Improving tests quality and automatic REST API documentation validation

Ivan A. Perl

ORACLE

2016 CEE-SEC(R) Software Engineering Conference in Russia

Agenda

- Что мы разрабатываем?
- Как тестировать???
- А хороши ли тесты?
  - И что с этим делать :)
  - TDD и BDD
- Автоматическая генерация, что и зачем

ORACLE

Что мы разрабатываем?

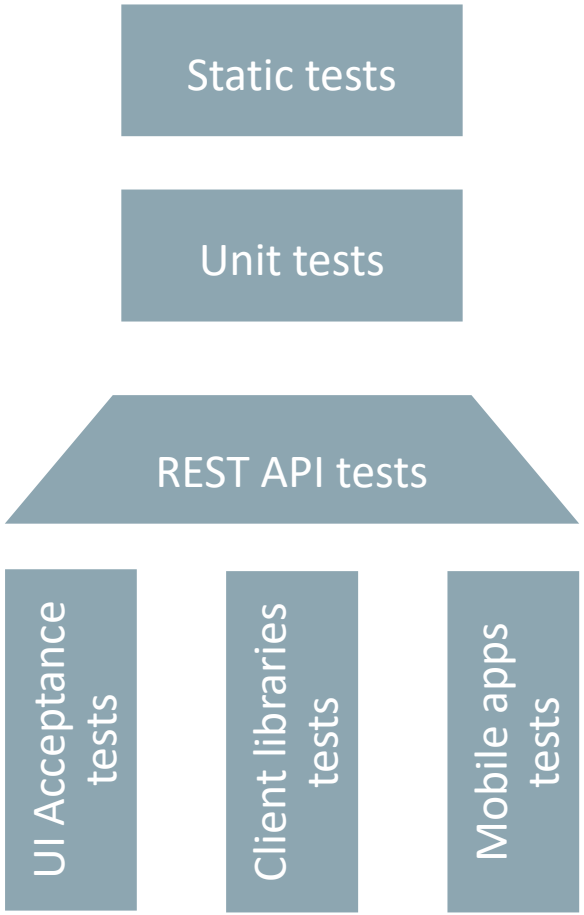
- Oracle Internet of Things

Что мы разрабатываем?

- Oracle Internet of Things
  - Несколько крупных независимых компонентов
  - Около 100 различных REST API вызовов
    - Используемые в вызовах модели имеют обязательные и опциональные поля
  - Различные конфигурации развёртывания

ORACLE

Как тестировать???



- Статический анализ кода: Checkstyle and Findbugs
- Тестирование на уровне классов: JUnit/TestNG, Mockito, PowerMock, jsTestDriver
- Тестирование REST API: JUnit/TestNG, Mockito, PowerMock
- Тестирование для REST API: JUnit/TestNG + platform specific automation tools

ORACLE

А хороши ли тесты? :)

- Статистика по тестам:

