Тестирование Frontend

Виды тестирования по уровням

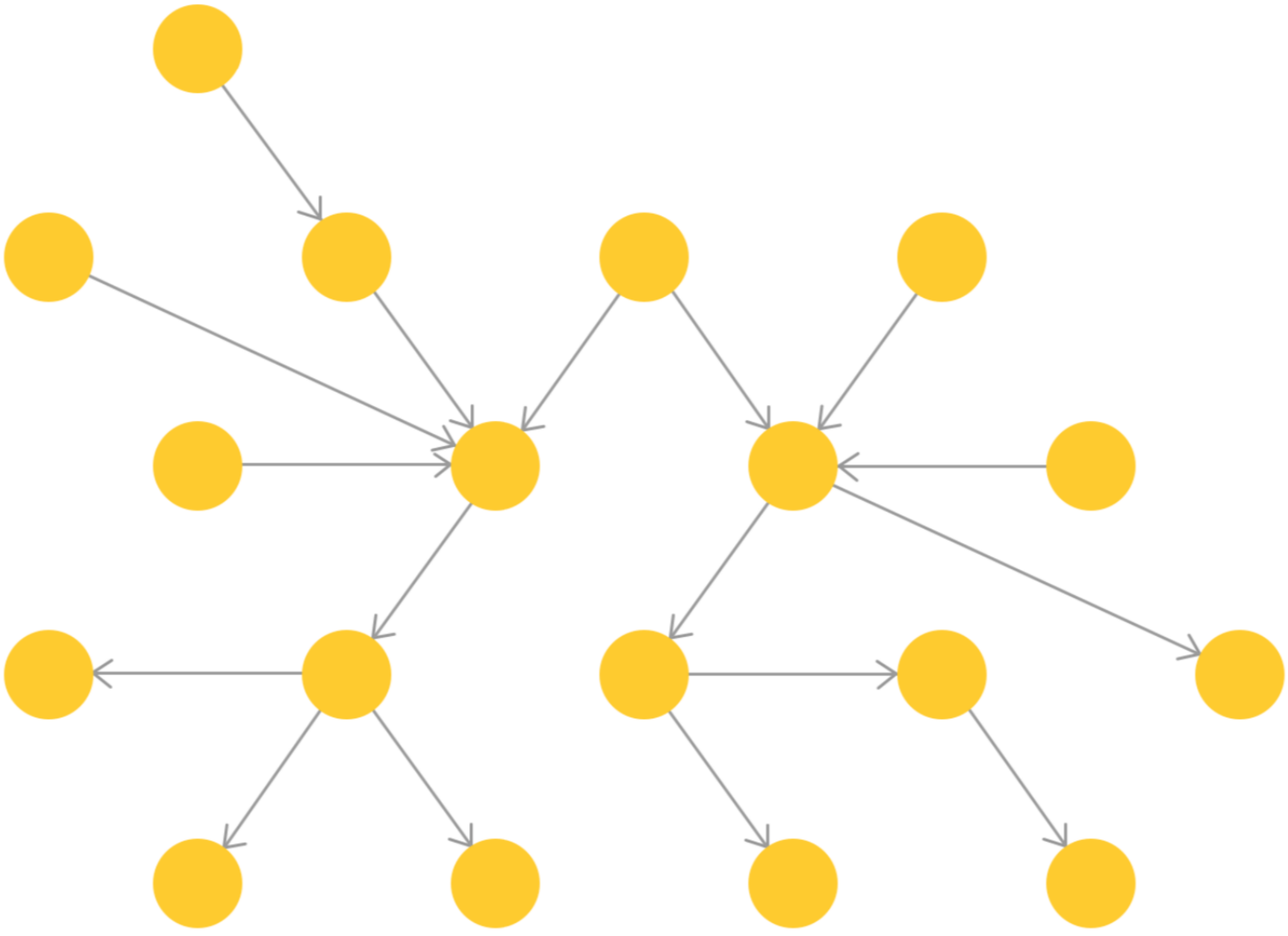
- Компонентное / Модульное
- Интеграционное
- Системное
- Приемочное

