Software Engineering and Architecture
Lecture 5: Agile Testing (1)

Olivier Liechti
olivier.liechti@heig-vd.ch

MSE | MASTER OF SCIENCE IN ENGINEERING

Today.

Agile testing
15h - 16h

OO Reengineering Patterns (3 x 10’)
16h - 16h30

Open Affect, API specification and validation
16h30 - 17h20

Week	Theory	OO Reengineering	Practice
#1	Agile, Scrum		Intro to Docker
#2	Software evolution	Introduction	Specify and implement a micro-service
#3	Continuous delivery (1)	*Setting Directions*	Intro to Jenkins & Travis
#4	Continuous delivery (2)	*First Contact*	Our first build pipeline
#5	**Agile testing (1)**	***Initial Understanding***	**Intro to Cucumber**
#6	Agile testing (2)	*Detailed Model Capture*	Add tests to pipeline
#7	Agile metrics	*Tests: Your Life Insurance!*	Add Sonar to pipeline
#8	Continuous improvement	*Migration Strategies I*	Add non-functional tests
#9	Wrap-up	*Migration Strategies II*	

Agenda

- **Testing in software development process**
  - Is it done as the **last phase** of the development cycle?
  - Is it done by an **independent organization**, which enforces quality levels?
- **Agile testing & quality**
  - How do testing activities fit in agile development methods?
  - What is the role of test engineers in agile organizations? Where do they fit?
  - How can we get apprehend the vast topic of “testing” and “validation”?
- **Agile testing quadrants**
  - Selected topics: unit tests, integration tests, automated acceptance tests, systemic tests

Product development cycle (waterfall?)

Analyze

Specify

Build

Test

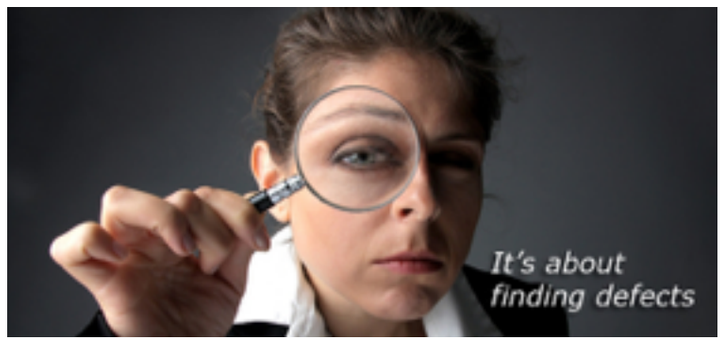
So... anything wrong with that?

Product development cycle

- How much **time** do we need to go through all the phases?
- How often do we go through these phases (is it a **cycle**)?
- How much "**functionality**" is moving through these phases?
- **Who** is working in each phase?
- How do people **collaborate**?

Product development cycle

Analyze → **Specify** → **Build** → **Test**




AUSTIN
AUSTIN

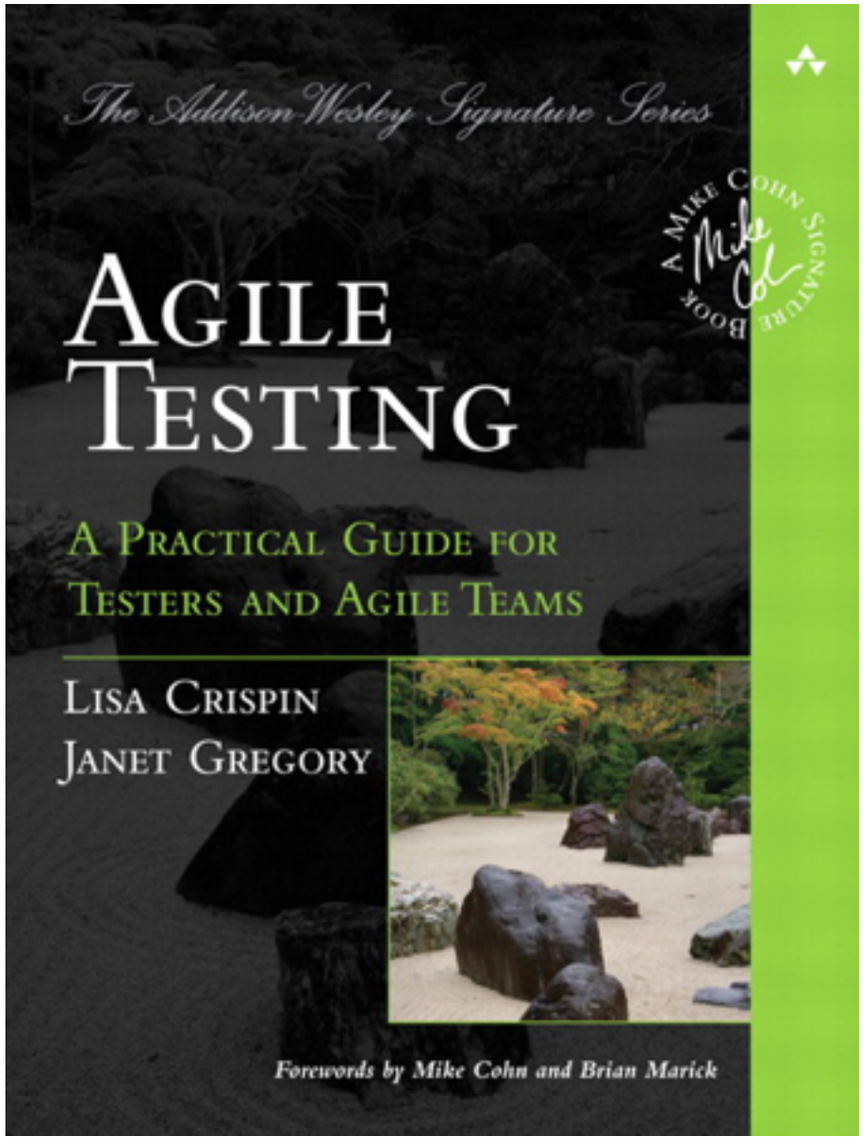

Don’t look at me, it’s his responsibility.
Browman

MSE | MASTER OF SCIENCE IN ENGINEERING

Agile Testing

*"One of the biggest differences in agile development versus traditional development is the "**whole team" approach**.*

*With agile, it's not only the testers or a quality assurance team who feel responsible for quality. [...]. **Everyone on an agile team gets "test-infected"**. Tests, from the unit level on up, drive the coding, help the team learn how the application should work, and let us know when we're "done" with a task or story."*



Janet Gregory and Lisa Crispin
http://www.informit.com/articles/article.aspx?p=1316250&seqNum=5

MSE | MASTER OF SCIENCE IN ENGINEERING

***Small autonomous teams**, which are given the responsibility of a complete "product" are very effective.*

*Communication, collaboration and coordination are much easier than across **organizational silos**.*

*People are much more likely to feel **empowered** and **accountable**.*

http://blog.jasoncrawford.org/two-pizza-teams

Elisabeth Hendrickson, Google Tech Talks

https://www.youtube.com/watch?v=bqrOnIECCSg

MSE | MASTER OF SCIENCE IN ENGINEERING

Agile Testing Quadrants

*"**Software quality**" is a **broad concept** and has many aspects (reliability, efficiency, usability, maintainability, etc.).*

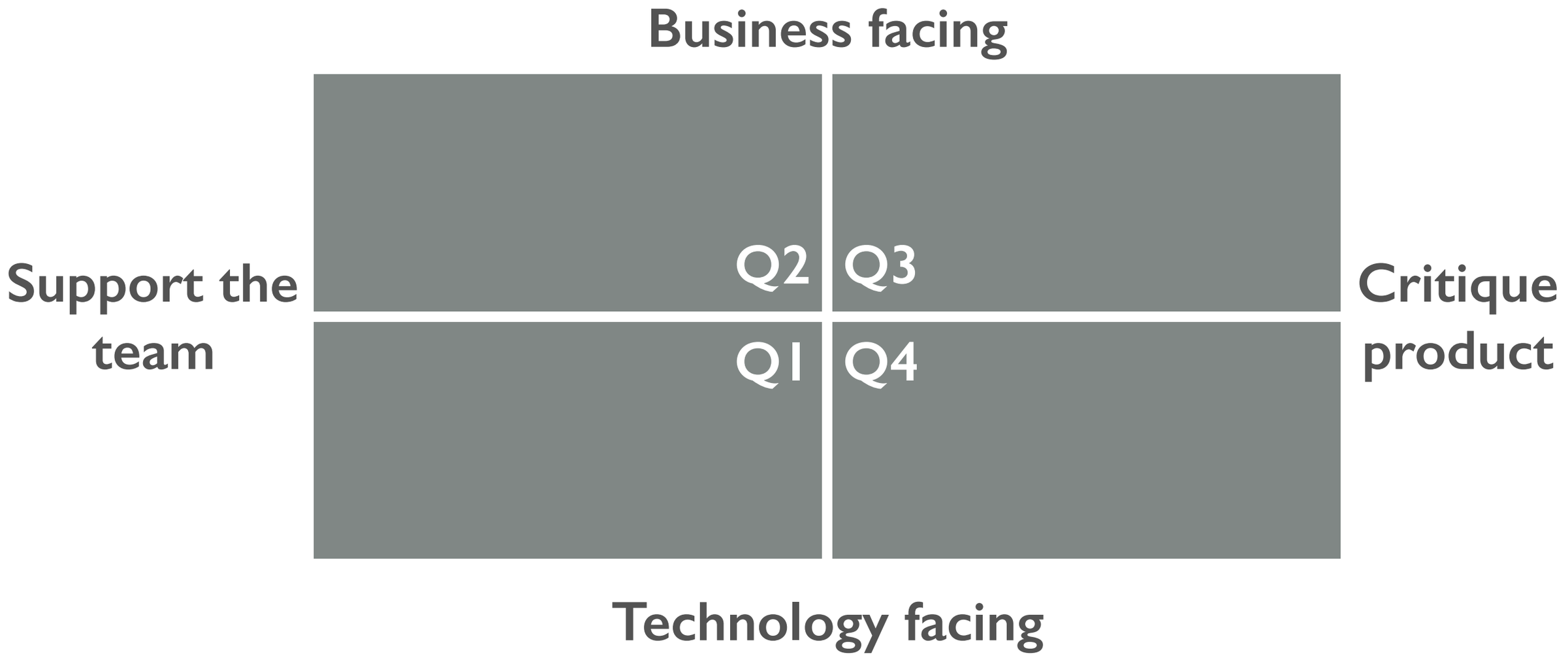
*"**Software testing**" refers to methods and techniques for **assessing** certain aspects of the quality of a software system. **There are many, many of them**.*