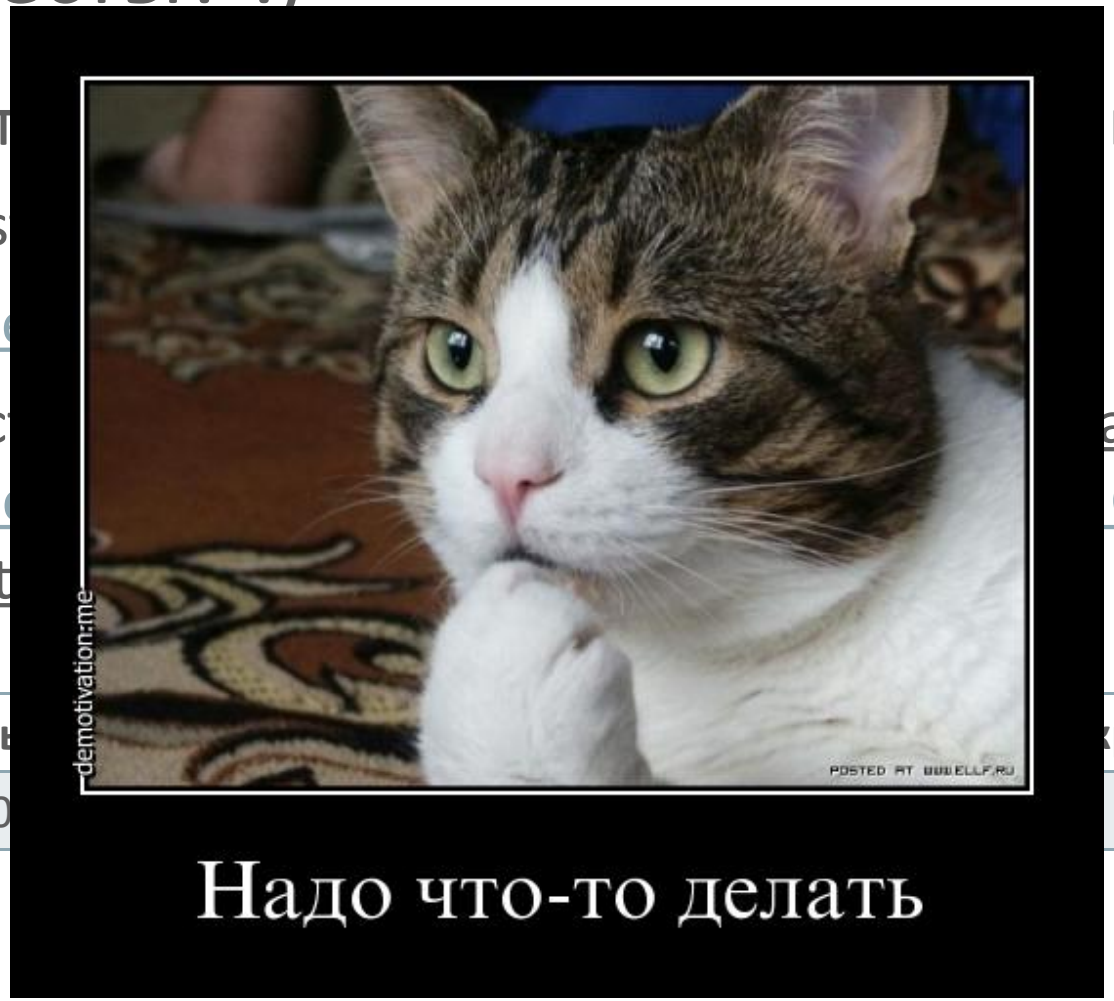
Уровень	Тесты	Покрытие
Unit tests	~8000	76% (jacoco)
Integration (REST API)	~1500	79% (API вызовов)
System tests (with native client library)	~450	15% (API вызовов)
Total:	**~10450**	

ORACLE

А хороши ли тесты? :)

- Проверим качест нием
  - Инструмент: Pites
  - Ссылка: http://pit
  - Набор мутаций (c ator - Increments Mutator - Invert N Conditionals Mutator - Return Values Mut

Уровень	Тесты	крытие хорошими тестами
Unit tests	~800	51% (jacoco)



Немного о «мутационном тестировании»

- Обычное тестирование:
  - Тестируем «хороший» код тестами, чтобы убедиться, что код работает как надо
- Мутационное тестирование:
  - У нас есть «хороший» код и «зелёные» тесты. Мы ломаем код и смотрим, чтобы тесты выявили внесённые ошибки
- Типичные мутации:

