Automatic identification of web API change types in consumer applications

Graciela Vidal[1], Franco Herrera[2], Sandra Casas[3]

[1,2,3]Instituto de Tecnología Aplicada – Universidad Nacional de la Patagonia Austral Campus Universitario – Piloto Lero Rivera S/Nro. Río Gallegos – Santa Cruz - Argentina

gvidal@uarg.unpa.edu.ar,
fherrera@uarg.unpa.edu.ar,scasas@uarg.unpa.edu.ar

Abstract. *The evolution of the web API affects consuming applications. Its modification and/or adaptation to be compatible with API changes is usually critical, since it is manual, and requires higher costs and efforts. Previous studies record types and/or patterns of changes in web APIs. However, proposals that help developers automatically identify changes in their applications are insufficient. This work proposes an approach that allows to automatically identify a set of types of changes. The strategy is to use a dictionary of consumed APIs and automatic checking for inconsistencies with the current selection of APIs included in the dictionary.*

Resumen *La evolución de las API web afecta a las aplicaciones consumidoras. Su modificación y/o adaptación para que sea compatible con los cambios de las APIs suele ser crítica, ya que es manual, y requiere mayores costes y esfuerzos. Estudios previos registran tipos y/o patrones de cambios en APIs web. Sin embargo, las propuestas que ayuden a los desarrolladores a identificar automáticamente cambios en sus aplicaciones son insuficientes. Este trabajo propone un enfoque que permite identificar automáticamente un conjunto de tipos de cambios. La estrategia consiste en utilizar un diccionario de APIs consumidas y la verificación automática de inconsistencias con la especificación actual de las APIs incluidas en el diccionario.*

1. Introducción

Las APIs (Interfaz de Programación de Aplicaciones) permiten reutilizar componentes de software en el contexto del desarrollo de software y así mejorar la productividad [Robbes y otros, 2019]. Una API puede verse como un contrato entre un proveedor y un consumidor de una funcionalidad dada [Maalej y Robillar, 2013]. Las APIs se han convertido en grandes aliadas del desarrollo de software moderno, tanto de aplicaciones móviles como aplicaciones web.

La nueva generación de APIs, denominadas API de servicios web, ofrecen un enfoque sistemático y extensible, a diferencia de las APIs vinculadas estáticamente o locales [Curbera y otros, 2002][Vinoski, 2008]. En particular las APIs web facilitan el intercambio de datos inter e intra organizacionales disponibles a través de endpoints específicos [Dekel y Herbsleb, 2009]. Los catálogos como API Harmony, PublicAPI o

ProgrammableWeb enumeran miles de APIs web y dan cuenta de la proliferación de este tipo de recursos.

Las APIs web evolucionan y cambian por diversos motivos tales como proporcionar nueva funcionalidad, solucionar problemas de seguridad, aumentar la facilidad de uso de la API a los consumidores, etc. En [Stocker y Zimmermann , 2021] se proporcionan diversas causas que impulsan a los proveedores de APIs web a realizar cambios. La cooperación entre los proveedores y los consumidores de APIs idealmente consistiría en que los proveedores desarrollan APIs estables, proporcionando documentación detallada y útil para que los consumidores de la API la utilicen sin dificultad, y que además las mejoras adicionales de la API no afecten el funcionamiento de las aplicaciones. No obstante, esto no siempre sucede, puesto que ante la evolución de las APIs se suelen presentar incompatibilidades con las aplicaciones consumidoras, produciendo generalmente impactos negativos [R. Koçi y otros, 2019][Espinha y otros, 2014][Dig y Johnson, 2006][Li y otros, 2013].

Se ha denominado co-evolución [Eilertsen y Bagge, 2018] cuando el software del cliente debe modificarse para mantenerse compatible con los cambios de las APIs. En el caso de APIs web, puede ser crítico, en tanto las aplicaciones consumidoras pueden perder la robustez, si la evolución tiene efectos conocidos como "breaking changes" (el código del cliente se "rompe", no compila, o se ejecuta erróneamente) y requiere que se reescriba el código correspondiente [Jezek y Dietrich, 2017][Xavier y otros, 2017]. Los cambios de metadatos o cambios sintácticos y varios niveles de cambios semánticos, cuando suceden, pueden requerir un esfuerzo significativo de reparación e incluso descubrimiento a partir del fallo. Esto se ve agravado en el contexto de APIs, los componentes de software se caracterizan por presentar ciclos de lanzamiento independientes. Según [Brito y otros, 2020] los "breaking changes" tienen impactos importantes en las apps clientes. Otros inconvenientes para los desarrolladores de app consumidoras se presentan, como el hecho de que el mecanismo de informar la obsolescencia (deprecado) de un elemento de la API, no es una práctica habitual en el contexto de APIs web, según los estudios empíricos de [Yasmin y otros, 2020].

Existen estudios que registran tipos y/o patrones de cambios de las APIs web [R. Koçi y otros, 2019] [Sohan y otros, 2015][Espinha y otros, 2015], sin embargo, aún son insuficientes las propuestas que ayuden a los desarrolladores de las aplicaciones consumidoras de APIs web a identificar automáticamente los cambios en relación directa a las aplicaciones. Es notable que la evolución de las APIs locales (estáticas) cuenta con diversas herramientas que asisten con diferentes niveles de automatización al mantenimiento (la identificación, migración, refactorización de los cambios), no obstante, en el ámbito de las APIs web, estas tareas continúan siendo manuales [Lamothe y otros, 2021].