*Some "**software testing**" techniques do not only measure quality after the fact, but **help the team to proactively** maintain the quality of the software to an appropriate level.*

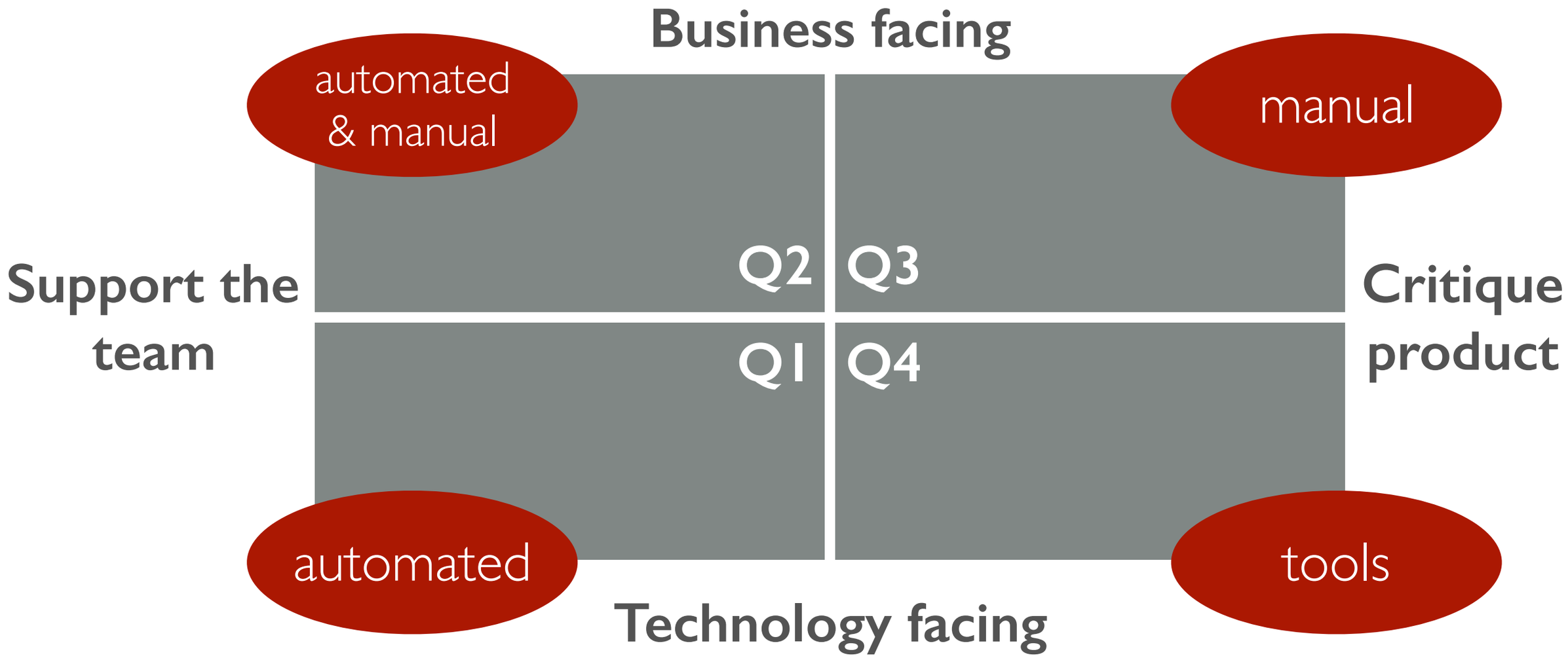
*Is there a way to **classify** all these methods, so that we can see how they relate to each other?*

*These techniques are aligned with the principles of **agile** software development.*

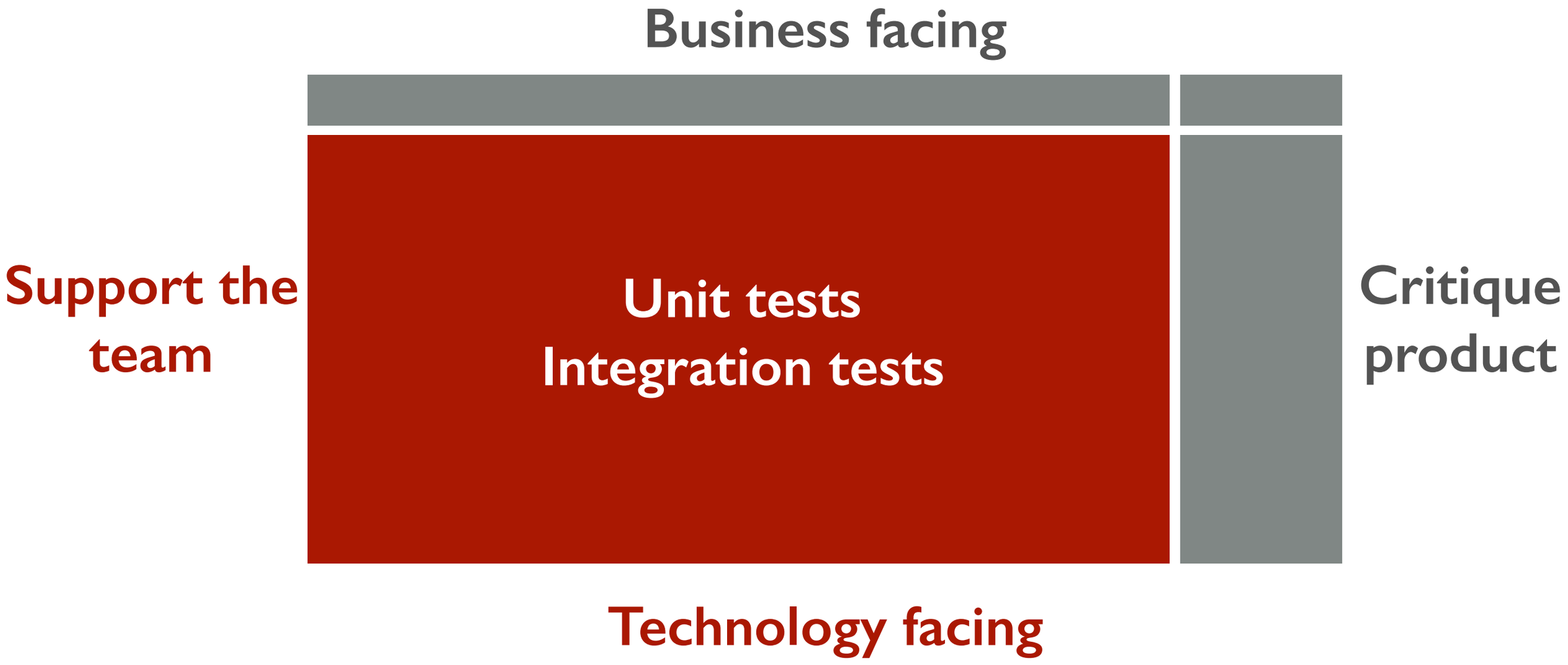
Agile testing quadrants



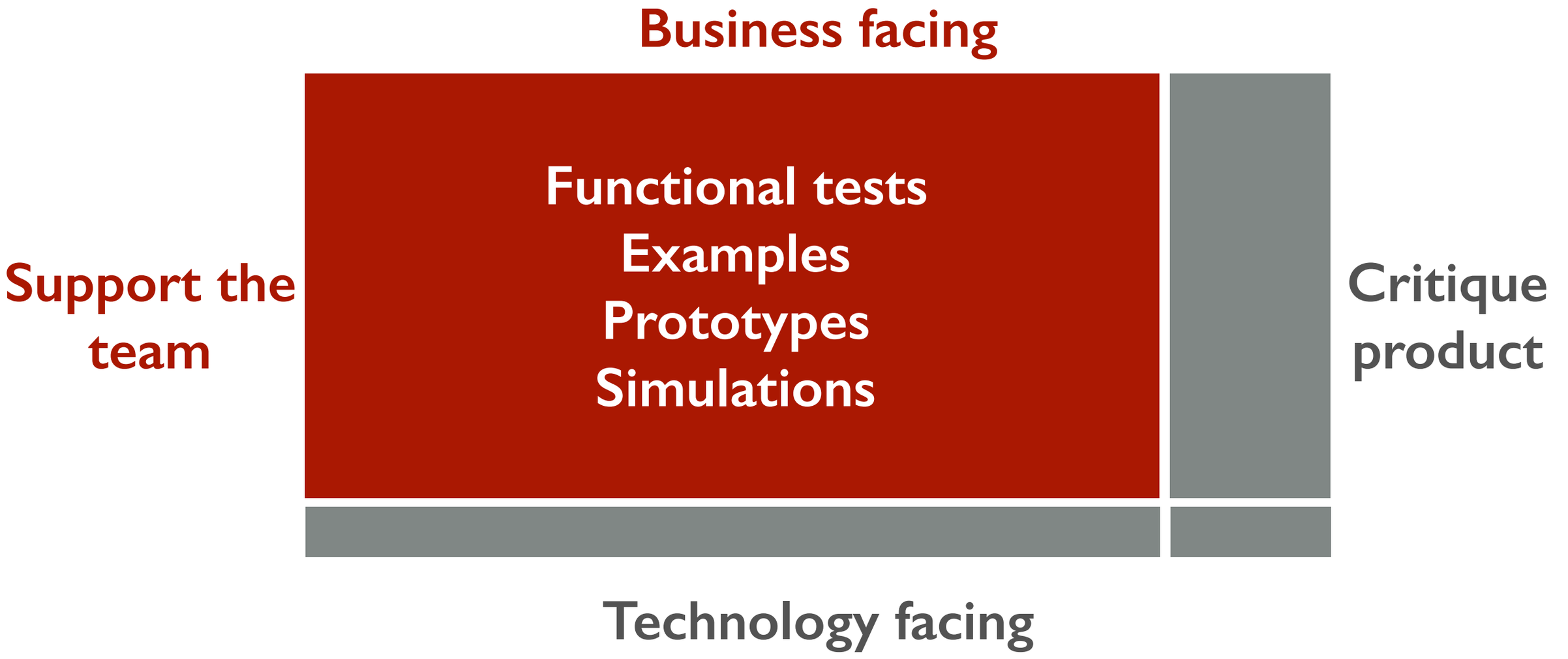
Support the team	Some of these tests **help individual team members** while they do their job. Sometimes, creating a "test" helps me **specify and/or design** the product. Other tests **facilitate team collaboration**, especially between "business" and "technical" people (shared language).
Critique product	Some of these tests allow humans to evaluate the quality of a software from the **users point of view** (is it easy to use? is it easy to learn? does it solve the user's problem?). Other tests aim to detect issues with **non-functional (systemic) qualities**.
Technology facing	Some tests are created and executed by **technical team members**. They are highly automated. They relate to the "Are we building the product right?" question.
Business facing	Some tests are created by (or at least with) **business-oriented team members**. They also relate to the "Are we building the right product?" question.


