АВТОМАТИЧЕСКИЕ ТЕСТЫ

ТЕОРИЯ И ПРИМЕРЫ

Идеальная пирамида тестирования

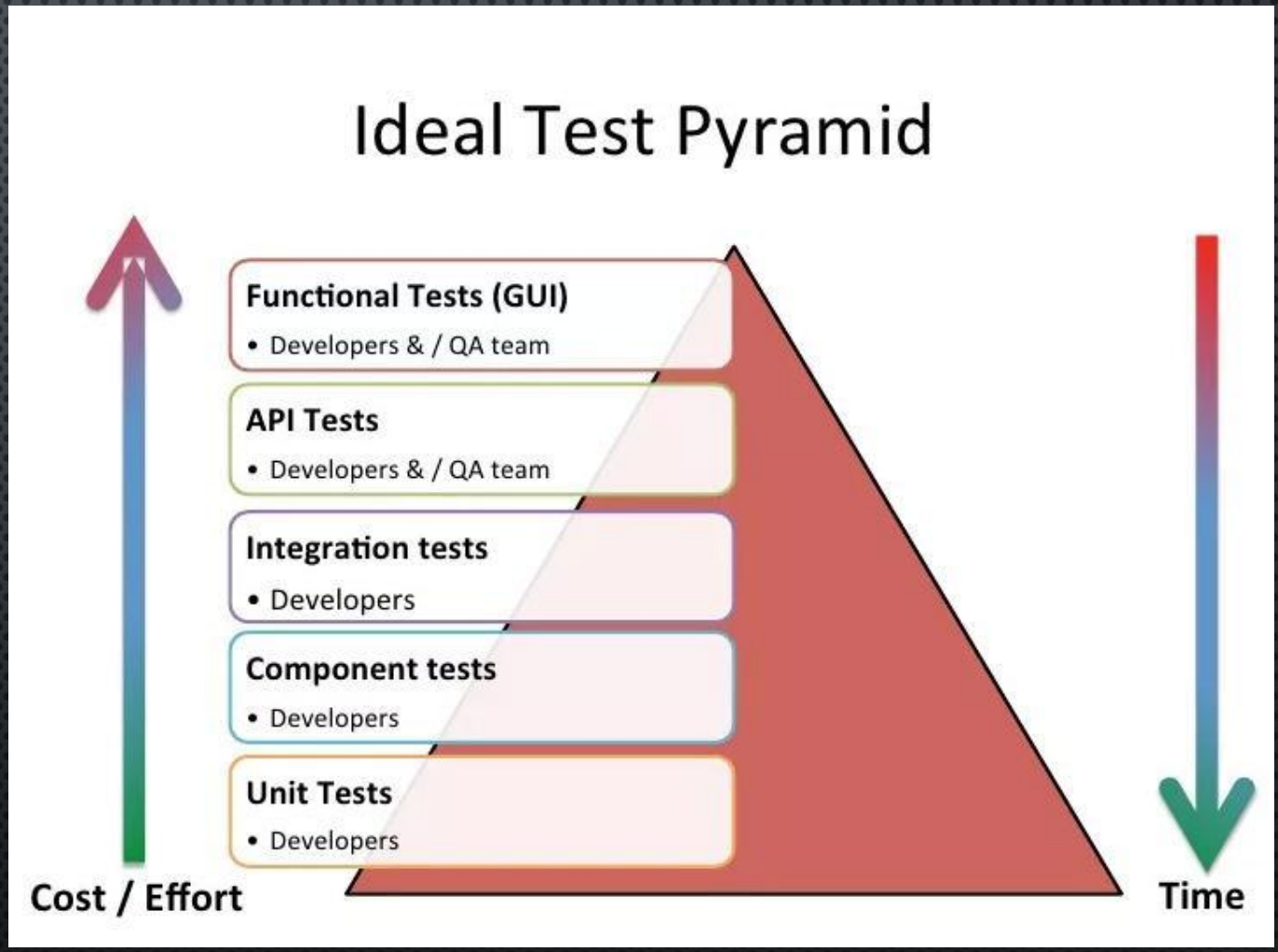
Основные труды на эту тему:

«Забытый слой пирамиды автоматических тестов» (Майк Кон, 2009)

«Пирамида тестов» (Мартин Фаулер, 2012)

Суть:

- Основная и самая бюджетная часть автоматических тестов приложения – Unit тесты.
- На разработку Unit теста затрачивается наименьшее время.
- Unit тесты – самые быстрые.
- Самые дорогие и ресурсоемкие – функциональные тесты.