En esta dirección, este trabajo propone un enfoque que permite identificar automáticamente un conjunto de tipos de cambios que afectan a una aplicación consumidora (web o móvil). La estrategia consiste en usar un diccionario de las APIs consumidas (DAC) y un proceso que automáticamente verifica incompatibilidades y discordancias con la OAS (OpenAPI Specification) vigente de las APIs incluidas en el diccionario.

Este trabajo está estructurado de la siguiente manera, en la Sección II se describe la metodología, en la Sección III se brindan conceptos sobre evolución y tipos de cambio en la APIs web, en la Sección IV se presenta la propuesta para identificar cambios en aplicaciones consumidoras, en la Sección V se desarrollan ejemplos de uso, en la Sección VI se presentan los trabajos relacionados, en la Sección VII se exponen las limitaciones de la propuesta, finalmente en la Sección VIII se presentan las conclusiones.

2. Metodología

En este trabajo se aplicó el enfoque denominado Design Science Research (DSR) [Peffers y otros, 2007]. Generalmente se utiliza en investigaciones relacionadas con la ingeniería, la informática y los sistemas de información. DSR propone la producción de artefactos, como instancias, construcciones, modelos o métodos. La Fig. 1 presenta los pasos que se han seguido en este estudio.

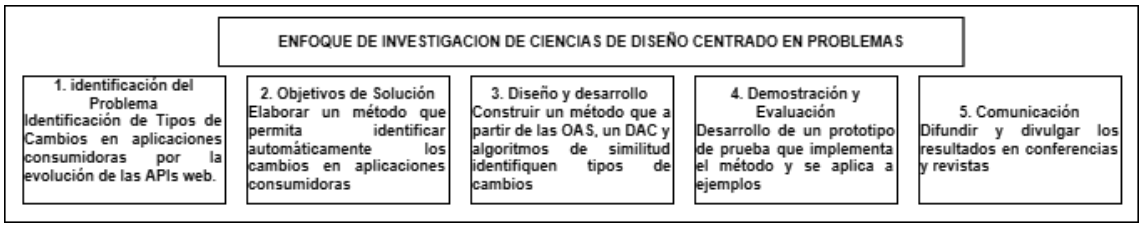

Figura 1. Pasos seguidos en el estudio.

El primer paso (1) fue la identificación del problema para el que se busca una solución. El segundo paso (2) fue establecer los objetivos de una solución inferida del problema identificado que ayudó a determinar los requisitos del marco. El tercer paso (3) fue el diseño y desarrollo del artefacto, este consiste en la definición de un diccionario de APIs web consumidas (DAC), la selección de tipos de cambios que se pretenden identificar y el proceso de verificación automática de cambios. El DAC se implementa en JSON. Los cambios resultan ser un subconjunto que ha sido reportado por estudios de otros autores. La verificación se realiza con la OAS, y es de carácter sintáctica. El cuarto paso (4) se refiere a la evaluación, para lo cual se desarrolló un prototipo de herramienta que da soporte a los artefactos presentados en el paso anterior y mediante algunos ejemplos muestra la factibilidad de la propuesta.

3. Evolución y Tipos de Cambios en APIs web

Durante el ciclo de vida las APIs experimentan iteraciones producto de su evolución, en cada fase del ciclo, los proveedores realizan modificaciones, lo que a veces hace necesario el lanzamiento de nuevas versiones [R. Koçi y otros, 2019]. Esto significa que las aplicaciones consumidoras también deben adaptarse (co-evolución) a estos cambios para continuar funcionando correctamente.

En contextos de APIs vinculadas estáticamente, algunos autores afirman que pueden surgir numerosos cambios, sin embargo, los consumidores pueden elegir no migrar a la nueva versión de la API. Cuando se trata de la evolución de una API web esto no es posible y los desarrolladores deben migrar y adaptarse a los cambios [R. Koçi y otros, 2019] [Espinha y otros, 2015].

Diversos tipos de cambios, producto de la evolución de las APIs web han sido reportados, los cuales están relacionados a diferentes componentes de una API. En la Tabla 1, se mencionan algunos que se han informado en estudios recientes, según el componente de la API al que afectan.