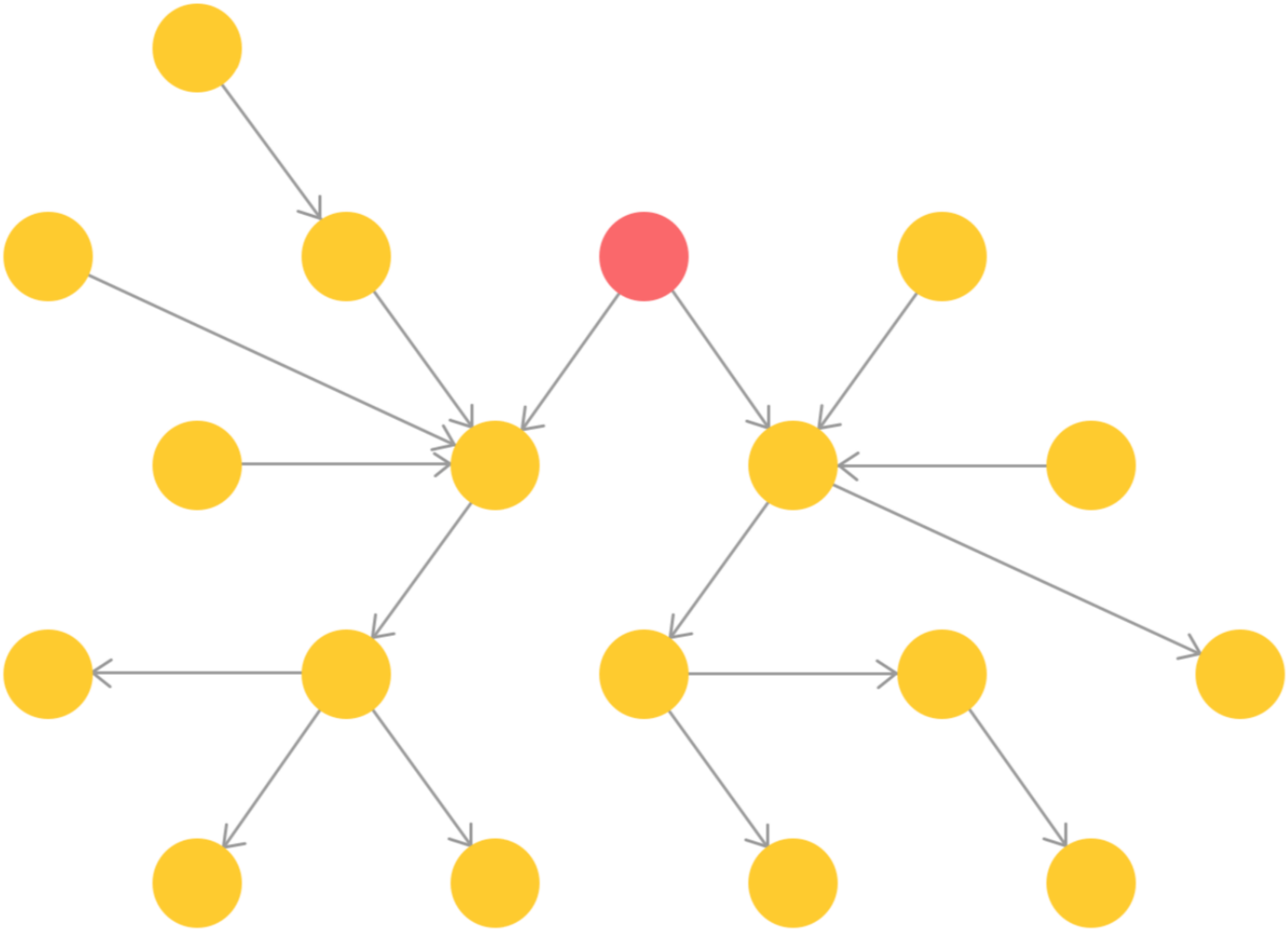
Уровни тестирования ПО

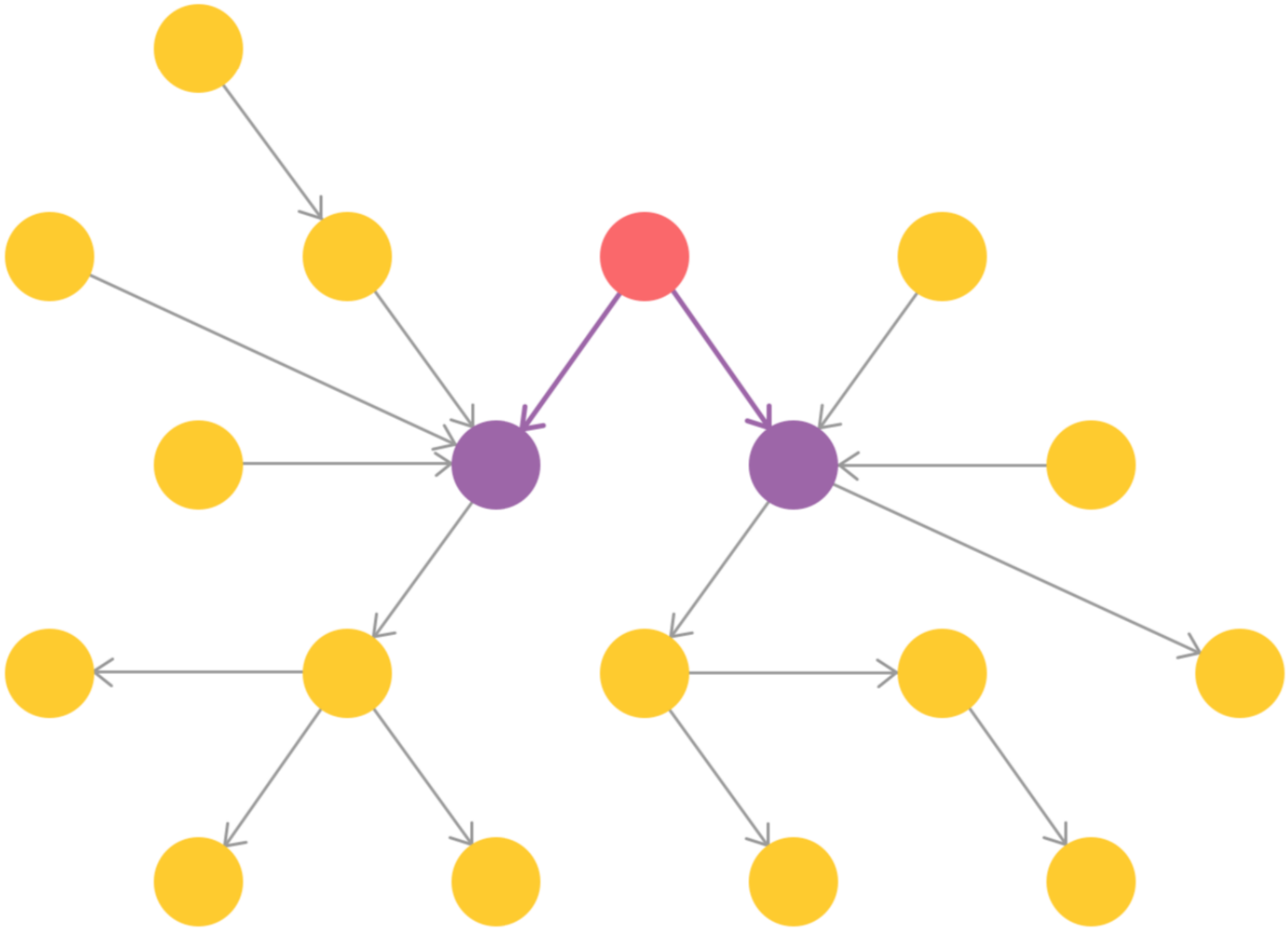
Тестирование – проверка, что ваш код работает правильно

