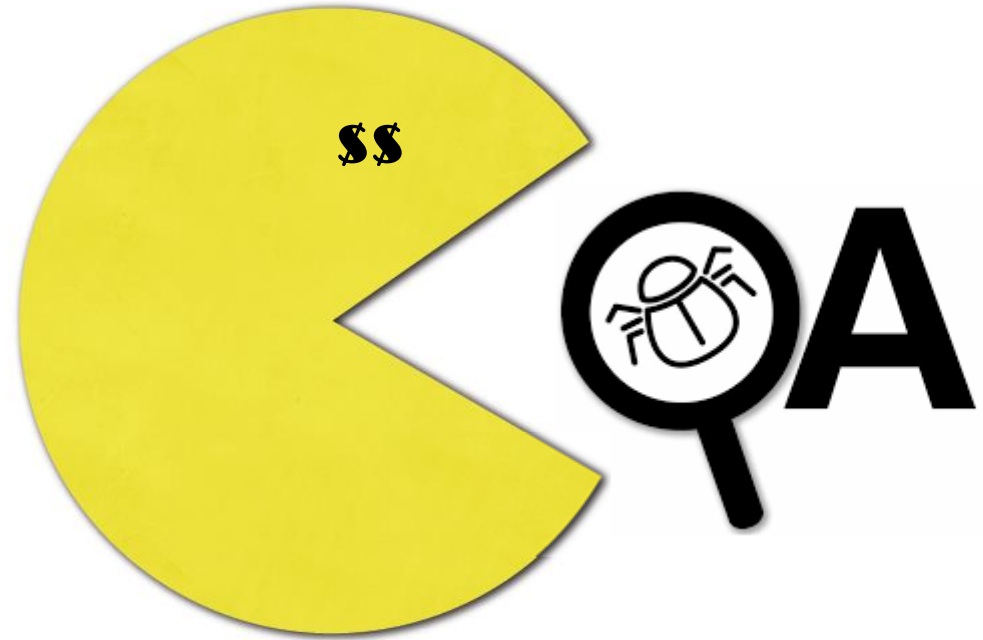


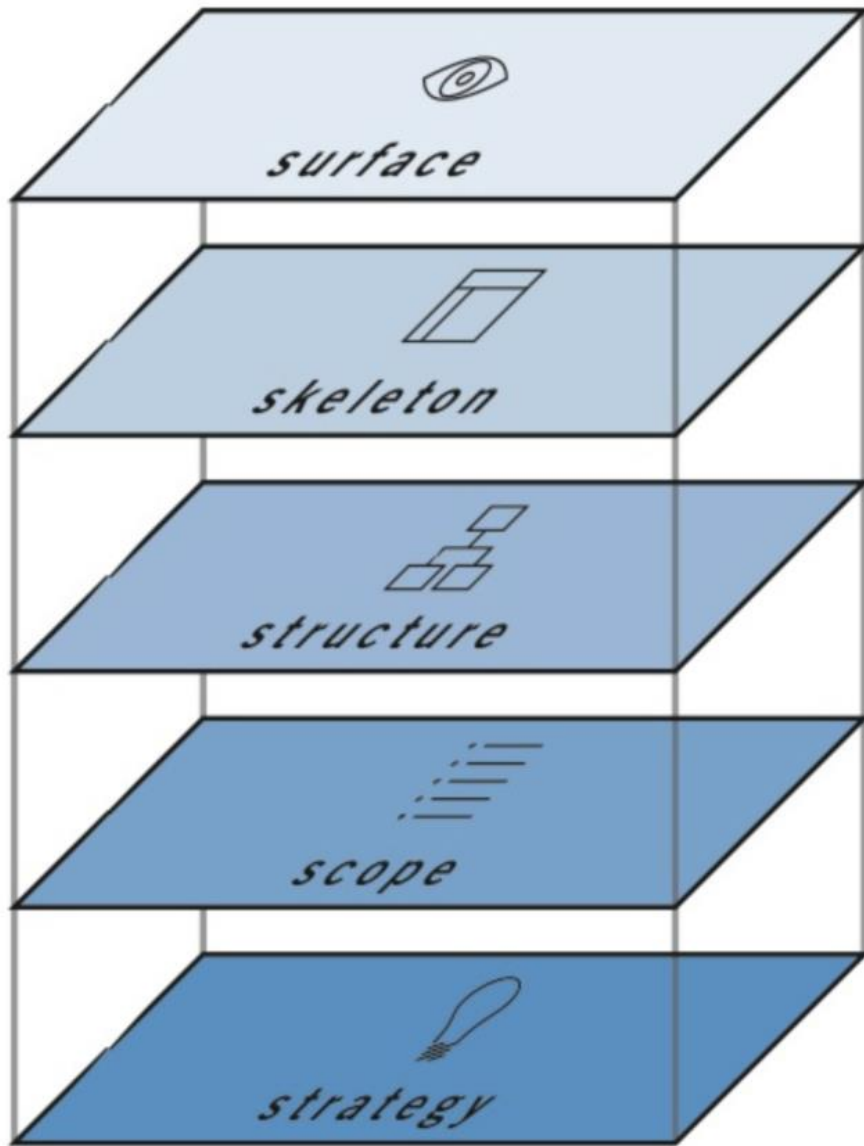
nexign



■ Percentage of chart that looks like PacMan



Jesse James Garrett



<http://www.jjg.net/elements/>

Concrete



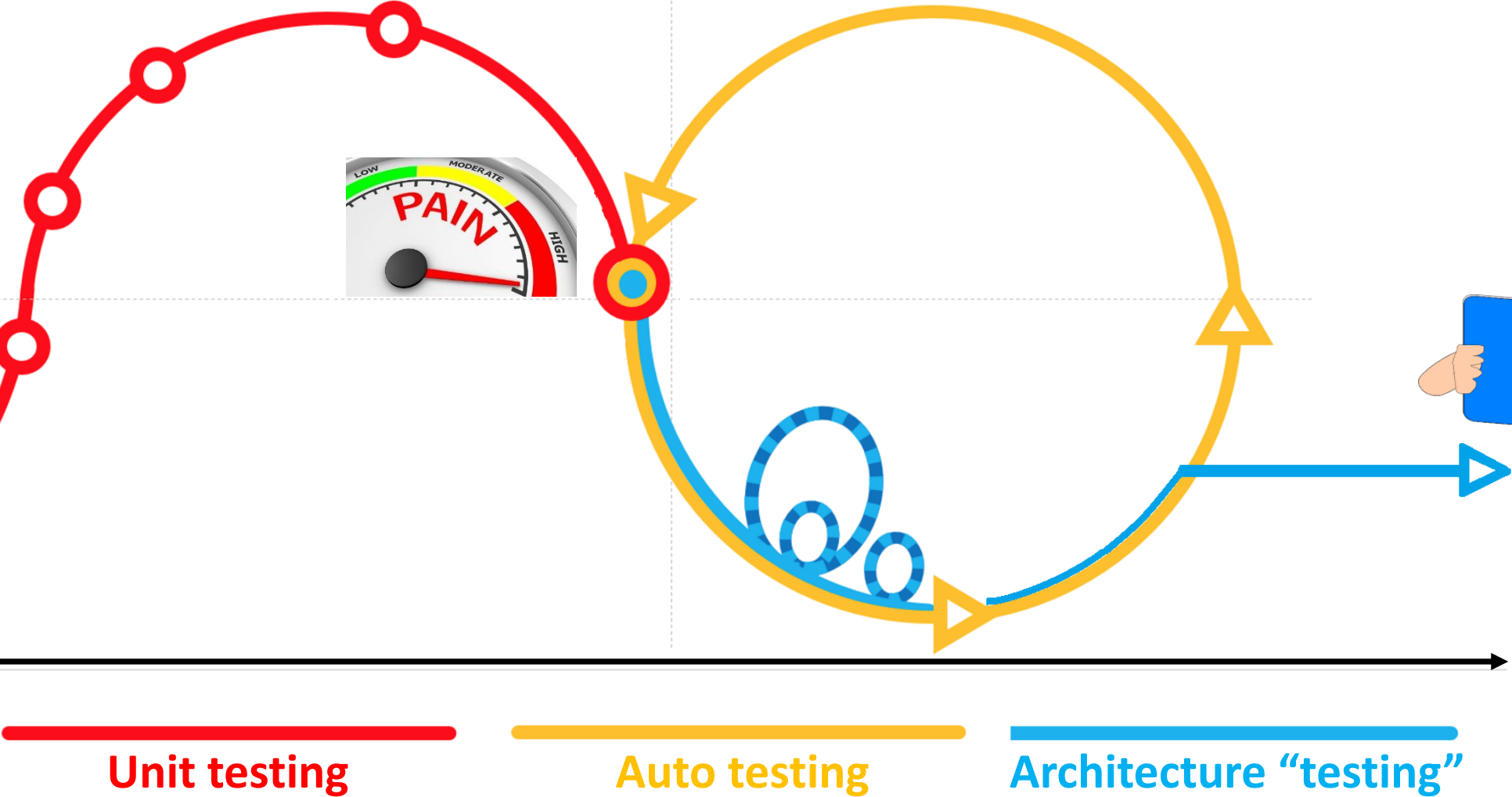
Abstract



Concrete

Our QA way

Abstract



UNIT TESTING

Nexign [C:\Victor\Nexign] - ...\monitor\monitor.py [Nexign] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Nexign > monitor > monitor.py

Project

- message_server.py
- notifier
- sensors
 - _init_.py
 - kettle.py
 - router.py
 - sensor.py
- .coverage
- _init_.py
- _main_.py
- config.py
- monitor.py
- test_messageHandler.py
- config.yaml
- features.txt
- releasenotes.txt
- requirements.txt
- run_monitor.cmd

External Libraries

Scratches and Consoles

```
10
11 class MessageHandler(asyncore.dispatcher_with_send):
12
13     _sockets = dict()
14     _sensors = dict()
15
16     def __init__(self, p_sock):
17         super().__init__(p_sock)
18         MessageHandler._sockets[str(self.addr)] = self
19
20     def handle_read(self):
21         print('handler handle_read')
