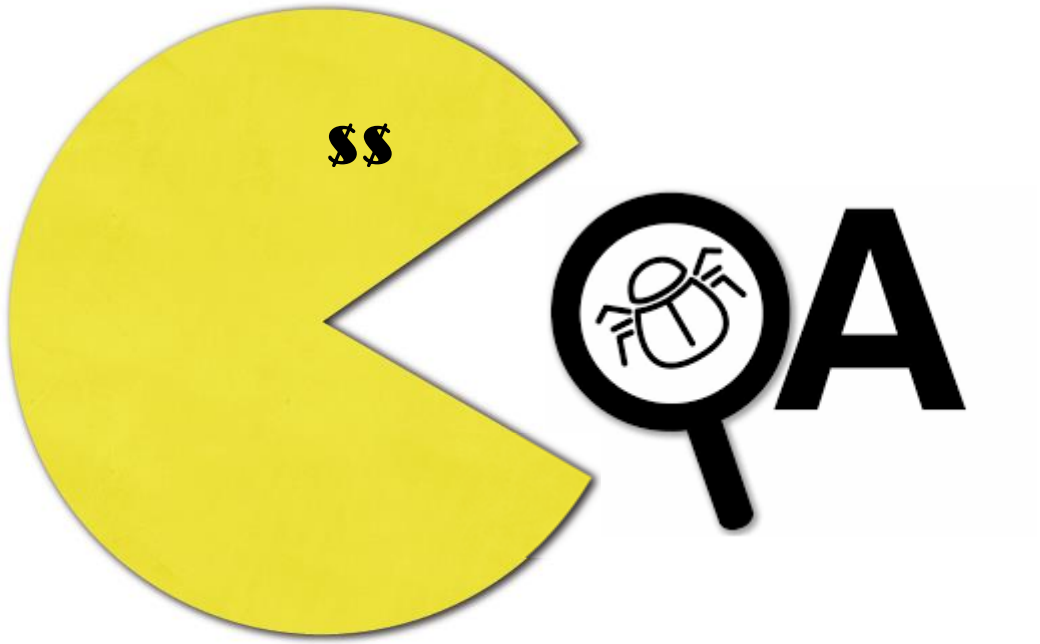