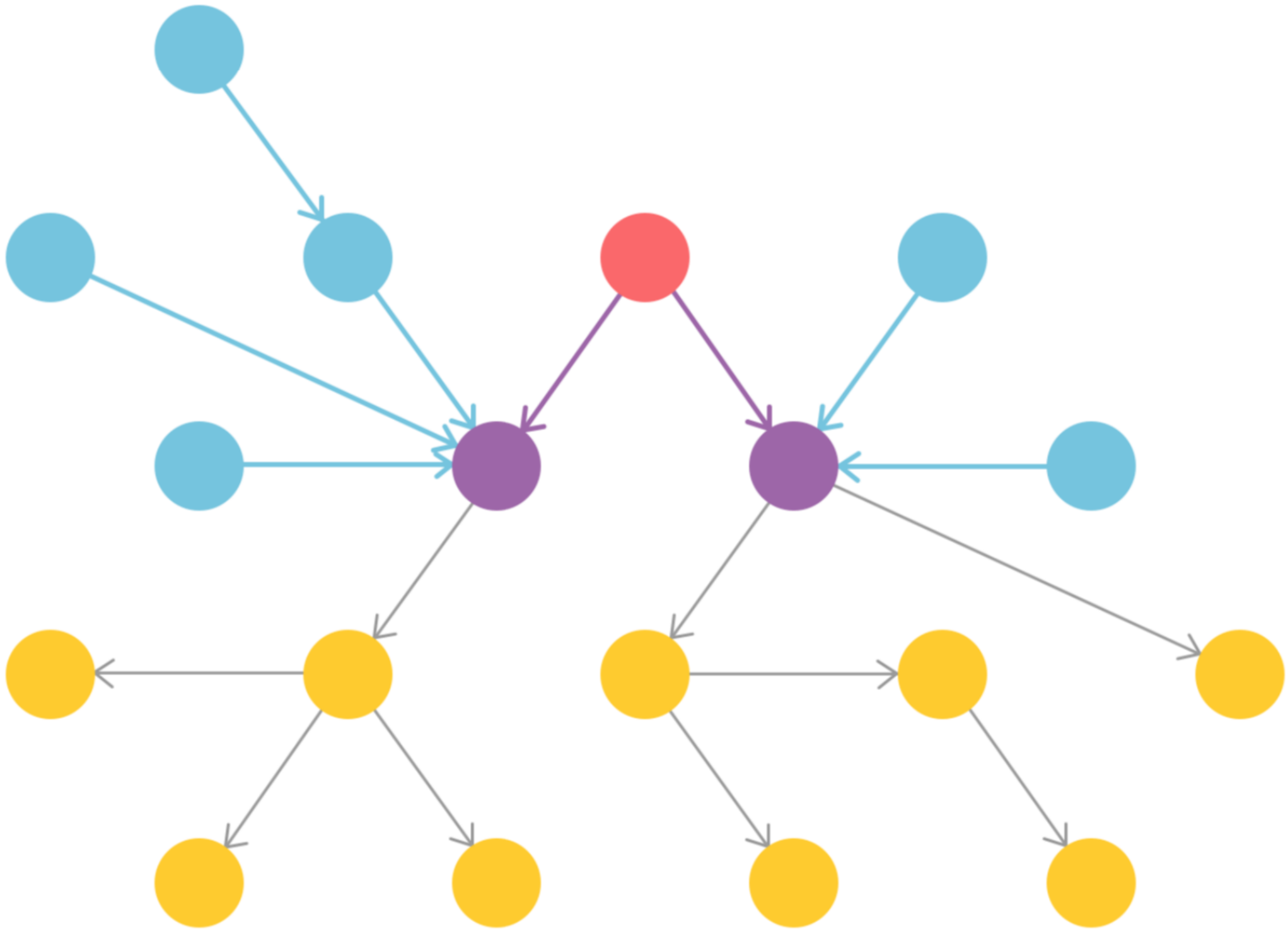
Результат

Автотесты



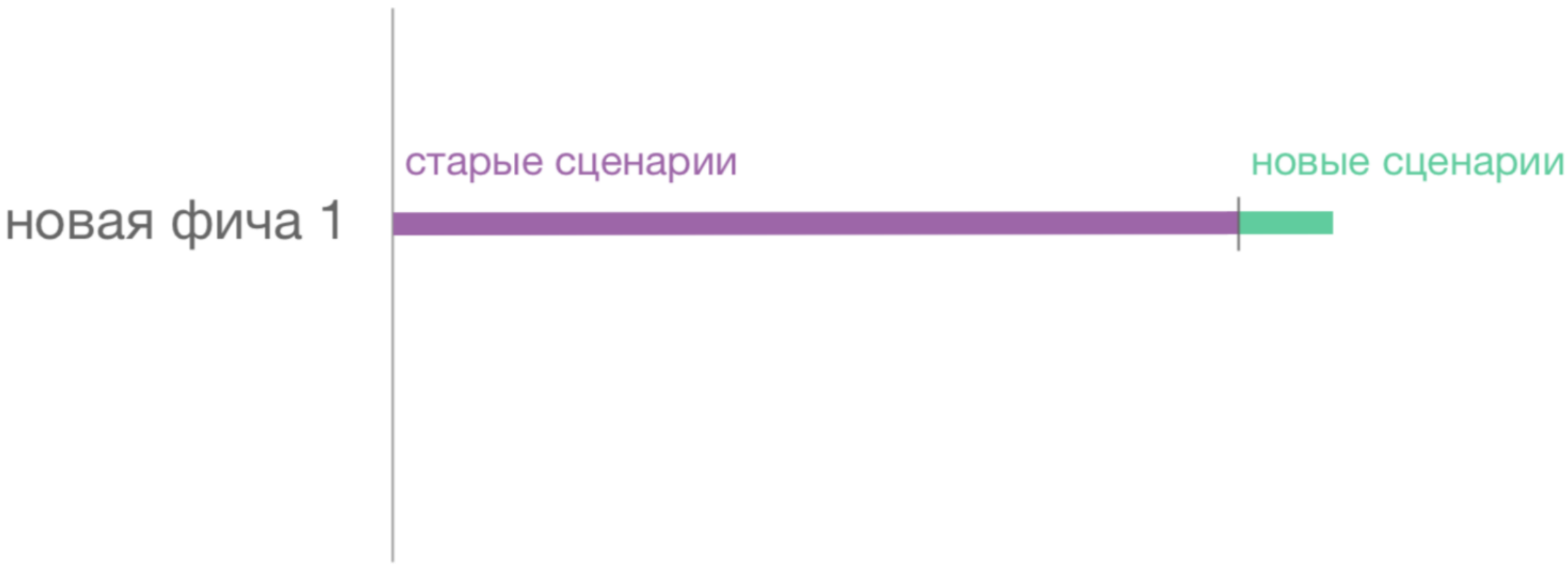


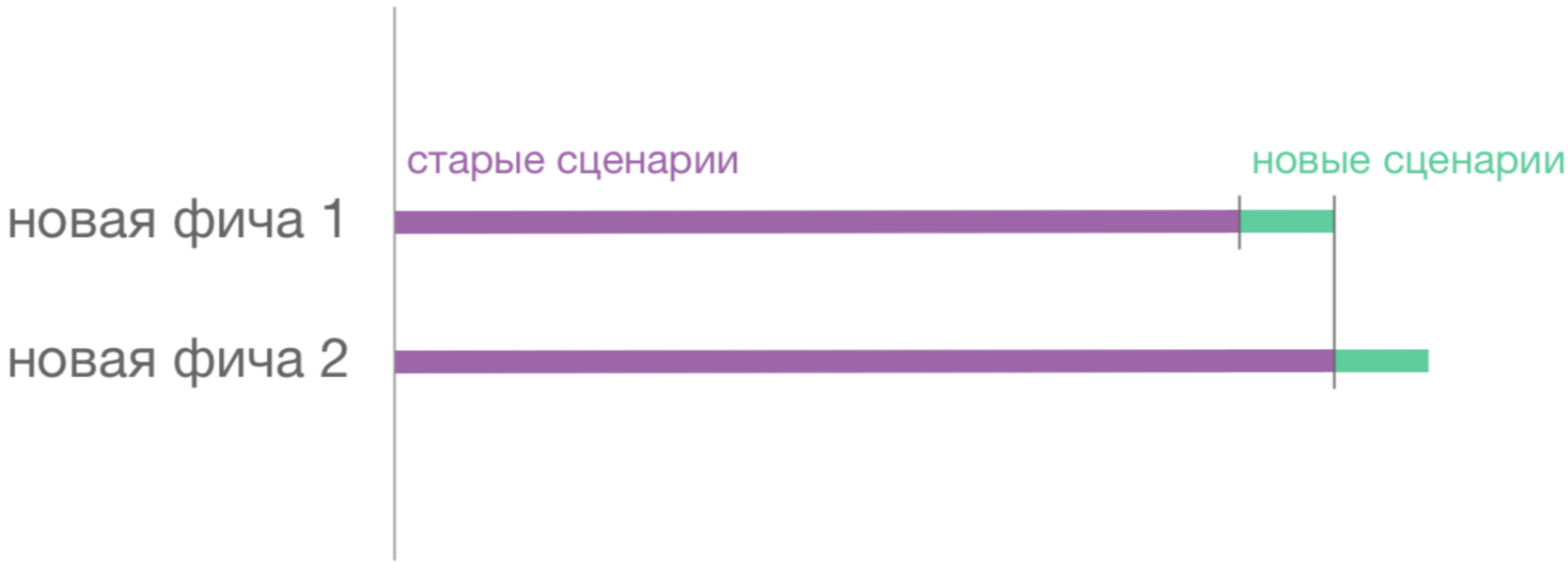




Что делать?