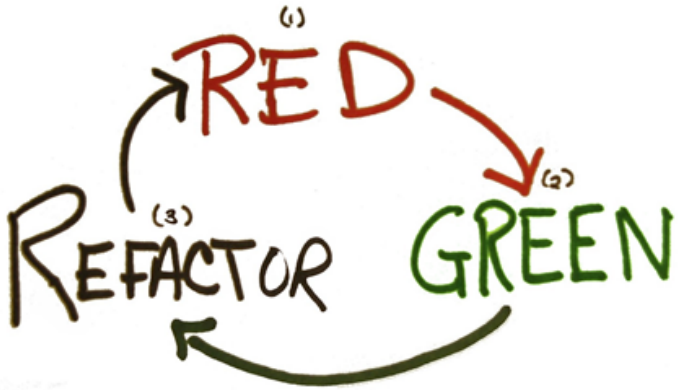
Conditionals Boundary Mutator	**Math Mutator**	**Inline Constant Mutator**
Negate Conditionals Mutator	**Increments Mutator**	**Return Values Mutator**
Remove Conditionals Mutator	**Invert Negatives Mutator**	**Void Method Call Mutator**

ORACLE

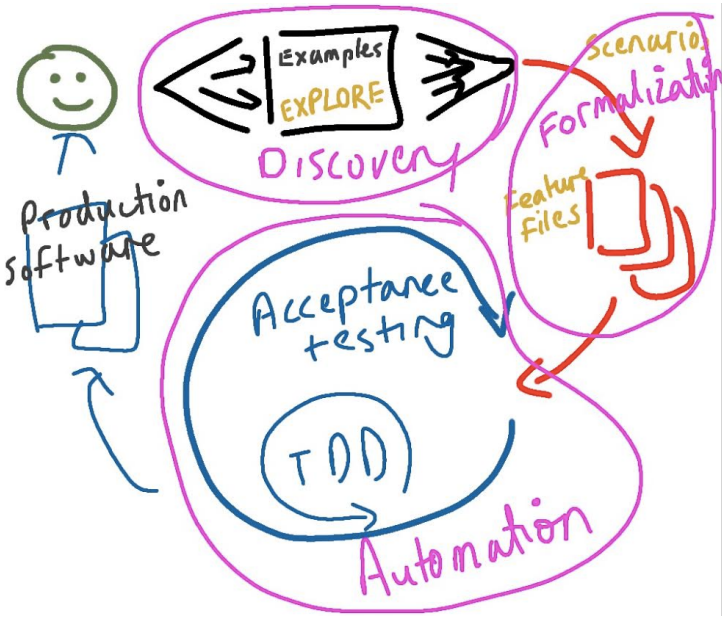
«Мутационное» тестирование в API тестах

- Зачем нужно?
  - Чтобы проверить насколько дырявые тесты
- Основные проблемы
  - Мутации очень просто приводят код в совершенно нерабочее состояние
  - Запускать такие тесты очень долго и дорого
- «Решения»
  - Очень точный выбор классов, которые будут подвержены мутациям
  - Использование ограниченного набора мутаций