МОДЕЛЬ АВТОМАТИЧЕСКОГО ТЕСТИРОВАНИЯ ПРИЛОЖЕНИЯ – ОТКРЫТИЯ ТЕМА.

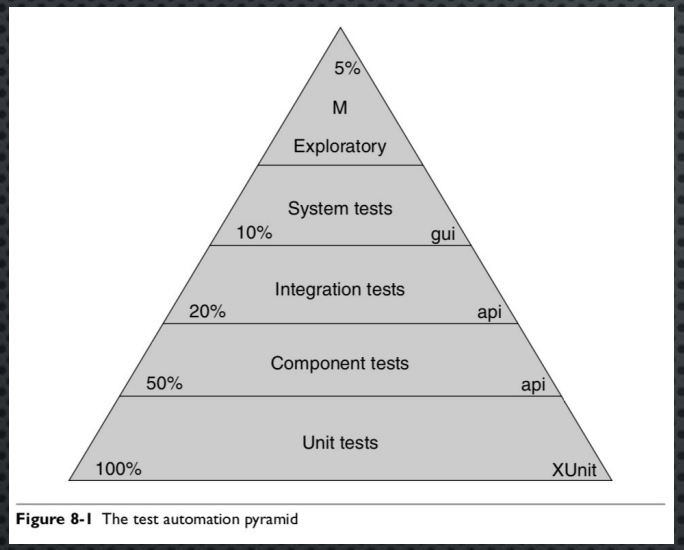
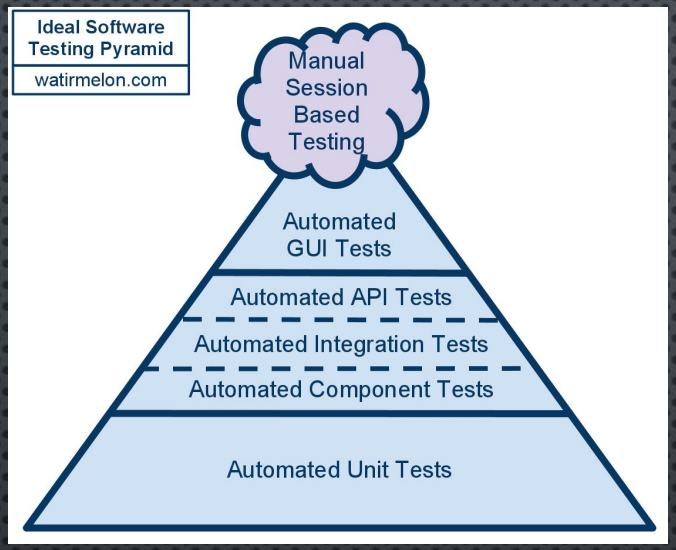