Business facing
automated & manual
manual
Support the team
Critique product
Q2
Q3
Q1
Q4
automated
tools
Technology facing

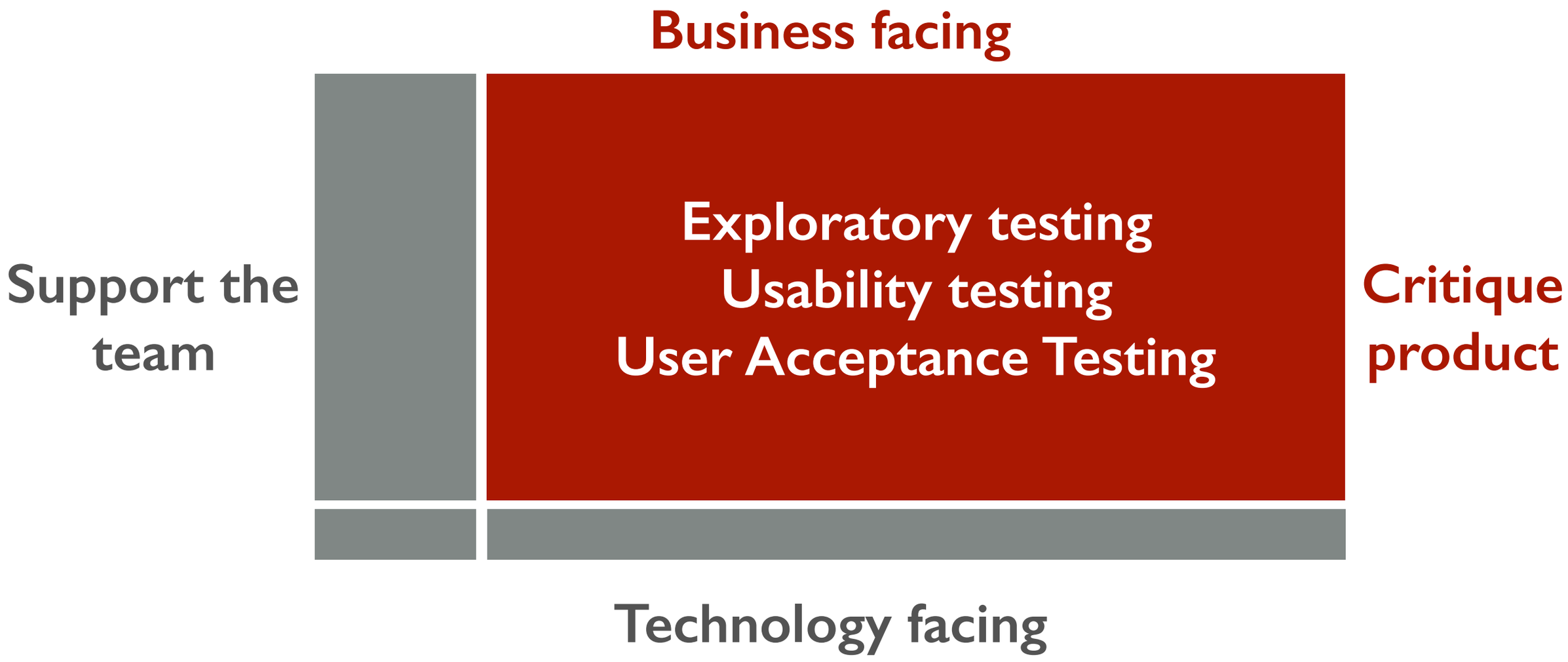
Agile testing quadrants: Q1



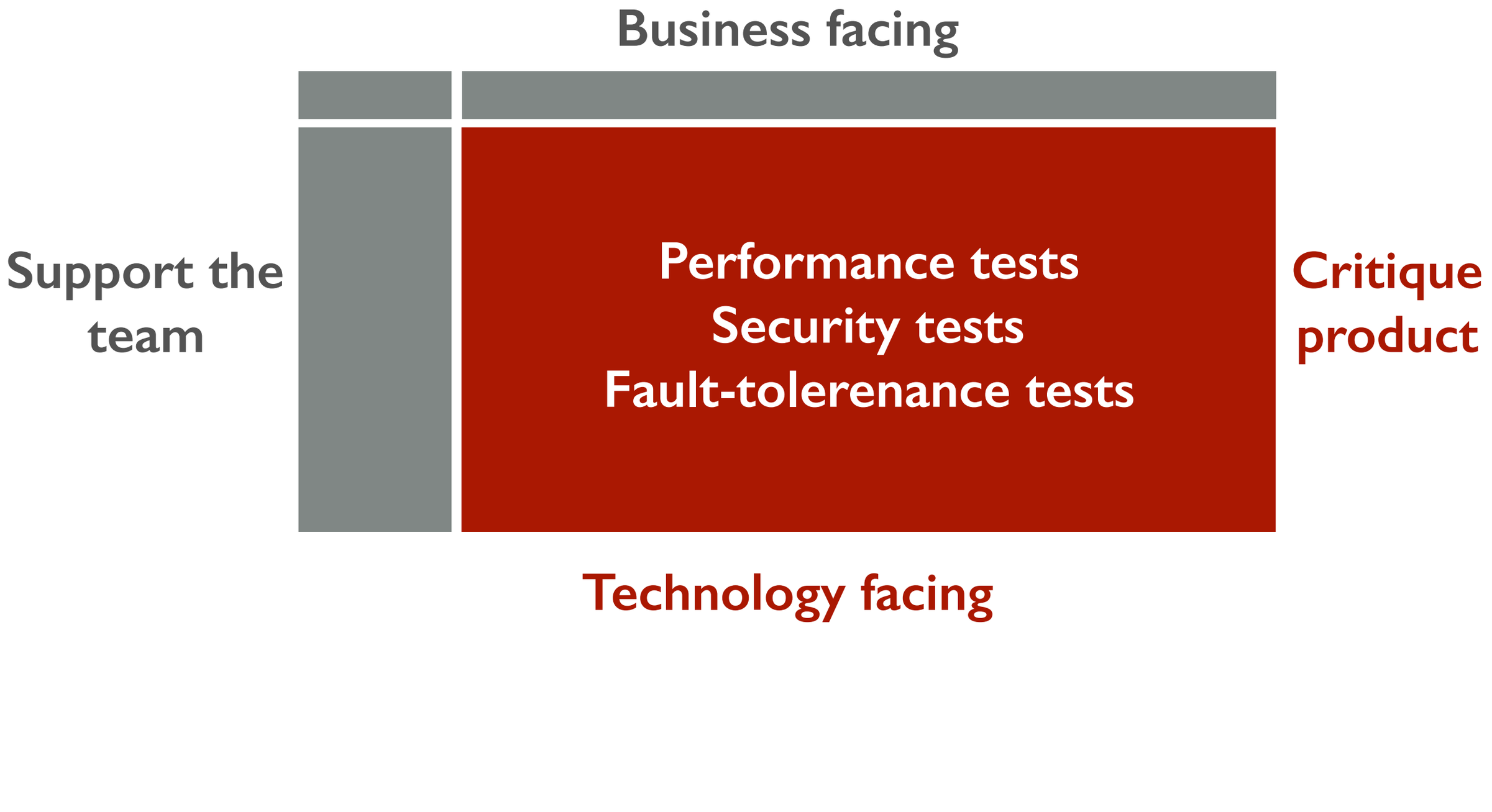
Agile testing quadrants: Q2



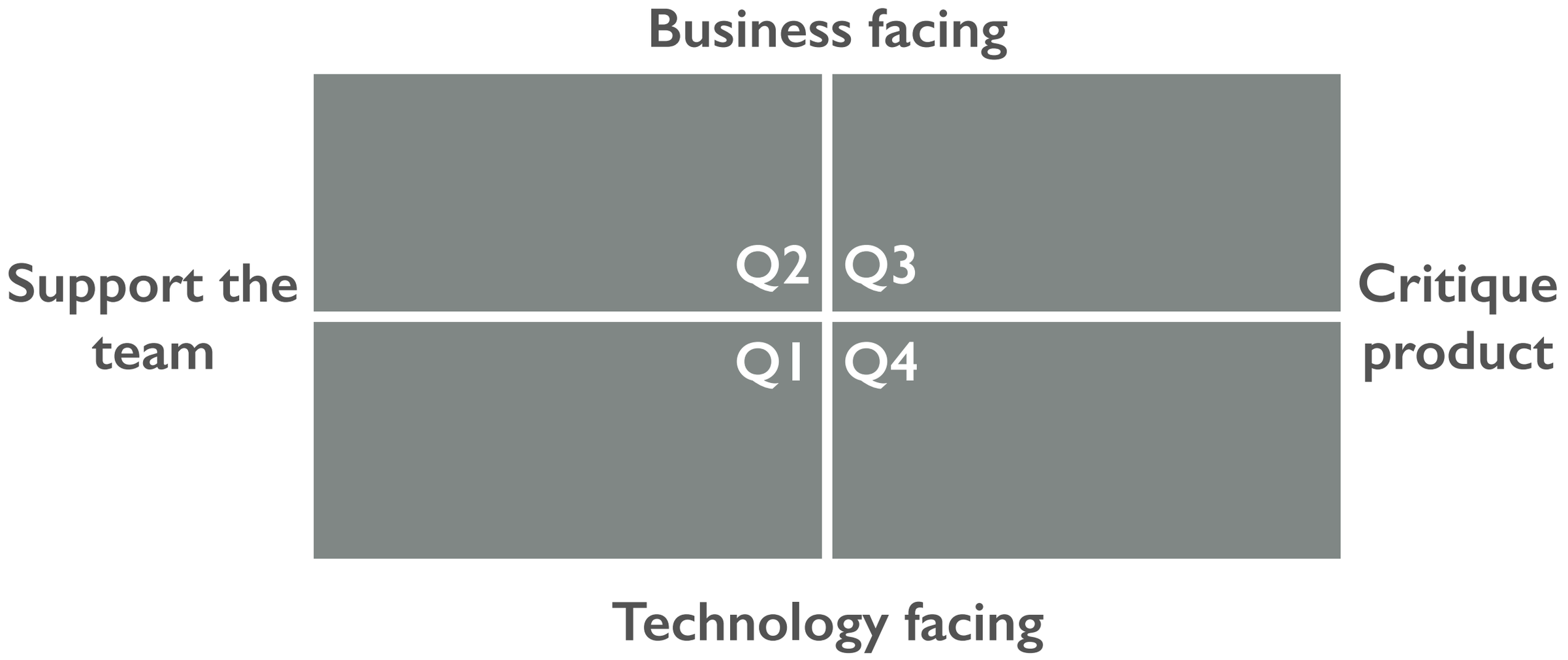
Agile testing quadrants: Q3



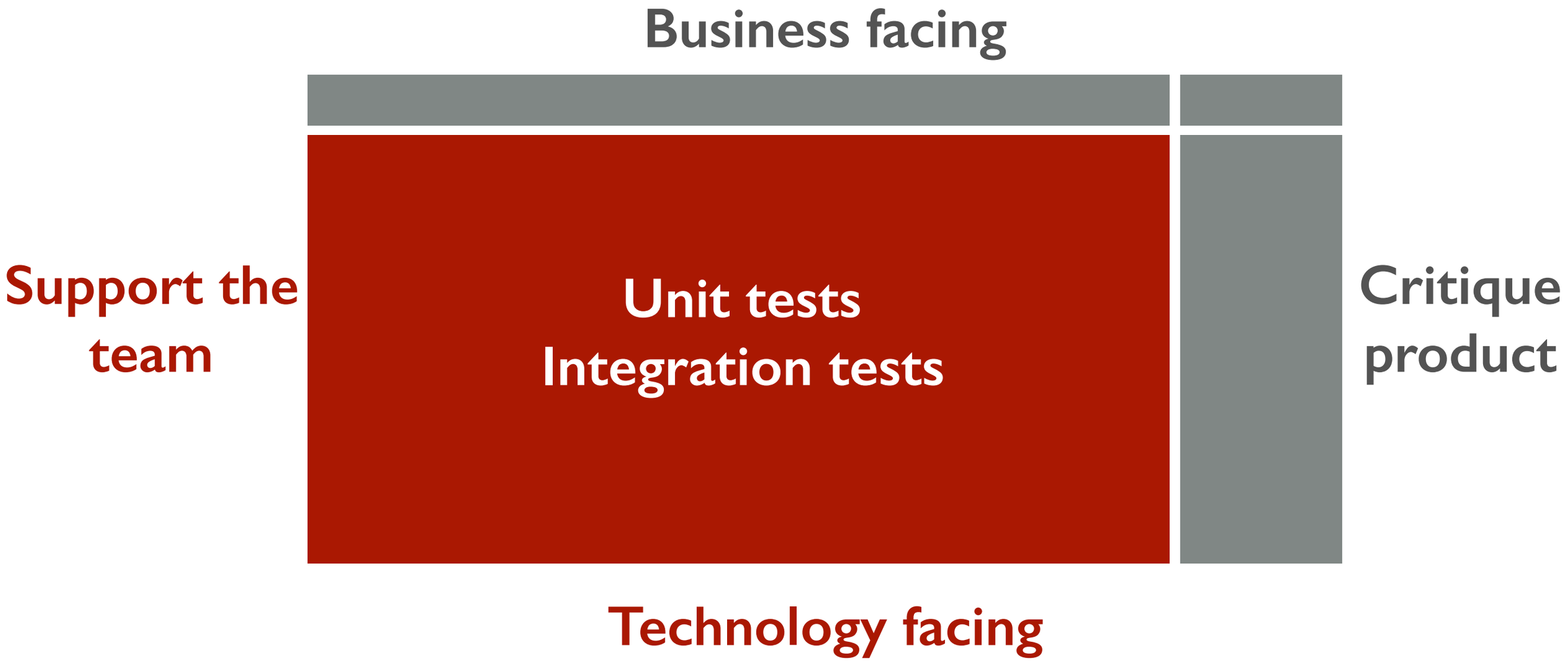
Agile testing quadrants: Q4



Agile testing quadrants



Agile testing quadrants: Q1



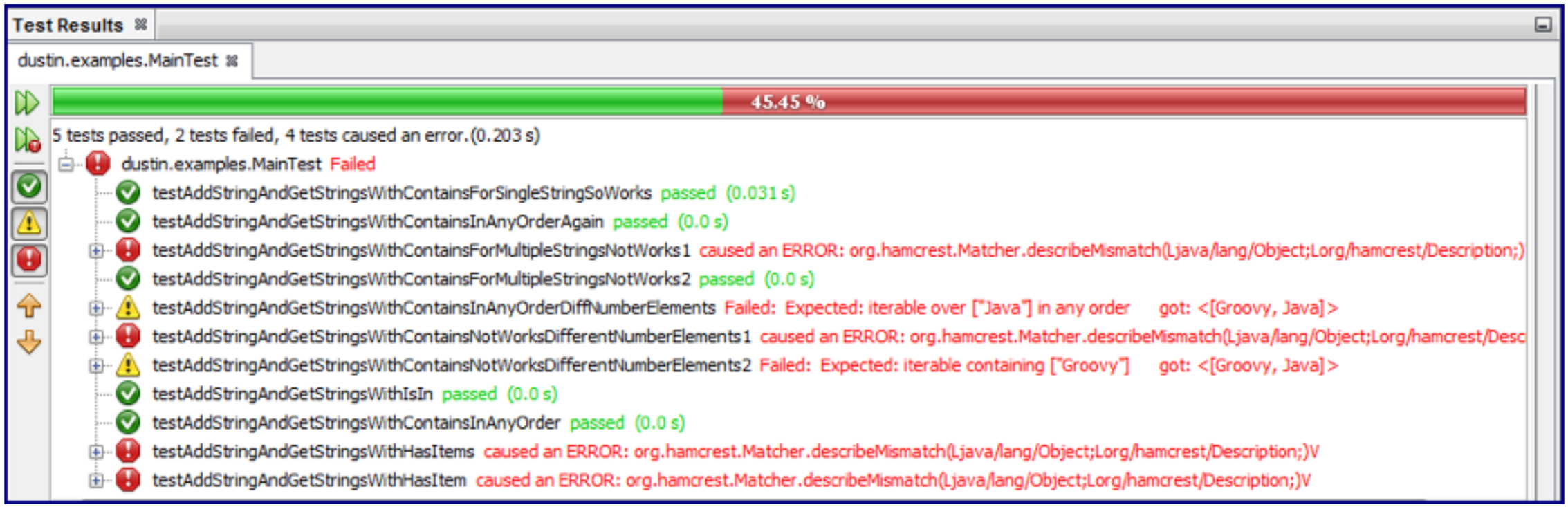
Unit tests: testing is not an after-the-fact activity

- **Writing tests** if not only about checking that a piece of code is doing what it is supposed to do.
- **Writing unit tests** helps us **design** better software (more modular, easier to maintain, easier to evolve). This is the core idea of Test-Driven Development (TDD).
