# 2. Standards

## 2.1 API Development Lifecycle

There are multiple API development methodologies used by various vendor platforms and development teams. Traditional methodology applied to any general software development is to plan, design, develop, test, and deploy a software component or platform, following a basic waterfall approach to all phases of the development lifecycle. However, there are many flaws with this traditional lifecycle approach to development, especially when focusing on microservice and API development. The key flaws identified with the traditional, waterfall, development lifecycle, are:

* Poor fit with the new Agile project methodologies. Waterfall project methodologies are inflexible to requirements and design changes and are typically geared towards large, monolithic projects, rather than towards microservices based architectures which can be delivered iteratively in smaller and quicker releases
* API contracts, and hence API specifications, are not well defined up front. This would cause delays for microservice consumers to start the integration with your microservice or API, until after the Development phase of the project is completed.
* Testing is performed only at later stages of the development lifecycle, post design and development stages, and not realizing the opportunity to test early, often, and iteratively

As a result of our analysis of multiple development lifecycle methodologies, the recommendation is to follow these key guidelines in all of the API development:

* Design First instead of Build First
* Follow Test Driven Development
* Create API Contracts and Mocks Early
* Iterate Frequently and Often
* Engage Your API Consumers

Each of the following key guidelines is elaborated further in the sections below.

### 2.1.1 Design First instead of Build First

There are two well-established approaches when it comes to API development:

1. **Design First**: Create a well-defined API contract, following Open API or RESTful API Modelling standards, such as Swagger or RAML, from which the code is then developed and tested. Mulesoft platform has been designed specifically with the Design First approach in mind. The API Contract is distributed to service consumers before the API is developed, so they can begin developing their consumption design.
2. **Code First**: Based on the business plan, API is directly coded, from which the Swagger or RAML API contract is then generated.

The biggest reason to go with the Design First approach is when the API audience are external consumers and partner platforms. In such a case, the API is a key distribution channel that your API consumers can use to consume your services provided, and a solid design plays a key role in your service adoption. The API contract is the cornerstone of the API design, and, as such, can act as the central draft the keeps all of the developers aligned on what your API’s objectives are, and how the API resources are exposed. Identifying bugs and issues with the API architecture and design becomes easier once a formal API contract is documented in a form of Swagger or RAML. Furthermore, mocking an API response becomes as straight-forward as taking the API contract and adding sample data values for each response object and attribute returned back in the API operation responses.

Code First approach becomes important when speedy code delivery is of utmost importance and the team has very strict and fast-paced timelines to deliver the software component or service.

When using Mulesoft Anypoint Platform, Design First approach is the best practice approach to all API development. A RAML contract is first created within the Design Center or using Anypoint Studio, published into Anypoint Platform and advertised to API consumers via Exchange. The consumers at this point can review the RAML contract and use the API Console to test out the sample inputs and outputs into the API, using Mock API capabilities.

### 2.1.2 Follow Test Driven Development

Test Driven Development (TDD) is an API development process that relies on iterative short development lifecycle:

1. Requirements are gathered
2. Requirements are turned into specific test cases
3. API is designed, mocked, developed and tested to pass the test cases defined above in one complete iteration on the API

This approach to API development is targeted primarily for small, component-based APIs that fit easily into microservice architecture. Microservice APIs are typically small functional components that are independent of the rest of the API platform

VA API Implementation projedts should follow the Test Driven Development approach for all of the APIs developed, as this approach has strong benefits over some of the traditional, Design-Develop-Test, approaches:

* Direct traceability from requirements to test cases to development components and service operations
* Strong focus on short iterative development lifecycle, with a strong fit for microservice development
* Testing is now an integrated function of all API development, as opposed to being a function of a central QA team isolated from the development lifecycle
* Testing frameworks such as MUnit, JUnit, Mocha, Jasmine, and Cucumber, have a strong fit into the overall microservices architecture

### 2.1.3 Create API Contracts and Mocks Early

Based on the guidelines mentioned above, to follow Design First, Test Driven Development, creating API contracts and mocks is an essential first step after the API requirements are delivered to your team to create an API. Either an Open API Standards contract such as Swagger 1.x or 2.x, or a RESTful API Modelling Language (RAML) contract are required to be defined and delivered to the development team and to the identified target API consumers.