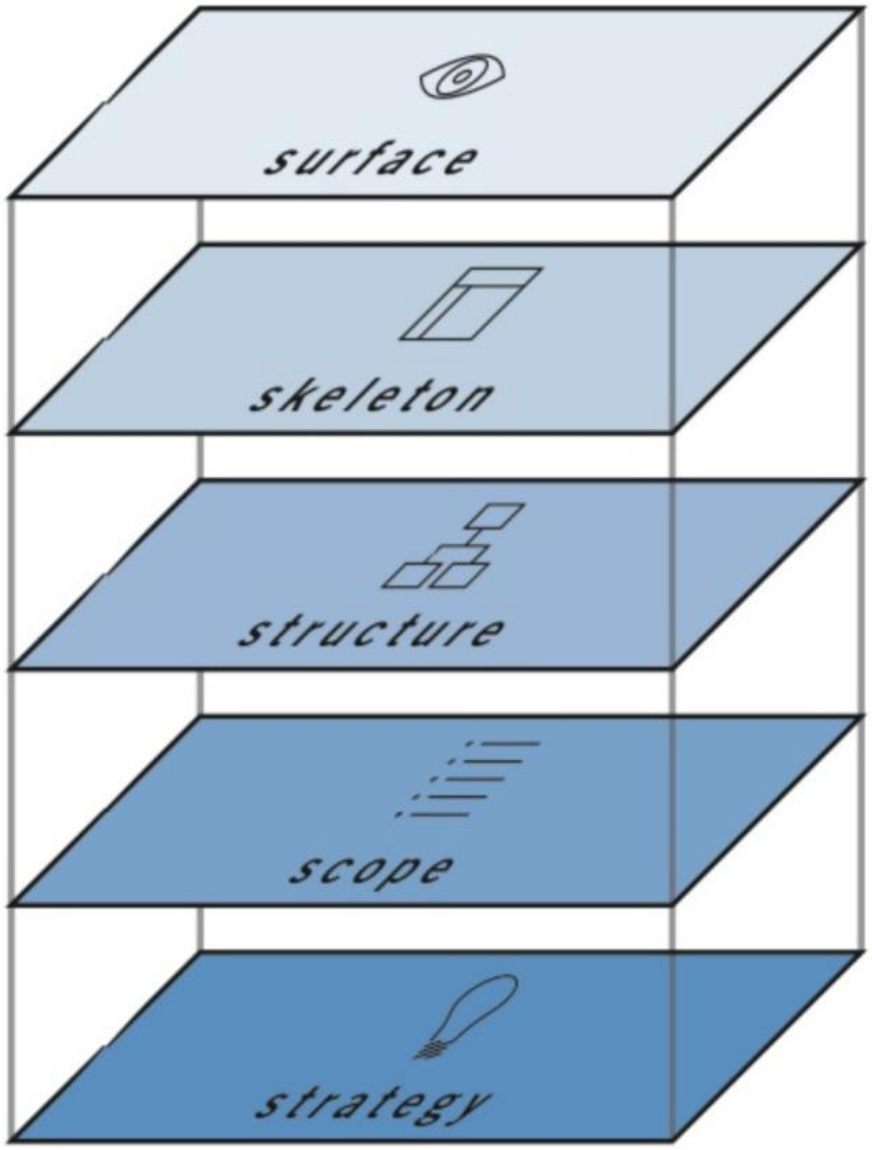
nexign