- Writing unit tests, i.e. building a **test harness**, allows us to **refactor** our code with confidence.
- **If unit tests help us design, could other tests help us specify the... behaviour of our system? Could they also give us confidence that we don't impact end-users with our changes?**

A harness can be useful and cool.

Whenever *they modify their code, developers can re-run the tests and get a "**green light / red light**" status in **milliseconds or seconds**. Even if there are thousands of unit tests.*

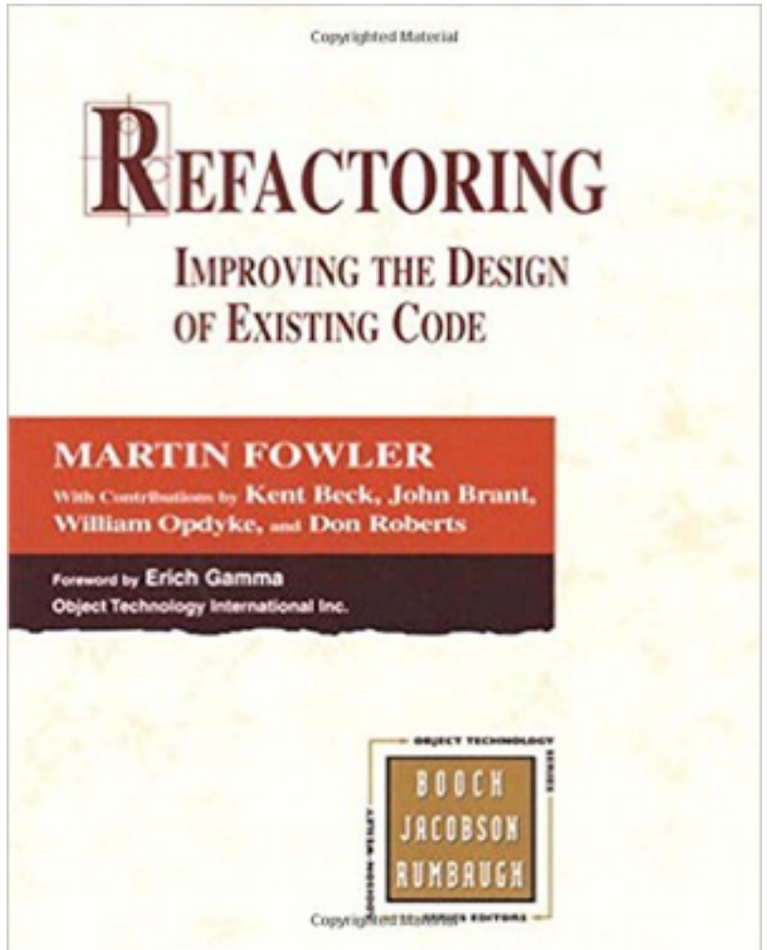
*A unit test **MUST BE extremely fast**. For this reason, a unit test MUST NOT do slow IO operations (no network calls, very few file operations).*