Figure 8-1 The test automation pyramid



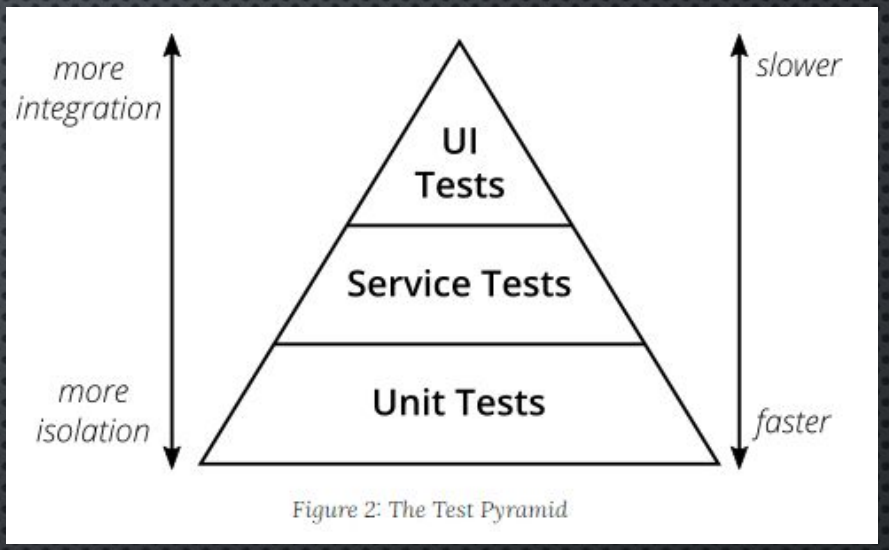
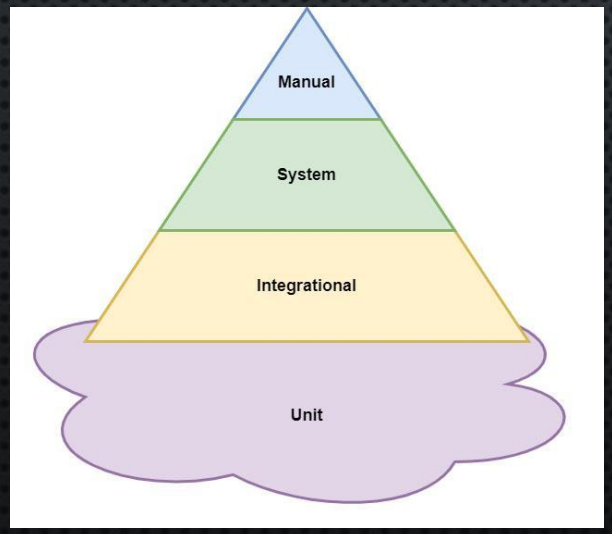
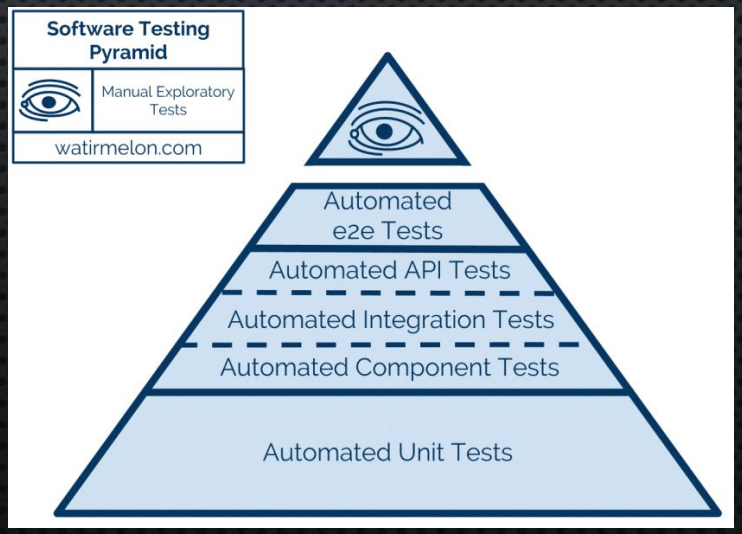
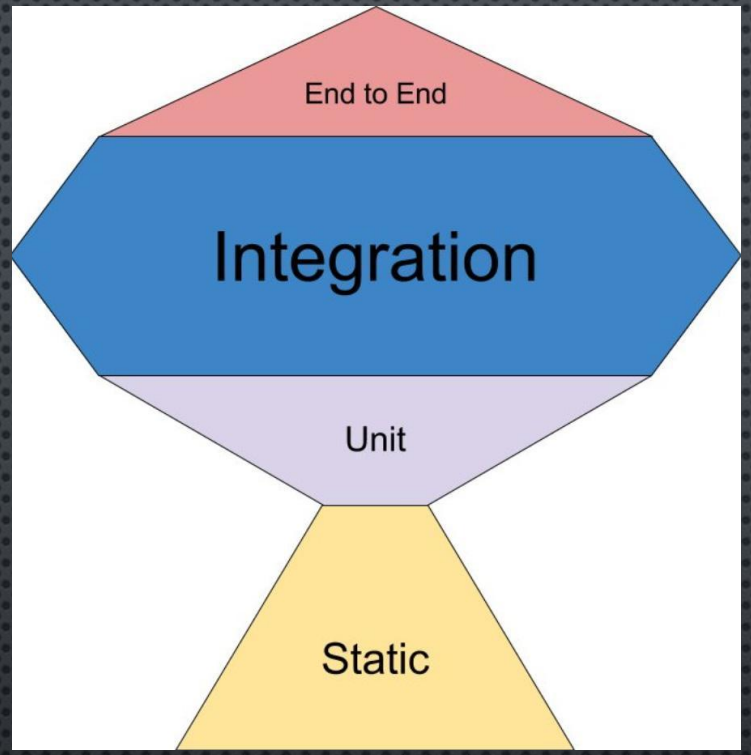


Figure 2: The Test Pyramid







Антипаттерн идеальной пирамиды

https://kentcdodds.com/blog/write-tests

Kent C. Dodds предложил иную схему.