Percentage of chart that looks like PacMan

actum

Jesse James Garrett

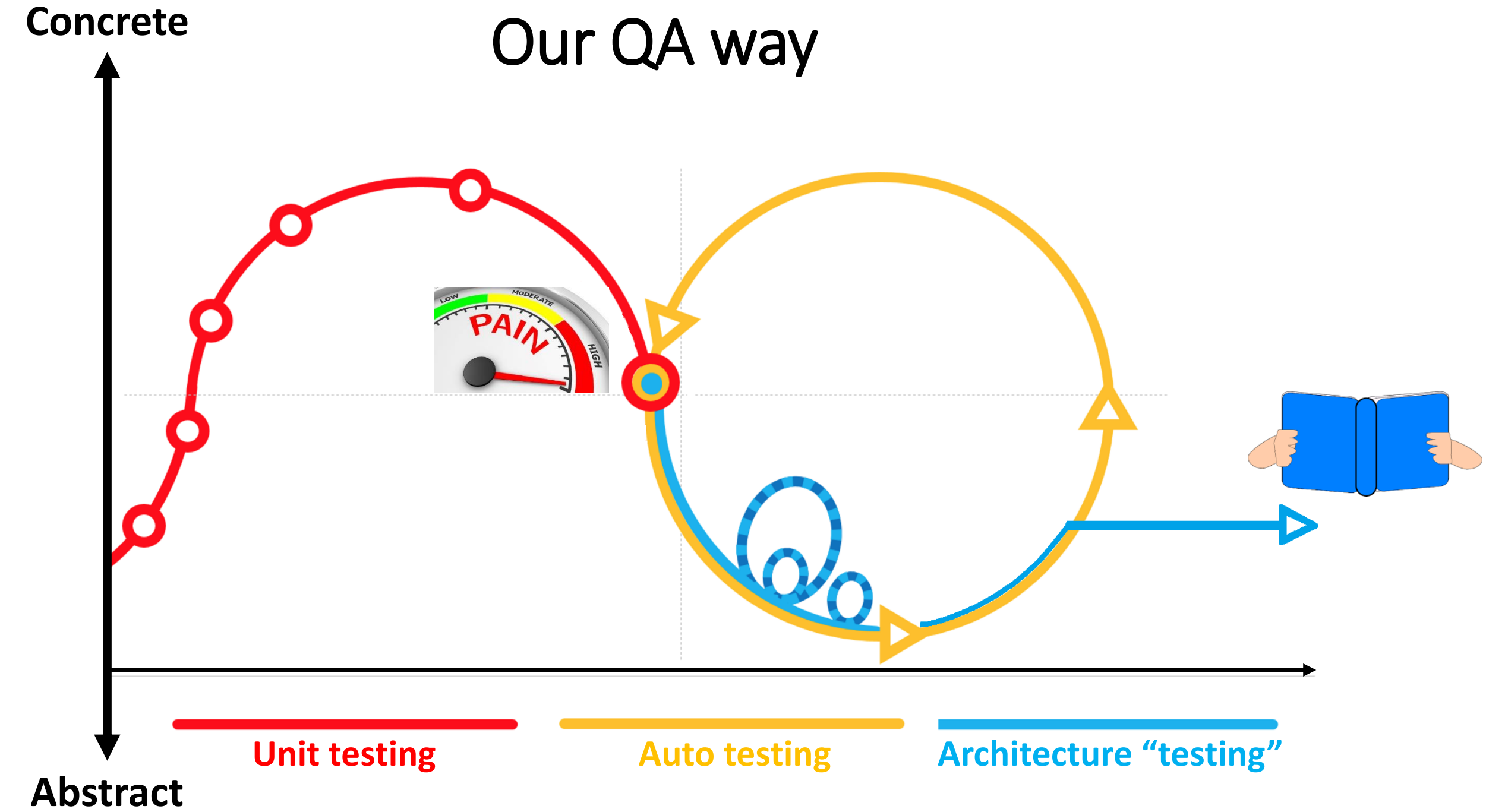


Concrete

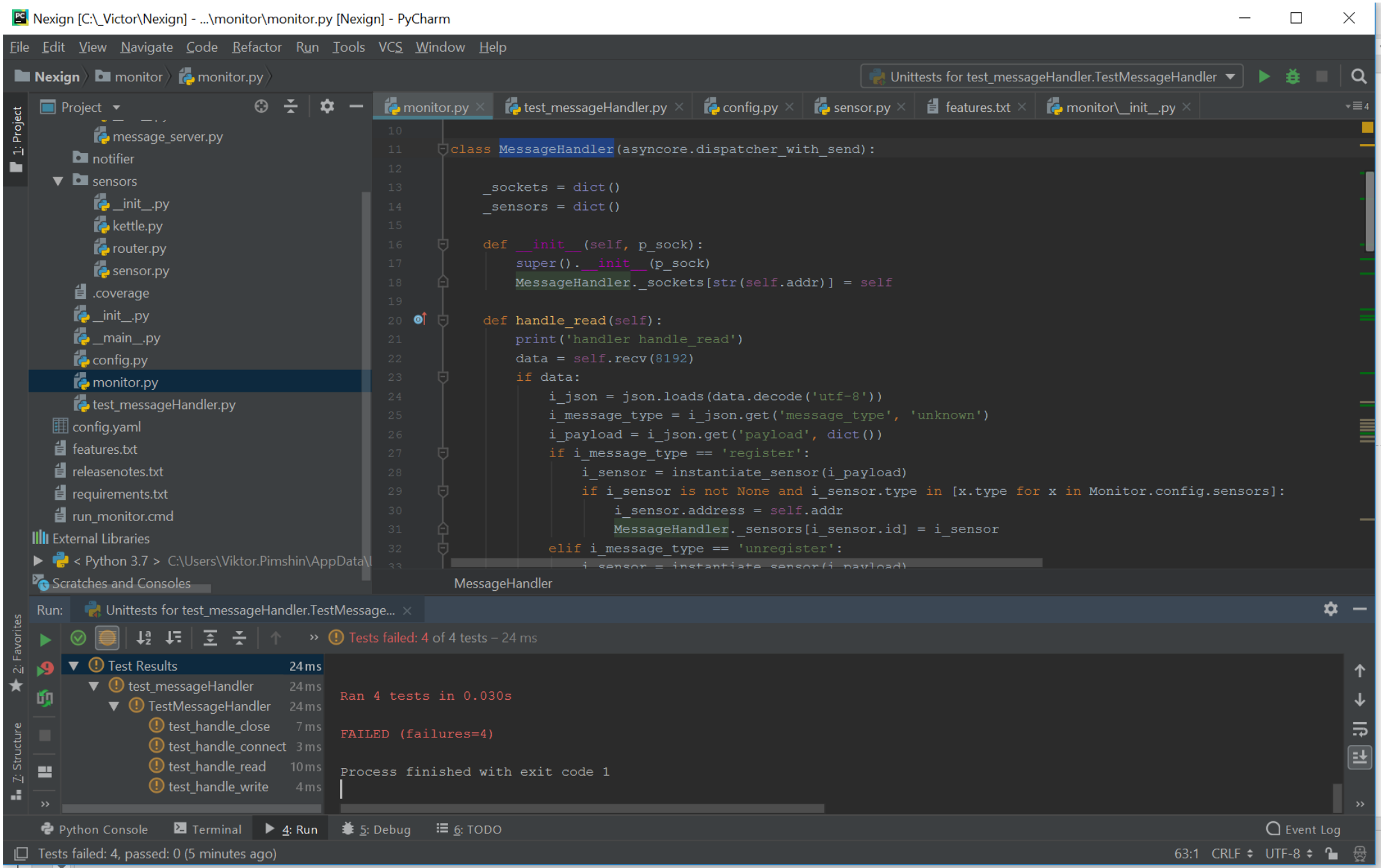
Abstract

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

Our QA way


Concrete
LOW
MODERATE
PAIN
HIGH
Abstract
Unit testing
Auto testing
Architecture “testing”