Tabla 1: Tipos de cambios agrupados por componente

Endpoints
▪ Modificación de endpoints - [R. Koçi y otros, 2019] ▪ Cambios en los métodos de solicitud (GET,POST,DELETE) - [R. Koçi y otros, 2019] ▪ Cambios en valores por defecto de parámetros - [Li y otros, 2013] ▪ Cambios en encabezados HTTP - [Sohan y otros, 2015] ▪ Cambio de home de API - [Sohan y otros, 2015]
Operaciones
▪ Remoción de operación /método (GET, POST, DELETE) - [Li y otros, 2013] [Fokaefs y otros, 2011] [Sohan y otros, 2015] ▪ Renombrar método - [Li y otros, 2013] [Sohan y otros, 2015] ▪ Combinación de métodos - [Li y otros, 2013] ▪ Separación de métodos - [Li y otros, 2013] ▪ Método request no soportado - [Li y otros, 2013] ▪ Cambio de post condiciones - [Sohan y otros, 2015]
Parámetros
▪ Cambios en tipo de parámetro - [Li y otros, 2013] [Espinha y otros, 2015] ▪ Modificación de parámetros - [R. Koçi y otros, 2019] ▪ Renombrar parámetros - [Li y otros, 2013] [Sohan y otros, 2015] ▪ Cambio en el formato del parámetro - [Li y otros, 2013] ▪ Cambio en los valores límites de los parámetros - [Li y otros, 2013] ▪ Cambios en los tipos de las entradas - [Fokaefs y otros, 2011] ▪ Remover parámetro - [Li y otros, 2013] [Sohan y otros, 2015] ▪ Agregar atributos en los parámetros - [Fokaefs y otros, 2011]
Campos
▪ Remoción de campos - [Sohan y otros, 2015] [Espinha y otros, 2015] ▪ Cambio en el valor de retorno - [Li y otros, 2013] ▪ Cambio en tags de XML - [Li y otros, 2013] ▪ Renombrar campos - [Sohan y otros, 2015] ▪ Cambios en los tipos de salida - [Fokaefs y otros, 2011]
Seguridad
▪ Cambios en niveles de autoridad - [R. Koçi y otros, 2019] ▪ Restricciones de acceso a la api - [Li y otros, 2013] ▪ Cambio en los protocolos de autorización - [Sohan y otros, 2015] ▪ Cambio de autenticación [Sohan y otros, 2015]
Errores
▪ Cambio en condiciones de error - [Sohan y otros, 2015]
Estructura - Formato
▪ Cambiar de formato de la respuesta (XML por JSON) - [Li y otros, 2013] ▪ Modificar la jerarquía (ubicación) de un componente en la respuesta - [Li y otros, 2013]

La evolución de las APIs web ocasiona diferentes tipos de cambios, los cuales afectarán a los consumidores de diferentes formas. Acciones como identificar el tipo de cambio, examinar el código de las aplicaciones para analizar el impacto que el cambio provoca y finalmente realizar las modificaciones para adaptarse a la nueva versión, son todas tareas que requieren tiempo y esfuerzo extra y por lo tanto son costosas.

4. Identificación de cambios en APIs web en aplicaciones consumidoras

La propuesta consiste en verificar automáticamente si las APIs web consumidas por una determinada aplicación, han experimentado un cambio y ofrecer información que reporte y proporcione detalles del mismo. El eje que fundamenta la estrategia, reside en que cuando una API web evoluciona, los cambios se reflejan en la especificación OAS. Así una nueva versión de la OAS se genera, lo cual está denotado en la misma descripción. Para realizar el proceso de identificación, se utiliza un artefacto que se denomina Diccionario de APIs web Consumidas (DAC) y la/s OAS vigentes (al momento de la verificación). A continuación, se presenta el DAC, los tipos de cambio a verificar y el proceso de verificación.

Diccionario de APIs web consumidoras

El registro de las APIs web consumidas por una aplicación en un diccionario sirve de documentación a las aplicaciones consumidoras y además permite identificar inconsistencias, incompatibilidades y cambios en las APIs. Este artefacto, debe ser creado durante el proceso de desarrollo de aplicaciones, comprende una estructura general para todas las aplicaciones. Cada aplicación dispondrá de su propio diccionario, el cual incluirá todas las APIs consumidas.

La información que describe la aplicación en el DAC, también en formato JSON, se agrupa en tres objetos: info, server y apis. El objeto info describe datos generales tales como title y description, el objeto servers indica el acceso a la página principal del sitio y apis es un vector que contiene las APIs web consumidas, ya que puede utilizar *n* APIs (n>=1). Los elementos definidos para cada API son: name, version, servers, paths (endPoints) y components. Los objetos paths y components definen la información más importante de cada API utilizada, en paths se especifican las operaciones (GET, POST, DELETE, etc.), se incluyen parameters con sus atributos (in, name, type) de cada uno de ellos. En components se definen schemas y security schema, el objeto schemas permite definir datos de entrada y salida como también sus propiedades (required, style, type, etc.) en este caso se definen campos esperados en response. Estos últimos representan los campos que el usuario utiliza de la API, de no contar con estos datos la aplicación web no estaría funcionando correctamente. En el objeto security schema se define el tipo de autenticación que requiere la API, la cual puede tomar los valores APIkey o auth2.

En la Fig. 2 se presenta la estructura del DAC para la aplicación web Kitelife[1]. El objeto info incluye la siguiente información, el nombre de la aplicación que consume la API, una breve descripción del sitio y además el número de versión de la aplicación. En este caso, se define un sitio web que permite buscar el lugar apropiado para hacer

[1] https://clayjuneau.com/Kiteflite/

volar una cometa dependiendo de las condiciones climáticas, se trata de la versión 1.0 de la aplicación.