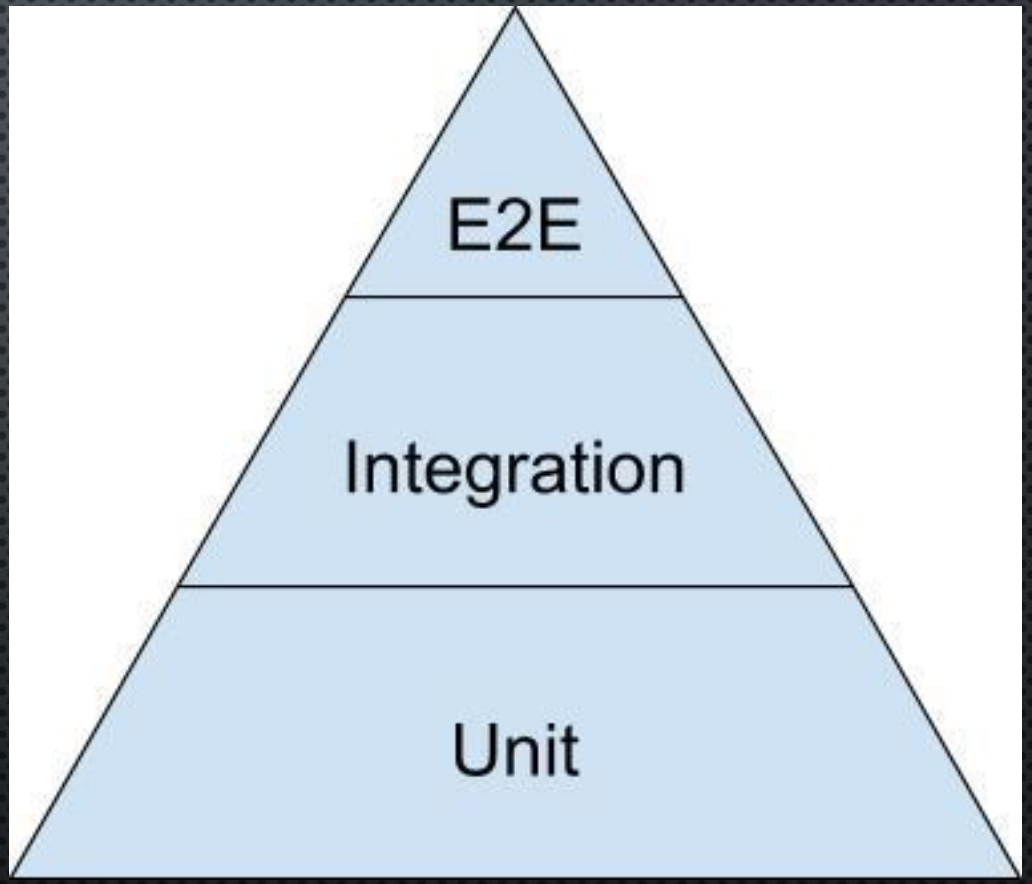
- В современных веб-приложениях большое значение имеют именно интеграционные тесты.
- Вся структура автотестов держится на статической (строгой) типизации. Статическая типизация в PHP поддерживается с 7.x.x версий (declare(strict_types=1);). Статическая типизация — когда тип переменной точно известен.

Упрощенная пирамида тестирования

- **Unit tests (модульные тесты)** – тесты, которые проверяю, что отдельные изолированные части кода работают должным образом. Тестируется один класс/метод/функция (или наименьшая рабочая единица для этой конкретной функциональности), а всё остальное имитируется/заменяется.
- **Integration tests** – тесты, фокусируемые на целом компоненте. Также именуемые сервисными тестами или даже компонентными тестами. Проверяют, что несколько сервисов работают вместе в гармонии.

  Это может быть набор классов/методов/функций, модуль, подсистема или даже само приложение.