22         data = self.recv(8192)
23         if data:
24             i_json = json.loads(data.decode('utf-8'))
25             i_message_type = i_json.get('message_type', 'unknown')
26             i_payload = i_json.get('payload', dict())
27             if i_message_type == 'register':
28                 i_sensor = instantiate_sensor(i_payload)
29                 if i_sensor is not None and i_sensor.type in [x.type for x in Monitor.config.sensors]:
30                     i_sensor.address = self.addr
31                     MessageHandler._sensors[i_sensor.id] = i_sensor
32             elif i_message_type == 'unregister':
33                 i_sensor = instantiate_sensor(i_payload)
```

MessageHandler

Run: Unittests for test_messageHandler.TestMessage... x

Test Results 24 ms

- test_messageHandler 24 ms
 - TestMessageHandler 24 ms
 - test_handle_close 7 ms
 - test_handle_connect 3 ms
 - test_handle_read 10 ms
 - test_handle_write 4 ms

Ran 4 tests in 0.030s

FAILED (failures=4)

Process finished with exit code 1

Python Console Terminal Run Debug TODO

Tests failed: 4, passed: 0 (5 minutes ago)

63:1 CRLF UTF-8

AUTO TESTING

Coverage for **sample.py** : 42%

12 statements 5 run 7 missing 0 excluded

```
1  # -*- coding: utf-8 -*-
2
3  def sum(num1, num2):
4      return num1 + num2
5
6
7  def sum_only_positive(num1, num2):
8      if num1 > 0 and num2 > 0:
9          return num1 + num2
10     else:
11         return None
12
13
14  def test_sum():
15      assert sum(5, 5) == 10
16
17
18  def test_sum_positive_ok():
19      assert sum_only_positive(2, 2) == 4
20
21
22  def test_sum_positive_fail():
23      assert sum_only_positive(-1, 2) is None
24
25
```

Use `coverage report` to report on the results:

```
$ coverage report -m
Name                               Stmts  Miss  Cover   Missing
-----
my_program.py                      20     4    80%    33-35, 39
my_other_module.py                 56     6    89%    17-23
-----
TOTAL                              76    10    87%
```

Name	Stmts	Miss	Cover	Missing

sensor.py	31	16	48%	7-8, 11-12, 15-18, 22, 26, 30, 34, 38, 44, 47, 53

ARCHITECTURE “TESTING”