Unit tests enable refactoring

"Refactoring is a disciplined technique for ***restructuring an existing body of code****, altering its internal structure* ***without changing its external behavior****."*

http://www.refactoring.com/



*The **code works** (all tests are green) **but is a mess**. It is difficult to maintain and makes the development of new feature slow.*

*The code is now **nicely structured**. It is meant to do the same thing as before (no behavior change). Because all my unit tests are still green, I have some confidence that this is true.*

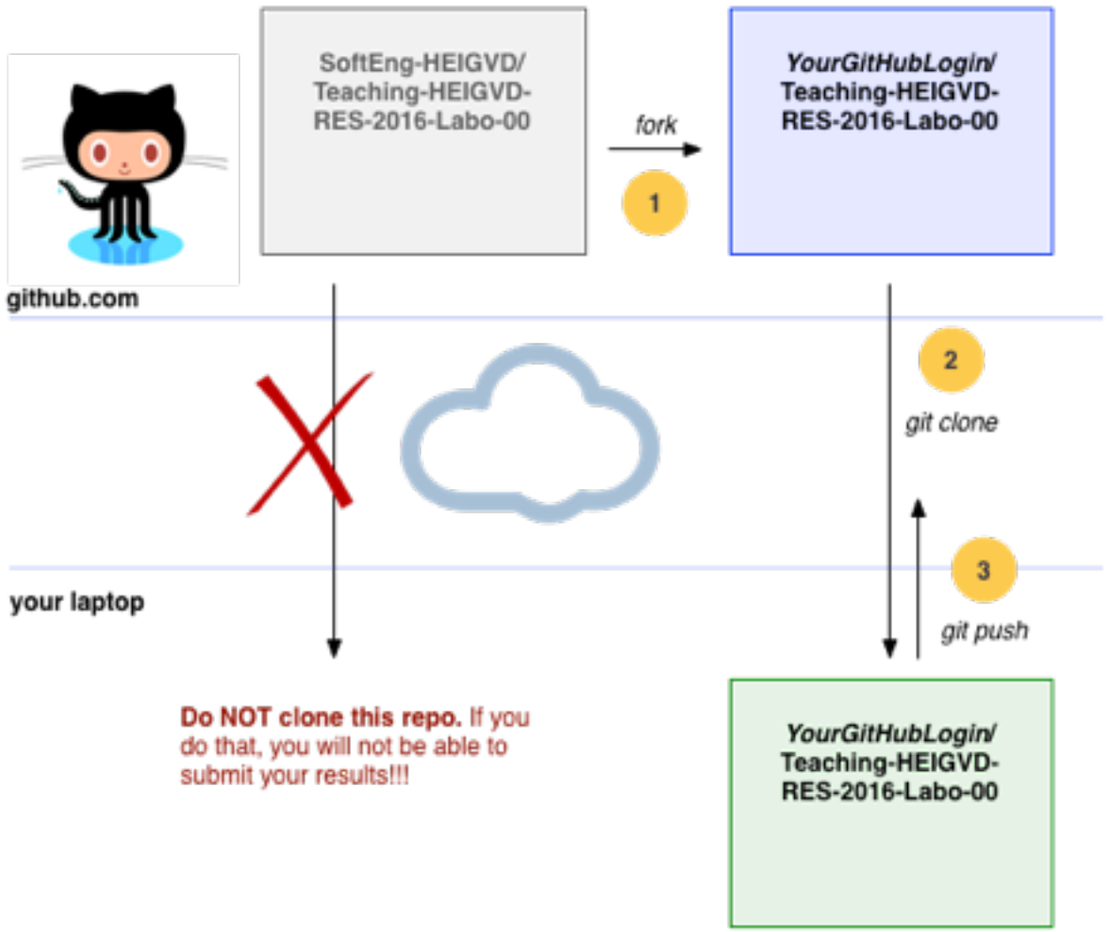
testArguments passed (3.592 s)
testInputOutput passed (0.924 s)
testWelcome passed (2.208 s)
testQuote passed (8.346 s)
testSubProjects passed (3.608 s)
testPi passed (0.598 s)
testFreeway passed (0.012 s)
testFractal passed (15.9 s)
testLexYacc passed (1.363 s)
testMP passed (4.338 s)
testHello passed (0.012 s)
testHelloQtWorld passed (0.009 s)
testProfilingDemo passed (0.025 s)

testArguments passed (3.592 s)
testInputOutput passed (0.924 s)
testWelcome passed (2.208 s)
testQuote passed (8.346 s)
testSubProjects passed (3.608 s)
testPi passed (0.598 s)
testFreeway passed (0.012 s)
testFractal passed (15.9 s)
testLexYacc passed (1.363 s)
testMP passed (4.338 s)
testHello passed (0.012 s)
testHelloQtWorld passed (0.009 s)
testProfilingDemo passed (0.025 s)

Integration tests

- Integration Tests are another type of **automated tests**.
- They validate the **interactions between several units**.
  - Does my “prediction algorithm” work with my “database service”?
  - Does my “web front-end” work with my “REST API”?
- Integration tests are **slower than unit tests**. We do not run them as often as unit tests (not to slow developers down), but we run them very often (at least before sharing one’s code with the team).

Do you want to see/try it in practice?



https://github.com/SoftEng-HEIGVD/Teaching-MSE-SEA-2017-Instruments

Description of the lab - first round

- Students have to **fork**, then **clone** a **GitHub** repository.
- The repository contains very little code (interfaces and some classes with partial implementations).
- The repository contains an **executable specification**: a list of **unit tests** that describe the behaviour that the application should have at the end of the implementation.
- Students implement the required classes.
- They use **apache maven** and **JUnit** to build the code and run the tests. You can think ot that as a “mini continuous integration pipeline” (it is actually the “commit” phase of a full pipeline).
- In the end, they have a list of green unit tests.

```java
@Test
public void thereShouldBeAnIInstrumentInterfaceAndATrumpetClass() {
  IInstrument trumpet = new Trumpet();
  assertNotNull(trumpet);
}

@Test
public void itShouldBePossibleToPlayAnInstrument() {
  IInstrument trumpet = new Trumpet();
  String sound = trumpet.play();
  assertNotNull(sound);
}

@Test
public void aTrumpetShouldMakePouet() {
  IInstrument trumpet = new Trumpet();
  String sound = trumpet.play();
  Assert.assertEquals("pouet", sound);
}

@Test
public void aTrumpetShouldBeLouderThanAFlute() {
  IInstrument trumpet = new Trumpet();
  IInstrument flute = new Flute();
  int trumpetVolume = trumpet.getSoundVolume();
  int fluteVolume = flute.getSoundVolume();
  Assert.assertTrue(trumpetVolume > fluteVolume);
}
```

```java
@Test
public void thereShouldBeAMethodToAddIntegers() {
  Application application = new Application();
  int sum = application.add(40, 2);
  assertEquals(42, sum);
}
```

Tests passed: 80.00 %

4 tests passed, 1 test failed.(0.016 s)

- ch.heigvd.res.lab00.ApplicationTest Failed
  - thereShouldBeAClassNamedApplication passed (0.012 s)
  - thereShouldBeAGetterForMessage passed (0.0 s)
  - theDefaultValueForMessageShouldBeDefined passed (0.0 s)
  - theGetterForMessageShouldReturnTheCorrectValue passed (0.0 s)
  - thereShouldBeAMethodToAddIntegers Failed: expected:<42> but was:<80>

```
expected:<42> but was:<80>
java.lang.AssertionError
 at org.junit.Assert.fail(Assert.java:88)
 at org.junit.Assert.failNotEquals(Assert.java:834)
 at org.junit.Assert.assertEquals(Assert.java:645)
 at org.junit.Assert.assertEquals(Assert.java:631)
 at ch.heigvd.res.lab00.ApplicationTest.thereShouldBeAMethodToAddIntegers(ApplicationTest.java:53)
```

```java
public int add(int a, int b) {
   return a * b;
}
```