Всё перетестировать


старые сценарии
новые сценарии
новая фича 1


старые сценарии
новые сценарии
новая фича 1
новая фича 2

Что делать?

~~Всё перетестировать~~

И так сойдет

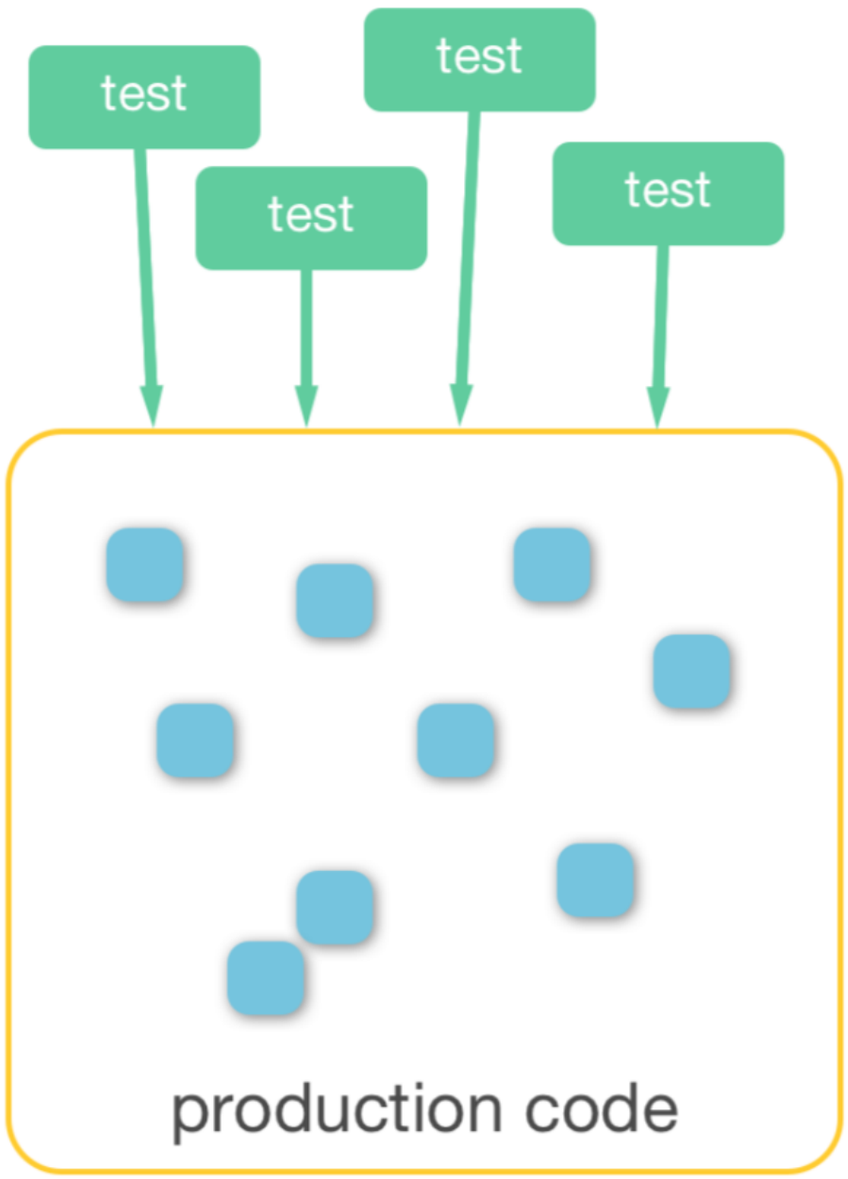
Что делать?

~~Всё перетестировать~~

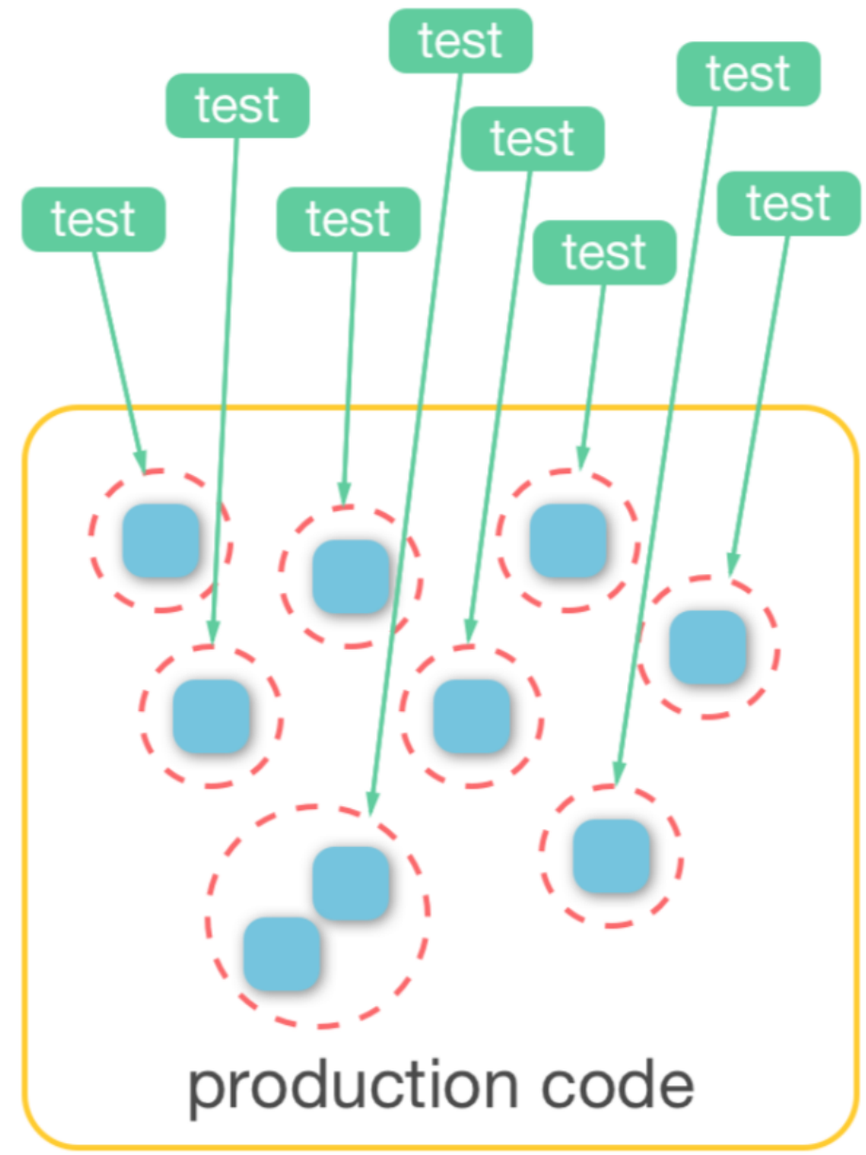
~~И так сойдет~~

Пишем автотесты

Integration testing



Unit testing



Модульный тест

Три вопроса

1. Что будем тестировать?