NX_1 [C:_Victor\NX_1] - ...\monitor\sensors\sensor.py [Nexign] - PyCharm

File Edit View Navigate Code Refactor Run Tools VCS Window Help

NX_1

monitor

sensors

sensor.py

monitor

Project

NX_1 [Nexign] C:_Victor\NX_1

monitor

message_service

notifier

sensors

__init__.py

kettle.py

router.py

sensor.py

.coverage

__init__.py

__main__.py

config.py

monitor.py

config.yaml

features.txt

releasenotes.txt

requirements.txt

run_monitor.cmd

External Libraries

Scratches and

sensor.py

```
4 class Sensor:
5
6     def __init__(self, p_sensor_json):
7         self._json = p_sensor_json
8         self._last_update = datetime.now()
9
10    def update(self, p_sensor):
11        self._last_update = datetime.now()
12        self._json['status'] = p_sensor.status
13
14    def check_status(self, p_timeout):
15        i_timedelta = (datetime.now() - self._last_update)
16        i_since_last = (i_timedelta.microseconds + (i_timedelta.seconds + i_timedelta.days * 24 * 3600) * 10 ** 6) / 10 ** 6
17        if i_since_last > p_timeout:
18            self._json['status'] = 'offline'
19
20    @property
21    def id(self):
22        return self._json.get('id')
23
24    @property
25    def type(self):
26        return self._json.get('type')
```

PC NX_1 [C:_Victor\NX_1] - ...\mon...

PC NX_2 [C:_Victor\NX_2] - ...\mon...

PC NX_3 [C:_Victor\NX_3] - ...\mon...

PC NX_4 [C:_Victor\NX_4] - ...\mon...

6: TODO

Event Log

31:1 CRLF UTF-8

PC

PC

PC

PC

ARCHITECTURE “TESTING”

1.0.0.		comments	comments	1.1.0.	2.0.0.	2.1.0.
Sensor		2 объявления класса	_json.get('timeout', 30)		Sensor	
ActiveSensor	Update() Check_status()				Update() Check_status()	
PassiveSensor					Check_rule() Fill_message()	
	id type status ('status', 'offline')					status('status', 'unknown')
ActiveSensor						ActiveSensor
	Kettle Router				Kettle Router	Fridge
PassiveSensor						
Config						
	Sensor	2 объявления класса				
Server						
F1. Listener	Incoming_from_Active()				Incoming_from_Active()	
F5. Listener	Incoming_from_Active()				Json_config	
	Status (O/I) Set_Status() Set_offline()	return self._json.get('status', 'offline')			Data (from cache)	
F6. Notification()	Incoming_from_Active()				Make_Notification()	
	Json_config Data Make_Notification()					
F4. InOut_Sensors	Show_Params()					
F3. Sys_Params	Change_Params()					
	Text					
F8. Kettle_Incoming	Incoming_from_Kettle()				Incoming_from_Kettle()	
					Data	
F10. Router_Incoming	Incoming_from_Router()				Incoming_from_Router()	
					Data	

Bug report #1

Поле	Комментарий
Короткое описание (Summary)	Двойное объявление класса Sensor
Проект (Project)	Сервер датчиков в квартире
Компонент приложения (Component)	В фалах sensor.py и config.py
Номер версии (Version)	1.0.0
Серьезность (Severity)	S1 Блокирующий (Blocker) S2 Критический (Critical) S3 Значительный (Major) S4 Незначительный (Minor) S5 Тривиальный (Trivial)
Приоритет дефекта (Priority)	P1 Высокий (High) P2 Средний (Medium) P3 Низкий (Low)
Статус (Status)	Новый баг
Назначен на (Assigned To)	Менеджер, ответственный за разработку компонента
ОС / Сервис Пак и т.д. / Браузера + версия / ...	Python 3.6, PyCharm IDE
Шаги воспроизведения (Steps to Reproduce)	Открыть код sensor.py – 4 строка кода: class Sensor: config.py – 5 строка кода: class Sensor:
Фактический Результат (Result)	Обнаружили 2 объявления одного и того же класса – были крайне удивлены
Ожидаемый результат (Expected Result)	Ожидал что каждый класс будет объявляться 1 раз
Прикрепленный файл (Attachment)	Скриншоты приведены в приложении