ORACLE

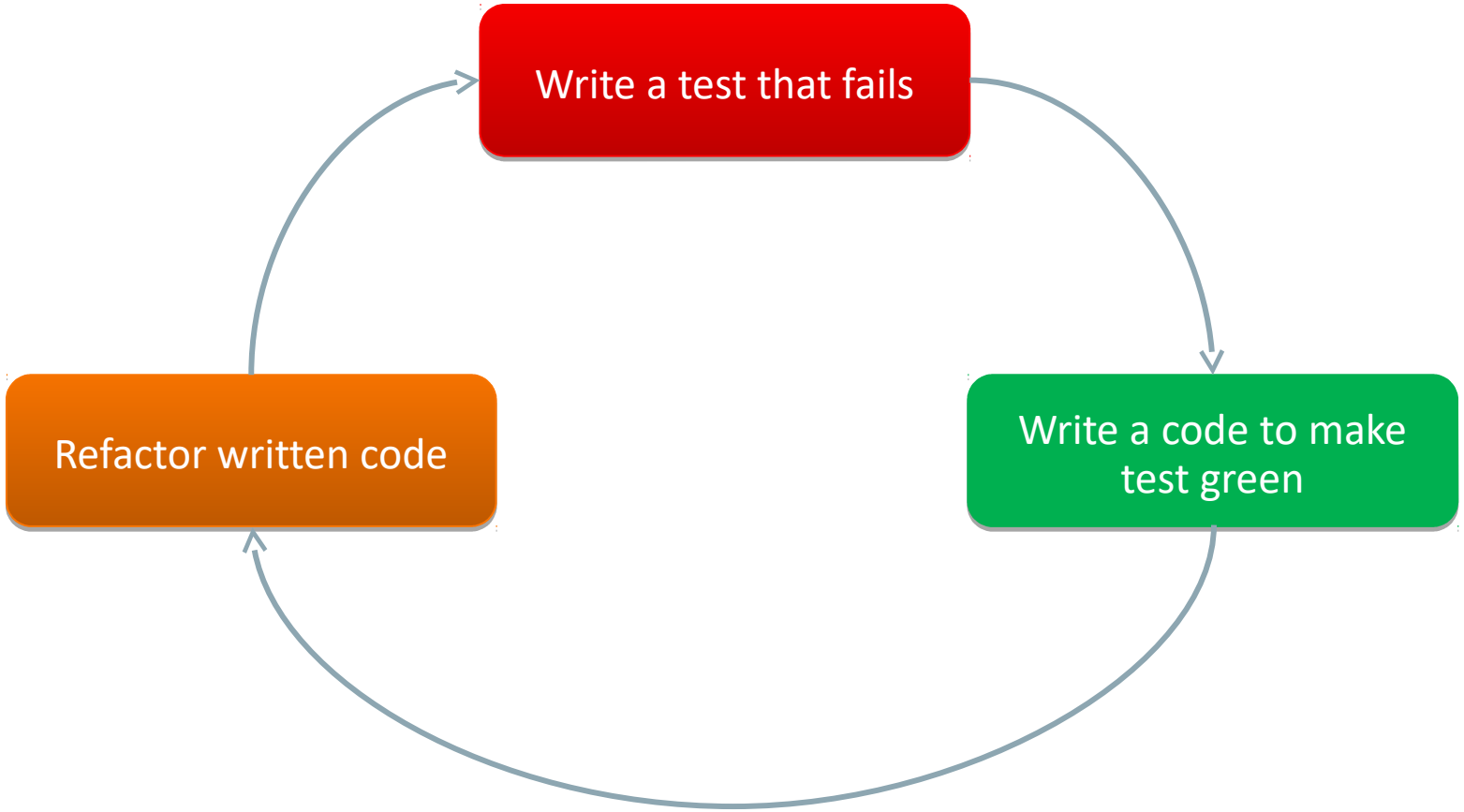

(1)
RED
(2)
GREEN
(3)
REFACTOR

TDD & BDD


Examples
EXPLORE
Discovery
Scenario
Formalization
Feature Files
Production software
Acceptance testing
TDD
Automation

TDD

- *“Red. Green. Refactor.”* – главная мантра разработки по TDD



TDD

- Тест разрабатывается для одной небольшой несуществующей функции. Запускается и «падает»

- Функция реализуется – После перезапуска тест проходит.

- Анализируем написанный код. Можно ли его улучшить?
  - Вся ли функциональность реализована?
  - Можно ли реализацию сделать проще?

Если ответ нет, то добавляем тесты и проводим рефакторинг.