UNIT TESTING



AUTO TESTING

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

12 statements | 5 run | 7 missing | 0 excluded

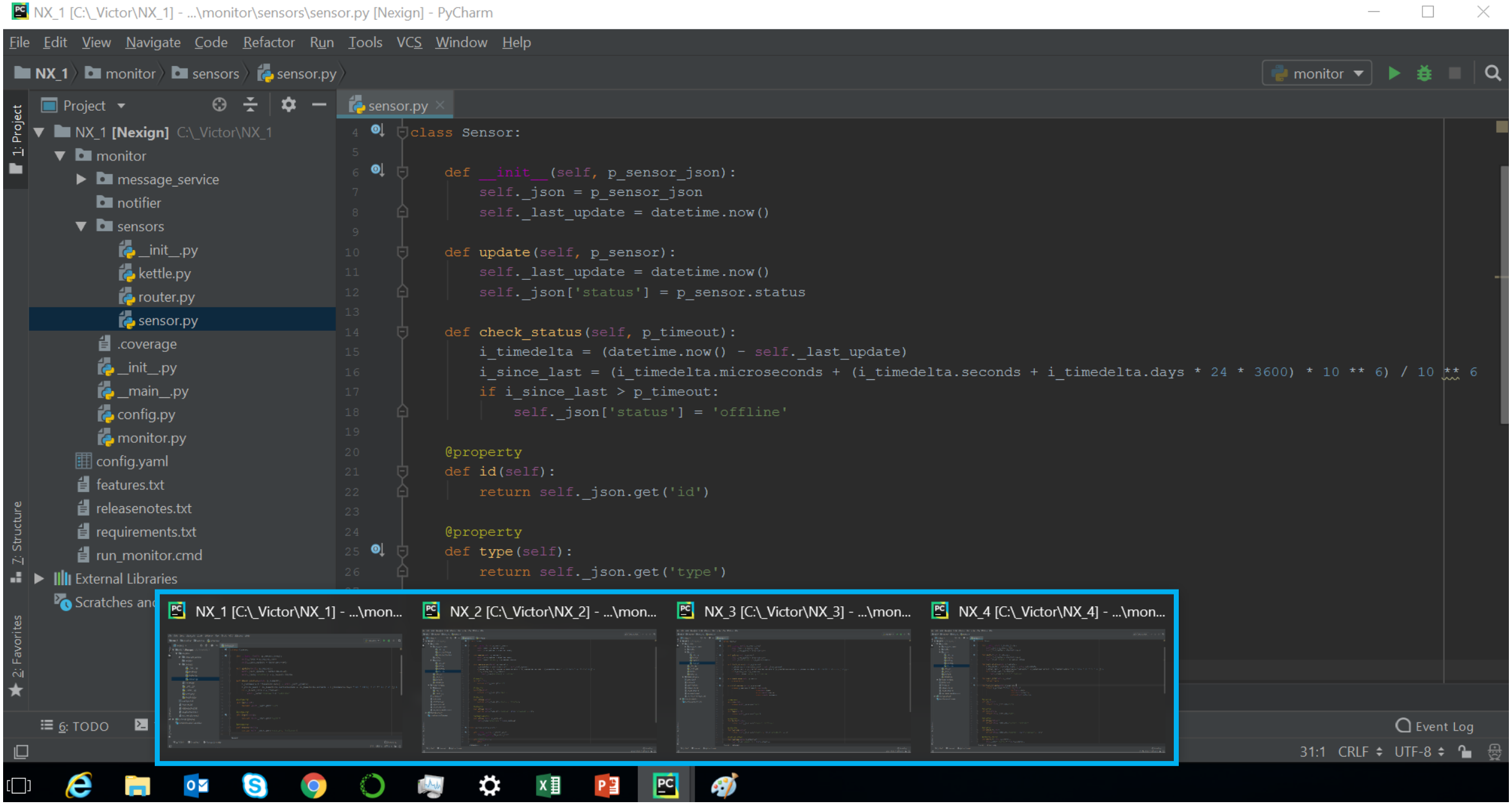
```python
# -*- coding: utf-8 -*-

def sum(num1, num2):
    return num1 + num2


def sum_only_positive(num1, num2):
    if num1 > 0 and num2 > 0:
        return num1 + num2
    else:
        return None


def test_sum():
    assert sum(5, 5) == 10


def test_sum_positive_ok():
    assert sum_only_positive(2, 2) == 4


def test_sum_positive_fail():
    assert sum_only_positive(-1, 2) is None


```

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”



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()				Update()	
	Check_status()				Check_status()	
PassiveSensor					Check_rule()	
					Fill_message()	
	id					
	type					
	status ('status', 'offline')					status('status', 'unknown')
ActiveSensor						ActiveSensor
	Kettle					Kettle
	Router					Router
						Fridge
PassiveSensor						
Config						
	Sensor	2 объявления класса				
Server						
F1. Listener	Incoming_from_Active()					
F5. Listener	Incoming_from_Active()					
	Status (O/I)					
	Set_Status()	return self._json.get('status', 'offline')				
	Set_offline()					
F6. Notification()	Incoming_from_Active()				Incoming_from_Active()	
	Json_config				Json_config	
	Data				Data (from cache)	
	Make_Notification()				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