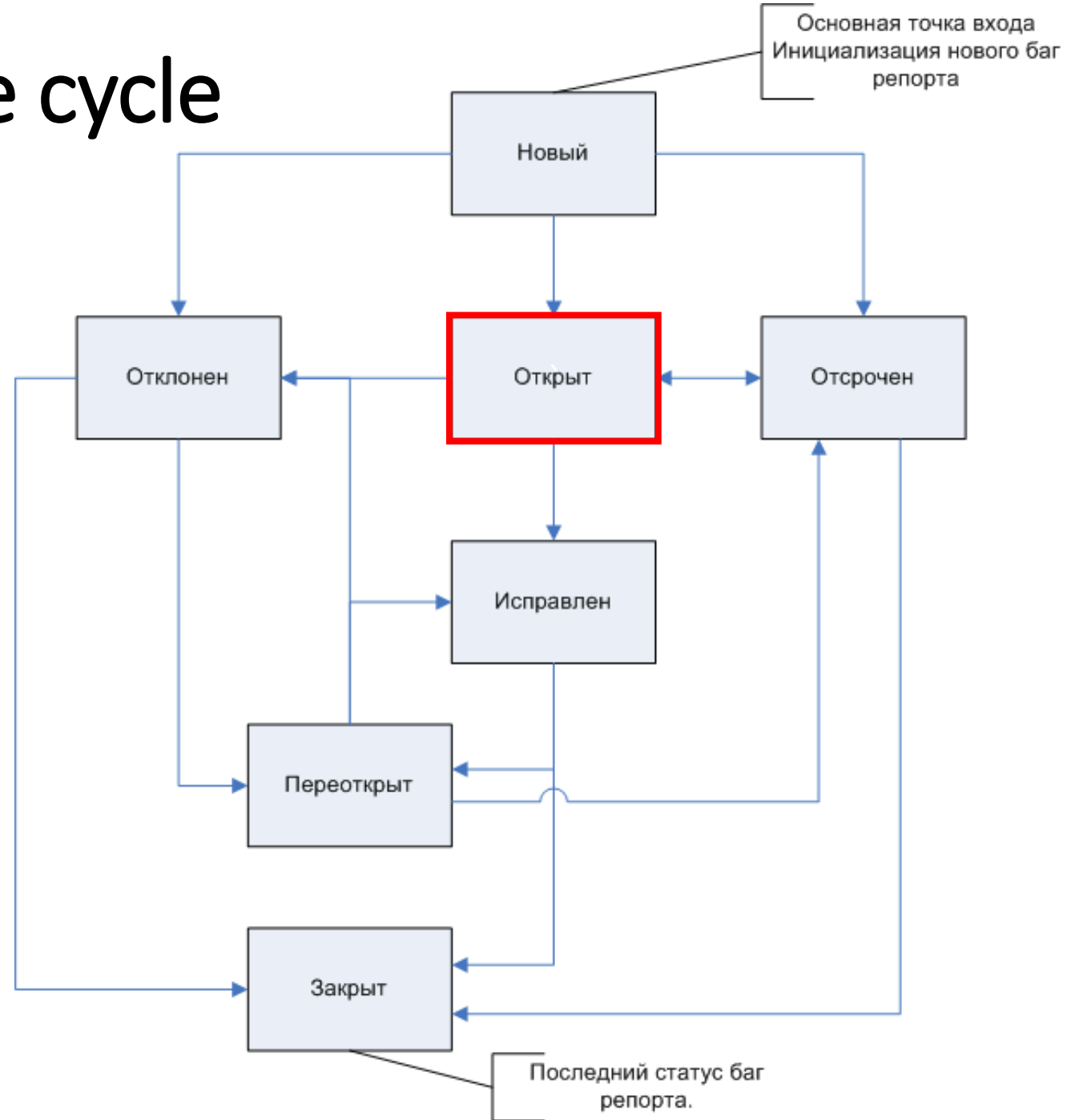
Bug report #2

Поле	Комментарий
Короткое описание (Summary)	Изменение статуса сенсора с 'offline' на 'unknown'
Проект (Project)	Сервер датчиков в квартире
Компонент приложения (Component)	В фалах sensor.py
Номер версии (Version)	2.1.0
Серьезность (Severity)	S1 Блокирующий (Blocker) S2 Критический (Critical) S3 Значительный (Major) S4 Незначительный (Minor) S5 Тривиальный (Trivial)
Приоритет дефекта (Priority)	P1 Высокий (High) P2 Средний (Medium) P3 Низкий (Low)
Статус (Status)	Новый баг – или фича, нужно выяснить
Назначен на (Assigned To)	Менеджер, ответственный за разработку компонента
ОС / Сервис Пак и т.д. / Браузера + версия / ...	Python 3.6, PyCharm IDE
Шаги воспроизведения (Steps to Reproduce)	Открыть код sensor.py – 41 строка кода: return self._json.get('status', 'unknown')
Фактический Результат (Result)	Нужно провести тестирование какой вариант правильный и поправить во всех версиях
Ожидаемый результат (Expected Result)	Отсутствие расхождений в версиях без комментариев об изменениях
Прикрепленный файл (Attachment)	Скриншоты приведены в приложении