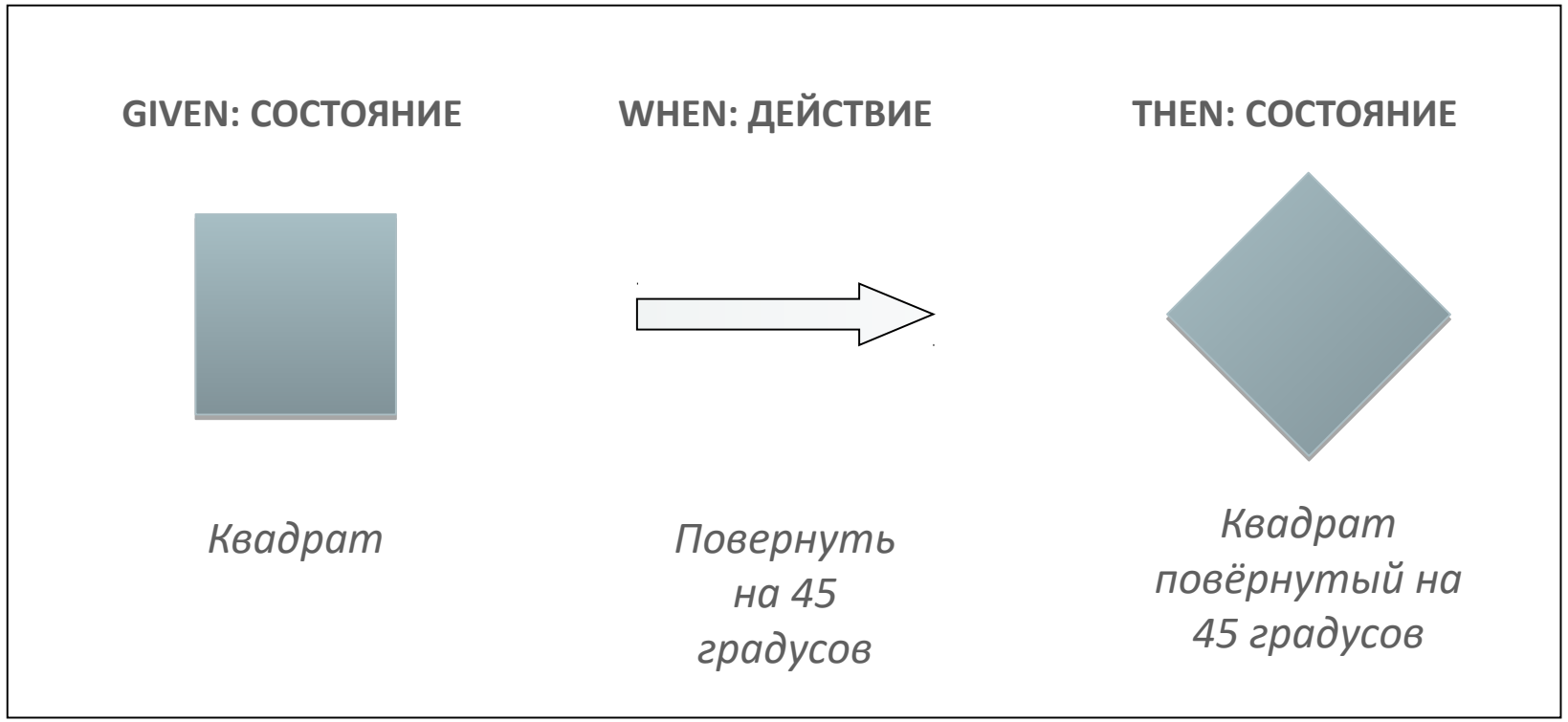
Что такое BDD?

- Формально:
  - BDD = **B**ehavior **D**riven **D**evelopment
- По смыслу:
  - BDD = Выполнение операция по смыслу, а не как простые вызовы функций
- **Behavior Driven Development:**
  - Это процесс разработки, возникший из test-driven development (TDD). BDD принципы и практики TDD с идеями доменно-ориентированной архитектуры объектно-ориентированного анализа.

ORACLE

Что такое BDD? cont.