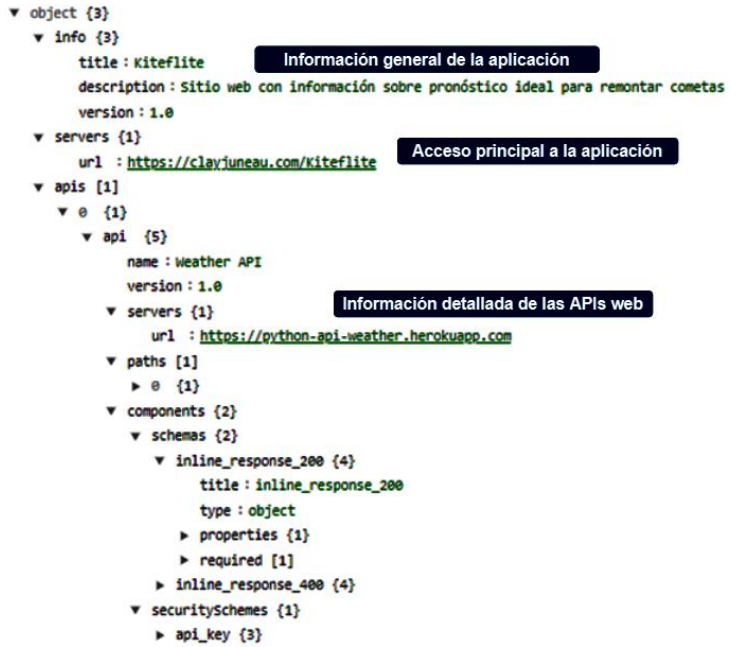

Fig.2 Estructura del DAC del sitio web Kiteflite

En el objeto servers se especifica la url de acceso a la aplicación. En el objeto apis se detallan los elementos utilizados de cada API consumida. La API utilizada es Weather API y su es versión 1.0. De cada API utilizada se registra la siguiente información: name, version y servers de acceso, los paths y components. En este DAC se define un path, luego se refiere a dos schemas y finalmente los datos de seguridad, precisamente es el tipo de autenticación requerido para acceder a la Weather API versión 1.0.

Tipos de cambios

La Tabla 2 presenta los siete tipos de cambios que este estudio aborda.

Tabla 2. Tipos de cambios y elementos OAS asociados

Tipo de Cambio	Elemento OAS
Modificación de endPoint	Elemento path
Cambio de Tipo de Propiedad	Elemento type del objeto properties
Remoción de Parámetro	Propiedad name del objeto parameter
Cambio de esquema	Propiedad name del objeto Schema
Cambio de Tipo de Dato de Parámetro	Propiedad type del objeto parameter
Cambio de Tipo de Autenticación	Propiedad type del objeto Schema Security
Modificación de Condición de Parámetro	Propiedad required del objeto parameter

Para cada uno, además se indica el elemento de la OAS que se puede analizar para identificar la ocurrencia del evento. Los mismos son un subconjunto de los mencionados en la Sección 3 y afectan a distintos elementos como los parámetros, los endPoints, componentes tales como esquemas (datos de entrada y salida) y esquemas de seguridad (autenticación).

Para verificar cada tipo de cambio se accede a cada objeto y luego a sus propiedades. Si alguna de las propiedades o sus valores definidos en el DAC son modificados o removidos de la OAS dejan de ser compatibles y significa que se produjo un tipo de cambio.

Proceso de verificación de cambios

El objetivo de este proceso es identificar automáticamente si se han producido cambios y proporcionar información sobre tales inconsistencias. En esta instancia la verificación es capaz de identificar los siete tipos de cambios antes descriptos.

El proceso requiere dos entradas, el DAC y la última versión de la especificación OAS, sobre la cual se realizará el análisis, ambos documentos en formato JSON. Luego se debe acceder a cada uno de los elementos definidos en el DAC y se lo compara con su equivalente en la OAS. Cada vez que encuentra una desigualdad, informa el tipo de cambio, el objeto sobre el cual se produce y de ser posible, los nuevos valores. En Listado 1 se describe el algoritmo que realiza el análisis de los parámetros y endPoints.

```
Entrada: dac(app) and OAS(x)
Salida : Cambio : reporte / set?

FOR EACH elemento in dac(app) where apis.api.name = x
  FOR EACH dac.path
    IF dac.apis.api.paths !found() in x
        Cambio ← Cambio + Modificacion Punto Final
    ELSE
      FOR EACH dac.apis.api.paths.parameters
        IF dac.apis.api.paths.parameters.name !found() in x
            Cambio← Cambio + Remocion de Parametro (name)
        ELSE
            IF dac.apis.api.paths.parameters.name.type < > en x
                Cambio← Cambio + Cambio de tipo de dato de parametro (name)
            ELSE
                IF dac.apis.api.paths.operationId.parameters.requerid < > en x
                    Cambio← Cambio + Cambio de condicion de parametro (name)
                endIf
            endIf
        endIf
      endForeach
    endIf
  endForeach
```

Listado 1. Algoritmo que identifica cambios en los endPoints y parámetros.

De manera similar, se han definido dos algoritmos que analizan los schemas y aspectos de seguridad.

5. Ejemplos de Uso

El proceso de verificación de cambios fue implementado en un prototipo desarrollado en Python y Django. El prototipo permite seleccionar el DAC y la OAS de la API web sobre la cual se realiza la verificación.

En la Fig.3 se presenta el prototipo, sobre la imagen a la izquierda, en primer lugar (a) se debe seleccionar el archivo correspondiente al diccionario de la aplicación consumidora, luego (b) se debe seleccionar la API web sobre la cual se realizará el

proceso. A la derecha de la imagen se presentan los resultados luego de la ejecución del proceso de verificación. En este caso, aplicado al diccionario denominado artefactoClima3.json y a la OAS de la API Weather versión 1.0. Se encontraron cuatro cambios en la OAS Weather: (a) modificación de condición de parámetro, se trata del parámetro city del endPoints /weather/get, en el cual se modificó la condición required, el nuevo valor es true, (b) remoción de parámetro, el parámetro state perteneciente al endPoints /weather no se encontró disponible, ahora se denomina states, (c) modificación de tipo de dato propiedad, el tipo de dato de la propiedad lon del esquema inline_response_200 fue modificada a string, (d) cambio de tipo de autenticación, se registró cambio en el schema de seguridad, el tipo de autenticación ahora es api_key.