Три вопроса

1. Что будем тестировать?

Тестируем логические блоки

Три вопроса

1. Что будем тестировать?
2. Какой сценарий будем проверять?

Три вопроса

1. Что будем тестировать?
2. Какой сценарий будем проверять?

Тестируем интерфейс, который используют внешние сущности

Один тест – один сценарий

Три вопроса

1. Что будем тестировать?
2. Какой сценарий будем проверять?
3. Что будем считать результатом этого сценария?

```
class MyInput {

  // результат функции
  toString() {
    return this.value.toFixed(2);
  }

  // изменение внутреннего состояния
  setValue(value) {
    this.value = value;
  }

  // обращение к внешней системе
  sync() {
    fetch('http://my-server/save', {
      method: 'POST',
      body: this.value
    });
  }
}
```

Модульный тест – пример

TodoList.js

```
class TodoList {
  constructor(names = []) {
    this.items = names.map(name => new TodoItem(name));
  }

  addItem(name) {
    this.items.push(new TodoItem(name));
  }

  done(index) {
    this.items[index].done();
  }

  clear() {
    this.items = this.items.filter(item => !item.isDone);
  }
}
```

TodoList.js (продолжение)

```
class TodoItem {
  constructor(name) {
    this.name = name;
    this.isDone = false;
  }

  done() {
    this.isDone = true;
  }
}


module.exports = {TodoList, TodoItem};
```

test.js

```
function todoListCanAddItem {
  // подготовка
  let list = new TodoList();

  // действие
  list.addItem('test');

  // проверка
  if (!list.items.some(item => item.name === 'test')) {
    throw 'todoListCanAddItem - ERROR';
  }
}

todoListCanAddItem();
```

```
[redacted]/test.js:12
        throw 'todoListCanAddItem - ERROR';
        ^
todoListCanAddItem - ERROR
```

test.js

```
function todoListCanCheckItem {
  // подготовка
  let list = new TodoList(['item1', 'item2']);

  // действие
  list.done(1);

  // проверка
  if (!list.items[1].isDone) {
    throw 'todoListCanCheckItem - ERROR';
  }
}

todoListCanCheckItem();
```

Минусы подхода

1. Если тест не упал с ошибкой – ничего не выводится в консоль. А прогонялся ли вообще тест?

2. Нужно отдельно запускать каждую из функций в `test.js`

3. Неинформативные сообщения об ошибках

Библиотеки для модульного тестирования

MochaJS

ChaiJS

```
function todoListCanAddItem {
   // подготовка
  let list = new TodoList();

   // действие
  list.addItem('test');

   // проверка
  if (!list.items.some(
      item => item.name === 'test'
  )) {
    throw 'todoListCanAddItem - ERROR';
  }
}

todoListCanAddItem();
```

```
it('можно добавлять элементы' , () => {
   // подготовка
  let list = new TodoList();

   // действие
  list.addItem('test');

   // проверка
  if (!list.items.some(
      item => item.name === 'test'
  )) {
    throw 'todoListCanAddItem - ERROR';
  }
}
```

```js
function todoListCanCheckItem {
  // подготовка
  let list = new TodoList([
    'item1',
    'item2'
  ]);

  // действие
  list.done(1);

  // проверка
  if (!list.items[1].isDone) {
    throw 'todoListCanCheckItem: ERROR';
  }
}

todoListCanCheckItem();
```

```js
it('можно чекать элементы по индексу', () => {
  // подготовка
  let list = new TodoList([
    'item1',
    'item2'
  ]);

  // действие
  list.done(1);

  // проверка
  if (!list.items[1].isDone) {
    throw 'todoListCanCheckItem: ERROR';
  }
}
```

Тесты успешно пройдены

```
node test.js
```

```
mocha test.js


  ✓ можно добавлять элементы
  ✓ можно чекать элементы по индексу

  2 passing (6ms)
```

Тесты упали с ошибкой

```
/test.js:12
        throw 'todoListCanAddItem - ERROR';
        ^
todoListCanAddItem - ERROR
```

```
> mocha test.js


  ✓ можно добавлять элементы
  1) можно чекать элементы по индексу

  1 passing (8ms)
  1 failing

  1) можно чекать элементы по индексу:
     Error: the string "todoListCanCheckItem - ERROR" was thrown, throw an Error :)



npm ERR! Test failed.  See above for more details.
```

Использование ChaiJS

```
it('можно добавлять элементы' , () => {
   // подготовка
  let list = new TodoList();

   // действие
  list.addItem('test');

   // проверка
  if (!list.items.some(
      item => item.name === 'test'
  )) {
    throw 'todoListCanAddItem - ERROR';
  }
}
```

```
it('можно добавлять элементы' , () => {
   // подготовка
  let list = new TodoList();

   // действие
  list.addItem('test');

   // проверка
  expect(list.items.map(i => i.name))
    .to.include('test');
}
```

ChaiJS добавляет Stack trace в сообщении об ошибке

```
> mocha test.js



  ✓ можно добавлять элементы
  1) можно чекать элементы по индексу

  1 passing (8ms)
  1 failing

  1) можно чекать элементы по индексу:
     Error: the string "todoListCanCheckItem - ERROR" was thrown, throw an Error :)



npm ERR! Test failed.  See above for more details.
```

```
  1) можно добавлять элементы
  ✓ можно чекать элементы по индексу

  1 passing (10ms)
  1 failing

  1) можно добавлять элементы:
     AssertionError: expected [] to include 'test'
      at Context.<anonymous> (test.js:12:44)


npm ERR! Test failed.  See above for more details.
```