Bug report #3

Поле	Комментарий
Короткое описание (Summary)	Снижение производительности модуля - sensor_emulator
Проект (Project)	Сервер датчиков в квартире
Компонент приложения (Component)	В работе emulator'a
Номер версии (Version)	После 2.0.0
Серьезность (Severity)	S1 Блокирующий (Blocker) S2 Критический (Critical) S3 Значительный (Major) S4 Незначительный (Minor) S5 Тривиальный (Trivial)
Приоритет дефекта (Priority)	P1 Высокий (High) P2 Средний (Medium) P3 Низкий (Low)
Статус (Status)	Новый баг
Назначен на (Assigned To)	Менеджер, ответственный за разработку компонента
ОС / Сервис Пак и т.д. / Браузера + версия / ...	Python 3.6, PyCharm IDE
Шаги воспроизведения (Steps to Reproduce)	Открыть код Запустить - sensor_emulator
Фактический Результат (Result)	Непонятная задержка в выполнении, м.б. слишком долго опрашиваем сенсоры
Ожидаемый результат (Expected Result)	Ожидал быстрой работы программы, т.к. сенсоры в квартире – их там немного
Прикрепленный файл (Attachment)	Скриншоты приведены в приложении

Bug workflow and life cycle



FINAL -> RE-CODE

► In [1]: `from datetime import datetime`

```
class Sensor:
```

```
    def __init__(self, p_sensor_json):
        self._json = p_sensor_json
        self._last_update = datetime.now()
```

```
    def update(self, p_sensor):
        self._last_update = datetime.now()
        self._json['status'] = p_sensor.status
```

```
    def check_status(self, p_timeout):
        i_timedelta = (datetime.now() - self._last_update)
        i_since_last = (i_timedelta.microseconds + (i_timedelta.seconds + i_timedelta.days * 24 * 3600) * 10 ** 6) / 10 ** 6
        if i_since_last > p_timeout:
            self._json['status'] = 'offline'
```

```
    @property
    def id(self):
        return self._json.get('id')
```

```
    @property
    def type(self):
        return self._json.get('type')
```

```
    @property
    def status(self):
        return self._json.get('status', 'offline')
```

```
    @property
    def address(self):
        return self._json.get('address', str(('localhost', -1)))
```

```
    @address.setter
    def address(self, p_address):
```

ПРИЛОЖЕНИЯ

Bug report #1 - скриншоты

```
sensor.py x config.py x
1 from datetime import datetime
2
3
4 class Sensor:
5
6     def __init__(self, p_sensor_json):
7         self._json = p_sensor_json
8         self._last_update = datetime.now()
9
10    def update(self, p_sensor):
11        self._last_update = datetime.now()
12        self._json['status'] = p_sensor.status
13
14    def check_status(self, p_timeout):
15        i_timedelta = (datetime.now() - self._last_update)
16        i_since_last = (i_timedelta.microseconds + (i_timedelta.seconds + i_timedelta.days * 24 * 3600) * 10 ** 6) / 10 ** 6
17        if i_since_last > p_timeout:
18            self._json['status'] = 'offline'
19
```

```
sensor.py x config.py x
1 import os
2 import yaml
3
4
5 class Sensor:
6
7     def __init__(self, p_sensor_json):
8         self._json = p_sensor_json
9
10    @property
11    def type(self):
12        return self._json.get('type')
13
```


Модульное тестирование (unit testing)

Единичное тестирование, или **модульное тестирование** (англ. **unit testing**) — процесс в программировании, позволяющий проверить на корректность единицы исходного кода, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Интеграционное тестирование

Интеграционное тестирование (англ. Integration testing, I&T) — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности.

Системное тестирование

Системное тестирование программного обеспечения — это тестирование программного обеспечения (ПО), 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.

Альфа-тестирование и **бета-тестирование** являются подкатегориями системного тестирования.