Introduction

Software development has transformed in recent years. Traditional software development methodologies are disrupted and becoming obsolete as customer awareness of technology continues to grow. We live in an application economy where customer feedback drives every aspect of business. The application economy has also transformed every business into a software business. Companies in any type of industry have more software developers today than they ever had in the past decade. Modern applications allow businesses to get closer to their customers with offerings that can be accessed 24/7, anywhere and on any device (personal computer, tablets, smart phones, etc.). This adds another layer of complexity for companies. They now need to focus equally, if not more on the complete user experience of the applications along with the core services they offer. Customers need to ‘love’ the application they are using, not just ‘like’ it. This small difference may have huge impact on revenue for a business.

Application Programming Interfaces (APIs) are key contributors to the application economy and core to modern applications. Traditional software development approaches did not allow development teams to focus on API design as the release cycles were generally long and customer feedback was usually focused on functionality of the system rather than API design. Customers did not have a chance to work with the APIs in early stages of development. This created a huge gap between the provider’s design and customer expectation. The result was dissatisfied customers which translated into loss of revenue for the company.

With the introduction of Agile software development methodology, software development teams now have an opportunity to demonstrate a working model of their application to customers multiple times during a release cycle. This allows teams to make required adjustments based on the feedback from customers, and new requirements from changing markets. This approach can be combined with technologies like service virtualization and applied to APIs, allowing customers to use the APIs even before they are developed. This forms the basis for API driven development.

API driven development allows development teams to develop applications with APIs as the key focus. The APIs used by the application are given highest priority and developed to meet customer requirements. The API forms the foundation for the other application layers like the user interface, integration points, etc. Within the Agile framework, the APIs are developed in increments and are regularly reviewed by the customers and suggestions are made. This approach works well to ensure all customer requirements are considered. However, there are still some challenges such as:

- Significant effort to include design changes
- Uncertainty around performance
- Inability to be used for integrations till the complete API is ready

I am going to focus on a practical approach to address the challenges mentioned above and achieve API-driven development with the use of service virtualization and test automation products. The focus of this article is primarily on Representational State Transfer (REST) based APIs as they are most widely used today.

Before we get into the details of the approach, I would like to provide a brief introduction about the general capabilities of service virtualization and test automation products. I will leverage CA products in my examples but the steps can be achieved using similar products.
Service Virtualization
Service virtualization products simulate constrained or unavailable systems across the software development lifecycle (SDLC), allowing developers, testers and performance teams to work in parallel for faster delivery and higher application quality and reliability. In API-driven development, service virtualization products act as the foundation by allowing developers to develop a virtual API according to the design specifications.

Test Automation
Test automation products help developers to:
- Achieve lower testing cycle times and costs with end-to-end automation.
- Improve quality by significantly reducing or eliminating manual testing efforts.

The Approach
The key to a successful API is its design, and we can have a well-designed API if we ask ourselves “What is the goal of this API?” Some examples could be:

- To improve developers’ ability to integrate with third-party systems
- To access services from mobile applications
- To compete with another product in market that has similar capabilities

Each example requires a different design approach for achieving the objectives. So, understanding the goal and designing a specification is the first step in the journey of “API-driven development.”

The following picture explains the different phases of API-driven development that can be implemented in an iterative way. Teams following Agile methodology for development will benefit more as I will explain the approach in the context of Agile development. All the steps or some of the steps described below can be implemented in each sprint/iteration.

Figure 1:
Phases of API-Driven Development
Design/Enhance API Specification:
In today's application economy, APIs can be among the biggest assets of a company. This can be achieved through a well-designed API which has the following characteristics:

- Easy to learn and use
- Easy to expand
- Simple yet sufficient to fulfil the customer’s requirement

At the same time a poorly designed API will quickly become a liability and support teams will be flooded with issues. Fortunately, there has been a lot of work done and number of modeling languages are available to describe the API design. I will briefly explain two API design techniques that I believe are easy to understand and use:

a. RESTful API Modeling Language (RAML);
b. Web Application Description Language (WADL)

RAML is a simple and succinct way of describing practically RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices.