Заглушки

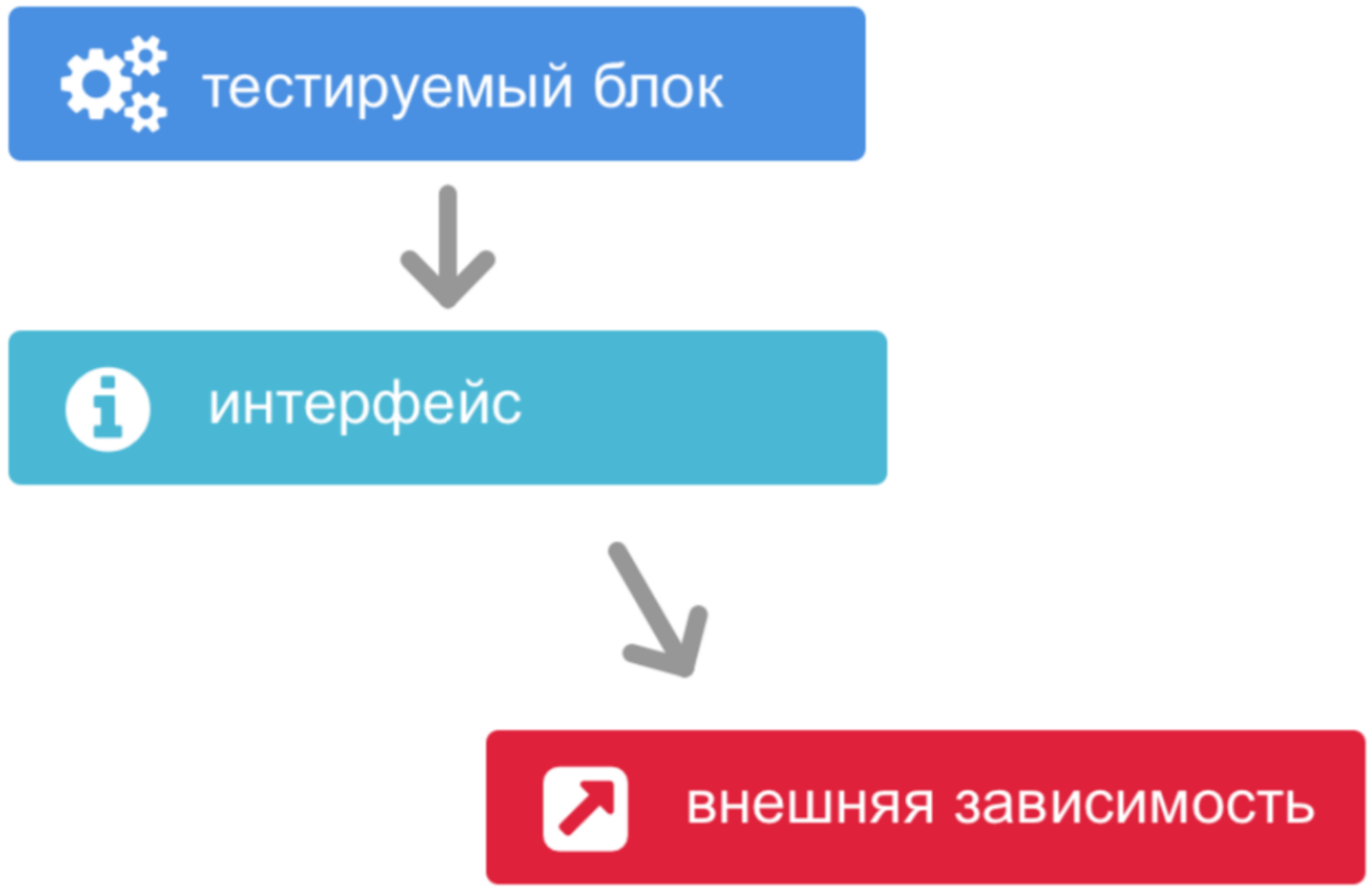
SKYTEAM
С.МИХАЛКОВ

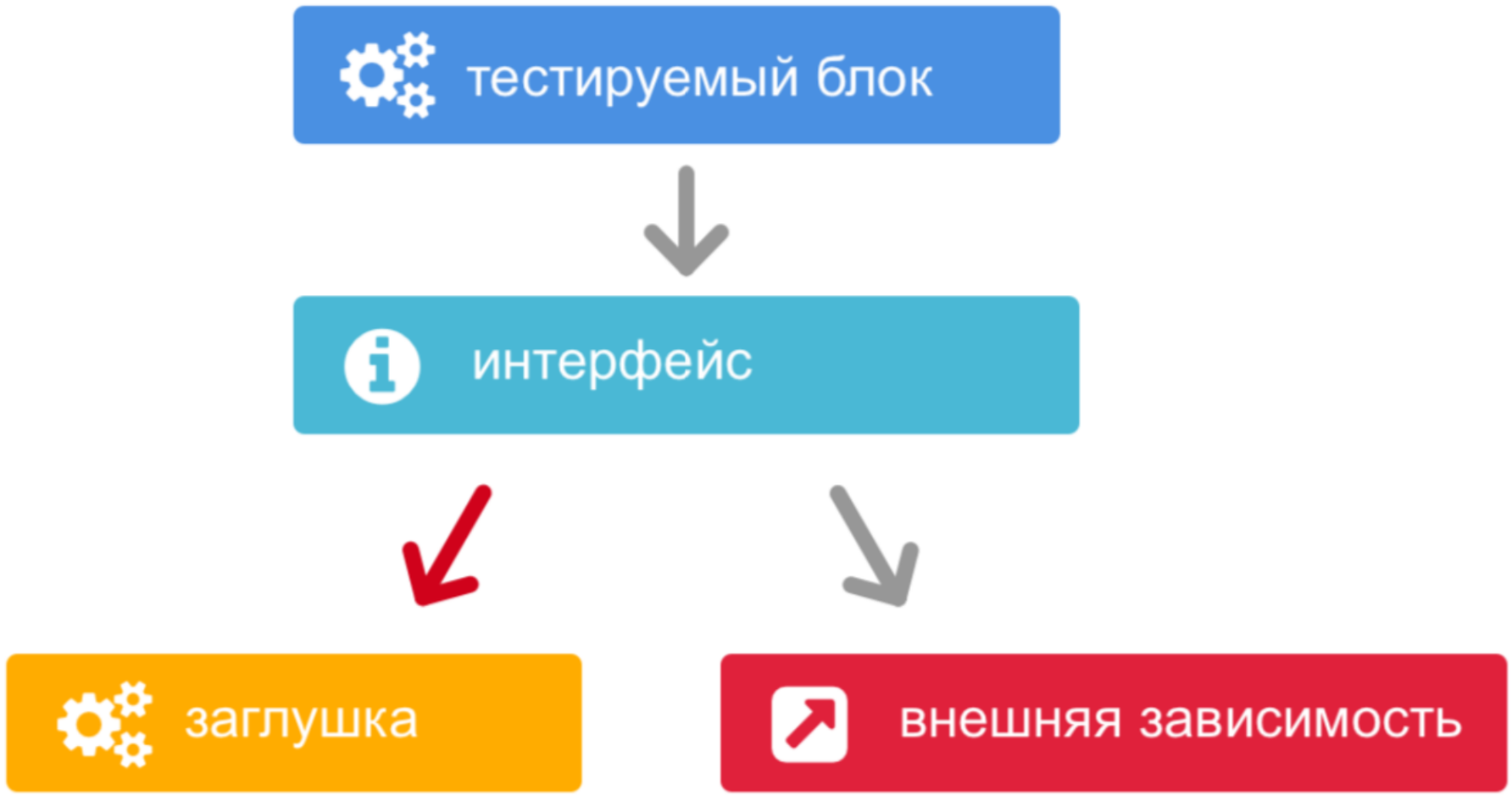
Внешняя зависимость – объект, с которым работает тестируемый блок, но который мы не контролируем.

Заглушка (stub) — управляемая замена зависимости.

- находим интерфейс
- делаем фальшивую реализацию
- подменяем настоящую реализацию заглушкой

тестируемый блок
внешняя зависимость


тестируемый блок
интерфейс
внешняя зависимость


тестируемый блок
интерфейс
заглушка
внешняя зависимость

Пример – заменяем зависимость заглушкой

Добавляем метод `load()` в `TodoList.js`

```
class TodoList {
  constructor(names = []) {
    this.items = names.map(name => new TodoItem(name));
  }

  load() {
    return fetch(`http://localhost:3000/load`)
      .then((names) => {
        this.items = names.map(name => new TodoItem(name));
      });
  }

  ...
}
```

Тест для проверки метода `load()`

```
it('можно загружать элементы с бэкенда' , () => {
  // подготовка
  let list = new TodoList();

  // действие
  list.load();

  // проверка
  expect(list.items).to.include(???);
}
```

Точки расширения — это места в программе, в которых можно подключить другую функциональность взамен существующей.