- Основная идея – код должен описывать переход объекта из состояния в состояние:



BDD и TDD

- BDD подход применим на всех уровнях тестирования

Unit tests

- Given – настройка mock объектов
- When – вызов кода тестирование которого производится
- Then – проверка результатов вызова

Integration tests

- Given – настройка окружения
- When – вызов кода тестирование которого производится
- Then – проверяем результаты вызовов

Acceptance tests

- Given – переход на нужный экран в интерфейсе
- When – производим необходимые действия
- Then – проверяем результаты действий

ORACLE

BDD и TDD, пример теста

- Возьмём обычный тест:

```
@Test
public void testEnrollment_BAD_REQUEST_AppAttributes_AppVersion_Empty() {
  AppEnrollmentRequest request = createAppRequestWithAttributes(appName());
  request.setVersion("");
  ClientResponse clientResponse = appEnrollment(request);
  assertFailureHttpStatusOnly(clientResponse, ClientResponse.Status.BAD_REQUEST);
}
```

- И перепишем его в соответствии с идеями BDD:

```
@Test
public void testEnrollmentShouldReturnBadRequetWhenAppVersionIsEmpty() {
    givenEnrollmentRequest();        //This will create a request instance in class scope
    givenRequestAppVersionIsEmpty(); //Setting request app version to ""
    ClientResponse clientResponse = whenEnrolmentRequestSent(); //Performing method call
    thenResponseHasStatusBadRequest(clientResponse);            //Validating result
}
```

ORACLE

Что мы на-BDD-TDD-ли

- Эффективное покрытие кода тестами, так как пишется только код, которые проверяется тестами
  - + не пишется лишний код (зацепки вида «вдруг пригодится в будущем»)
  - Более безопасный рефакторинг, что важно при гибкой разработке и меняющихся требованиях
- Более читаемые тесты в которых очень просто разбираться
- Более читаемые stacktrace’ы
- Улучшилась архитектура
- Выросло качество требований

ORACLE

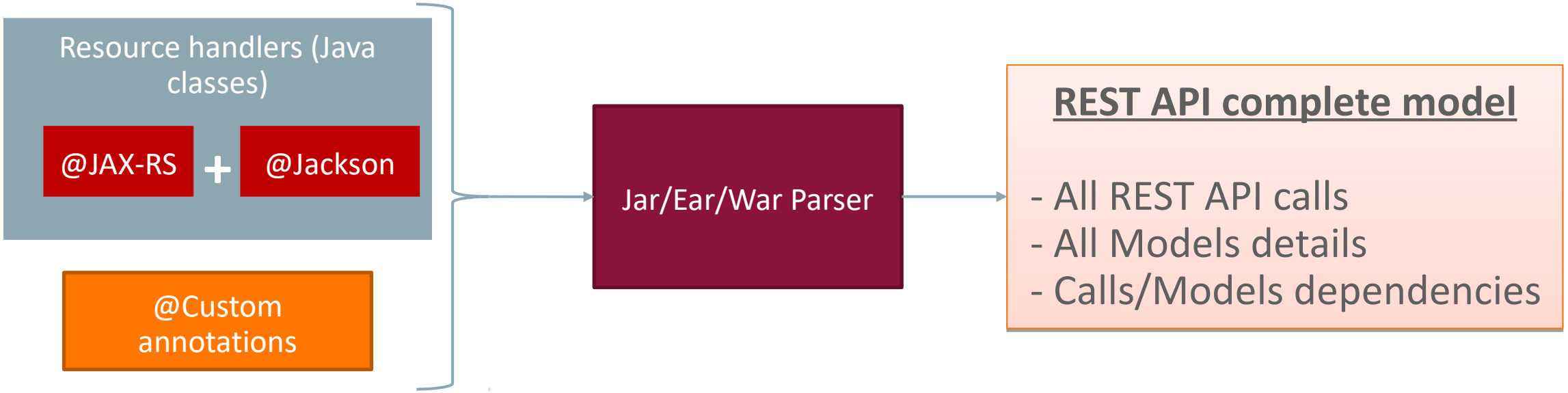
Автоматическая генерация, что и зачем

- В проекте более 100 REST API вызовов
- Модели, которые используются для вызовов имеют обязательные и опциональные поля
- Проблема №1 – документация
  - Она всегда устаревшая :)
- Проблема №2 – точность реализации API
  - На правильных данных все запросы работают
  - Все обязательные поля правда обязательные (а все опциональные опциональные)
  - Проверка значений полей по шаблонам

