat** и **OrbitData**.

Объекты класса **Sat** содержит основную информацию о спутнике: номер в базе NORAD, международный идентификатор и название. NORAD - целое положительное число, международный идентификатор и название - строки из любых символов, а также ссылку на самый “свежий” соответствующий ему объект OrbitData.

Объекты класса **OrbitData** содержат информацию, используемую в модели SGP4 для построения траекторий спутника, а также дату, когда эта информация была получена. Таким образом, каждому спутнику (Sat) соответствуют один или несколько объектов OrbitData; несколько в случае если есть информация о положении спутника на несколько различных моментов времени.

SatView содержит список известных спутников, получаемый из БД (**DBController**). При возникновении соответствующих событий от интерфейса пользователя (GUI), SatView может запрашивать у DBController новый список спутников (*reload_sats()*), соответствующие определенному спутнику объекты OrbitData (*render()*) для последующего перенаправления в рендерер (Renderer), или же направлять в DBController загруженные им из файла TLE объекты Sat (и OrbitData) для добавления их в БД (*process_tle()*).

Так как тяжело проверить на верность большинство данных модели SGP4, объект Sat считается “верным”, если:

- его NORAD-номер больше 0;
- его название - непустая строка;
- его международный ID - непустая строка;
- OrbitData, на который ссылается этот объект, также верен.

OrbitData считается верным, если:

- год эпохи - неотрицательное число в пределах 0-99 (числа меньше 57 соответствуют годам 2000..2056, числа >= 57 соответствуют годам 1957..1999);
- время эпохи (день в году) - неотрицательное число в пределах 0-365.

Для проверки корректности взаимодействия были написаны следующие интеграционные тесты:

Метод reload_sats(self)

Тесты:

32	Условия	dbc выдаёт ошибку доступа к базе данных
	Ожидаемый результат	- Вывод ошибки (gui.show_error)

33	Условия	dbc возвращает пустой список Sat
	Ожидаемый результат	- satview.sats == [] - Очистка списка спутников (gui.clear_sats) - Список в интерфейсе пуст (gui.set_sats вызван с аргументом [])

34	Условия	dbc возвращает непустой список корректных Sat
	Ожидаемый результат	- satview.sats == dbc.get_all() != [] - Очистка списка спутников (gui.clear_sats) - Список в интерфейсе не пуст (gui.set_sats вызван с аргументом satview.sats/dbc.get_all())

35	Условия	dbc возвращает непустой список Sat, некоторые или все из которых некорректны (sat.is_valid() == False)
	Ожидаемый результат	- satview.sats содержит только корректные Sat - Очистка списка спутников (gui.clear_sats) - Список в интерфейсе не пуст, но содержит только корректные Sat (gui.set_sats() вызван с аргументом satview.sats != dbc.get_all())

Метод render(self, ta, tb, norads)

Тесты:

36	Условия	dbc выдаёт ошибку доступа к базе данных
	Ожидаемый результат	- Вывод ошибки (gui.show_error)

37	Условия	dbc возвращает верные OrbitData для всех спутников в norads
	Ожидаемый результат	- Создание траекторий всех спутников из norads (rdr.render() вызван с аргументом dbc.get_orbits_between(n, ta, tb) для каждого n из norads); - Отображение траекторий всех спутников из norads (gui.render_paths() вызван со списком пар (n, orbs) для каждого n из norads, где orbs - непустой список сегментов линии траектории)

38	Условия	dbc возвращает верные OrbitData не для всех спутников в norads
	Ожидаемый результат	- Создание траекторий только тех спутников из norads, для которых из dbc пришли верные OrbitData (rdr.render() вызван со списком длины менее dbc.get_orbits_between(n, ta, tb) для каждого n из norads, или вообще не вызван для этого n); - Отображение траекторий тех спутников из norads, для которых пришли корректные OrbitData (gui.render_paths() вызван со списком пар (n, orbs) для каждого “корректного” n из norads) - Вывод предупреждения (gui.show_warning)

Метод process_tle(self, txt)

Тесты:

39	Условия	dbc выдаёт ошибку доступа к базе данных
	Ожидаемый результат	- Ошибка поднимается наверх (DBError)

40	Условия	Загружены верные TLE и успешно переданы в dbc
	Ожидаемый результат	- Спутники успешно добавлены в БД (dbc.sat_exists() == True)

Анализ покрытия кода интеграционными и модульными тестами

После добавления интеграционных тестов уровень покрытия вырос до 77%:

Coverage report: 77%

Module ↓	statements	missing	excluded	coverage
modules\coordconv.py	29	0	0	100%
modules\dbcontroller.py	89	6	0	93%
modules\decoder.py	19	3	0	84%
modules\events.py	17	6	0	65%
modules\gui.py	23	11	0	52%
modules\gui_qt.py	144	112	0	22%
modules\gui_qt_utils.py	72	55	0	24%
modules\renderer.py	136	40	0	71%
modules\satellite.py	66	3	0	95%
modules\satview.py	105	25	0	76%
modules\tledecoder.py	110	1	0	99%
modules\utils.py	17	1	0	94%
test_misc.py	20	0	0	100%
test_renderer.py	37	0	0	100%
test_satview.py	143	0	0	100%
test_satview_dbcontroller.py	61	0	0	100%
test_tledecoder.py	62	0	0	100%
Total	**1150**	**263**	**0**	77%

Системное тестирование

Системное тестирование — это тестирование программного обеспечения, выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования «чёрного ящика», и, тем самым, не требует знаний о внутреннем устройстве системы.

Системное тестирование было выполнено с использованием фреймворка **PQAut** для автоматизации тестирования ПО с графическим интерфейсом на **PyQt5**. Фреймворк позволяет описывать сценарии тестирования на языке **Gherkin**. Например, один из тестов модуля Renderer:

```
Scenario: Rendering nothing
    Given I have no satellites selected
    When I tap on "Render"
    then there are no lines on the canvas
```

When-, then- и given-выражения затем реализуются на Python:

```
@when('I tap on "{name}"')
def i_tap_on(context, name):
    pqaut.tap(name)

@then('there are no lines on the canvas')
def there_are_no_lines(context):
    cv = pqaut.find_element("mapCanvas", "")
    assert_equal(cv["items"], 0)
```

Регрессионное тестирование

Регрессионное тестирование — это выборочное тестирование, позволяющее убедиться, что изменения не вызвали нежелательных побочных эффектов, или что измененная система по-прежнему соответствует требованиям.

Для того чтобы знать, какие тесты перезапускать после того или иного изменения в программе, нужно определить, от каких конкретно частей программы (модулей, методов, и т.п.) зависит результат каждого теста. Для этого часто используется **управляющий граф**, отображающий поток управления программы, по которому легко отследить зависимости одних блоков/модулей/методов от других.

Мы построили один из вариантов управляющего графа: **граф вызовов**, показывающий, какие методы или функции вызывают какие. Граф был построен с помощью утилиты **pycallgraph**, результатом работы которого является ориентированный граф, где вершинами являются методы, а ребрами вызовы одних методов другими.

Проследив по этому графу иерархию методов, можно было легко установить, какие тесты зависят от поведения каких методов. Результат представлен в таблицах ниже.

Таблица зависимости тестов от методов

Метод	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Renderer.render	+	+	+																	
Renderer.__get_coord			+																	
Renderer.__get_lines			+																	
Renderer.__render_border			+																	
Renderer.__get_coords_from_nearest			+																	
Renderer.__render_mid			+																	
Renderer.__set_steps			+																	
Line.clamp			+	+	+	+	+	+	+	+	+									
TLEListDecoder.decode												+	+	+	+	+	+	+	+	+
TLEListDecoder.__sat_from_tle														+	+	+	+	+	+	+
OrbitData.init_date	+		+																	
OrbitData.init_sgp4			+																	
OrbitData.propagate			+																	
wgs84_to_lonlat			+																	
normalize_lon			+																	
lonlat_to_mercator			+																	
wgs84_to_ndc			+																	
jt_to_gmst			+																	
tle_checksum																	+	+		

Таблица зависимости тестов от методов (продолжение)

Метод	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
SatView.reload_sats												+	+	+	+					
SatView.sat_by_norad					+	+											+	+		
SatView.save_tle							+													
SatView.process_tle																			+	+
SatView.load_tle								+												
SatView.render									+	+						+	+	+		
SatView.frame											+									
DBController.get_all														+			+	+		
DBController.get_orbits_between																	+	+		
DBController.get_orbit_latest														+						
DBController.sat_exists																				+
DBController.sat_add																				+
DBController.sat_delete																				+
DBController.orbdata_add																				+
DBController.add																				+
DBController.sync																				+
DBController.__init_db														+			+			+
DBController.__close_orbits																	+			
DBController.__closest_orbit																	+			
DBController.__rows_to_orbdata																	+			
DBController.__row_to_orbdata																	+			
DBController.__row_to_sat														+			+			
Renderer.render										+							+	+		
Renderer.__get_coord										+							+	+		
Renderer.__get_lines										+							+	+		
Renderer.__render_border										+							+	+		
Renderer.__get_coords_from_nearest										+							+	+		
Renderer.__render_mid										+							+	+		
Renderer.__set_steps										+							+	+		
Line.clamp										+										
TLEListDecoder.decode	+	+	+	+																
TLEListDecoder.__sat_from_tle	+	+	+	+																
OrbitData.init_date	+	+	+	+						+							+	+		
OrbitData.init_sgp4										+							+	+		
OrbitData.propagate										+							+	+		
OrbitData.is_valid									+	+							+	+		+
Sat.is_valid									+	+							+	+		+
wgs84_to_lonlat										+							+	+		
normalize_lon										+							+	+		
lonlat_to_mercator										+							+	+		
wgs84_to_ndc										+							+	+		
jt_to_gmst										+							+	+		
tle_checksum	+	+	+	+																