Рефакторинг — изменение кода без изменения функциональности.

Добавляем в TodoList точку расширения

```
class TodoList {
  constructor(names = []) {
    this.fetch = global.fetch;
    this.items = names.map(name => new TodoItem(name));
  }

  load() {
    return this.fetch(`http://localhost:3000/load`)
      .then((names) => {
        this.items = names.map(name => new TodoItem(name));
      });
  }

  ...
}
```

Тест для проверки метода `load()`

```
it('можно загружать элементы с бэкенда' , () => {
  // подготовка
  let list = new TodoList();
  list.fetch = () => {
    return Promise.resolve(['item1', 'item2']);
  });

  // действие
  list.load();

  // проверка
  expect(list.items.map(i => i.name)
    .to.eql(['item1', 'item2']);
}
```

При запуске – ошибка из-за асинхронности

```
✓ можно добавлять элементы
✓ можно чекать элементы по индексу

2 passing (12ms)
1 failing

1) можно загружать элементы с бэкенда:

   AssertionError: expected [] to deeply equal [ 'item1', 'item2' ]
   + expected - actual

   -[]
   +[
   +  "item1"
   +  "item2"
   +]

   at Context.it (test.js:15:44)
```

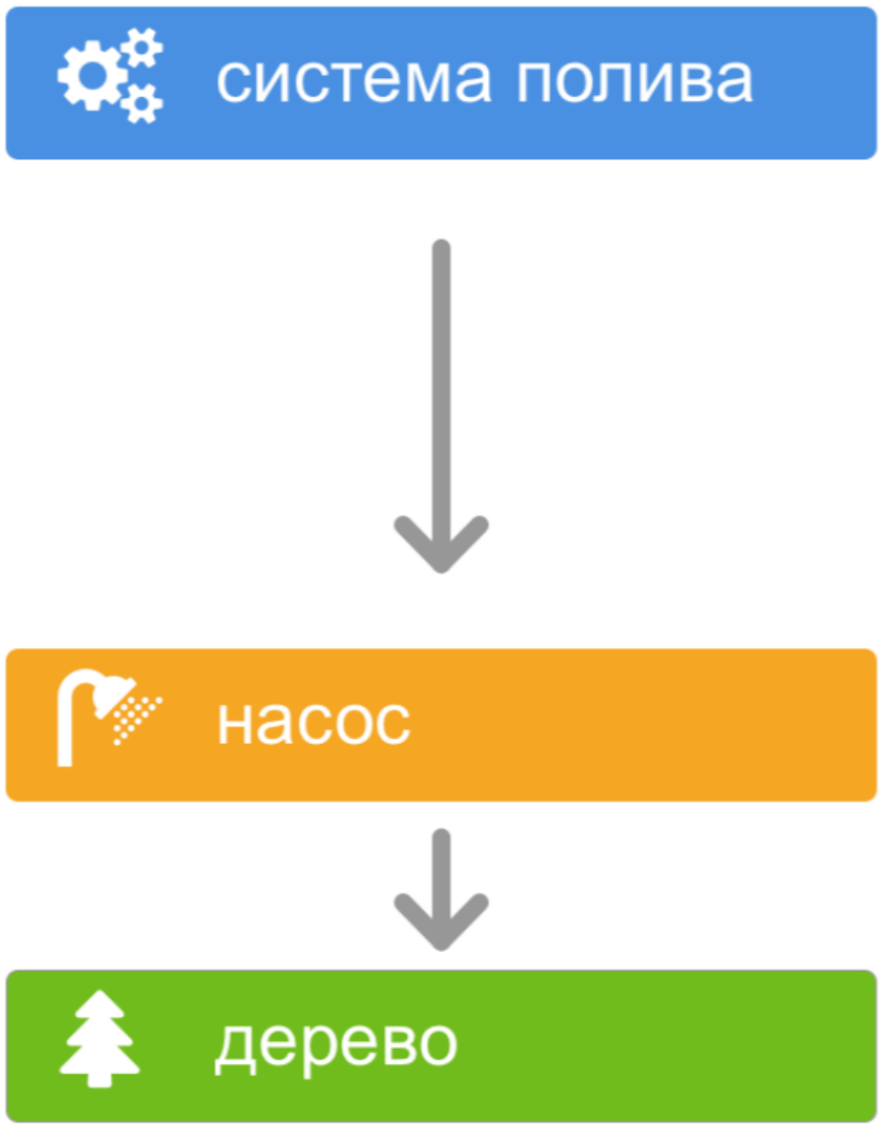
Тест для проверки метода `load()` – асинхронный

```
it('можно загружать элементы с бэкенда' , async () => {
  // подготовка
  let list = new TodoList();
  list.fetch = () => {
    return Promise.resolve(['item1', 'item2']);
  });

  // действие
  await list.load();

  // проверка
  expect(list.items.map(i => i.name)
    .to.eql(['item1', 'item2']);
}
```

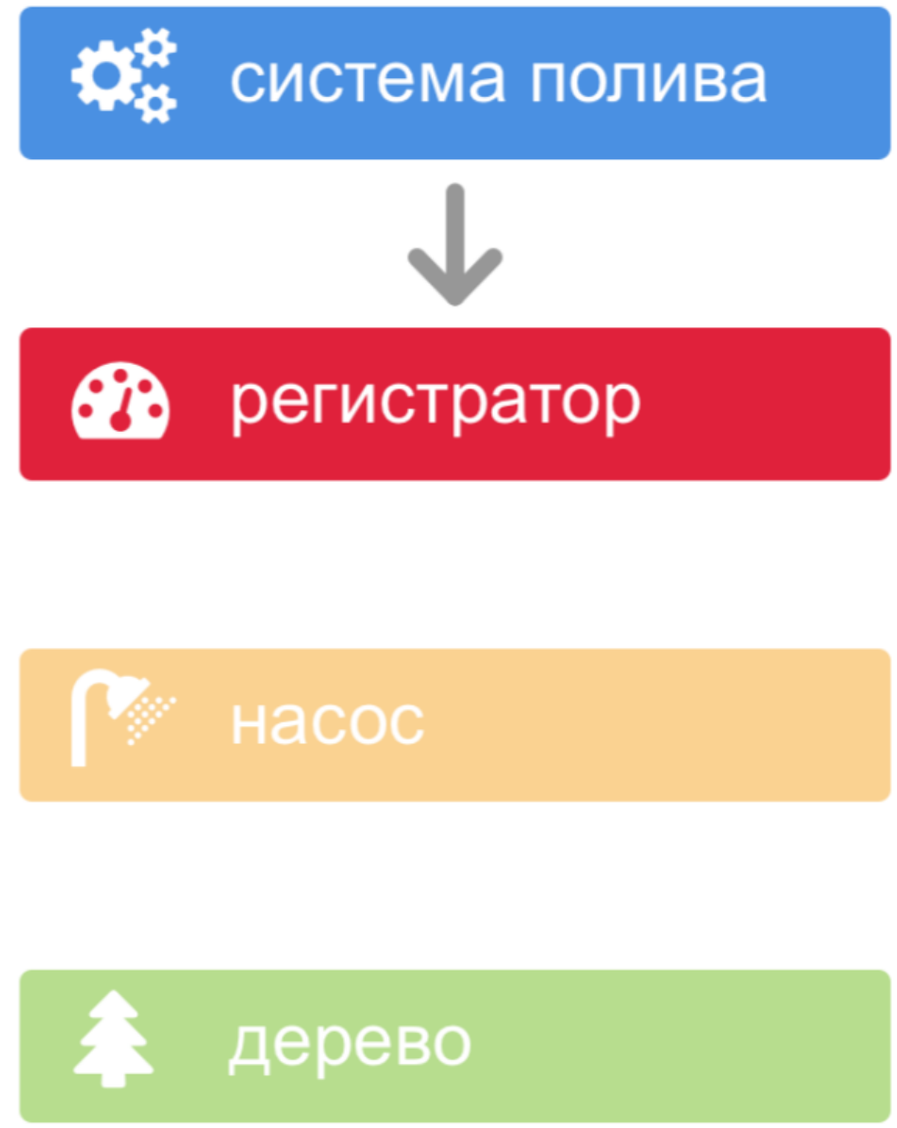
```
✓ можно загружать элементы с бэкенда
✓ можно добавлять элементы
✓ можно чекать элементы по индексу

3 passing (11ms)
```