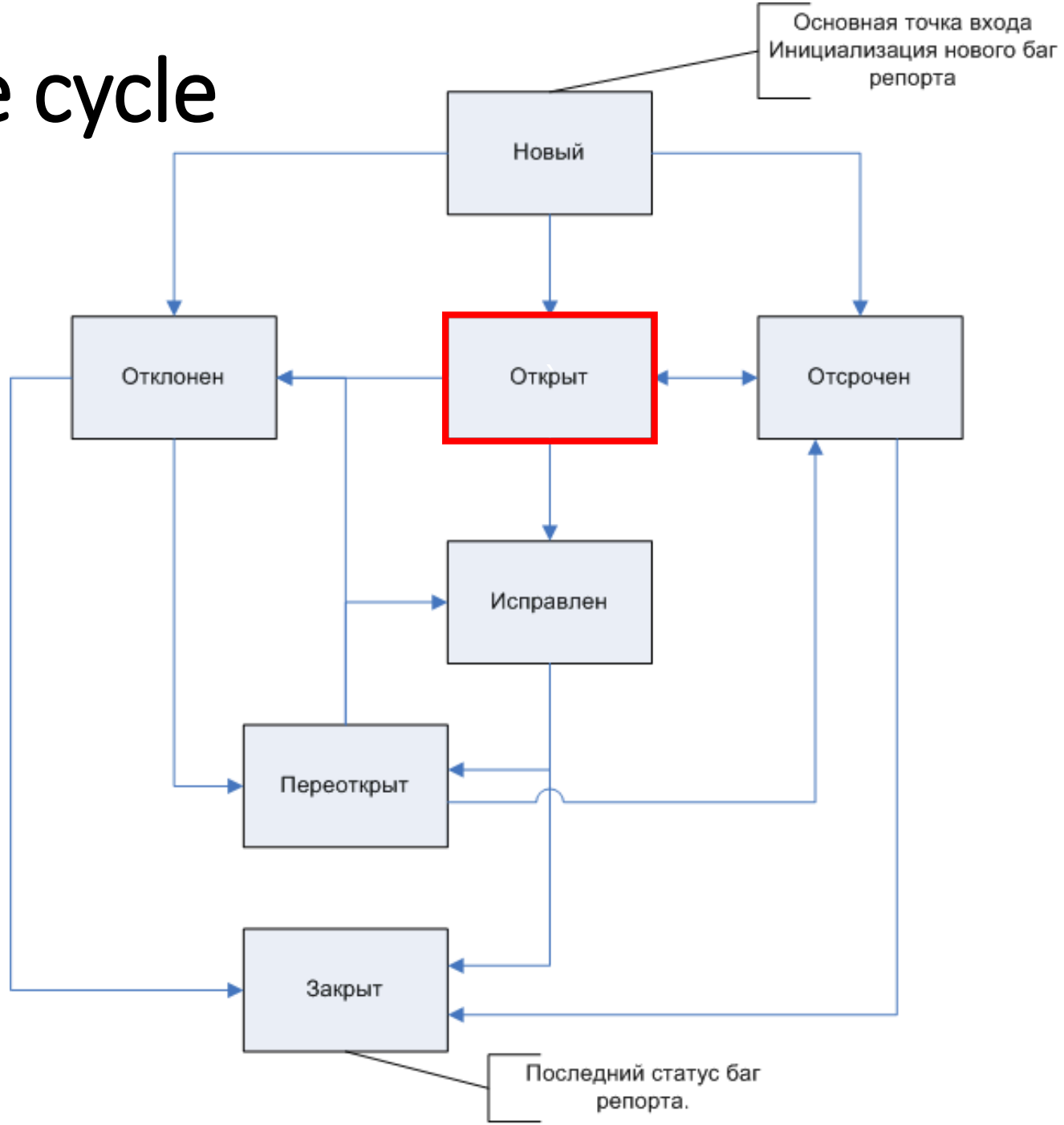
Bug report #2

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 фитча, 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
```

ПРИЛОЖЕНИЯ

Bug report #1 - скриншоты

sensor.py | config.py

```python
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'

```

sensor.py | config.py

```python
import os
import yaml


class Sensor:

    def __init__(self, p_sensor_json):
        self._json = p_sensor_json

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

```

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.