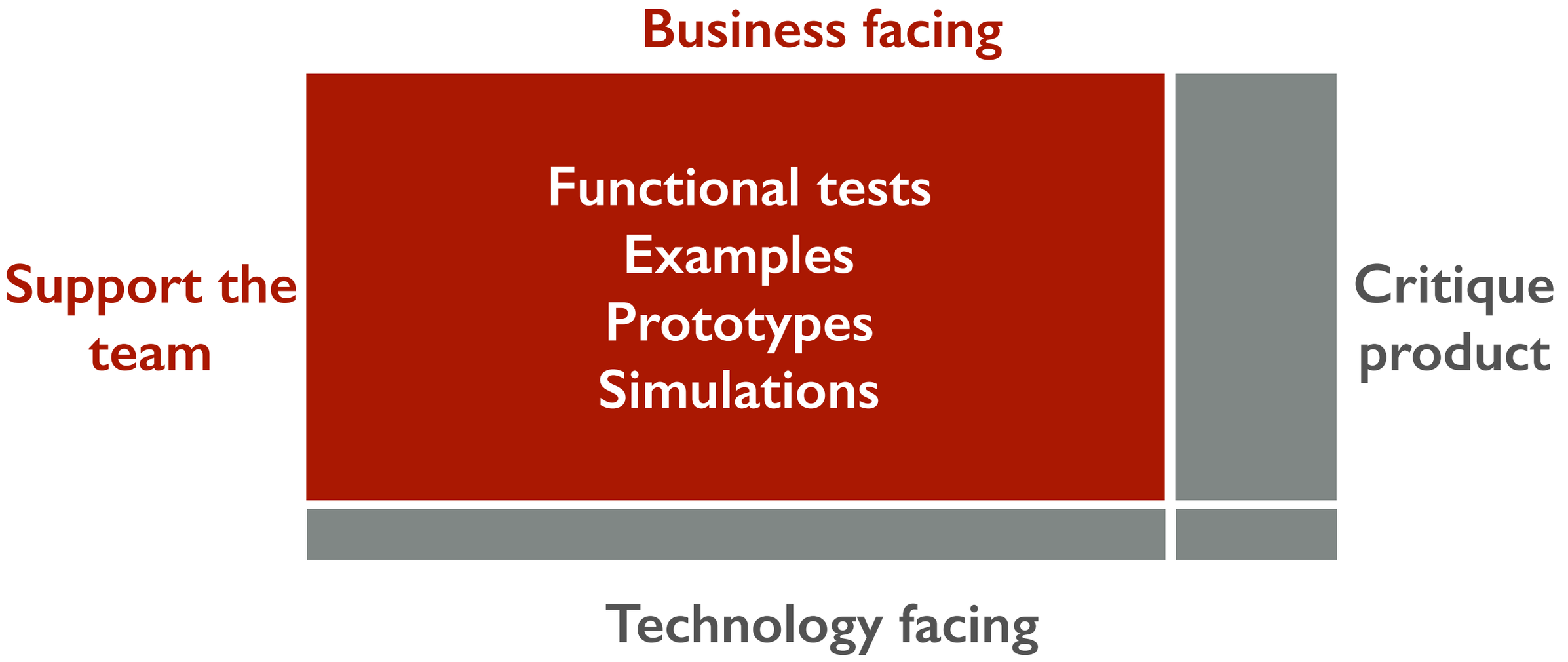
Tests passed: 80.00 %

4 tests passed, 1 test failed.(0.016 s)

- ch.heigvd.res.lab00.ApplicationTest Failed
  - thereShouldBeAClassNamedApplication passed (0.012 s)
  - thereShouldBeAGetterForMessage passed (0.0 s)
  - theDefaultValueForMessageShouldBeDefined passed (0.0 s)
  - theGetterForMessageShouldReturnTheCorrectValue passed (0.0 s)
  - thereShouldBeAMethodToAddIntegers Failed: expected:<42> but was:<80>

```
expected:<42> but was:<80>
java.lang.AssertionError
 at org.junit.Assert.fail(Assert.java:88)
 at org.junit.Assert.failNotEquals(Assert.java:834)
 at org.junit.Assert.assertEquals(Assert.java:645)
 at org.junit.Assert.assertEquals(Assert.java:631)
 at ch.heigvd.res.lab00.ApplicationTest.thereShouldBeAMethodToAddIntegers(ApplicationTest.java:53)
```

Next week


Business facing
Functional tests
Examples
Prototypes
Simulations
Support the team
Critique product
Technology facing

MSE | MASTER OF SCIENCE IN ENGINEERING

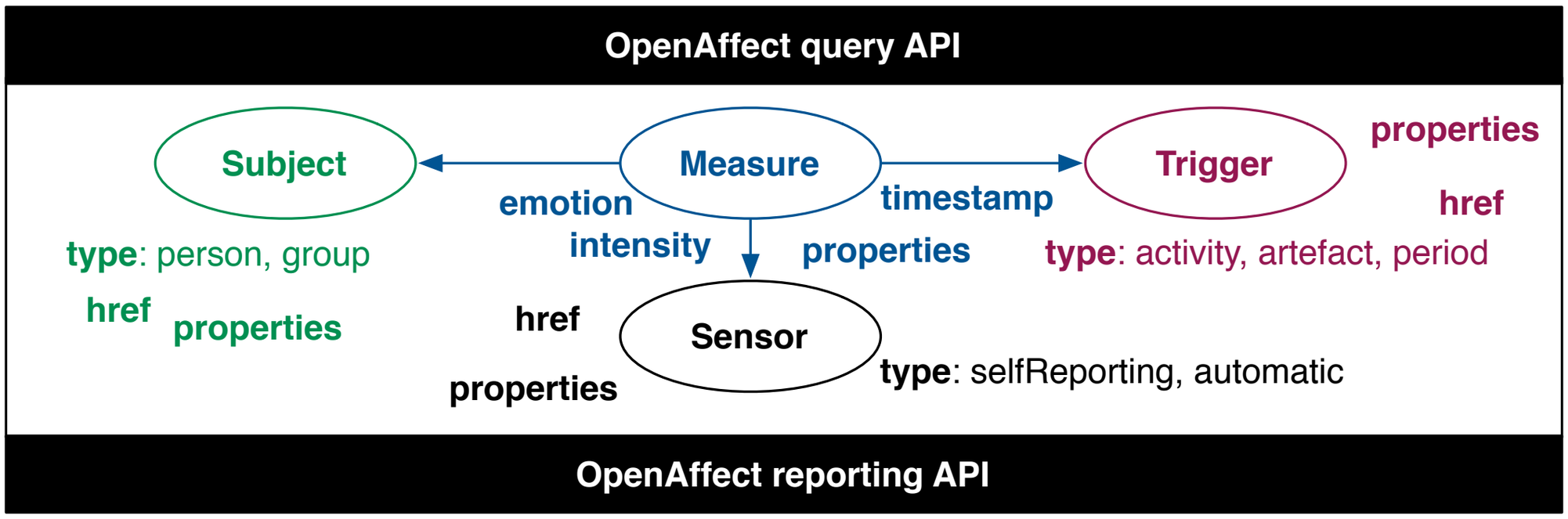
Open Affect

MSE | MASTER OF SCIENCE IN ENGINEERING

MSR studies
(across soft repositories)

Analytics dashboards
(trends, insights)

Information radiators
(WoT, notifications)



Research support tools
(labelling tools, machine learning, etc.)

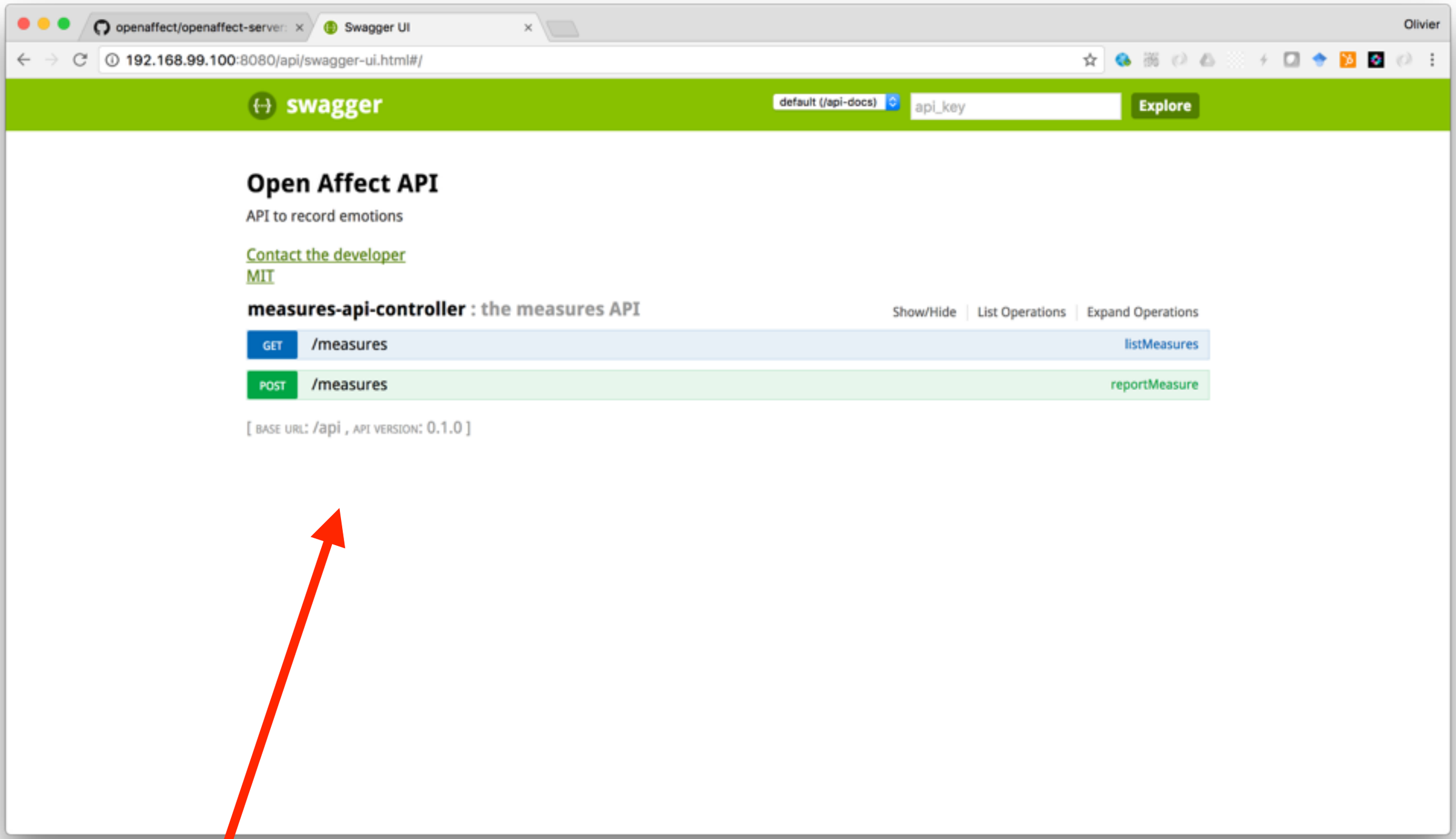
Software engineering tools
(issue trackers, IDEs, git tools, etc.)