Mock

Тестируем состояние



Тестируем взаимодействие



Добавляем метод `save()` в `TodoList.js`

```
class TodoList {
  constructor(names = []) {
    this.items = names.map(name => new TodoItem(name));
  }

  save() {
    let names = this.items
      .filter(item => !item.isDone)
      .map(item => item.name);

    return this.fetch('http://localhost:3000/load', {
      method: 'POST'
      body: names
    });
  }

  ...
}
```

Тест для проверки метода `save()`

```
it('можно сохранять элементы на сервер' , () => {
  // подготовка
  let list = new TodoList(['item1', 'item2']);

  // действие
  list.save();

  // проверка
  expect(???).to.eql(???);
}
```

Тест для проверки метода `save()`

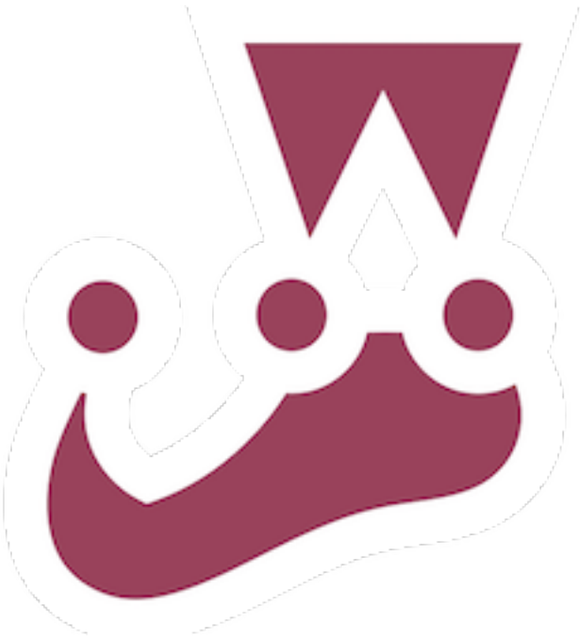
```
it('можно сохранять элементы на сервер' , () => {
  // подготовка
  let list = new TodoList(['item1', 'item2']);
  list.fetch = () => {
    return Promise.resolve();
  }

  // действие
  list.save();

  // проверка
  expect(???).to.eql(???);
}
```

Тест для проверки параметров запроса `save()`

```
it('можно сохранять элементы на сервер' , () => {
  // подготовка
  let latestArgs;

  let list = new TodoList(['item1', 'item2']);
  list.fetch = (...args) => {
    latestArgs = args;
    return Promise.resolve();
  }

  // действие
  list.save();

  // проверка
  expect(latestArgs[1].body).to.eql(['item1', 'item2']);
}
```

SinonJS

Тестирование при помощи СНИМКОВ



JestJS

Снэпшот-тест для компонента `Link`

```
import React from 'react';
import Link from '../Link.react';
import renderer from 'react-test-renderer';

it('отображается корректно', () => {
  const tree = renderer.create(
    <Link page="http://www.facebook.com">Facebook</Link>
  ).toJSON();
  expect(tree).toMatchSnapshot();
});
```

Содержимое файла со снэпшотом для компонента `Link`

```
exports[`отображается корректно 1`] = `
<a
  className="normal"
  href="http://www.facebook.com"
  onMouseEnter={[Function]}
  onMouseLeave={[Function]}
>
  Facebook
</a>
`;
```

Обновляем компонент или тест

```
import React from 'react';
import Link from '../Link.react';
import renderer from 'react-test-renderer';

it('отображается корректно', () => {
  const tree = renderer.create(
    <Link page="http://www.instagram.com">Instagram</Link>
  ).toJSON();
  expect(tree).toMatchSnapshot();
});
```

Обновление снэпшота

```
2. node
FAIL __tests__/Link.react-test.js
  ● renders correctly

    expect(value).toMatchSnapshot()

    Received value does not match stored snapshot 1.

    - Snapshot
    + Received

     <a
       className="normal"
    -  href="http://www.facebook.com"
    +  href="http://www.instagram.com"
       onMouseEnter={[Function]}
       onMouseLeave={[Function]}>
    -  Facebook
    +  Instagram
     </a>

      at Object.<anonymous> (__tests__/Link.react-test.js:14:16)

  × renders correctly (10ms)

Snapshot Summary
 › 1 snapshot test failed in 1 test suite. Inspect your code changes or press `u` to update
```

```
jest --updateSnapshot
```

Написание хороших тестов

Хороший тест – это тест, который ...

1. заслуживает доверия
2. легко сопровождать
3. легко понять

Надежность тестов

Надежные тесты — без ошибок и проверяют всё то, что нужно.

- Проверять, что тест корректно ломается
- Проверять только один сценарий и один тип результата
- Не добавлять логику в тесты
- Тесты должны быть изолированы
- Мониторить покрытие сценариев тестами

Пример библиотеки мониторинга test coverage

IstanbulJS

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	98.92	94.36	99.49	100	
yargs	99.17	93.95	100	100	
index.js	100	100	100	100	
yargs.js	99.15	93.86	100	100	
yargs/lib	98.7	94.72	99.07	100	
command.js	99.1	98.51	100	100	
completion.js	100	95.83	100	100	
obj-filter.js	87.5	83.33	66.67	100	
usage.js	97.89	92.59	100	100	
validation.js	100	95.56	100	100	

Лёгкость понимания

- Давать тестам осмысленные названия
- Использовать осмысленные названия переменных в тестах
- Писать утверждения со смыслом
- Отделять проверки от действий
- Поддерживать соответствие между тестами и кодом на уровне файловой системы

Спасибо!