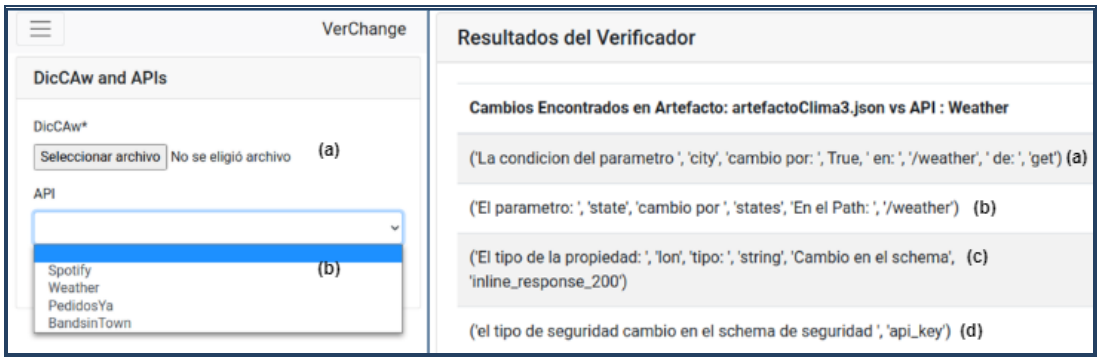

Fig.3 Prototipo de verificación y resultados

A continuación, se describen dos de los tipos de cambio identificados:

Modificación en condición de parámetro: Primero se analiza la coincidencia del nombre de la API y luego del endPoint, en este caso es "/weather". Luego se verificaron los parámetros, primero el atributo *name*, cuyo valor es "city", el cual se encontraba en la especificación, es decir que no existían anomalías hasta ese punto. Luego se analizaron las demás propiedades de este parámetro, en el caso de *required* que en el DAC está definido "false", y en la OAS fue modificado a "true". Entonces el verificador informó el tipo de cambio y el nuevo valor del campo. Con esta información el desarrollador es capaz de tomar decisiones respecto a las actualizaciones necesarias para que la aplicación consumidora continúe siendo compatible con la API utilizada. En Fig.4 se muestran las estructuras analizadas, (a) corresponde a la OAS vigente, (b) es el DAC y en (c) se presentan los resultados obtenidos por el prototipo.

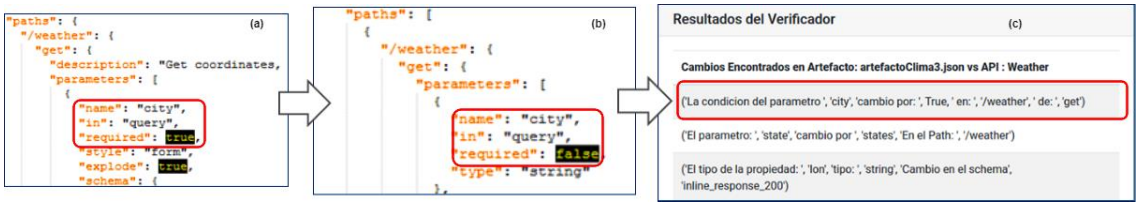

Fig.4 . Identificación del cambio: modificación de condición de parámetro

Remoción de parámetros: nuevamente el proceso se inició verificando el nombre de la API y los endPoints, luego se accedió a los parámetros definidos para el mismo, para cada uno de ellos se buscó su equivalente en la especificación OAS verificando el atributo *name*. Cada parámetro fue verificado sobre la OAS, en este caso el *name* del parámetro definido en el diccionario es "state". El mismo no se encontraba en la

especificación, por lo tanto, se informó el tipo de cambio al desarrollador. En la Fig.5 se presentan las especificaciones analizadas y los resultados obtenidos por el verificador.

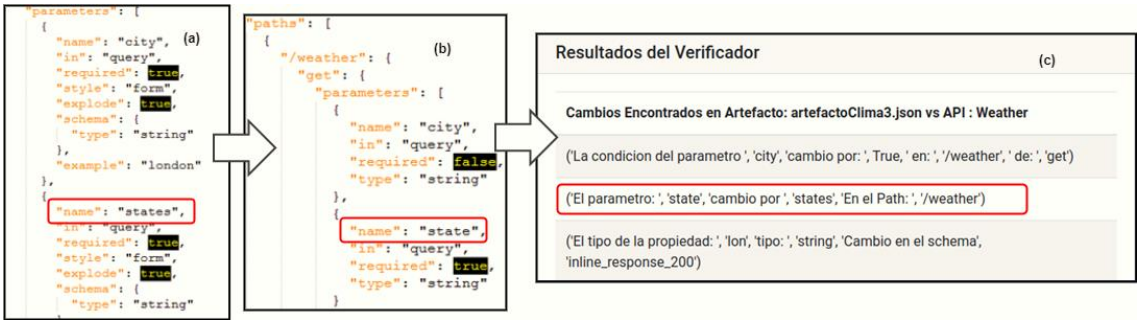

Fig.5. Identificación del cambio: remoción de parámetro

6. Trabajos Relacionados

En [Lamothe y otros, 2021] se presenta una revisión sistemática de la literatura (RSL) referente a la evolución de las APIs, que analiza 369 artículos en el periodo 1994-2020, con el objetivo de examinar la evolución del tema e identificar los desafíos de investigación. El estudio demuestra claramente que una mayor cantidad de trabajos de investigación en APIs locales, y particularmente en relación a las APIs de Java y Android. En consecuencia, existen mayores evidencias y soluciones a problemas de este nicho, se reportan numerosas herramientas, técnicas, recomendaciones con los objetivos de documentación, migración, adaptación, deprecación, etc. La evolución y cambios de APIs locales por su naturaleza estática orientan a soluciones y enfoques que no son aplicables al mismo fenómeno, pero si en el contexto de APIs web. La RSL destaca como uno de los desafíos actuales y futuros, la propuesta de herramientas que ayuden a los desarrolladores que usan APIs web a manejar la evolución, ya que se ha realizado escaso esfuerzo en este tema [Wittern, 2018].