Quantified-self tools
(mobile apps, "agile bots", retrospective tools)

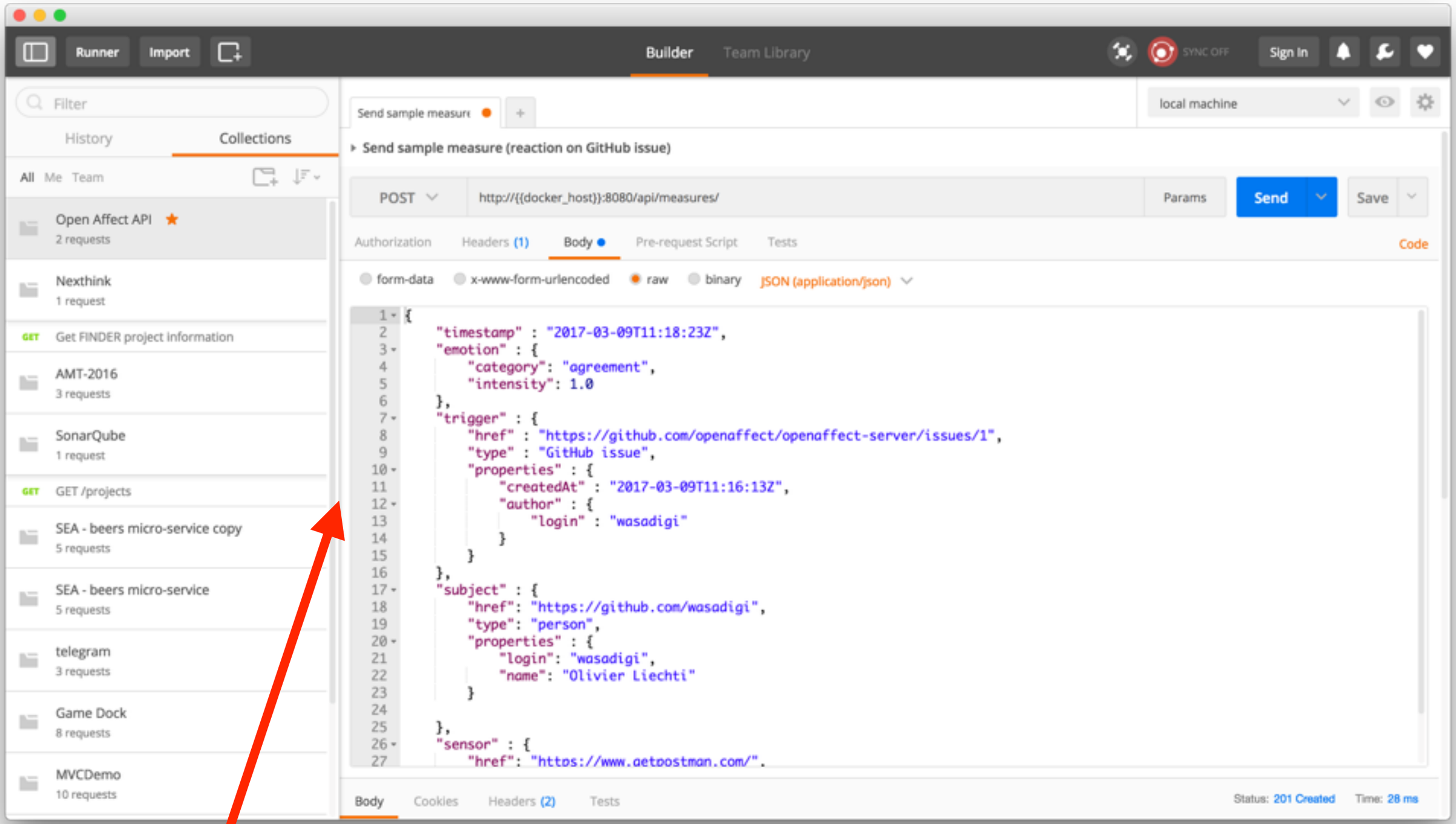
- **Sensors** report **measures** indicating that **subjects** have felt **emotions** about **triggers**.
- *A **git plugin** reports a **measure** indicating that **Bob** has felt **pride** about **commit e77f23**.*
- *A **mobile app** reports a **measure** indicating that **Alice** has felt **joy** about the **sprint 32**.*
- *A **Jira plugin** reports a **measure** indicating that the **John** has felt **anger** about **bug OA-283**.*

```
{
     "timestamp": now,
     "emotion": {
          "category": "joy",
          "intensity": 1.0,
          "properties": {}
     },
     "subject": {
          "href": "http://people",
          "type": "",
          "properties": {}
     },
     "trigger": {
          "href": "https://avalia.atlassian.net/rest/api/latest/issue/AVA-2",
          "type": "story",
          "properties": {
          }
     },
     "sensor": {
          "href": "http://emotion4jira.io/plugins/jira",
          "type": "selfReporting",
          "properties": {}
     },
     properties: {}
}
```

- **Sensors**, **subjects** and **triggers** are "**resources**", identified by a URL.
- They are typically **managed by external systems** (GitHub, Jira, etc.)
- **Descriptive properties** can be attached to resources, so that they can be **queried directly** on the Open Affect server.
- The initial list of **emotions (categories)** has been defined as: *admiration, anger, disappointment, distress, fear, fears-confirmed, gloating, gratification, gratitude, happy-for, hate, hope, joy, love, pity, pride, relief, remorse, reproach, resentment, satisfaction, shame*.



- I have an online, interactive documentation of the REST API exposed by the Open Affect server. It has been generated during the build process.



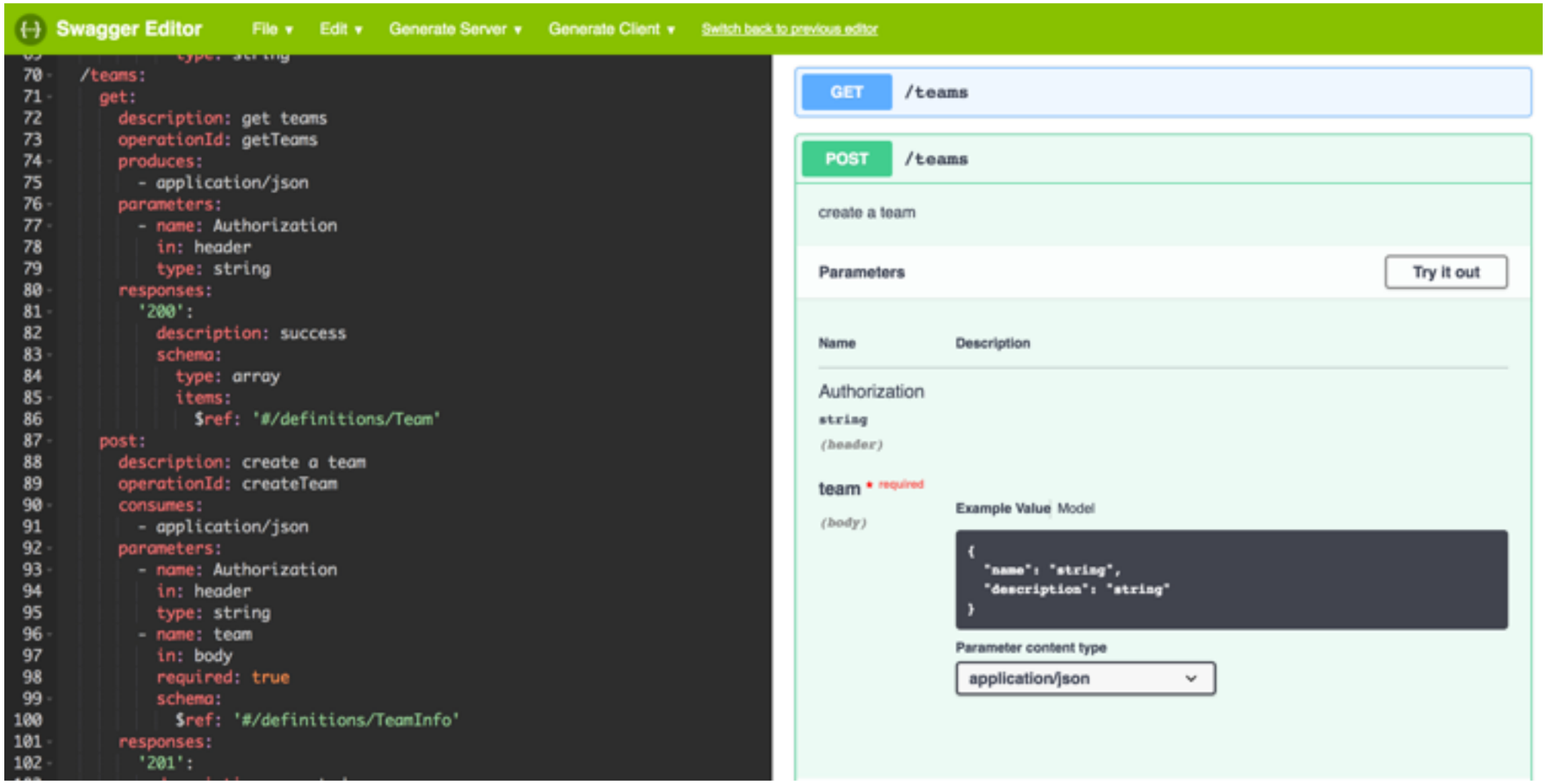
- I can use Postman and the provided sample collection (in the git repo, look for examples/OpenAffectAPI.postman_collection) to send API requests.