  Если в тесте используется **БД**, сеть для вызова другого компонента/приложения, внешняя система (например, очередь или почтовый сервер), читает/записывает файлы, то это интеграционный тест, а не модульный тест.
- **E2E (end to end) Tests** – тесты, которые часто называются функциональными, системными или **User UI tests**. Как правило, эти тесты реализует робот-помощник, который ведет себя как пользователь, щелкая по приложению и проверяя его правильность работы.



TDD И BDD ПОДХОДЫ

```php
/**
 * A basic test example.
 *
 * @return void
 */
public function testBasicTest()
{
    $param = '+7 (999) 888-77-66';

    $result = Helper::convertPhoneNumber($param);

    $this->assertEquals($result, '9998887766');
}
```

```
Feature: ls
  In order to see the directory structure
  As a UNIX user
  I need to be able to list the current directory's contents

  Scenario:
    Given I am in a directory "test"
    And I have a file named "foo"
    And I have a file named "bar"
    When I run "ls"
    Then I should get:
      """
      bar
      foo
      """
```

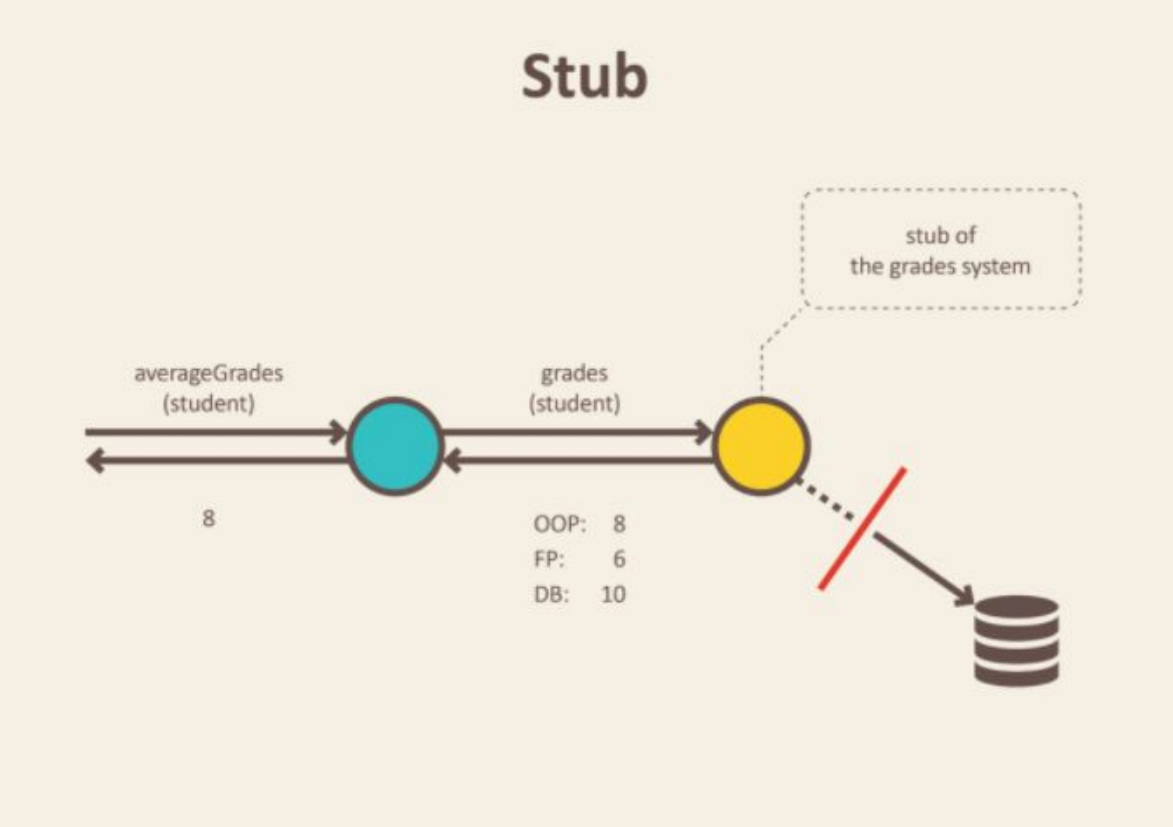
TDD (Test Driven Development) - Разработка на основе тестов.