En [Stocker y Zimmermann, 2021], se presenta una encuesta a proveedores de APIs web, cuyo objetivo fue comprender las razones que obligan a los arquitectos de software y desarrolladores de API a cambiar una API y también las consecuencias de dichos cambios. Los autores llegan a las siguientes conclusiones: a) las modificaciones a las API suelen ser causadas principalmente por cambios en los requisitos funcionales, impulsados por el cliente o el proveedor, y menos por problemas no funcionales. b) en cuanto al segundo factor de cambios (no funcionales) se indicó que ocurren problemas de calidad y también dan como resultado cambios en la API. La usabilidad y mantenibilidad, seguidos de rendimiento y escalabilidad, son los atributos de calidad que más cambios impulsan. c) en cuanto a las tácticas de cambio y mitigación que los proveedores aplican al realizar los cambios en las APIs que ofrecen, estas son, actualizar la especificación o el código, actualizar la documentación de la API, ajustan el número de versión de la API, y ajustar los clientes de la API. Luego los autores proponen un catálogo de refactoring candidatos pero para desarrolladores de APIs.

El estudio de [R. Koçi y otros, 2019] está orientado a la identificación y clasificación de tipos de cambios, que se producen en una API e investigan como estos cambios se reflejan en distintos artefactos tales como la documentación, notas de versión, seguimiento de problemas, y los registros de uso de APIs. El análisis de cada paso del cambio, su implementación y el impacto sobre los consumidores permite una

amplia visión de la evolución de la API. Para esto, proponen un framework de clasificación de cambios que pueden ocurrir en una API y las causas que los originan. El Framework se limita a la identificación y clasificación de los cambios. Principalmente el trabajo se enfoca en cómo se reflejan los cambios y en qué medida son documentados.

En [Li y otros, 2013] presentan un estudio empírico sobre la evolución de las APIs web, como resultado presentan un conjunto de patrones de cambio. Estos patrones y su frecuencia son presentados a los desarrolladores como información de referencia sobre los cambios que pueden descubrir. Afirman que es posible que los patrones sirvan en un proceso de migración automatizado en un futuro. Por último, presentan un análisis comparativo con APIs locales.

En [Sohan y otros, 2015] se presenta un conjunto de recomendaciones para profesionales e investigadores basadas en un caso de estudio realizado sobre un conjunto de APIs. El mismo consiste en un esquema de codificación de tipos de cambios, para definir patrones de cambio, además identificaron las formas de documentar y comunicar los cambios. Si bien las recomendaciones se ajustan a diversas aplicaciones web que utilizan APIs, no ofrecen automaticidad que asista a los consumidores de APIs en el proceso de identificar y tratar los cambios ocasionados por la evolución.

En [Espinha y otros, 2015] sostienen que las APIs web son necesarias en el desarrollo de software moderno y al mismo tiempo reconocen que su evolución es inevitable. Por lo cual, también es necesaria la actualización del software para que siga siendo compatible con la API web que utiliza. Presentan un estudio sobre la reacción de aplicaciones móviles, ante cambios o fallas en una API web utilizadas. La experiencia sobre 48 aplicaciones y la simulación de mutaciones en API web, proporciona información sobre los enfoques de control de versiones, además realizaron entrevistas a desarrolladores.

En [Fokaefs y otros, 2011] se presenta un trabajo empírico sobre la evolución de servicios web, para esto utilizó las descripciones WSDL. Luego un algoritmo fue diseñado para identificar cambios sobre estos documentos. Finalmente, con los resultados se analizan los cambios que pueden ocurrir entre versiones de diferentes servicios, esto permite discutir los posibles efectos en el mantenimiento de estos sistemas. A diferencia de nuestro trabajo, se enfoca en especificaciones de servicios web.

Analizar los cambios y evolución de software ayuda al desarrollador en la tarea de mantenimiento de sistemas, en [Chaturvedi y otros, 2021] se presenta un trabajo en el cual aplicaron minería de cambios y de evolución a sistemas distribuidos. El mismo se enfoca en descubrir cambios provocados por la evolución de servicios. Para esto utilizan como entradas dos versiones de especificaciones de servicios, posteriormente aplican la minería de cambios. Luego proponen cuatro métricas de evolución del sistema. Ambos son la base de un modelo de análisis de evolución, además crearon una herramienta inteligente denominada gestión automática de cambios de servicios web. Las métricas pueden ser útiles en el mantenimiento de versiones y el desarrollo de futuras versiones. También pueden ayudar al consumidor a analizar funcionalidades. Si bien se trata de un proceso que analiza dos especificaciones como entrada, la diferencia con este trabajo se encuentra en que es aplicada a especificaciones de servicios web y en esta propuesta se

utiliza una OAS y un DAC, este último es un documento técnico personalizado y definido por el desarrollador. El mismo, no solamente documenta a la aplicación, sino que también asiste en un paso importante en la identificación automática de los cambios en las APIs web.