WADL is a XML description of HTTP-based Web applications and in this context REST Web services. It models the resources and relationships provided by a service.

If you plan to use service virtualization products for API-driven development, using RAML as the preferred design language will help as most products support building virtual service from a RAML file. After selecting the design language, it’s worthwhile to spend time on understanding API design best practices that have all the characteristics mentioned earlier. A number of websites provide useful information on this topic. At the end of the design phase, you should have a RAML file with the API design that can be distributed as-is for reviews or can be used to build a virtual API.

Develop and test virtual API:
With an approved API design available, the next step is to develop a virtual API. Every service virtualization product has its own way of creating and deploying virtual services. The deployed virtual service can be tested using a set of automation tests created using CA Application Test.

This approach aligns perfectly with Agile teams that follow test-driven development, as the automated tests are available to test the real API when the development is complete.

In the following example, CA Service Virtualization is used to build a Virtual Service Image (VSI) using RAML and deploy to a Virtual Service Environment (VSE).
The following screenshots highlight the steps involved in creating a virtual service using RAML:

**Figure 2:**
Providing the RAML file in CA Service Virtualization

![Figure 2: Providing the RAML file in CA Service Virtualization](image)

a. Specify the RAML file to import the resources and transactions.

**Figure 3:**
Specifying the protocols in CA Service Virtualization

![Figure 3: Specifying the protocols in CA Service Virtualization](image)

b. Specify the Request Side protocols.
c. The virtual service image creation is complete.

d. Deploy the Virtual Service Image to a Virtual Service Environment and ensure its running.
A sample GET call from any REST client can be used to verify the status of the virtual service. In the example below, REST app in Google chrome is used to test the virtual API.

**Figure 6:**
Accessing the virtual API using REST app in Google chrome.

```json
{
  "size": 1,
  "authors": [{
    "id": 1,
    "name": "J. R. R. Tolkien"
  },
  "books": [{
    "id": 1,
    "name": "The Hobbit; or, There and Back Again",
    "isbn": "0544792822X",
    "author_id": 1
  }]
}
```
The virtual service can now be used to create automated tests using any automation tool. The automated tests can also be executed against the real API when it’s ready. Further, the virtual service can be used for performance testing. Service virtualization products offer ways to configure virtual services to mimic a real system. This allows identification of performance issues early enough to make required design changes.

In the following example CA Application Test is used to automate the validation of virtual service.

Figure 7:
Test execution in CA Application Test
Demonstrate the virtual API
In an Agile development process, at the end of each sprint teams would demonstrate a version of the virtual API to customers and stakeholders to gain valuable feedback early in the development phase.

A virtual service is a great way to demonstrate the behavior of API as it allows to provide a WYSIWYG (What You See Is What You Get) experience to stakeholders and customers. The automated tests will provide the list of scenarios validated to demonstrate the number of requirements met in a given sprint/iteration.

Customers will have a greater insight of the API even before the real API is available and can provide feedback early and often which can be incorporated in the virtual service delivered in the next increment. The features that are signed off by customers can move forward for real implementation.

Develop the real API
At the beginning of each sprint/iteration, development team starts building the real API based on the signed off features and continue to improve the virtual service based on the feedback received. The automated tests created for virtual service can be used against the real API to validate its functionality.

This cycle can be implemented in every sprint/iteration to ensure the requirements are met regularly and prevent design changes late in the development cycle.

Conclusion
There are multiple ways to introduce an API-driven development process in development teams and this article explained one of them using a combination of service virtualization and test automation products to deliver powerful APIs to customers. API-driven development is a practice that improves the overall usability and integration of products and solutions over a period of time, making companies successful in the current application economy.

References
3. RAML definition, http://raml.org/

For more information, please visit ca.com

CA Technologies (NASDAQ: CA) creates software that fuels transformation for companies and enables them to seize the opportunities of the application economy. Software is at the heart of every business, in every industry. From planning to development to management and security, CA is working with companies worldwide to change the way we live, transact and communicate – across mobile, private and public cloud, distributed and mainframe environments. Learn more at ca.com.