MSE | MASTER OF SCIENCE IN ENGINEERING

Getting started with Swagger

API specification with Swagger

Interactive documentation



Top-Down

code generation

VS

Bottom-Up

annotations

MSE | MASTER OF SCIENCE IN ENGINEERING

Editor v2

vs

Editor v3

vs

codegen

MASTER OF SCIENCE
IN ENGINEERING

Step 1: describe

Let's look at an example

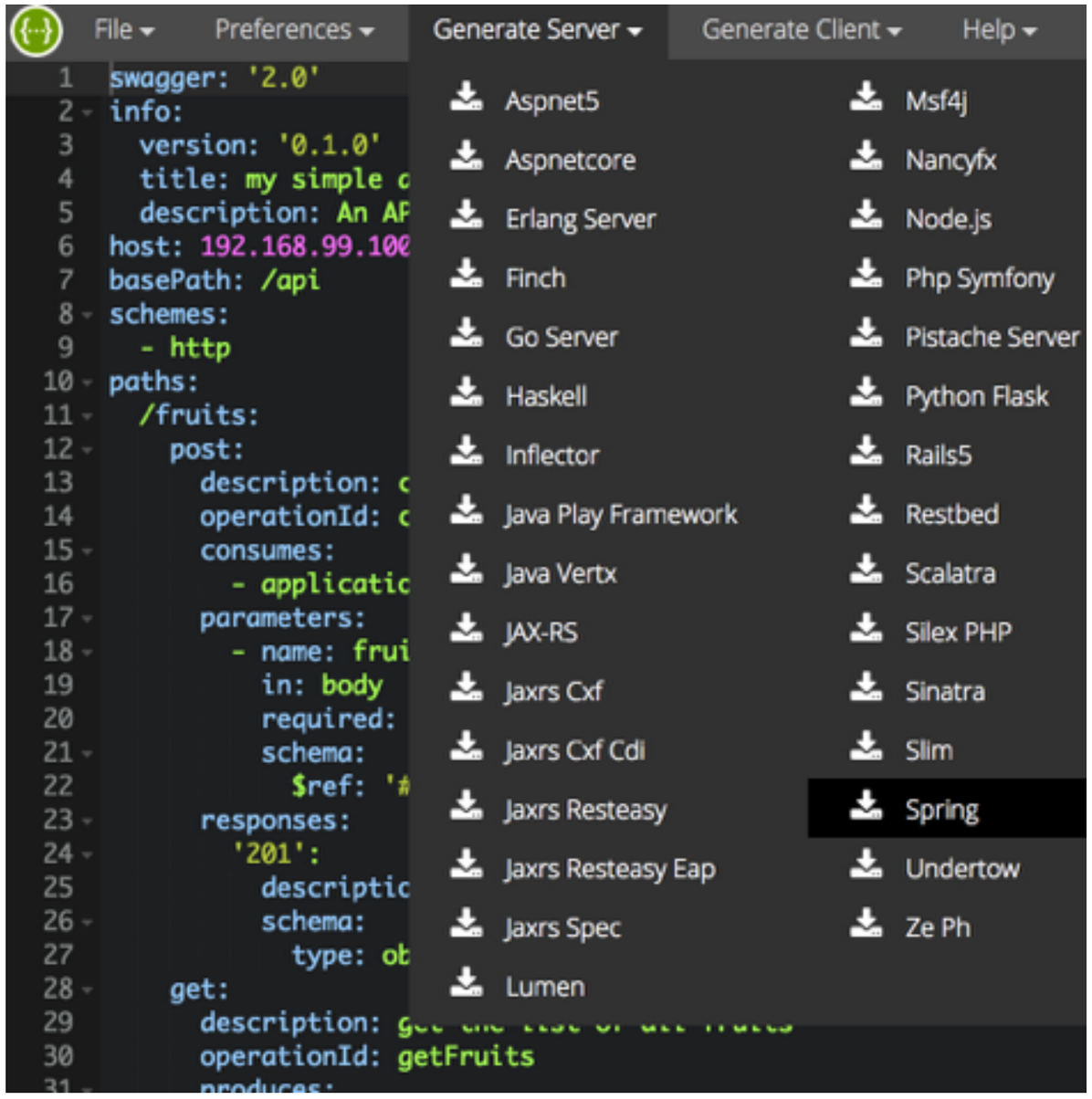
- **Clone our repo**: https://github.com/AvaliaSystems/TrainingREST
  - Checkout the **swagger-intro branch**
  - Open the **./swagger/examples/fruits-api.yml** file, copy content
- Open the **Swagger Editor v2**: http://editor2.swagger.io, paste content
- Read the **specification**, look at the interactive **documentation**

Resources, operations and types

```
paths:
  /fruits:
    post:
      description: create a fruit
      operationId: createFruit
      consumes:
        - application/json
      parameters:
        - name: fruit
          in: body
          required: true
          schema:
            $ref: '#/definitions/Fruit'
      responses:
        '201':
          description: created
          schema:
            type: object
```

```
definitions:
  Fruit:
    type: object
    properties:
      kind:
        type: string
      colour:
        type: string
      size:
        type: string
```

Step 2: implement




```
spring-server
├── README.md
├── pom.xml
└── src
    └── main
        ├── java
        │   └── io
        │       └── swagger
        │           ├── RFC3339DateFormat.java
        │           ├── Swagger2SpringBoot.java
        │           ├── api
        │           │   ├── ApiException.java
        │           │   ├── ApiOriginFilter.java
        │           │   ├── ApiResponseMessage.java
        │           │   ├── FruitsApi.java
        │           │   ├── FruitsApiController.java
        │           │   ├── NotFoundException.java
        │           ├── configuration
        │           │   ├── HomeController.java
        │           ├── SwaggerDocumentationConfig.java
        │           └── model
        │               └── Fruit.java
        └── resources
            └── application.properties

9 directories, 14 files
```

Let's generate Java from the the spec

- In the editor, go to "**Generate Server**", "**Spring**"
- Unzip the skeleton and open the project in your **IDE**
- Fix **dependencies** in the **pom.xml** file
- Configure the **maven plugin** in the **pom.xml** (depends on your IDE)
- **Run**, either from command line (mvn spring-boot:run) or the IDE.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <!--<scope>provided</scope>-->
</dependency>
```

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <fork>true</fork>
    </configuration>
</plugin>
```

```
public class Fruit   {
  @JsonProperty("kind")
  private String kind =
null;

  @JsonProperty("colour")
  private String colour =
null;

  @JsonProperty("size")
  private String size =
null;
...
}
```

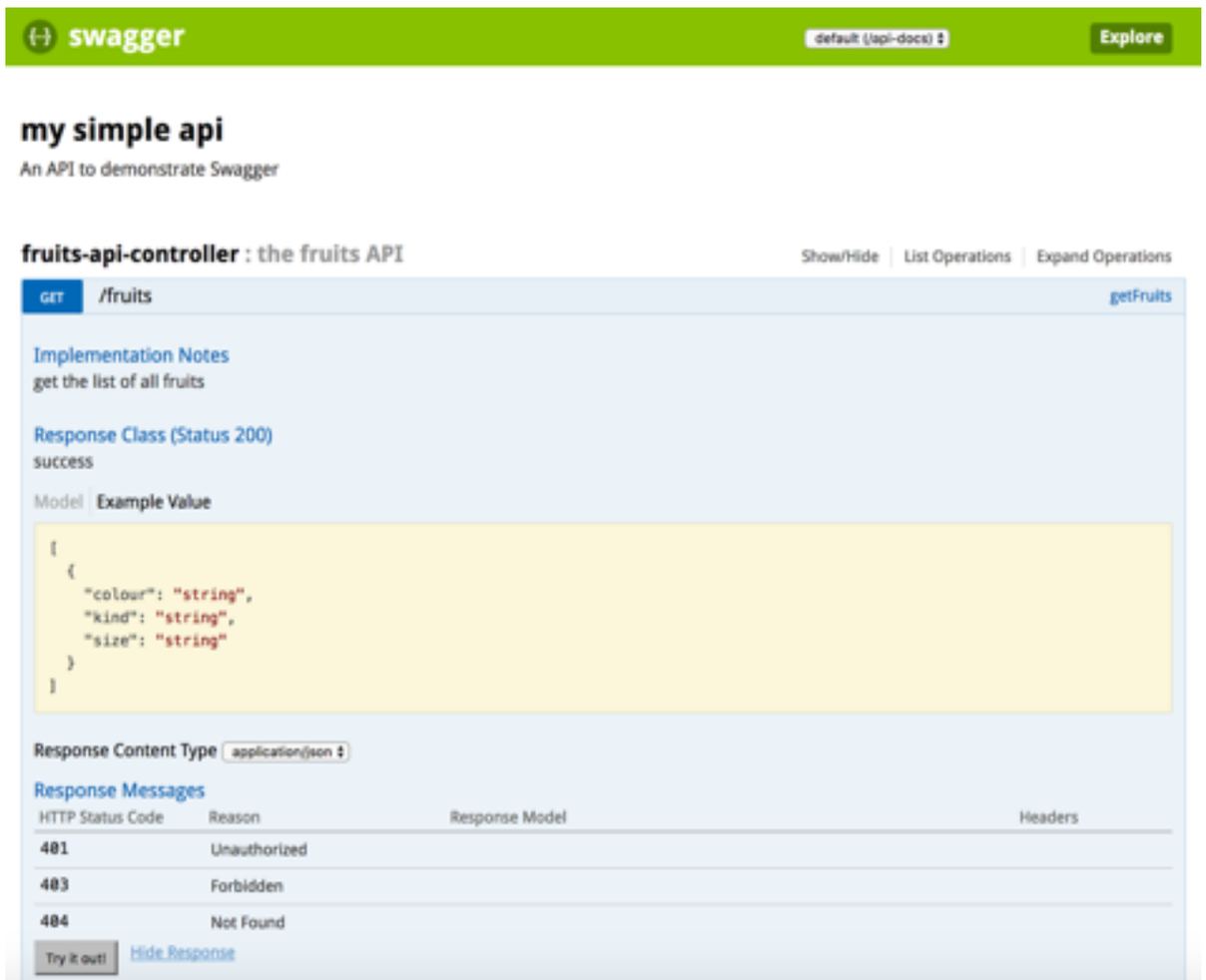
https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-requestmapping

```
@Controller
public class FruitsApiController implements FruitsApi {

    public ResponseEntity<Object> createFruit(@ApiParam(value = "" ,required=true )  @Valid
@RequestBody Fruit fruit) {
        // do some magic!
        return new ResponseEntity<Object>(HttpStatus.OK);
    }

    public ResponseEntity<List<Fruit>> getFruits() {
        // do some magic!
        return new ResponseEntity<List<Fruit>>(HttpStatus.OK);
    }

}
```

Access documentation in the browser

- In the editor, go to "**Generate Server**", "**Spring**"
- Unzip the skeleton and open the project in your **IDE**
- Fix **dependencies** in the **pom.xml** file
- Configure the **maven plugin** in the **pom.xml** (depends on your IDE)

http://localhost:8080/api/swagger-ui.html



Add persistence with Spring Data

- https://spring.io/guides/gs/accessing-data-jpa/
- Update dependencies in pom.xml
- Add a Fruit entity (DTO vs Entity!!)
- Add a Repository
- Inject dependency on Repository into API controller

```
<dependency>
    <groupId>org.springframework.boot</
groupId>
    <artifactId>spring-boot-starter-data-
jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

```
public interface FruitRepository extends
CrudRepository<FruitEntity, Long>{
}
```

```
@Entity
public class FruitEntity implements
Serializable {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private long id;

    private String kind;
    private String size;
    private String colour;

    public long getId() {
        return id;
    }

    public String getKind() {
        return kind;
    }
...
}
```