Diversos tipos de fallas que afectan a la producción de las aplicaciones que consumen APIs web, son analizadas por [Aué y otros, 2018] El trabajo identifica 11 causas generales de fallos, las cuales se pueden atribuir principalmente a los datos de solicitud no válidos o faltantes. Consideramos que algunos de estos tipos de fallos también podrían estar relacionados a cambios por evolución de las APIs web. Si los cambios de las APIs web se producen de forma brusca y no informada o documentada, solicitudes que eran válidas dejarán de serlo.

El estudio empírico de [Yasmin y otros, 2020] revela que más del 80% de los cambios que producen "breaking changes" no fueron informados en las versiones de OAS anteriores mediante el uso del protocolo "deprecar-remover", aplicado para anunciar la futura obsolescencia de elementos. En consecuencia, los desarrolladores no tienen la oportunidad de adaptar y/o migrar las aplicaciones consumidoras de manera anticipada y evitar los impactos. Por ello, propuestas como la que nuestro trabajo presenta resultan necesarias.

En [Tashtoush y otros, 2019][Wittern y otros, 2017][SungGyeong y otros, 2014] se presentan herramientas que usando alguna especificación (OpenApi/Swagger o Web IDL), verifican la consistencia y coherencia de las peticiones o solicitudes codificadas en las aplicaciones consumidoras. Las herramientas son específicas para aplicaciones basadas en tecnologías Ajax, Jquery y Javascript. Estas herramientas están dirigidas a identificar errores por solicitudes inválidas, ocasionados principalmente por documentación incompleta de las APIs, como los enumerados por [Aué y otros, 2018]. También pueden ser útiles para identificar algunos de los errores de ejecución causados por la evolución de las APIs, como ser los que implican cambios y modificaciones en los endPoints.

Como se abordó en la Sección 3, existen diversos estudios empíricos, que han identificado y caracterizado los tipos de cambios producto de la evolución de las APIS web. Estos trabajos [Stocker y Zimmermann, 2021][R. Koçi y otros, 2019][Li y otros, 2013][Sohan y otros, 2015][Espinha y otros, 2015][Fokaefs y otros, 2011] sirven para catalogar, clasificar y analizar los cambios, pero en ninguno de ellos se presentan propuestas de identificación, asistencia o manejo destinada a los desarrollares de aplicaciones consumidoras de APIs web.

7. Amenazas a la validez y limitaciones

En cuanto a la validez interna, nuestra propuesta requiere de las OAS de las APIs web. Mientras las OAS a utilizar por el verificador estén correctamente elaboradas y actualizadas, el proceso de verificación y el DAC identificarán los cambios definidos. En el caso de OAS incompletas, con errores o desactualizadas, el verificador podría no detectar los cambios y/o informar cambios incorrectos. En cuanto a la validez externa, dado que el proceso de verificación solo requiere de entradas documentos JSON (OAS y DAC) nuestra propuesta es independiente de todo detalle de implementación (lenguajes de programación, frameworks, sistemas operativos, etc.) de las aplicaciones

consumidoras. Por otro lado, para aquellas APIs web cuyas especificaciones respondan a otros esquemas (WADL, API Blueprint, RAML), existen diversos procesos automáticos de conversión entre especificaciones, lo cual amplia la aplicación de nuestro enfoque.

Observamos algunas limitaciones en nuestra propuesta. Como se dijo anteriormente es válida para aplicaciones que utilicen APIs web cuya especificación se encuentre disponible en formato OAS. Además, requiere que el desarrollador analice y registre en el DAC, por un lado, los datos de la aplicación consumidora y además los elementos que utiliza de cada una de las APIs web. Esto demandará tiempo y esfuerzo inicialmente, sin embargo, es posible compensarlo en el futuro, en la tarea de descubrimiento de cambios en las APIs una vez que la aplicación está en línea.

8. Conclusiones

Este trabajo presentó un enfoque para identificar un conjunto de cambios que ocurren frecuentemente en APIs web en aplicaciones consumidoras. La propuesta consiste en un Diccionario de Aplicaciones Consumidoras de APIs y un proceso que automáticamente identifica siete tipos de cambios de APIs web. El DAC y la OAS más reciente de una API web seleccionada, representan las entradas del proceso. El mismo analiza ambos archivos y en caso de encontrar diferencias informa el tipo de cambio y los elementos que han sido modificados o ya no están disponibles, como así también los nuevos valores, cuando es posible.

El DAC también sirve para documentar a la aplicación web consumidora de manera personalizada, es definido por el desarrollador e incluye únicamente la información requerida por la aplicación. Además, puede ser utilizado para otros procesos o integrarse a herramientas en el proceso de desarrollo.

El DAC es compatible con el estándar OpenAPI, debido a que su contenido posee una estructura equivalente a OAS, pero incluye únicamente los campos necesarios para describir la aplicación consumidora. Esto significa que el diccionario es comprendido fácilmente por los desarrolladores familiarizados con OAS.

El trabajo continúa con la extensión del diccionario ya sea para incorporar nueva información de la aplicación, como así también para adaptar su estructura a nuevos campos en las nuevas versiones de OpenAPI. No se descarta trabajar en la automatización de la creación del diccionario, lo cual representaría una tarea menos para el desarrollador. La extensión del diccionario posibilitará ampliar las capacidades del verificador, es decir aumentar la cantidad y la naturaleza de los tipos de cambios a identificar.

Los tipos de cambios han sido abordados por estudios previos, sin embargo, escasos están orientados a APIs web, nuestro trabajo se enfoca específicamente sobre aplicaciones web consumidoras de APIs web.