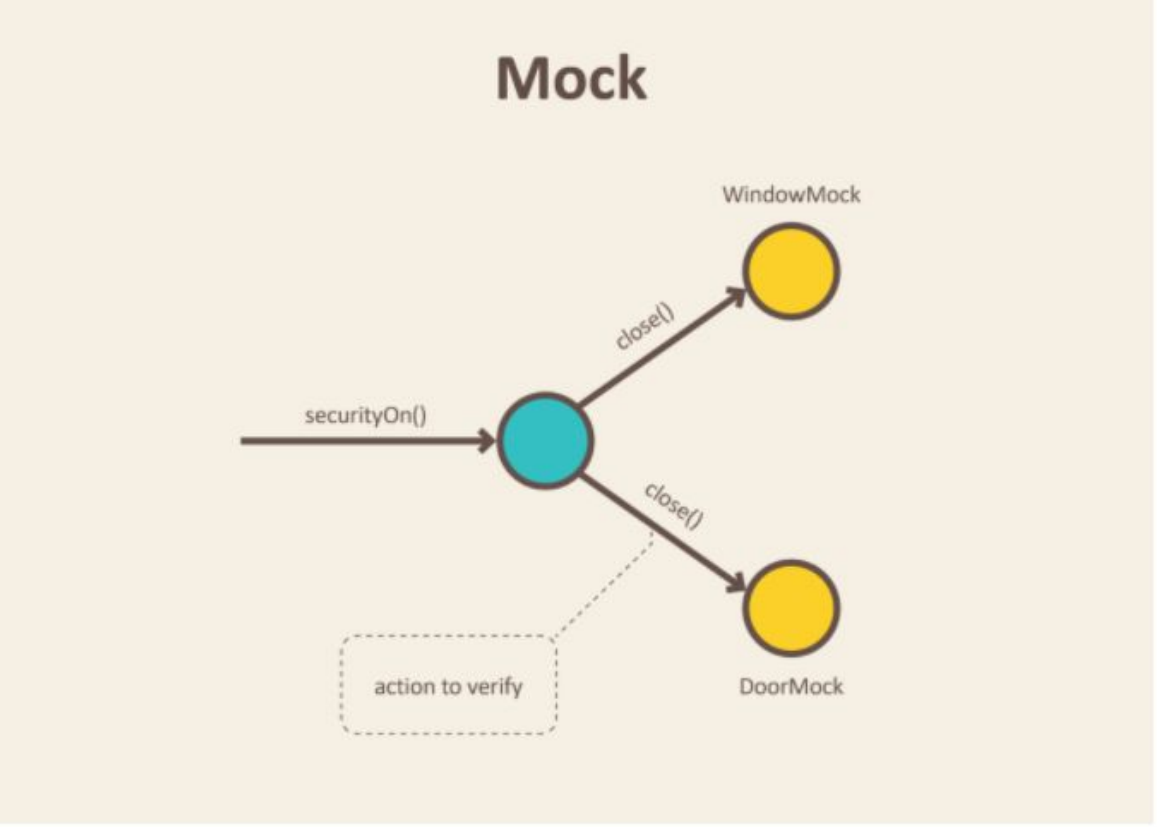
- юнит-тестирование,
- тесты сразу реализуются в коде
- юнит-тесты пишут сами разработчики.
- проверяет работу функций.

BDD (Test Driven Development) - Разработка на основе поведения.

- интеграционное и E2E тестирование,
- описываются шаги на языке, понятном всем, а не только разработчикам.
- требует объедения усилий разных членов команды.
- Тест-кейсы воплощаются в код тестировщиком-автоматизатором.
- Проверяет пользовательские сценарии.

MOCKS И STUBS





ПРАКТИЧЕСКИЙ ПРИМЕР

ТЕГ	Описание
1.0.0	Начальная форма подтверждения номера телефона
1.1.0	Разработан unit тест на преобразование номера телефона к виду 9999999999 Видео: https://www.loom.com/share/6ff55e3de66e42ef855044d216674ed2
1.2.0	Разработан интеграционный тест с внешним сервисом по передаче номера телефона. API метод: https://qa.dev.sebekon.ru/api/crm/handle Видео: https://www.loom.com/share/de24f37167b845fbb538934b5a76a2ba
1.3.0	Разработан функциональный тест по работе с формой подтверждения номера телефона. Видео: https://www.loom.com/share/0454019ba27e4d5bb99da1b2809818d0
1.4.0	Внешний сервис CRM перестал быть доступным.
1.5.0	Неработоспособность формы, которая не покрывается ни unit тестами, ни интеграционными тестами.
1.6.0	Верстальщик изменил правило маски телефона. Интеграционный тест перестал успешно отрабатывать. Показать перезапуск юнит-теста при изменении регулярного выражения.