ORACLE

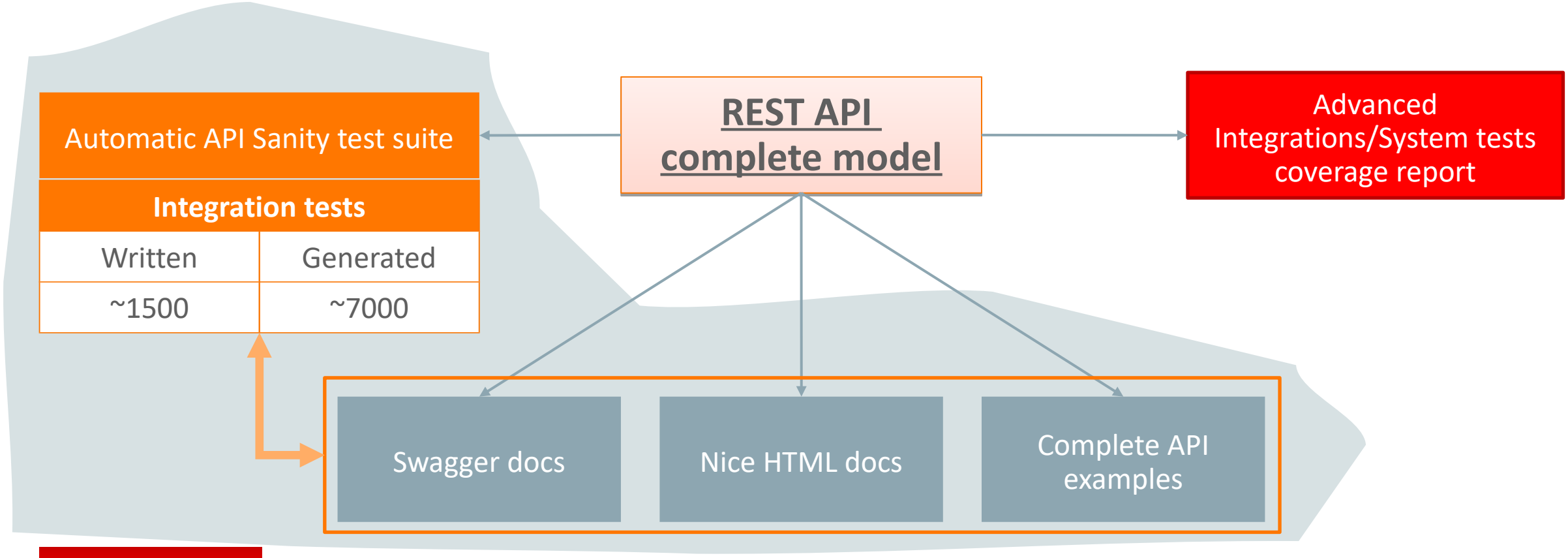
Автоматическая генерация, что и зачем

- Решение



Автоматическая генерация, что и зачем

- Что получили



Что мы нагенерировали

- Документация для REST API в разных форматах для разных целей
  - Так как по реальному собранному продукту строится модель документации, то сериализовать её можно в любой дополнительный формат очень быстро
- Sanity тесты, проверяющие базовые детали работы REST API
  - Проверка обязательности\необязательности полей моделей
  - Проверка соответствия полей шаблонам в запросах и ответах
  - Проверка корректности ошибок возвращаемых сервером
- Автоматическая проверка того, что API работает именно так как оно описано в документации

ORACLE

Спасибо за внимание!

Вопросы?