Для проверки правильности составленной таблицы в программу были внесены изменения. Поведение метода **Line.clamp()** было изменено так, чтобы результат его в некоторых граничных условиях изменился (когда y-координаты располагались в порядке возрастания или x-координаты не попадали в промежуток [-1, 1]). По таблице можно отследить, что от поведения этого метода зависят тесты 3-11 и 30. Действительно, перезапустив тесты, мы обнаружили, что изменился результат тестов 3-11:

```
======================================================================
FAIL: test_clamp (test_renderer.TestLine)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\figgis\Desktop\Work\lab_prj\tests\test_renderer.py", line 29, i
n test_clamp
    self.assertTrue(fabs(a[i] - b[i]) < 1e-6)
AssertionError: False is not true
```

Таким образом, достаточно было перезапустить лишь указанный набор тестов.

Автоматизация генерации тестов

Реализация тестов — достаточно затратный процесс, поэтому часто прибегают к тем или иным средствам, его облегчающим. Например, часто используются специальные BDD-фреймворки, позволяющие описывать те или иные функции (возможности, особенности) программы простыми сценариями, состоящими из заранее реализованных элементов.

Использованный нами в процессе системного тестирования фреймворк PQAut основан на BDD-фреймворке **Behave**, позволяющем описывать тесты на функции (features) программы на языке Gherkin. Поэтому в нашем случае можно использовать для генерации любых тестов тот же подход, что и при разработке системных тестов, описанный выше.

Пример

На данный момент в программе при попытке рендеринга при отсутствии выбранных спутников просто ничего не происходит: на карте не рисуется ничего нового, никакие другие элементы интерфейса не изменяются. В ходе системного тестирования среди прочих тестов мы реализовали тест, это проверяющий, состоящий из сценария на Gherkin, использующего заранее реализованные блоки:

```
Scenario: Rendering nothing
    Given I have no satellites selected
    When I tap on "Render"
    then there are no lines on the canvas
```

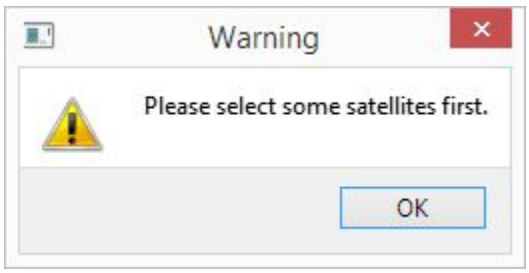
Реализация блоков:

```
@given('I have no satellites selected')
def i_have_no_satellites_selected(context):
    pass  # по умолчанию после запуска программы ничего не выбрано


@when('I tap on "{name}"')
def i_tap_on(context, name):
    pqaut.tap(name)


@then('there are no lines on the canvas')
def there_are_no_lines(context):
    cv = pqaut.find_element("mapCanvas", "")
    assert_equal(cv["items"], 0)
```

Предположим, что в новой версии приложения мы хотим предупреждать пользователя о попытке рендеринга пустого списка спутников с помощью gui.show_warning():



После добавления этого функционала описанный выше тест также должен выполняться (на карте и так ничего не станет рисоваться). При этом требуется написать новый тест, чтобы удостовериться, что диалог действительно показывается. Это можно сделать, просто написав новый Gherkin-сценарий для данной ситуации, используя те же блоки:

```
Scenario: Warning when rendering nothing
    Given I have no satellites selected
    When I tap on "Render"
    then a “Warning” dialog pops up
```

Блок ‘a “{}” dialog pops up’ уже был реализован и использован нами в ходе системного тестирования:

```
@then('a "{title}" dialog pops up')
def dialog_pops_up(context, title):
      cv = pqaut.find_element("dialog", "QDialog")
      assert_not_equal(cv, None)
      assert_equal(cv["title"], title)
```

После добавления этой новой функции было достаточно исполнить лишь упомянутый выше тест на рисование пустого списка и этот новый тест. Оба теста выполнились успешно.