КРАТКИЕ СВЕДЕНИЯ ПО АВТОТЕСТАМ

	UNIT TESTS	INTEGRATION TESTS	E2E
Преимущества	• Скорость работы • Скорость разработки, • Повторные запуски тестов для разработки. • Изолированность системы • Заменяет документацию и комментарии • Быстрый поиск багов	• Незаменимы при микросервисной архитектуре	• Более надежное покрытие функционала, в том числе и user interface.
Недостатки	• Применение Mocks и Stubs • Не покрывают общую функциональность сервисов	• Требуют особой настройки архитектуры взаимодействия сервисов (тестовые контуры)	• Низкая скорость работы, • Более длительный поиск багов
Инструменты, PHP	PHPUnit, Codeception	PHPUnit, Codeception	Codeception, Selenium, Dusk

ОСОБЕННОСТИ ВНЕДРЕНИЯ АВТОМАТИЧЕСКИХ ТЕСТОВ В БИТРИКС

Структура автотестов для bitrix:
https://git.sebekon.ru/sebekon-repository/backend/bitrix/bitrix-tests

Проблема: компонентная архитектура приложения. Сложно внедрять unit и интеграционные тесты.

Решение:

- Тестировать часть методов классов компонентов без метода executeComponent(
- Покрывать unit и интеграционными тестами все кастомные классы приложения.
- Покрытие E2E тестами.

```php
/** @global CIntranetToolbar $INTRANET_TOOLBAR */
global $INTRANET_TOOLBAR;

use Bitrix\Main\Context,
    Bitrix\Main\Type\DateTime,
    Bitrix\Main\Loader,
    Bitrix\Iblock;

CPageOption::SetOptionString( module_id: "main",  name: "nav_page_in_session",  value: "N");

if(!isset($arParams["CACHE_TIME"]))
    $arParams["CACHE_TIME"] = 36000000;

$arParams["IBLOCK_TYPE"] = trim($arParams["IBLOCK_TYPE"]);
if(strlen($arParams["IBLOCK_TYPE"])<=0)
    $arParams["IBLOCK_TYPE"] = "news";
$arParams["IBLOCK_ID"] = trim($arParams["IBLOCK_ID"]);
$arParams["PARENT_SECTION"] = intval($arParams["PARENT_SECTION"]);
$arParams["INCLUDE_SUBSECTIONS"] = $arParams["INCLUDE_SUBSECTIONS"]!="N";
$arParams["SET_LAST_MODIFIED"] = $arParams["SET_LAST_MODIFIED"]==="Y";

$arParams["SORT_BY1"] = trim($arParams["SORT_BY1"]);
if(strlen($arParams["SORT_BY1"])<=0)
    $arParams["SORT_BY1"] = "ACTIVE_FROM";
if(!preg_match( pattern: '/^(asc|desc|nulls)(,asc|,desc|,nulls){0,1}$/i', $arParams["SORT_ORDER1"]))
    $arParams["SORT_ORDER1"]="DESC";

if(strlen($arParams["SORT_BY2"])<=0)
{
    if (strtoupper($arParams["SORT_BY1"]) == 'SORT')
    {
        $arParams["SORT_BY2"] = "ID";
        $arParams["SORT_ORDER2"] = "DESC";
    }
    else
```

РЕЗЮМЕ

- Увидели, что написание теста зачастую не увеличивает время разработки, а наоборот, сокращает его.
- Мы практически смогли поймать баги уже на этапе запуска тестов, а не после выгрузки на боевой стенд.
- Важно не только создавать тесты, но и поддерживать их, как и документацию проекта.
- Нужно овладеть основными методиками написания тестов, чтобы они покрывали максимум всех возможных состояний системы.
- Необходимо добавить в регламент разработки обязательность написания тестов для всех кастомных классов вне компонентов и модулей Битрикс.
- CI/CD, Pipelines, автоматический запуск тестов.