Mocking API requests and responses is an important next step, after the API contract is designed. In the Mulesoft Anypoint Platform, this is achieved either automatically using an API Console inside the Design Center, or manually, by developing and deploying a mock API based on the contract defined. In other API platforms, it can be also an automated function using SwaggerUI, as an example, or a manually developed mock API solution. For manually developed mock APIs, the developer can choose to hard-code a JSON or SOAP-based response for a given API operation, or leverage a mock framework such as Mockito, for example.

### 2.1.4 Iterate Frequently and Often

API development should be an iterative process, with initial requirements collected, test cases defined, an API contract designed, and mock responses created and published as the initial version of the API. Following this mock API deployment, an actual API should then be developed by replacing mock operation responses with “live” service operations. Each API development iteration should be clearly labelled and versioned in the source code repository and the API Management console.

Mulesoft API Manager provides ability to tag APIs with versions, which should be used during each successive API iteration deployment.

Furthermore, many packaging and build frameworks such as Maven, have clearly identified version tags for each new API iteration. In Maven, pom.xml should be modified to iterate new API version as a Maven artifact version Id.

### 2.1.5 Engage Your API Consumers

Engage your API consumer early, as soon as the requirements are converted into test cases and the API contract is defined in a form of a Swagger or RAML contract. At that point, the target API consumer can review the contract. Next, develop the mock service operations and deploy as the early API iteration, for the API consumers to start integrating with your API mock service.

### 2.1.6 Development Lifecycle Stages

Following the above API development guidelines, below are the key stages of the API development. As an API developer, follow the above guidelines and make sure to complete all stages in the below lifecycle.



1. **Design Phase** includes the following:
   * Analyze existing API requirements created by business stakeholders and your target API consumers.
   * Create an overall sequence flow diagram for the API service and its operations, covering all service integration points and interactions with other services, as well as parallel/sequential/rollback requirements.
   * Identify and document key use cases to be supported by this API.

**Deliverables** produced by each developer in this phase:

1. Sequence Flow Diagram.
2. RAML or Swagger API Specification for the API.
3. Key Use Cases???
4. **Mock Phase** includes creating Mock service to provide sample request and response in the service endpoint. The service at this point should be deployed in its initial iteration with mock inputs and outputs, for target users to be able to consume that service to integrate with their end user platforms.

**Deliverables** produced by each developer in this phase:

1. Mock service operations returning sample output for successful service requests, following the RAML or Swagger API contract defined in the Design phase.
2. Mock service operations returning sample error output for negative service requests.
3. Deployment of the mock service.
4. **Test Phase** follows the test-driven development approach and requires the developers to produce test cases and steps (required for all services developed). Initial unit and functional tests should also be developed after the API initial iteration has been developed. The tests have to cover use cases identified in the Design phase.

**Deliverables** produced by each developer in this phase:

1. Test Cases and Steps as part of the overall Test Plan. Each Test Case should have traceability to the requirements.
2. Unit tests covering mock API operations (using MUnit, Mocha/Chai, JUnit, etc).
3. Functional tests to test how the APIs are to be consumed (using SoapUI, Postman, or automated Gerkin/Cucumber functional test framework).
4. **Build Phase** requires the full development of the API, replacing mock operations with “live” ones, with the complete code review and baseline check-in into the Master branch of the source code repository. At the end of this phase, the “live” API is ready for deployment into non-production environment, with the live operations (not mocks).

**Deliverables** produced by each developer in this phase:

1. Development of the source code.
2. Peer Code Review by the development lead/designated peer.
3. Deployment and validation into non-production environment of the live microservice (no longer mock).
4. **Publish Phase** requires publishing the API manually or via the configured CI/CD pipeline (e.g., Cloudbees, Jenkins, etc) to automate deploy and publish process of the API. In this phase, all required dependencies and components for this API have to also be published into the internal dependency management system such as Artifactory.

**Deliverables** produced by each developer in this phase:

1. Externalizing all test data and configurations to support multi-environment deployments.
2. Deployment and validation into non-production environments manually and via the CI/CD pipeline.
3. Execution of all the functional tests in each of the non-production environments (via SoapUI, Postman and/or automated functional test suite such as Cucumber).
4. **Operate and Engage Phases** require target API consumers and representatives of the business teams to validate the published services in the QA and UAT environments to provide feedback and identify defects.

**Deliverables** produced by each developer in this phase:

1. Resolve defects identified by others.
2. Redeploy the APIs with fixed defects across environments.