Referencias

Aué, J., Aniche, M., Lobbezoo, M., & van Deursen, A. (2018). “An Exploratory Study on Faults in Web API Integration in a Large-Scale Payment Company”. In ICSE-SEIP '18: 40th International Conference on Software Engineering: Software

Engineering in Practice Track (pp. 13-22). Association for Computing Machinery (ACM). https://doi.org/10.1145/3183519.3183537

Brito A., Valente M.T, Xavier L. (2020) “You broke my code: understanding the motivations for breaking changes in APIs,” Empirical Software Engineering 25, 1458–1492.

Chaturvedi, A. Tiwari A., Binkley D. and Chaturvedi S.(2021) "Service Evolution Analytics: Change and Evolution Mining of a Distributed System," in *IEEE Transactions on Engineering Management*, vol. 68, no. 1, pp. 137-148. doi: 10.1109/TEM.2020.2987641.

Curbera F., Duftler M., Khalaf R., Nagy W., Mukhi N., and Weerawarana S.. (2002) “Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI,” Internet Computing, vol. 6, no. 2, pp. 86–93, 2002.

Dekel U. and Herbsleb J.D. (2009) “Reading the documentation of invoked API functions in program comprehension” ICPC’09, pp. 168–177.

Dig D. and Johnson R.E (2006) “How do APIs evolve? A story of refactoring.” In Journal of Software Maintenance, vol. 18, pp. 83-107.

Eilertsen A.M. and Bagge A.H. (2018) “Exploring API / Client Co-Evolution”, Gothenburg, Sweden 2nd International Workshop on API Usage and Evolution.

Espinha T. Zaidman A. and Gross H.G (2014) “Web API growing pains: Stories from client developers and their code”. IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). IEEE CS, pp. 84–93.

Espinha T., Zaidman A., and Gross H.G. (2015) “Web API Fragility: How Robust is Your Mobile Application?” in Proceedings of the IEEE MOBILESoft, pp. 12–21.

Fokaefs M., Mikhaiel R., Tsantalis N, Stroulia E., Lau A. (2011) “An Empirical Study on Web Service Evolution”. IEEE International Conference on Web Services.

Jezek K. and Dietrich J.(2017) “API Evolution and Compatibility: A Data Corpus and Tool Evaluation”. Journal of Object Technology , 2:1–23. https://doi.org/10.5381/jot.2017.16.4.a2

Lamothe M., Guéhéneuc Y.G, and Shang W.(2021) ” A Systematic Review of API Evolution Literature”. ACM Comput. Surv. 54, 8, Article 171 .36 pages. https://doi.org/10.1145/3470133

Li J, Xiong Y., Liu X., and Zhang L. (2013) “How does web service API evolution affect clients?” 20th Conf. on Web Services (ICWS). IEEE, pp. 300–307.

Maalej W. and Robillard M.P. (2013) “Patterns of knowledge in API reference documentation,” IEEE Trans. Softw. Eng., vol. 39, no. 9, pp. 1264–1282.

Peffers K, Tuunanen T., Rothenberger A. and Chatterjee S. (2007) “A design science research methodology for information systems research”. Journal of Management Information Systems, 24(3). Pp 45–77 https://doi.org/10.2753/MIS0742-1222240302

R. Koçi R., Franch X., Janovic P and Abelló A (2019) “Classification of Changes in API Evolution”. IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)

Robbes R., Lungu M. and Janes A.(2019) “API fluency” ICSE-NIER '19: Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results. pp 97–100. https://doi.org/10.1109/ICSE-NIER.2019.00033

Sohan S.M., Anslow C, Maurer F. (2015) “A Case Study of Web API Evolution”. IEEE World Congress on Services. Doi: 10.1109/SERVICES.2015.43.

Stocker M. and Zimmermann O. (2021) “From Code Refactoring to API Refactoring: Agile Service Design and Evolution”.Conférence Symposium and Summer School on Service-Oriented Computing Pages 174-193 - Éditeur Springer, doi 10.1007/978-3-030-87568-8_11.

SungGyeong B., Hyunghun Ch., Inho L., and Sukyoung R. (2014). SAFEWAPI: web API misuse detector for web applications. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering . Pages 507–517. DOI:https://doi.org/10.1145/2635868.2635916

Tashtoush T., AlRashdan M.N., Salameh O. and Alsmirat M. (2019) "Swagger-based jQuery Ajax Validation," IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC) pp. 0069-0072, doi: 10.1109/CCWC.2019.8666542.

Vinoski S.(2008) “Restful web services development checklist,” IEEE Internet Computing, vol. 12, no. 6, pp. 96–95.

Wittern E. (2018).”Web APIs—Challenges, design points, and research opportunities”. In Proceedings of the 2nd International Workshop on API Usage and Evolution. ACM Press, New York, NY, 18–18.

Wittern E., Ying A.T.T, Zheng Y, Dolby J. and Laredo J.A. (2017) "Statically Checking Web API Requests in JavaScript," IEEE/ACM 39th International Conference on Software Engineering (ICSE) pp. 244-254, doi: 10.1109/ICSE.2017.30.

Xavier L., Brito A., Hora A., Valente M.T. (2017) Historical and Impact Analysis of API Breaking Changes: A Large-Scale Study. IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). Doi 10.1109/SANER.2017.7884616.

Yasmin J., Tian Y. and Yang J. (2020) "A First Look at the Deprecation of RESTful APIs: An Empirical Study," in IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, Australia, pp. 151-161.doi: 10.1109/ICSME46990.2020.00024